ЩАСЛИВСЬКИЙ НАВЧАЛЬНО-ВИХОВНИЙ КОМПЛЕКС

О С Т А П Е Ц Ь
Володимир Степанович

ПОЗАКЛАСНА РОБОТА З ПРОГРАМУВАННЯ У ШКОЛІ

навчально-методичний посібник

2007 рік

Схвалено: науково-методичною радою
Київського обласного інституту
післядипломної освіти педагогічних кадрів

Рецензенти: **Валерій Анатолійович Федорчук**,
завідуючий
центром інформаційних технологій навчання
Київського обласного інституту
післядипломної освіти педагогічних кадрів

Ірина Вільївна Скляр,
викладач УФМЛ КПІ ім. Т. Шевченка,
редактор газети ″Інформатика″,
Заслужений учитель України

Посібник видано за сприяння
Бориспільської районної Ради
та благодійного фонду «Борисове поле»

Анотація: У посібнику на основі 20-річного досвіду роботи досліджується питання ефективності вивчення алгоритмізації та програмування у загальноосвітніх школах. На базі задач, що пропонувались на ІІ-у та ІІІ-у етапах Всеукраїнських олімпіад з інформатики у Київській області, розглядається методика побудови математичних моделей, опису, дослідження та оцінки алгоритмів, складання та тестування програм. Порушуються питання вивчення у загальноосвітніх школах деяких складних і цікавих тем, зокрема опрацювання багаторозрядних чисел, методу динамічного програмування та теорії графів. Посібник адресується учням старших класів, що цікавляться основами програмування, та учителям інформатики.

І. ЗАМІСТЬ ПЕРЕДМОВИ

До шкільних навчальних предметів вже два десятиріччя належить базова дисципліна "Основи інформатики". За цей час погляди на шкільну інформатику пройшли велику трансформацію. У перші роки відповідно до принципу академіка А.П. Єршова: “шкільна інформатика повинна йти від математики“ курс зводився до алгоритмізації та програмування. Але з появою потужних персональних комп'ютерів та поширенням інформаційних технологій він поступово набирав технологічного характеру. Сьогодні згідно універсального профілю 83% навчального часу в школах відводиться на вивчення операційних систем та прикладного програмного забезпечення загального призначення.

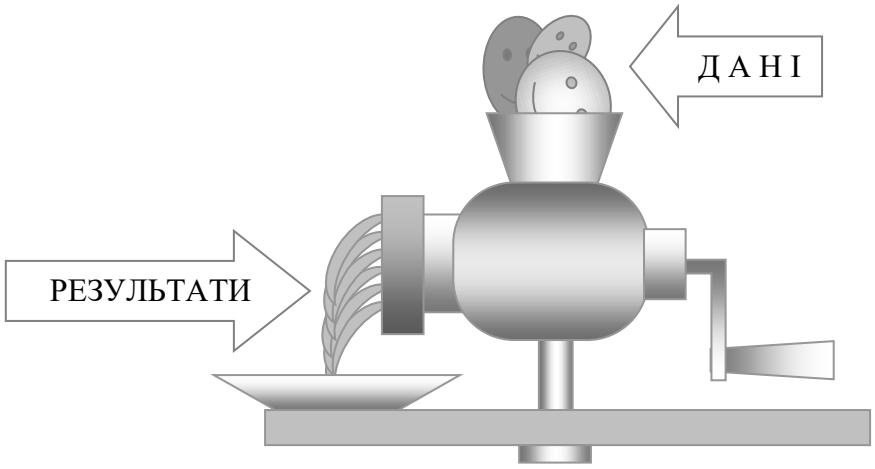
І все ж стержневим залишається розділ "Алгоритмізація та програмування". Шкільну інформатику не можна зводити до безсистемної маси знань про комп’ютер та його застосування, адже, як і в інших навчальних предметах, мова повинна йти про знайомство з наукою. Як відомо, будь-яка галузь знань лише тоді може у повному розумінні бути наукою, коли вона включає математичні методи дослідження. Звідси перший висновок: розділ “Алгоритмізація і програмування“, який по суті є розділом прикладної математики, не може втратити актуальності в курсі інформатики.

Але поглянемо на питання ще з одного боку. За відомим висловом М.В. Ломоносова, математику ще й тому слід вивчати, що вона розум до ладу приводить. Це – постулат, який ніким не ставився під сумнів. Математику, хоча б в обсязі загальноосвітньої школи, вивчають всі люди, бо вона є невід’ємною складовою загальної середньої освіти. На наш погляд, інформатика, як прикладна галузь знань, *дає своєрідне уточнення та узагальнення не тільки таких понять, як інформація та алгоритм, а й багатьох інших загальноосвітніх понять, від лінгвістичних, наприклад, поняття алфавіту та мови, до природничих і математичних, як, наприклад, поняття числа, величини, типу, або виду, виразу, формули, функції, процедури тощо*. Іншими словами, *інформатика, покликана робити наступний за математикою крок у “приведенні розуму до ладу“*.

Про що тут ідеться? Візьмемо поняття “алфавіт“. У початкових класах його розуміють лише як множину знаків, що позначають звуки конкретної національної мови. Згідно тези Р. Декарта, *точні визначення поняттям звільняють людство щонайменше від половини його помилок*, тому дорослі люди повинні розуміти навіть такі звичні й знайомі поняття, як “алфавіт“ ширше, а саме - як набір символів для висловлення думок у певній мові. Учнів корисно, а головне, легко переконати, що до українського алфавіту на рівних правах, наприклад, з буквою “а“, входять арабські цифри, розділові знаки і т.д. Якщо формування такого уявлення про алфавіт не передбачено в початкових класах та на уроках рідної мови, то вивчення мови програмування вже оправдано хоча б цим. Але, на наше глибоке переконання, будь-який учитель-класовод чи мовник повинні у свій час узагальнити учням і поняття алфавіту, і поняття мови, бо

не можливе розуміння процесу пізнання без принципу “від споглядання до узагальнення“, адже аналіз та синтез – це α і ω мислительсьного процесу.

Давайте порівняємо поняття величини чи функції у математиці та фізиці – з одного боку, та в інформатиці – з іншого. Величина у першому випадку має характеристики *ім’я* та *значення*, про *тип* майже не згадується, адже з точки зору математики і фізики величини фактично бувають лише числовими, правда, крім скалярних розрізняють ще й векторні. З програмування учні дізнаються про кодування величин, про потребу їх оголошення та опису, про різноманіття типів, особливо у мові Паскаль, у якому тільки цілочисельних типів кілька. На перший погляд це дивує, але коли пояснити принципи представлення величин у пам’яті комп’ютера, здивування проходить. Ім’я величини в математиці не обговорюється. Час – *t*, швидкість – *v* і т.д. Мови про позначення величини, як деякого набору допустимих символів, нема. Про це говориться в інформатиці. Як виявляється ім’я, чи ідентифікатор може бути не тільки у величини, а й у файла, папки, людини врешті. Коли пояснюєш, що ім’я, в тому числі і ваше – це лише позначення, деякий набір символів, зникає абсурдна і шкідлива для наукового світогляду містика імен. Учні перестають дивуватись і погоджуються, коли їм говорять, що ім’я “Володимир“ для американського індіанця так само є лише набором символів, як для нас “Чингачгук“. У одній із передач “Что, где, когда?“ говорилось, що в момент пізнання істини людина відчуває здивування. У нас ідеться саме про це – про *процес пізнання через здивування*.



Мал. 1

Необхідно хоча б коротко сказати про поняття функції. Пригадаймо, як складно було його усвідомити нам самим за класичним означенням у курсі математичного аналізу. Математика розглядає дуже формально і обмежено уявлення про функцію, бо не акцентує, як інформатика, що функція, не лише залежність між двома змінними, при якій кожному значенню однієї відповідає єдине значення другої, а й *дія*, яка певне значення аргументу перетворює у цілком визначене значення функції. Для учнів цікаво, що термін “дія“ тут слід розуміти, як “автомат“, подібно тому, як електром’ясорубка – це автомат для перетворення м’яса у фарш. У процесі таких міркувань у свідомості учнів виникає колосальний світоглядний скачок, багато різних за формою явищ зводяться, так би мовити, “до спільного знаменника“. А якщо пригадати, що математика, на-

віть вища, практично обмежується розглядом функцій двох числових аргументів, а інформатика, точніше розділ "Алгоритмізація і програмування", розглядає поняття функції багатьох аргументів, причому, не тільки числових типів, а саме *поняття функції у певному розумінні зводиться до поняття процедури*[1], то можна сказати, що абстрактне математичне поняття функції в інформатиці набуває живий, зрозумілий без складного математичного апарату образ. От і спробуй не погодитись, що інформатика, як сказано вище, дуже доречно доповнює математику у "приведенні розуму до ладу", тобто у формуванні наукового світогляду.

Так склалось, що сьогодні домінують дискретні способи зберігання, опрацювання і передачі інформації з допомогою комп'ютерних технологій, а тому інформатика, не зважаючи на свій дуже молодий вік, опинилась буквально в центрі уваги. *Курс інформатики покликаний не тільки забезпечувати реалізацію дуже важливої функції цементування в єдине ціле всієї сукупності сучасних базових знань, а й формувати окремі компоненти загальної освіти - логічне, алгоритмічне і структурне мислення, необхідні тепер будь-якій пересічній людині*.

Безперечно, ідучи в ногу з часом, слід враховувати стрімке удосконалення обчислювальної техніки, доступність сучасних комп'ютерів, нове і ефективне програмне забезпечення, ширше використовувати дидактичні можливості інформаційних технологій. Але сподіваємось, що вищенаведені міркування переконали читача, що шкільну інформатику ні в якому випадку не слід зводити тільки до курсу користувача. Глибоко помилкова також думка, яку висловлюють інколи навіть авторитетні та знані люди, про те, що на часі зміщувати акценти від так званої "предметизації", характерної для старої, ще радянської школи, до формування сучасної інформаційної культури, яка повинна забезпечити оперативність добування і використання інформації.

мал. 2

Але, відштовхуючись від уявлення про комп'ютер, як інструмент для інтенсифікації та автоматизації розумової діяльності, слід не забувати, що крім стандартних функцій, пов'язаних із використанням ППЗ загального призначення, він володіє важливою властивістю – здатністю до розширення обсягу влас-

[1] Пригадавши, що функції та процедури – це підпрограми, можна зробити висновок: функція – це процедура, яка повертає *єдиний результат*, в той час, як процедура може мати кілька результатів, або не мати їх зовсім.

них можливостей завдяки програмуванню. У зв’язку з цим *необхідно не скорочувати, а, навпаки, нарощувати присутність алгоритмізації, причому, як шляхом збільшення відведеної кількості навчальних годин, так і наданням їй домінуючої функції у всіх темах курсу*.

Наведені вище загальні міркування про місце інформатики у сучасній системі освіти не можна відділяти від акцентуванні на місці розділу "Алгоритмізація та програмування" в курсі інформатики. Точка зору відомого ученого С. Пейперта полягає в тому, що *дитина повинна програмувати комп'ютер* і, роблячи це, *не тільки опановувати часточкою найсучаснішої техніки, але й прилучатись до багатьох глибоких ідей природознавства, математики, а також до мистецтва інтелектуального моделювання*.

Отже, програмування – це не екзотична частина шкільної інформатики, не данина моді, або чийсь снобізм. Програмування –це арматура інформатики, а через неї і всієї сучасної інформаційної культури. Кожна сфера, чи галузь має свої передові рубежі, перспективу та стратегічні напрями. Всім відомо, що без систематичної і продуманої позакласної роботи досягти значних успіхів у вивченні будь-якого предмету не можливо. Позакласна робота з програмування, не зважаючи на те, що предмет "програмування" відсутній у шкільних навчальних планах, відіграє, на наш погляд, роль перспективи та стратегічного напрямку в інформатиці. Такими міркуваннями ми намагались пояснити науково-методологічну базу предмету "інформатика". Нижче, розглядаючи так звані олімпіадні задачі, сконцентруємось на пошуках ефективних методів вивчення програмування в загальноосвітній школі.

II. СУТЬ І ЗАВДАННЯ СУЧАСНИХ УЧНІВСЬКИХ ОЛІМПІАД З ІНФОРМАТИКИ. КОЛИ, ЯК ТА З ЧОГО ПОЧИНАТИ ПІДГОТОВКУ?

Питання досить очевидне, але зовсім не риторичне. Відповідь має кілька складових. Пропедевтичну роботу з програмування бажано починати приблизно з сьомого класу, бо, з одного боку, в учнів необхідно встигнути сформувати дуже стрункy, чітку і об'ємну наукову систему знань, з іншого – необхідна математична культура мислення з'являється в учнів не раніше цього віку. Цілком зрозуміло, що доречно зосередитись на підготовці до олімпіад з інформатики (читай - програмування), адже олімпійський дух, атмосферу змагання та мотивацію успіху найкраще досягати в цій формі позакласної роботи. Початковими є І-й та ІІ-й, тобто шкільні та районні етапи.

Але в попередньому розділі вже зверталась увага на те, що під олімпіадами з інформатики треба розуміти олімпіади з відсутнього в шкільному навчальному плані предмету. Термін “Учнівські олімпіади з інформатики” давно потребує уточнення. Насамперед не можна ігнорувати той факт, що сам курс має дві компоненти: алгоритмізацію (програмування) і так званий *курс користувача*, причому акценти все більше зміщуються у бік останнього, для підтвердження досить навести динаміку змін у програмі за кілька останніх років. На-

приклад, на тему “Глобальна мережа Інтернет та її можливості” спочатку відводилось дві оглядові години, потім кількість їх була збільшена до шести, і, нарешті, у програмі, діючій у 2002-2003 навчальному році [1][2], вона доведена до 16-18 годин, причому, незалежно від типу операційної системи та техніки. Тому напрошується цілком логічне питання: що слід розуміти під олімпіадами з інформатики? Адже можна мати на увазі як курс користувача, так і курс алгоритмізації та програмування. Цілком можливий і третій варіант – змішаний. Це перша причина для уточнення.

Традиційні олімпіади, які ми матимемо на увазі далі, проводяться не з обчислювальної техніки чи інформатики, навіть не з розділу “Основи алгоритмізації та програмування”, а лише *відповідно до цього розділу*. Більше того, враховуючи, що на III-IV етапах пропонуються завдання та ставляться умови, що часто унеможливлюють їх розв’язання на базі шкільних курсів математики й основ інформатики, розглядаються й оцінюються лише програми, написані мовами Pascal та C, - це *олімпіади з програмування*, причому їх рівень інколи вищий аналогічних олімпіад для студентів. Тут очевидне протиріччя як із діючими програмами, так і з здоровим глуздом. Це друга причина для уточнення терміну "учнівські олімпіади з інформатики", причому, вона важливіша за першу, бо стосується рівня олімпіад.

Може виникнути питання: а чи варто уточнювати термін і міняти зміст вже класичної форми роботи з предмету? Думається, що варто, цьому теж є кілька важливих причин. Насамперед слід констатувати, що в силу діючих правил на I-II рівнях, найбільш масових, сьогодні спостерігається формальність підготовки та проведення, тому, на жаль, частина шкіл і районів не виставляють своїх команд на II та III етапи, або формують їх з випадкових людей. Це можна зрозуміти, адже забезпечити належний рівень викладання предмету через відсутність програмного забезпечення, літератури і недостатню кваліфікацію вчителів інформатики (в переважній більшості сільських шкіл це сумісники), не вдається. Тим більше, що олімпіади, як уже згадувалось вище, вузькопрофільні. Враховуючи проект [2], що дозволяє переведення вивчення інформатики з 10-11 у 7-9 класи, можна прогнозувати погіршення становища, адже учні середніх класів масово не готові сприймати програмування на необхідному для олімпіад рівні. Тоді ще більше закріпиться абсурдна тенденція, коли шкільні олімпіади з інформатики проводяться не для учнів загальноосвітніх шкіл, а лише *від їх імені*.

Не зважаючи на це, на вищих етапах та на міжнародному рівні, українські школярі щороку успішно виконують запропоновані завдання. Але це здебільшого представники спеціалізованих шкіл всеукраїнського рівня при університетах.

На наш погляд, вищесказане переконує в необхідності уточнення терміну “учнівські олімпіади з інформатики”, перегляду форм і методів їх підготовки та проведення, тобто зміни акцентів і узгодження рівня із шкільним предметом "інформатика". Такі олімпіади очевидно менш сприятимуть виявленню претен-

[2] Тут і далі номер у квадратних дужках відповідає номеру використаної літератури

дентів для участі на міжнародному рівні, але вони можуть стати розумною і корисною альтернативою.

Не акцентуючи далі на цьому, поставимо іншу мету – спробуємо сформулювати і узагальнити вимоги до олімпіад з інформатики в традиційному розумінні (далі – просто *олімпіади з інформатики*) та згідно сучасних вимог накреслити шляхи їх підготовки й проведення на нижчих (I-III) етапах.

Не зважаючи на окремі міркування, що висловлювались багатьма авторами на цю тему у різних педагогічних періодичних виданнях, на сьогодні вона ще практично не досліджена і не висвітлена в достатньому обсязі, нема єдиних у різних регіонах і на різних етапах проведення критеріїв та підходів до підготовки і проведення олімпіад, хоча потреба в методичних посібниках для вчителів та наявності інструктивно-методичних рекомендацій дуже гостра. Розуміючи, що тема надто широка, обмежимось її фрагментарним оглядом і орієнтовними методичними порадами, не вдаючись до рекомендацій, які повинні спиратись на нормативну базу та педагогічні дослідження.

Слід звернути увагу на суттєві проблеми. Перша проблема полягає у *надто великому розриві між рівнем підготовки юних програмістів згідно Державного стандарту та діючої програми і рівнем теоретичної та практичної підготовки з програмування, необхідної для участі навіть у I-II етапах олімпіади*. Програма шкільного курсу інформатики орієнтована на комбіноване вивчення "Курсу користувача" та "Алгоритмізації і програмування", але з ухилом на підготовку користувача [1]. Програмування в ній представлено лише оглядово. Ще багато років тому у [3], критикувався підхід до викладання шкільної інформатики в одному з пробних підручників. Процитуємо кілька не вихоплених з контексту, а характерних, рядків: ”…світовий досвід навчання роботі з ПЕОМ у середніх навчальних закладах показує, що програмістами стають 5-7 відсотків учнів, які цілеспрямовано переймались цим профілем. Тому й питома вага програмування у курсі інформатики на повинна перевищувати 10 відсотків. Та й вивчати програмування треба так, щоб якомога ширше показати можливості ЕОМ.” Виходить, для пересічного школяра достатньо лише початкового рівня навичок програмування. Але до участі в олімпіадах, які вимагають високого рівня культури програмування, залучаються виключно учні загальноосвітніх шкіл.

Пропоновані на III-IV, а останніми роками і на нижчих етапах, завдання вимагають знання ряду тем, що виходять за межі шкільних програм з інформатики та математики. Учням пропонуються задачі, розв’язування яких потребує не обізнаності, а свідомого розуміння і вільного використання відомостей з аналітичної геометрії, комбінаторики, теорії графів, інших математичних дисциплін, що вивчаються переважно у вузах а не в середніх навчальних закладах, причому не завжди на перших курсах і переважно на математичних та технічних спеціальностях. Стали необхідними також вміння користуватись *рекурсивним методом, методами довгої арифметики, перебору та динамічного програмування, вміння оцінювати програми на ефективність та швидкодію*. Але наведемо одну характеру цитату [4]: “… метод динамічного програмування для скінченого простору рішень справді досить простий, а його ідея зрозуміла на-

віть школярам з пересічною підготовкою. Але й годі вимагати від них самостійно додуматись за кілька годин до того, що було останнім словом науки лише 35 років назад". Як видно, вона стосується доступності для учнів методу динамічного програмування – одного з виділених курсивом кілька рядків вище розділів програмування. Перш за все вона дуже суперечлива. Чи справді він досить простий, та ще й для школяра з пересічною підготовкою? А як же з "… годі вимагати від них самостійно додуматись за кілька годин"? І, врешті, навіщо учням, навіть під час олімпіад давати завдання, для розв'язування яких необхідно знати, розуміти і вміти застосовувати те, що було останнім словом науки лише 35 років назад? Очевидно, що олімпіади з інформатики з самого початку поміщені в логічний вакуум, який знаходиться між влучними висловленнями Г.Г. Науменка [3] та О.Б. Рудика [4], що підтверджує сформульовану вище першу проблему. Очевидно, необхідно звузити цей логічний вакуум до тонкої і прозорої плівки, через яку пересічний учень може побачити горизонти програмування, а при бажанні чи потребі легко подолати її. Тому позакласну роботу з інформатики слід вести якісно, як додатковий навчальний предмет, причому, не тільки для забезпечення успішної участі учнів у олімпіадах, а також, щоб розвинути в них алгоритмічну культуру, логічне мислення, дати комп'ютерну грамотність, підготувати до навчання у вузах.

Друга проблема – *відсутність вікової диференціації.* Адже у програмуванні важко класифікувати задачі, як по віку, так і по складності необхідного теоретичного матеріалу. Через це на олімпіадах з інформатики здебільшого даються однакові задачі для 8-11 класів. Спробуємо знайти один із варіантів розв'язання цієї проблеми, навівши приклади завдань, окремих для 10 та 11 класів[3]:

10 клас

1) *Системи числення* (***Notations10***) Дано натуральне число X у четвірковій системі числення. Описати алгоритм його переводу у вісімкову систему числення.
2) *Просте число* (***SimplNumb10***) Описати алгоритм перевірки, чи буде простим дане натуральне число.
3) *Сходинки* (***Ladder10***) По східцях (21 сходинка) вгору піднімається десятикласник. Описати алгоритм підрахунку кількості способів дістатись до верхньої сходинки, якщо за один крок він може піднятись не більше, ніж на 3 сходинки.
4) *Обмін місцями* (***ExchangPlac10***) Потрібно описати стратегію обміну місцями *k* значків "×" та "o" на полі, що має вигляд смужки з *2k+1* клітинок (див. мал.), якщо "хрестики" можуть рухатись тільки в напрямку вправо, а "нулики" можуть рухатись тільки в напрямку вліво, стаючи на сусідню клітинку, якщо вона вільна, або "перестрибнувши" через різнойменний значок, якщо за ним є вільна клітинка:

×	×	…	×		o	o	…	o

[3] Завдання пропонувались під час теоретичного туру Бориспільської районної олімпіади з інформатики в 1999-2000 році.

11 клас

1) *Системи числення* (***Notations11***) Дано натуральне число *X* у *n*-ковій системі числення. Описати алгоритм його переводу у *m*-кову систему числення, якщо $m \neq n$ і $2<n<16$, $2<m<16$.
2) *Просте число* (***SimplNumb11***) Описати алгоритм підрахунку кількості простих чисел, які знаходяться між двома даними натуральними числами *M* та *N*.
3) *Сходинки* (***Ladder11***) По східцях ($k \leq 30$ - кількість сходинок) вгору піднімається одинадцятикласник. Описати алгоритм підрахунку кількості способів підняття на верхню сходинку, якщо за один крок він може піднятись не більше, ніж на *n* сходинок.
4) *Обмін місцями* (***ExchangPlac11***) Потрібно описати стратегію обміну місцями значків "×" та "o" на квадратному полі з $(2k+1)\times(2k+1)$ клітинок (див. мал., $k=3$), якщо кожен "хрестик" може рухатись тільки вправо або тільки вниз, а кожен "нулик" може рухатись тільки вліво або тільки вгору, стаючи на сусідню клітинку, якщо вона вільна, або "перестрибнувши" через різнойменний значок, якщо за ним є вільна клітинка:

×	×	×	×	×	×	×
×	×	×	×	×	×	×
×	×	×	×	×	×	×
×	×	×		o	o	o
o	o	o	o	o	o	o
o	o	o	o	o	o	o
o	o	o	o	o	o	o

Прокоментуємо наведені завдання. Перш за все в них замість вимоги написати програми згідно з даними технічними умовами, пропонується описати алгоритми, що важливо для теоретичного туру. Кожне завдання можна звести до прозорого, не дуже громіздкого алгоритму, опис якого наближений до звичайного програмного матеріалу з предмету, зводиться до формулювання ідеї, не потребує глибоких теоретичних знань. Нарешті, умови вибрано так, що завдання для 11-го класу є логічним ускладненням завдання для 10-го класу з таким же номером. На наш погляд, обидві сформульовані вище проблеми тут якщо не вирішені цілком, то, принаймні, зведені до мінімуму.

Ми не даремно щойно описали завдання теоретичного туру олімпіади. Очевидно у цьому випадку очікується практичний тур. Причини проведення олімпіади в два тури – це окрема розмова, яку ми перенесемо нижче. Тут акцентуємо увагу на третій проблемі, яка полягає у невизначеності *співвідношення між теоретичною та практичною частинами роботи над задачами*. З одного боку не можна ігнорувати роботу над описом математичної моделі, складанням, доведенням та оцінкою алгоритму, з іншого боку, тільки ступінь практичної реалізації у вигляді діючої програми дозволить зробити правильну оцінку. Офіційні суперечки про те, в якій формі - теоретичній, практичній чи комбінованій слід проводити олімпіади з інформатики, давно вважаються завершеними на користь практичної. Тому перевірка письмових робіт і аналіз програм тепер не

обов’язкові. Єдиним визнаним продуктом учасника олімпіади є програма. Така загальноприйнята позиція. Переконливо висловився про це доцент Дніпропетровського державного університету, головний експерт XIII Всеукраїнської олімпіади з інформатики О. Л. Хижа, [5]: "*Акцент усе більше зміщується від «краси алгоритму» до «діючої програми»... Журі олімпіад уже не розглядає тексти програм учасників*. Оцінювання діючої програми здійснюється як оцінювання *діючого «чорного ящика»*: на вхід подаються вхідні дані, а на виході має бути правильний результат. *Хитромудрі обмеження, що містяться в умові, задають необхідний рівень ефективності розв'язання"*.

Якщо в умовах задач вимагається описати алгоритм, то, очевидно, постає ще одна проблема, не логічного, як три попередні, а технічного характеру: *як описувати алгоритм*. Цій проблемі нижче присвятимо цілий розділ. Через відсутність однакових умов на місцях, *участь в олімпіадах з інформатики не може бути обов’язковою*, що на практиці, як згадувалось вище, має місце дефакто. Отже дані про участь і результати олімпіад з інформатики не можуть бути обов’язковими компонентами в статистичній звітності та при підведенні підсумків роботи навчальних закладів.

Проведення районних олімпіад з інформатики за зразком обласного етапу не досконале. Перш за все для організації районного туру, особливо у сільських районах, важко знайти єдиний населений пункт з достатньою кількістю комп'ютерів. Значна частина учнів, особливо 9-10 класів у листопаді-грудні ще не має достатніх навичок програмування, тому для значної (інколи переважної) кількості учасників практичний тур з використанням комп’ютерів не надто актуальний. Всі етапи Всеукраїнської олімпіади з інформатики мають спільну головну мету: відбір учасників наступного етапу. Але, очевидно, що будь-яка олімпіада покликана також популяризувати предмет, сприяти піднесенню загального рівня знань, умінь та навичок. У зв'язку з цим можна погодитись, що різні етапи олімпіади з інформатики повинні мати власні локальні завдання. Наприклад, перший (шкільний) етап у першу чергу покликаний виявити тих, хто цікавиться предметом більше за інших, володіє елементами програмування. Другий (районний) етап може відібрати серед призерів І-го етапу тих, хто має здібності до точних наук, схильність абстрактно, логічно, структурно та алгоритмічно мислити, має певні навички програмування. Третій (обласний) етап дає змогу відібрати учнів, які мають значний досвід програмування та перспективи удосконалення. IV-й етап виявляє обдарованих у галузі програмування. Умовно можна об'єднати у дві групи I-II та III-IV етапи. Очевидно, враховуючи це, послідовно від етапу до етапу слід змінювати методику підготовки завдань та умови проведення змагань. Так на IV-у етапі склалась традиція проводити два практичні тури. На наш погляд, на II-у етапі резонно також проводити два тури, але перший – теоретичний, а другий – практичний. По-перше, теоретичний тур дозволить виявити учасників, що мають здібності до точних наук, схильність абстрактно, логічно, структурно та алгоритмічно мислити, по-друге, він допоможе відібрати тих, хто має навички програмування. Олімпіади – це “колективний забіг”, це – марафон для особливо сильних. Тому після теоретичного туру до наступних змагань будуть допущена лише порівняно невелика кількість

здатних до такої специфічної сфери діяльності, як програмування. Проведенню районних етапів олімпіади з інформатики є кілька перепон. Серед них першою можна вважати включення районних олімпіад з інформатики, до етапів Всеукраїнської олімпіади, а це не дозволяє змінювати умови проведення. Проте ця перепона легко усувається - можна вважати теоретичний тур попереднім і неофіційним. Залишається лише дозволити в другому турі не брати участь навчальним закладам, які не можуть виставити достатньо підготовлених учасників.

Такі зміни дозволять проводити практичний тур районної олімпіади за всіма правилами, як зауважено в наведеній вище цитаті [5], можна буде змістити акцент від "краси алгоритму " до " діючої програми ", а журі одержить можливість виключно оцінювати дієвість програми, застосувавши в тестах обмеження, що містяться в умові і задають необхідний рівень ефективності розв'язання.

Існує ще одна, можливо, не дуже суттєва перепона впровадженню теоретичного туру. Справа в тому, що графік олімпіад та конкурсів-захистів у секціях МАН досить тісний, тому важко організувати два тури олімпіади з інформатики.

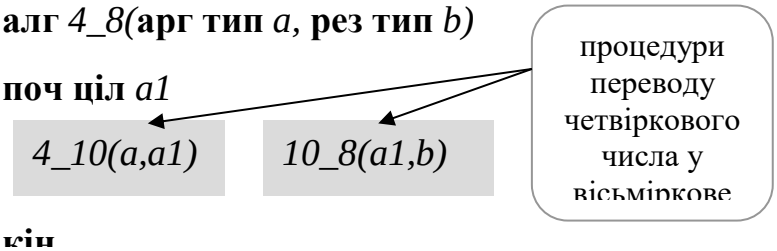
Результати районних олімпіад слід розглядати на семінарі вчителів інформатики, підготувавши методичні рекомендації. Вони можуть бути посібником для вчителів і учнів для наступної позакласної роботи з предмету. Наведемо зразок таких методичних вказівок, використавши наведені вище завдання для теоретичного туру районного етапу олімпіади.

Зауважимо, що завдання для теоретичного туру районної олімпіади були складені так, що відповідні номери завдань для 10-го і 11-го класів були споріднені за змістом. Враховуючи це ми будемо розглядати їх разом.

III. МЕТОДИЧНІ ПОРАДИ ЩОДО ПІДГОТОВКИ ТА ПРОВЕДЕННЯ РАЙОННИХ ОЛІМПІАД З ІНФОРМАТИКИ

Задача 1. Накладемо обмеження на умову першої задачі (***Notations***). Не порушуючи загальності, будемо вважати, що основи систем числення n і m задовольняють умовам $2<n<9$, $2<m<9$. Зробимо це для того, щоб не користуватись цифрами, відмінними від арабських, прийнятих у десятковій системі числення. Обмежимось також числами, які можна опрацьовувати на ПК без послуг "довгої арифметики".

При складанні алгоритму переведення натурального числа з n-річної системи числення в m-річну систему числення, де m та $n \in N$ і $m \neq n$, (для 10-го класу - з четвіркової у вісьміркову) доцільно взяти за базові два алгоритми, які переводять натуральне число з десяткової системи числення у n-річну ($n \in N$ і $n \neq 10$) і навпаки. Тоді алгоритм (представлений у вигляді процедури, так будемо робити і надалі) може мати такий вигляд:

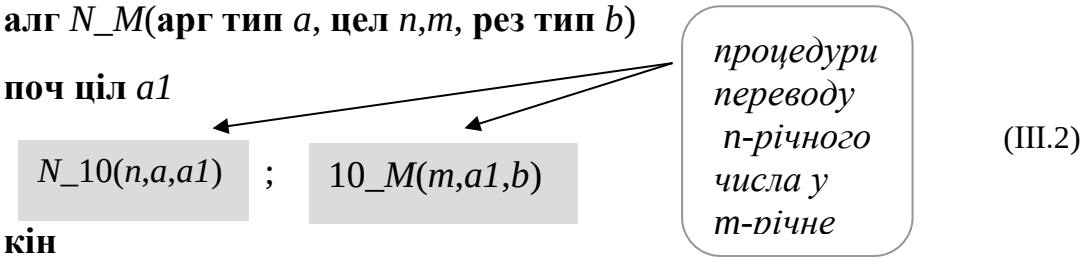


(III.1)

Тут аргумент і результат поки що мають невизначений тип, умовно позначений зарезервованим словом "**тип**", адже в різних системах числення числа представляються послідовністю символів, зміст якої різний. Конкретизуємо тип параметрів дещо пізніше.

Може здатись, що при такому підході є зайвою десяткова система числення, яка відіграє роль посередника. Але це тільки на перший погляд. Адже такий алгоритм є частинним випадком більш загального алгоритму, який забезпечить переведення натурального числа з *n*-річної системи числення в *m*-річну систему числення.

Звичайно, для складання саме такого алгоритму треба дещо змінити його вигляд:

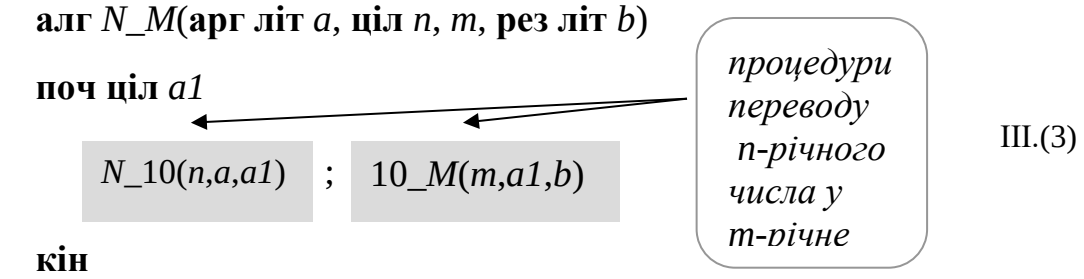


(III.2)

Серед аргументів доведеться вказувати ще й натуральні числа *m* та *n*, які є основами вхідної та вихідної систем числення.

А тепер про тип вхідного та вихідного чисел. Їх можна подавати як лінійною таблицею, так і у вигляді літерної величини. Кожен спосіб має свої переваги і недоліки. Нам здається, що більш універсальним є представлення аргументів і результатів у вигляді літерних величин.

Врахувавши ці міркування, отримаємо таку схему алгоритму:



III.(3)

В наведеному нижче алгоритмі виклик процедур та функцій і їх тіла виділені однаковим кольором, що допоможе при розборі алгоритму.

При роботі з алгоритмом, як видно із його заголовка та попередніх пояснень, число-аргумент і число-результат є ланцюжками символів. Це дозволить опрацьовувати числа, записані, наприклад в двійковій системі числення, навіть кількома десятками символів. Враховуючи, що тут описується теоретичний тур олімпіади, запишемо алгоритм переведення числа з *n*-річної системи числення в *m*-річну систему числення умовною навчальною алгоритмічною мовою (УНАМ), у якій прийнято використані процедури та функції записувати під головним алгоритмом, який виділено напівжирним шрифтом:

```
алг N_M(арг літ numb1,  ціл n, m,, рез літ numb2)
поч ціл a
¦  B_10(numb1, n, a); 3_10(a, m, numb2)
кін
алг B_10(арг літ a,цел b,рез ціл n)
поч ціл i,k,l
¦ k:=длин(a);n:=0
¦ пц для i від 1 до k
¦ ¦ n:=n+Літ_Чис(a[i:i])*b**(k-i)
¦ кц
кін
алг 3_10(арг ціл n, k, рез літ a)
поч ціл m, n1, ost, літ a1
¦ n1:=n; a:=""
¦ пц
¦ ¦ ost:=mod(n1,k);n1:=div(n1,k);  a:=Чис_Літ(ost)+a
¦ кц_при n1<=0
кін
алг літ Чис_Літ(арг ціл n)
поч ціл i
¦ знач:=""
¦ вибір
¦ ¦при n=0:знач:=знач+"0"
¦ ¦при n=1:знач:=знач+"1"
¦ ¦при n=2:знач:=знач+"2"
¦ ¦при n=3:знач:=знач+"3"
¦ ¦при n=4:знач:=знач+"4"
¦ ¦при n=5:знач:=знач+"5"
¦ ¦при n=6:знач:=знач+"6"
¦ ¦при n=7:знач:=знач+"7"
¦ ¦при n=8:знач:=знач+"8"
¦ ¦при n=9:знач:=знач+"9"
¦ все
кін
алг ціл Літ_Чис(арг літ a)
поч ціл i, k
¦ вибір
¦ ¦при a="0" : знач:=0
¦ ¦при a="1" : знач:=1
¦ ¦при a="2" : знач:=2
¦ ¦при a="3" : знач:=3
¦ ¦при a="4" : знач:=4
¦ ¦при a="5" : знач:=5
¦ ¦при a="6" : знач:=6
¦ ¦при a="7" : знач:=7
```

```
¦ ¦при а="8" : знач:=8
¦ ¦при а="9" : знач:=9
¦ все
кін
```

Засоби УНАМ прості та наочні, але не достатні для компактного опису навіть таких простих алгоритмів. У цьому легко переконатись, поглянувши на функцію ***Чис_Літ***, де для розшифрування числа, записанного у текстовому вигляді, довелось використати команду вибору, яка займає 12 рядків. Тому нижче наводимо дві функції, записані мовою Turbo Pascal, які переводять двійкове число у десяткове та десяткове у двійкове:

```
program Bin_Dec;
  var a:string; b:integer;
Function Translation_10_2 (n:integer):string;
   var d:integer;s,st:string;
begin
  d:=n;st:='';
  while d<>0 do                                                    begin
    m:=d mod 2;d:=d div 2;Str(m,s);st:=s+st; Translation_10_2:=st end;
end;
Function Translation_2_10(n:string):integer;
   var s:string;
        m,i,k,x,cod:integer;
begin
    s:=n;m:=Length(s);k:=1;x:=0;
    repeat
      Val(s[m],i,cod);  x:=x+i*k;  k:=k*2;  m:=m-1
    until m=0;
    Translation_2_10:=x
end;
Begin
   WriteLn('Ввести двійкове число:'); ReadLn(a);
   WriteLn(Translation_2_10(b));
   WriteLn('Ввести десяткове число:'); ReadLn(b);
   WriteLn(Translation_10_2(a));
end.
```

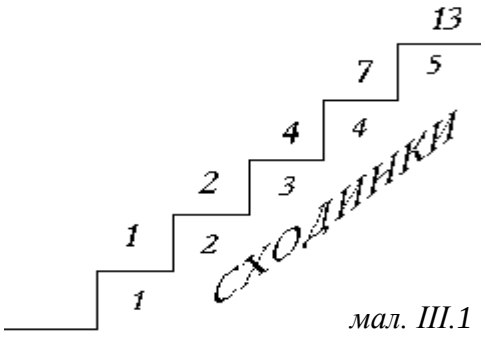
Задача 2. (***SimplNumb***) Алгоритми визначення та опрацювання простих чисел цікаві своєю прозорістю та простотою і в той же час великим смисловим навантаженням. Це видно на прикладі наступного алгоритму:

```
алг лог Prost(арг ціл n)
поч ціл i, m
¦ знач:=так
¦ m:=Int(Sqrt(n)); i:=2
¦ пц поки i<=m и знач
¦ ¦ якщо mod (n, i)=0
¦ ¦ ¦то знач:=ні
```

```
¦ ¦ все
¦ ¦ i:=i+1
¦ кц
кін
```

Алгоритм представлений у вигляді логічної функції. Ідея перевірки числа на "простоту" випливає з означення простого числа. Для перевірки, чи натуральне число *n* просте необхідно випробувати його на подільність на числа з діапазону [*2; n-1*]. Проте алгоритм можна оптимізувати, помітивши, що випробування достатньо проводити на діапазоні [*2; int(sqrt(n))*], де *int(sqrt(n))* - ціла частина арифметичного квадратного кореня з числа *n*.

Скориставшись вищенаведеною алгоритмом-функцією, просто написати алгоритм для задачі ***SimplNumb 11***.



мал. III.1

Задача 3. (***Ladder***) Алгоритм розв'язання задачі напрочуд легкий, хоча це спочатку непомітно. Це дозволило використати дану задачу на III-у етапі Всеукраїнської олімпіади у попередні роки.

Пояснимо алгоритм, скориставшись малюнком III.1.

Очевидно, що першої сходинки можна досягти лише одним способом, відповідно другої - двома. Неважко зрозуміти, що третя сходинка досягається чотирма способами.

А для всіх наступних сходинок кількість способів дорівнює сумі способів досягнення трьох попередніх сходинок. Отже справедлива рекурентна формула: $\begin{cases} C(1)=1; \\ C(2)=2; \\ C(3)=4; \\ n>3, C(n)=C(n-1)+C(n-2)+C(n-3), \end{cases}$

де *C(n)* - кількість способів досягнення сходинки № *n*.

```
алг ціл C(арг ціл n)
поч
¦ вибір
¦ ¦при n=1: знач:=1
¦ ¦при n=2: знач:=2
¦ ¦при n=3: знач:=4
¦ ¦інакше знач:=C(n-1) + C(n-2) + C(n-3)
¦ все
кін
```

Задача 4. (***ExchangPlac***) Варіант цієї задачі для 10-го класу запозичено з [6], де вона описана, як головоломка “Зануда”, а для 11-го класу задача отримана ускладненням цієї головоломки, і як часто буває в таких випадках, зовнішньо зовсім не схожа на першу.

Як легко помітити на конкретних прикладах, стратегія гри “Зануда”, ігровим полем якої є смужка з *2k+1* клітинок (див. мал. ІІІ.2), полягає у послі-

×	×	…	×		o	o	…	o

мал .ІІІ. 2

довному проведенні серій ходів (у таблиці 1 наводиться для смужки із 7 клітинок). Отже, відпадає проблема вибору значка для наступного ходу. Наприклад, як видно із таблиці 1, у п’ятій серії треба тричі підряд ходити значком “×”. Потрібно лише вибрати, який із “хрестиків” повинен переміщуватись в даний мо-

таблиця ІІІ.1

номер серії	ходи, з яких складається серія
1	×
2	oo
3	×××
4	ooo
5	×××
6	oo
7	×

мент, щоб на наступному кроці згідно правил гри також був можливий хід.

Легко помітити, що пуста клітинка повторює всі переміщення значків, але в протилежному напрямку. Тоді стає очевидним, що в алгоритмі легше буде забезпечувати “блукання” пустої клітинки серед “хрестиків” та “нуликів”, ніж самі ходи цих значків. Крім того, якщо “хрестики” замінити числами “-1”, а “нулики” числами “1”, і постійно пам’ятати у вигляді відповідного числа (-1 або 1), яка серія ходів здійснюється в даний момент, то можна легко визначати напрям переміщення пустої клітинки, яку позначимо числом 0.

Напрям руху пустої клітинки (далі число 0) вліво, як і на числовій прямій, позначатимемо від’ємним числом –1, а вправо – відповідно додатним числом 1. Нуль повинен переміщуватись на одну клітинку, якщо напрям співпадає

таблиця ІІІ.2

початкове положення	-1	-1	-1	0	1	1	1
після 1 кроку	-1	-1	0	-1	1	1	1
після 2 кроку	-1	-1	1	-1	0	1	1
після 3 кроку	-1	-1	1	-1	1	0	1
після 4 кроку	-1	-1	1	0	1	-1	1
після 5 кроку	-1	0	1	-1	1	-1	1
. . .	…	…	…	…	…	…	…
Після 11 кроку	1	-1	1	0	1	-1	-1
Після 12 кроку	1	0	1	-1	1	-1	-1
Після 13 кроку	1	1	0	-1	1	-1	-1
Після 14 кроку	1	1	1	-1	0	-1	-1
Після 15 кроку	1	1	1	0	-1	-1	-1

із числом, що стоїть у сусідній клітинці даного напрямку, або на дві клітинки в іншому випадку.

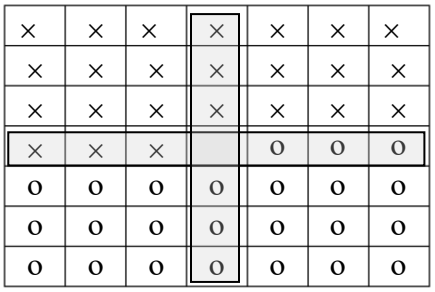
Прослідкувавши за кольорами в таблиці, можна наочно уявити переміщення "хрестиків" та "нуликів", що допоможе зрозуміти алгоритм.

```
алг ExchangPlac(арг ціл n, рез цілтаб M[1:2*n+1])
поч ціл i, l, напрям, номер, цілтаб L[1:2*n+1]
¦ П(n,M)
¦ К_кроків(n,L)
¦ напрям:=1; номер:=n+1
¦ пц для i від 1 до 2*n+1
¦ ¦ Сх(n,L[i], напрям, номер, M, напрям, номер)
¦ ¦ вивід нс, M; напрям:=-напрям
¦ кц
кін
алг П(арг ціл n, рез цілтаб P[1:2*n+1])
поч ціл i
¦ P[n+1]:=0
¦ пц для i від 1 до n
¦ ¦ P[i]:=-1;P[2*n+2-i]:=1
¦ кц
кін
алг К_кроків(арг ціл n, рез цілтаб K[1:2*n+1])
поч ціл i
¦ K[n+1]:=n
¦ пц для i від 1 до n
¦ ¦ K[i]:=i;K[2*n+2-i]:=i
¦ кц
кін
алг Сх(арг ціл n, m, t, s, арг рез цілтаб S[1:2*n+1],рез ціл k,p)
поч ціл i, нап, ном
¦ нап:=t; ном:=s
¦ пц для i від 1 до m
¦ ¦ якщо S[ном+нап]=нап
¦ ¦ ¦то S[ном+нап]:=0; S[ном]:=нап; ном:=ном+нап
¦ ¦ ¦інакше S[ном+2*нап]:=0; S[ном]:=нап; ном:=ном+2*нап
¦ ¦ все
¦ кц
¦ k:=нап; p:=ном
кін
```

Четверте завдання для 11-го класу (***ExchangPlac11***) [8] з першого погляду схоже на відповідне завдання для 10-го класу (***ExchangPlac10***) тільки зовнішньо. Проте досить уважно придивитись до наступних малюнків ІІІ.3 та ІІІ.4, щоб помітити, що воно є просто площинним варіантом завдання для 10-го класу.

На малюнку сірим кольором виділено рядок, що є ігровим полем для задачі ***ExchangPlac10***. Але стовпчик, виділений сірим кольором, є також ігровим полем для цієї задачі. Як видно з таблиці III.2, у сірому рядку буде "блукати" пусте поле (позначене нулем).

мал. III. 3

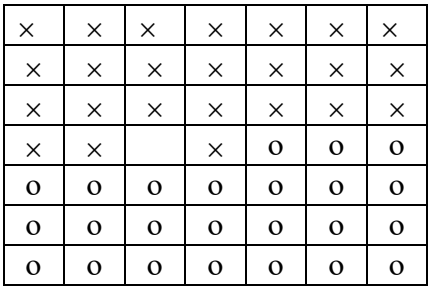


Користуючись цим, ми завжди зможемо виконати завдання ***Exchang-Plac10*** для стовпчика, який містить пусте поле, наприклад після першого ходу в сірому рядку:

Можна попробувати навіть застосувати рекурсію для алгоритму ***ExchangPlac***. Але на заваді цьому стоїть те, що в сірому рядку пусте поле може кілька разів займати одну і ту ж позицію. Це також видно із таблиці III.2. Отже для контролю над цим необхідно мати спочатку заповнений нулями лінійний масив з *2n+1* елементів, у якому відмічається, скільки разів поле з відповідним номером було пусте. Так можна уникнути подвійного опрацювання кожного стовпчика поля.

Проаналізовані завдання теоретичного туру підібрані так, що необхідності у дослідженні та оцінці алгоритмів нема, самі алгоритми також легко описа-

мал. III.4



ти словесно чи у вигляді блок-схем. Учням, навіть 11-х класів, достатньо мінімальних знань про масиви, рекурсії, підпрограми.

IV. ПОРАДИ ЩОДО ПІДБОРУ ЗАВДАНЬ

Надзвичайно важливими у процесі підготовки до олімпіади є ретельність підбору завдань та формулювання їх умов. Розглянемо зразок невдалого підбору. Ці завдання пропонувались для районного етапу в 1999-2000 н.р., отже для практичного туру.

9 клас

Задача 1.9[4] Обчислити $n!=1\cdot2\cdot3\cdot\ldots\cdot(n-1)\cdot n$.

Задача 2.9 Встановити, скільки елементів даного масиву належить проміжку (*a,b*) та знайти порядкові номери і значення цих елементів.

Задача 3.9 Знайти взаємне розташування відрізків *[a,b]* і *[c,d]* на числовій прямій.

Задача 4.9 В японському календарі кожен з 12 послідовних років має назву звіра (пацюк, бик, тигр, заєць, дракон, змія, кінь, вівця, мавпа, півень, собака, кабан), а кожен з 5 років має колір (зелений, червоний, жовтий, синій, чорний). З'ясувати, яка назва 2000 року, якщо 1984 - рік зеленого пацюка.

10 клас

Задача 1.10 Для даного масиву встановити найбільшу довжину послідовності однакових елементів, що розташовані поряд.

Задача 2.10 На інтервалі *(1000, 9999)* знайти всі прості числа, в записі яких сума першої і другої цифр рівна сумі третьої і четвертої.

Задача 3.10 З'ясувати, чи належить точка *(x,y)* кругу радіуса *r* з центром в точці *(a,b)*.

Задача 4.10 Алгоритм обробляє деякі сполучення букв В, П, М. Алгоритм переводить слово ПВПВМВ в слово ПВ, МВМВПВМВ в МВМВ, МВПВПВМВПВ в ПВ, ПВПВ в ПВПВ, МВПВПВПВ в ПВПВ. Для деяких слів, наприклад, ППВ. МВМП, ВВ, ПВПМПВ, алгоритм видає повідомлення "помилка", а). Описати такий алгоритм; б). Який сенс можна надати такому алгоритму?

11 клас

Задача 1.11 Знайти найменше та найбільше числа, які можна подати сумами деяких (можливо всіх) елементів даного масиву.

Задача 2.11 Знайти суму елементів матриці, розташованих на одній з ліній, яка паралельна:

а) головній діагоналі; б) бічній діагоналі.

Задана 3.11 Числа по спіралі. Заповнити квадратну таблицю *T(n,n)* послідовними цілими числами від 1 до n^2, розміщеними по спіралі, починаючи з лівого верхнього кута і рухаючись за годинниковою стрілкою.

Задача 4.11 За координатами вершин опуклого чотирикутника встановити:

а) його вид (квадрат, ромб, прямокутник, паралелограм, трапеція);

б) чи можна в нього вписати коло? в) чи можна навколо нього описати коло?

Завдання не можна вважати вдалими, хоча вони розбиті на класи, включаючи 9-й. По-перше, жодне із завдань не передбачає складання алгоритму чи програми. Формулювання „Знайти найменше та найбільше числа, які можна подати сумами деяких (можливо всіх) елементів даного массиву“ (1.11) зовсім

[4] Для зручності завдання помічені подвійними номерами, розділеними крапкою, наприклад, 3.10, що означає: третє завдання для 10-го класу.

не означає створення програми. По-друге, відсутність чітких формулювань, вимог щодо оцінювання, технічних вимог та тестів.

Але з приводу цих завдань можна висловити не лише негативні, а й позитивні міркування, кожен мінус можна повернути як плюс.

Безперечно, наявність завдань для різних класів окремо, включаючи і 9-й, є плюсом. Адже, коли завдання однакові, змагання для учнів не 11-го класу втрачає зміст, бо вони з самого початку поставлені в програшні умови - одинадцятикласики краще справляться із завданнями через більший досвід та теоретичну підготовку.

Відносна легкість. Серія завдань повинна обов'язково включати завдання різного рівня, в тому числі і "втішальні". Це гуманно, адже частина учасників, які залишаться розчарованими, буде меншою. В підсумковій таблиці буде "спектр" балів, що дозволить вибрати для наступного туру дійсно найсильніших. І, нарешті, не треба забувати, що учасники - переважно учні загальноосвітніх шкіл, які вивчають інформатику у недостатньому обсязі. Звичайно, ті, хто має значно якіснішу підготовку, при цьому не втратять балів і, безперечно, займуть призові місця.

А ось з приводу відсутності чітких формулювань, вимог щодо оцінювання, технічних вимог та тестів дійсно виникає багато думок, які суперечливі, а то й протилежні. Справді, в цих умовах ускладнюється робота журі, як під час проведення, так і в процесі перевірки. Це явний мінус. Технічні вимоги для практичного туру необхідні. Адже повинно бути чітко вказано, яким чином слід забезпечити ввід даних, у якому вигляді видати результати.

Для перевірки практичного туру необхідні тести. Інакше журі вже в процесі перевірки повинне складати свої, які через дефіцит часу можуть бути непродуманими або невдалими. Виникає також проблема вводу тестових даних. Дуже важливе тестування і для перевірки дії програми у екстремальних випадках.

Недоліки такого характеру можна побачити буквально у всіх задачах. Звернемо увагу на найбільш принципові. Як, наприклад, зрозуміти умову задачі 1.9? Визначення факторіала, залежно від технічних вимог, може бути дуже легким (для значення *n*, що не перевищує 18) і надзвичайно складним, коли навіть так звана "довга арифметика" не допоможе.

Наведемо короткі вказівки до розв’язування задач 1.1 – 3.4. Питання про тестування програм розглянемо пізніше. Для написання алгоритмів учням 8-9 класів було дозволено користуватись версією умовної навчальної алгоритмічної мови (УНАМ), реалізованою в пакеті „КУМИР“, що є її інтерпретатором. Детальний опис цієї, по суті, мови програмування є в [7]. При вивченні початкового курсу алгоритмізації корисно використовувати інтерпретатор алгоритмічної, зокрема згаданий пакет „КУМИР“, або інші, наприклад, ALGO[5] . Це дає змогу при виробленні початкових навичок програмування не відволікатись на вивчення англомовних зарезервованих слів, а писати програми (алгоритми), так би мовити, рідною мовою, максимально концентруючись на усвідомленні алго-

[5] Демонстраційна україномовна версія мови Паскаль, автор В. Петрів.

ритму. При описі розв'язань завдань для 10 і 11 класів будемо користуватись програмами, написаними на мові Turbo Pascal, якою користувались учні.

Задача 1.9 Поняття факторіалу відомо і зрозуміло учням (IV.1). Спочатку виникає думка написати циклічний алгоритм. Слід показати його неефективність і спрямувати їх на пошуки рекурентної формули і рекурсивного алгоритму.

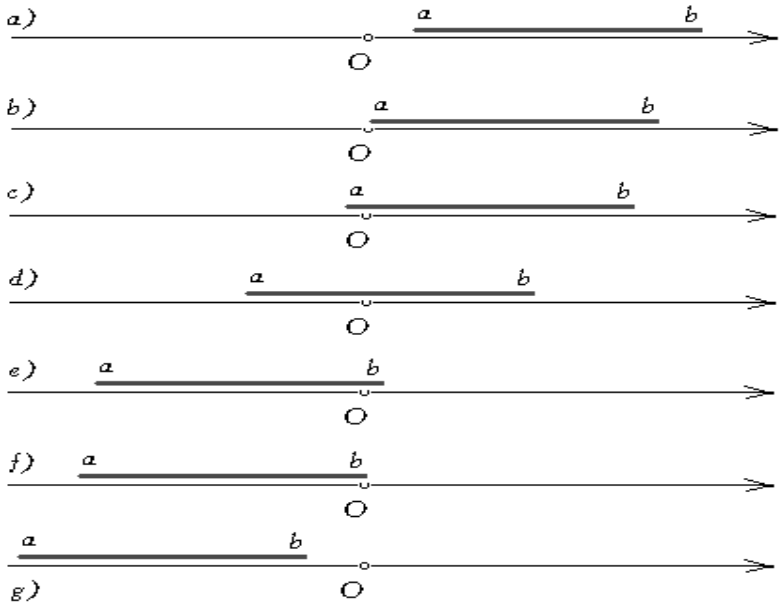
Для усвідомлення, як працює рекурсивний алгоритм, необхідно закріпити його на прикладі алгоритму визначення чисел Фібоначчі (IV.2) та алгоритма розв'язання задачі ***Ladder*** (див. теоретичний тур). Наведемо алгоритм, записаний у вигляді функції:

```
алг нат Факторіал (арг нат n)
поч
   якщо n=0 або n=1                                          IV.1
       то знач:=1
       інакше знач:= Факторіал (n-1)*n
    все
алг нат Фібоначчі (арг нат n)
поч
   якщо n=1 або n=2                                          IV.2
       то знач:=1
       інакше  знач:= Фібоначчі (n-1) + Фібоначчі (n-2)
    все
кін
```

Задача 2.3 Коментарі до цієї задачі зайві. Єдиним блоком виділено ввід масиву та його опрацювання. Пояснення до записів на алгоритмічній мові див. нижче.

```
нат n
вивод нр,"ввести n";ввод n
алг Кільк_елементів_масиву_що_належать_відрізку
поч нат i,  kilk,  ціл a, b,  цілтаб M[1:n]
¦ вивод нр,"ввести a";ввод a; вивод нр,"ввести b";ввод b
¦ нц для i від 1 до n
¦ ¦ вивод нс,"ввести M[",i,"]:";ввод M[i]
¦ кц
¦ kilk:=0
¦ нц для i від 1 до n
¦ ¦ если M[i]>=a и M[i]<=b
¦ ¦   ¦то вивод нр, i, M[i]; kilk:= kilk+1
¦ ¦ все
¦ кц
¦вивод нр,kilk
кін
```

Задача 3.9 Цю задачу проілюструємо малюнком IV.1 та алгоритмом учасника Бориспільської районної олімпіади 1999-2000 н.р., переможця серед восьмикласників, учня Щасливського НВК Хількевича Олексія:



мал. IV.1

```
алг Взаємне_розташування_відрізків(дійс a,b,c,d)
поч
¦ вибір
¦ ¦при a=b або c=d:вивод"Це не відрізки, а точки!"
¦ ¦при a<b і c<d: Варіант(a,b,c,d)  |оскільки в умові сказано,
¦ ¦при a<b і c>d: Варіант(a,b,d,c)  |що [a,b] і [c,d]-це відрізки,
¦ ¦при a>b і c<d: Варіант(b,a,c,d)  |то a не дорівнює b і
¦ ¦при a>b і c>d: Варіант(b,a,d,c)  |c не дорівнює d
¦ все
кон
алг Варіант(дійс a,b,c,d)
поч
¦ вибір
¦ ¦при c>b:вивод"Не мають спільних точок!"
¦ ¦при c=b:вивод"Мають одну спільну точку!"
¦ ¦при a=c і d=b:вывод"Відрізки повністю співпадають!"
¦ ¦при c<b:вывод"Мають спільний відрізок!"
¦ все
кін
```

Виділено допоміжну процедуру "Варіант" та команди її виклику.

Слід зауважити, що задачі 3.9 ізоморфна умова такої задачі: У правильному многокутнику проведено *n* різних діагоналей. Чи є серед них хоча б одна пара таких, що перетинаються? Про ізоморфізм можна спрочитати в [8].

Задача 4.9 Цю задачу також проілюструємо алгоритмом того ж учня:

```
алг Японський_Новий_ Рік (цел n)
 дано   |натуральне n - рік, назву якого треба взнати
```

```
                |літтаб H[1:2] - масив-назва року, де 1-ий елемент-
                | це колір тварини,  а 2-ий елемент - назва тварини
 треба|визнач. назву року, якщо 1984 рік- зеленого пацюка
поч ціл колір, тварина, літтаб H[1:2]
¦ колір:=mod(n,5); тварина:=mod(n,12)
¦ вибір
¦ ¦при колір=4:H[1]:="зелений"     |процес
¦ ¦при колір=0:H[1]:="червоний"    |визначення
¦ ¦при колір=1:H[1]:="жовтий"      |кольору року
¦ ¦при колір=2:H[1]:="синій"       |за остачею
¦ ¦при колір=3:H[1]:="чорний"      |"колір"
¦ все
¦ вибір
¦ ¦при тварина=4:H[2]:="пацюк"  |процес
¦ ¦при тварина=5:H[2]:="бик"    |визначення
¦ ¦при тварина=6:H[2]:="тигр"     |кольору
¦ ¦при тварина=7:H[2]:="заець"    |року
¦ ¦при тварина=8:H[2]:="дракон"  |за
¦ ¦при тварина=9:H[2]:="змія"     |остачею
¦ ¦при тварина=10:H[2]:="кінь"    |"зверь"
¦ ¦при тварина=11:H[2]:="вівця"
¦ ¦при тварина=0:H[2]:="мавпа"
¦ ¦при тварина=1:H[2]:="півень"
¦ ¦при тварина=2:H[2]:="собака"
¦ ¦при тварина=3:H[2]:="кабан"
¦ все
¦ вивод H[1],",",H[2]
кін
```

Задача 1.10 Ця задача відноситься до розряду втішальних. Вона не потребує поширеного додаткового пояснення. Слід переглянути даний масив (звичайно він повинен бути лінійним, що, на жаль, в умові не відмічено), перевіряючи, чи наступний елемент дорівнює поточному. Спеціальний лічильник *k* повинен збільшуватись на 1, якщо наступний елемент такий же, як поточний, та отримувати значення 1 у протилежному випадку. Слід ввести ще один лічильник *k1*, який запам'ятовує останнє не рівне 1 значення лічильника *k*, щоб порівняти його із новим значенням *k*, також відмінним від 1. Відповіддю буде остаточне значення *k1*.

Ускладнивши цю задачу, одержимо серію задач, які можна розв'язувати у порядку підвищення складності. Це буде непоганим тренуванням. Ось деякі завдання, що приводять до нової, складнішої задачі:

1) Вивести на екран значення елемента найдовшої послідовності однакових елементів;
2) Вивести на екран найдовшу послідовність однакових елементів;
3) Вивести на екран в стовпчик найдовші послідовності однакових елементів у порядку зростання кількості елементів.

```
program Z_1_10;
  const n=20;
  var i,k,k1,l:integer;
      MASS:array[1..n] of integer;
begin
  for i:=1 to n do
    Readln(MASS[i]);
  k:=1;k1:=1;l:=MASS[1];
  for i:=2 to n do
    if l=MASS[i] then k:=k+1
    else              begin
      if k1<k then k1:=k;
      k:=1;l:=MASS[i] end;
  if k1<k then k1:=k;
  writeln(k1);
end.
```

Задача 2.10 Для розв'язання цієї задачі слід скористатись в якості підпрограми функцією *PROST* (задача № 2 теоретичного туру). "Відсіяні" з її допомогою прості числа з діапазону *(1000, 9999)* слід перевіряти на рівність сум двох перших і двох останніх цифр. В цьому допоможе відомий алгоритм "Щасливий білет".

Задача 3.10 Ця задача є скоріше не олімпіадною з інформатики, а простою задачею на координатний метод для 9 класу. Адже все зводиться до перевірки умови: $\sqrt{(x-a)^2+(y-b)^2} \le r$. Приводити програму не будемо.

Задача 4.10 Цю задачу можна назвати "дивною". Описати алгоритм для пункту а) неважко, якщо зробити такий аналіз:

ПВ~~ПВМВ~~ $\rightarrow$ ПВ
МВМВ~~ПВМВ~~ $\rightarrow$ МВМВ
~~МВПВПВМВ~~ПВ $\rightarrow$ ПВ
ПВПВ $\rightarrow$ ПВПВ
~~МВПВ~~ПВПВ $\rightarrow$ ПВПВ

Звідси видно, що

1) обробляються лише буквосполучення, які не мають букв, крім "В", "П" та "М", мають парну кількість символів і кожна пара починається не на "В", а закінчується на "В";

2) для одержання результату викреслюються по черзі пари ПВ та МВ, тобто фрагменти типу МВПВ або ПВМВ.

Щодо відповіді на пункт б), то питання, на мій погляд поставлено не коректно. Якщо сенс конкретний, то це із дуже вузької області, не відомої широкому колу людей, якщо ні, то можна придумати багато інтерпретацій.

На мою думку, до цієї задачі необхідно було зробити додаткові пояснення.

Задача 1.11 На цьому прикладі зручно наочно проілюструвати учням початкові етапи розв'язування задачі на ПК (математичну постановку та опис алгоритму).

Перш за все очевидно, що легше буде розглядати різні можливі випадки, якщо упорядкувати даний масив по зростанню. Тоді для унаочнення можна буде скористатись моделлю числової прямої. Можливі такі варіанти розміщення чисел:

На малюнку *a* і b - це найменше і найбільше числа в упорядкованому масиві. Матимемо на увазі, що c) та e) - це випадки, коли лише одне число з масиву має відмінний знак від решти чисел, а b) та f) - випадки, коли найменше чи найбільше число рівне 0.

Очевидно, що у випадку a) найбільша та найменша суми будуть дорівнювати суму всіх чисел та двох найменших (аналогічно, але з точністю до навпаки буде у випадку g)).

У випадку d) результатами відповідно будуть суми всіх додатних та від'ємних чисел. Правда, може бути ще випадок, коли число 0 входить до масиву, але він ситуацію не змінює.

Випадки b), c), e), та f) між собою також схожі, наприклад, у випадку a) найменша сума - це сума числа *a* і наступного за ним числа, а найбільша сума - це сума всіх чисел, крім числа *a*.

```
{Програма визначає найменшу та найбільшу суми, які можна
отримати із елементів лінійного масиву цілих чисел}
Program S_min_S_max;
 var i,m,n,i1,s1,s2,indikator:integer;p:text;
 a:array[1..100] of integer;
 Procedure uporyadkuvannya;{упорядкування масиву}
  begin
   for i:=1 to n do for i1:=i to n do
      if (a[i]>a[i1])and(i<>i1) then   begin
         m:=a[i];a[i]:=a[i1];a[i1]:=m  end;
  end;
 Procedure dodatni(var s1,s2:integer);{всі додатні}
   var q:integer;
  begin
   s1:=a[1]+a[2];for q:=1 to n do s2:=s2+a[q];
  end;
 Procedure vidjemni(var s1,s2:integer);{всі від'ємні}
   var q:integer;
  begin
   s2:=a[1]+a[2];for q:=1 to n do s1:=s1+a[q];
  end;
 Procedure onevidjemne(var s1,s2:integer);{одне від'ємне}
   var q:integer;
  begin
   s1:=a[1]+a[2];for q:=2 to n do s2:=s2+a[q];
```

```
  end;
 Procedure onedodatne(var s1,s2:integer);{одне додатне}
   var q:integer;
  begin
   s2:=a[n]+a[1];for q:=1 to n-1 do s1:=s1+a[q];
  end;
 Procedure mishani(var s1,s2:integer);{більше, ніж по од-
ному додатному і від'ємному}
   var q:integer;
  begin
   q:=1;
   while a[q]<0 do begin s1:=s1+a[q];q:=q+1 end;
   q:=n;
   while a[q]>0 do begin s2:=s2+a[q];q:=q-1 end;
  end;
Begin
 assign(p,'1.dat'); reset(p);read(p,n);
 for i:=1 to n do
  begin read(p,a[i]);write(a[i]:3)end;
 uporyadkuvannya;
 m:=0;
 for i:=1 to n do
  if a[i]<0 then m:=m+1;{кількість від'ємних}
 if m=0 then indikator:=1;if m=n then indikator:=2;
 if m=1 then indikator:=3;if m=n-1 then indikator:=4;
 if (m<n-1)and(m>1) then indikator:=5;
 s1:=0;s2:=0;{визначення процедури}
 case indikator of
  1:dodatni(s1,s2);
  2:vidjemni(s1,s2);
  3:onevidjemne(s1,s2);
  4:onedodatne(s1,s2);
  5:mishani(s1,s2);
 end;{виконання процедури}
 writeln;write('min сума',s1,'max сума',s2);
End.
```

Для кращої читабельності програми розгляд окремих випадків, розглянутих вище, оформлено у вигляді процедур. Масив зчитується з файлу *1.dat*. Упорядкування забезпечується виділеною зеленим кольором процедури. Суттєвим у програмі є виділений бузковим кольором фрагмент, де підраховується кількість від'ємних елементів (*m*). Залежно від значення *m* вибирається значення величини *indikator*,яка у виділеному сірим кольором фрагменті виконує відповідну процедуру.

Задача 2.11 Ця задача цінна не своєю оригінальністю, а тим, що вона може бути першою в цілому ланцюжку цікавих задач на масиви, розміщених у

порядку складності (цікавості). Домовимось вибирати вузловий елемент, через який буде проходити лінія, паралельна одній з діагоналей матриці (матриця мається на увазі квадратною) вводом значень $1<i<n$ та $1<j<n$, $i \neq j$.

Нехай для ілюстрації (*таблиця IV.1*) в даній матриці ($n=7$) матимемо діагональні елементи рівні 0, решта 1, вузловий елемент підкреслено, потрібні лінії виділено сірим кольором.

Видно, що в такому випадку шукані суми S_1 та S_2 будуть відповідно рівні 4 та 6.

Алгоритм їх знаходження не потребує додаткового пояснення, тому можна просто привести текст програми.

таблиця IV.1

0	1	1	1	1	1	0
1	0	1	1	1	0	1
1	1	0	1	0	1	1
1	1	1	0	1	1	1
1	1	0	1	0	1	1
1	0	1	1	1	0	1
0	1	1	1	1	1	0

```
Program Suma_Po_Diagonalam;
 var i,n,i1,x,y,x1,y1,s1,s2:integer;p:text;
 a:array[1..100,1..100]of integer;
Begin
 assign(p,'2.dat');reset(p);read(p,n);
 {вивод масива}
 for i:=1 to n do                        begin
   writeln;
   for i1:=1 to n do                     begin
     read(p,a[i,i1]);write(a[i,i1]:3)end
                                         end;
 {кінець виводу масива}writeln;
 Write('Введіть координати x,y:');read(x); go-
toxy(30,n+2);read(y);
x1:=x;y1:=y;
 {шукаємо початок діагоналі}
 while (x1>1)and(y1>1) do
   begin x1:=x1-1;y1:=y1-1 end;
 while(x1<n+1)and(y1<n+1)do{опускаючись,шукаємоs1}
   begin s1:=s1+a[y1,x1];x1:=x1+1;y1:=y1+1 end;
 write('Сума по основній діагоналі дорівнює', s1);
 x1:=x;y1:=y;
 {шукаємо початок діагоналі}
 while(x1>1)and(y1<n) do
   begin x1:=x1-1;y1:=y1+1 end;
 while x1<n+1)and(y1>0) do{опускаючись шукаємо s2}
   begin s2:=s2+a[y1,x1];x1:=x1+1;y1:=y1-1 end;
```

```
writeln;
 write('Сума по бічній діагоналі дорівнює', s2);
End.
```

Під час розбору завдань зручно вводити деякі означення.

Означення 1. **Вписаним** у квадратну матрицю **числовим прямокутником** називатимемо множину елементів матриці, розташованих на лініях, які паралельні діагоналям матриці і обмежені елементами, один з індексів яких дорівнює 1 або *n* (як виділено в таблиці *IV.1*, самі ці лінії називатимемо **сторонами**, а елементи, що їх обмежують - **вершинами** вписаного у квадратну матрицю числового прямокутника.

таблиця IV.2

0	1	1	1	1	1	0
1	0	1	1	1	0	1
1	1	0	1	0	1	1
1	1	1	0	1	1	1
1	1	0	1	0	1	1
1	0	1	1	1	0	1
0	1	1	1	1	1	0

Означення 2. **Периметром** вписаного у квадратну матрицю числового прямокутника називатимемо суму елементів, які лежать його сторонах.

Означення 3. **Площею** вписаного у квадратну матрицю числового прямокутника називатимемо суму його периметра та елементів, що знаходяться в середині нього (виділені зеленим кольором).

Задача 1. Знайти периметр вписаного в квадратну матрицю числового прямокутника.

```
Program Perymetr;
  uses crt;
  var i,n,i1,x,y,x1,y1,s1,s2:integer;p:text;
      a:array[1..100,1..100]of integer;
Begin
  clrscr;
  assign(p,'2.dat');reset(p);read(p,n);
  for i:=1 to n do begin writeln;{вивод масива}
  for i1:=1 to n do                    begin
     read(p,a[i,i1]);write(a[i,i1]:3) end
  end;writeln;            {кінець виводу масива}
  repeat
    Write('Введіть координати x,y: ');
    read(x,y);
  until (x<0)or(y<0)or(x=y)or(x=n-i+1)
```

```
  {шукаємо вузлову вершину в першому стовпчику}
    x1:=x;y1:=y;
    while (x1>1)and(y1>1) do
      begin x1:=x1-1;y1:=y1-1 end;
    if x1=1then while(y1>1)do
      begin x1:=x1+1;y1:=y1-1 end;
   {опускаючись, шукаємо s1}
    while x1<n+1 do                        begin
      s1:=s1+a[y1,x1];x1:=x1+1;y1:=y1+1 end;
    x1:=x1-2;
   {опускаючись шукаємо наступну s1:=сторону+s1}
    while y1<n+1 do                         begin
       s1:=s1+a[y1,x1];x1:=x1-1;y1:=y1+1 end;
    y1:=y1-2;
   {опускаючись шукаємо наступну s1:=сторону+s1}
    while x1>0 do                           begin
       s1:=s1+a[y1,x1];x1:=x1-1;y1:=y1-1 end;
    x1:=x1+2;
   {опускаючись шукаємо наступну s1:=сторону+s1}
    while y1>1 do                           begin
       s1:=s1+a[y1,x1];x1:=x1+1;y1:=y1-1 end;
    else write(s1:4);
    readkey
End.
```

Спочатку програма від вузлового елемента *a[x,y]*, піднімаючись по стороні, паралельній головній діагоналі, шуває вузлову вершину вписаного прямокутника, що лежить в першому стовпчику, причому, якщо цей процес приводить до вершини у першому рядочку, то вершини у першому стовпчику досягаєм, спускаючись по стороні, паралельній бічній діагоналі.

Задача 2. Знайти площу вписаного в квадратну матрицю числового прямокутника.

Розв'язання цієї задачі спрощується, якщо відняти від суми всіх елементів матриці суму площ "трикутників", які залишаються після "вилучення" з матриці вписаного числового чотирикутника. Ця ідея реалізована в наступній програмі.

```
Program Ploscha;
 uses crt;
 var i,n,c,i1,x1,y1,m:integer;p:text;
 a:array[1..100,1..100]of integer;
Procedure Left_up;{шукаємо площу лівого верхнього трикут-
ника який не входить в площу шукану квадрата}
 begin
  for i:=1to x1-1do for i1:=1to x1-ido m:=m-a[i,i1]
 end;
```

```
Procedure Right_up;{шукаємо площу правого верхнього три-
кутника який не входить в площу шукану квадрата}
 begin
    for i:=1 to n-x1 do for i1:=x1+i to n do
      m:=m-a[i,i1]
 end;
Procedure Right_down;{шукаємо площу правого нижнього три-
кутника якўй не входить в площу шукану квадрата}
 begin
  c:=1;
  for i:=n downto n-x1 do begin c:=c+1;
    for i1:=n-x1+c to n do m:=m-a[i,i1]end
  end;
Procedure Left_down;{шукаємо площу лівого нижнього три-
кутника якўй не входить в площу шукану квадрата}
 begin
   c:=0;
   for i:=x1+1 to n do begin c:=c+1;
     for i1:=1 to c do m:=m-a[i,i1] end
 end;
Begin
  clrscr;
  assign(p,'2.dat');reset(p);read(p,n);
  for i:=1 to n do begin writeln; {вивод масива}
   for i1:=1 to n do begin
     read(p,a[i,i1]);write(a[i,i1]:3);m:=m+a[i,i1]
     end end;writeln;
 {кінець виводу масива}
  Write('Введіть координати x,y: ');
  read(x1);gotoxy(30,n+2);read(y1);
 {шукаємо початок діагоналі}
  while (x1>1)and(y1>1) do begin
    x1:=x1-1;y1:=y1-1      end;
  if x1=1 then {прив'язка до лівого краю}
    while (y1>1) do begin x1:=x1+1;y1:=y1-1 end;
  Left_up;Right_up;Right_down;Left_down;
  write(m:3);
  readkey
End.
```

Задача 3.11 Скориставшись цією задачею, продовжимо практикум на масиви. Але спочатку обговоримо саму задачу.

1	2	3	4	5	6	7
24	25	26	27	28	29	8
23	40	41	42	43	30	9

22	39	48	49	44	31	10
21	38	47	46	45	32	11
20	37	36	35	34	33	12
19	18	17	16	15	14	13

Як бачимо, назва задачі („Числа по спіралі“) цілком оправдана. Нескладна, давно відома, але діти її розв'язують із задоволенням, бо вона ховає в собі два сюрпризи: спочатку здається не зовсім легкою і очевидною, а в кінці виникає захоплення від легкості, про яку і не здогадувався. Такий ефект виникає через вкладені цикли. В програмі виділено синім кольором операторні дужки зовнішнього циклу, решта циклів обмежені попарно однокольоровими операторними дужками. Величини *x, y, kinets* є перемикачами, а *m* зберігає число, яке потрібно занести в чергову комірку.

```
program Spiral;
      var i, i1, x, y, m, n, kinets:integer;
       a:array[1..100,1..100]of integer;
Begin
  read(n); m:=1; kinets:=n; x:=1; y:=1;
   for i:=1 to n do                              begin
      for i1:=x to kinets do    begin
          a[i1,y]:=m;m:=m+1 end;
       x:=i1;
       for i1:=y+1 to kinets do
           begin  a[x,i1]:=m;m:=m+1 end;
        y:=i1;
        for i1:=x-1 downto n-kinets+1  do
           begin  a[i1,y]:=m;m:=m+1 end;
         x:=i1;
         for i1:=y-1 downto n-kinets+2  do
             begin  a[x,i1]:=m;m:=m+1 end;
          y:=i1;inc(x);kinets:=kinets-1         end;
    for i:=1 to n do                                    begin
        writeln; for i1:=1 to n do write(a[i1,i]:3) end;
End.
```

Задачу можна дещо ускладнити і зробити цікавішою, якщо взяти вже заповнений довільно масив $n \times n$, і, проходячи той же шлях, що і в попередній задачі, тобто по спіралі, перетворити масив у лінійний. Інакше, розгорнути, рухаючись по спіралі квадратну матрицю у лінійну. Можна піти й далі, коли початковий масив брати не квадратний, а прямокутний. Отже отримано дві цікаві задачі, які разом із задачею *3.11* утворюють ланцюжок однотипних, наростаючих по складності і цікавості задач. Досить допомогти учням розв'язати першу задачу, як вони самі вже зможуть розв'язати решту, причому почуватимуть себе першовідкривачами.

А ми сформулюємо ще одну, звичайно ж цікавішу задачу.

Задача. Написати програму згортання лінійного масиву розмірністю *n*, в прямокутний, якщо це можливо, так, щоб початковий масив укладався у кінцевий у вигляді спіралі.

Помічаємо, що коли *n* - просте число, то задача не має розв'язків, а, наприклад, при *n=36* вона згортається кількома способами: *2×18, 3×12, 4×9, 6×6.* Отже виникає додаткова задача, зовсім іншого характеру: знайти всі можливі представлення натурального числа у вигляді добутку двох натуральних множників. Це вже задача на повний перебір.

Цікавий пошук оптимальності цього перебору. Звичайно можна вважати, що *m×n* і *n×m* - це один і той же добуток. Тоді перебір значно скоротиться. Його можна провести для масиву із *n* елементів, випробовуючи подільність числа *n* на всі натуральні числа, починаючи з 2 і кінчаючи $[\sqrt{n}]$.

```
Program  Skrutka;
 var i,i1,n,k,number:integer;a:array[1..100,1..100] of integer;
Procedure Variant;
begin
  while i<=round(sqrt(n)) do                    begin
    if n mod i=0 then                         begin
      a[number,1]:=i;a[number,2]:=trunc(n/i);
      number:=number+1                        end;
     i:=i+1;                                    end;
  number:=number-1;
end;
Begin
   write('Ввести довжину масиву, який треба скрутити : ');
   read(n);i:=2;number:=1;
   Variant;
    writeln;
   if (n<4)or(number=0) then write('Невірно задані дані,
                                  або скрутити неможна!')
   else                                                  begin
  writeln('Можна розкласти стількома способами : ',number);
  for i:=1 to number do
      writeln(a[i,1],' * ',a[i,2]);
  p:=1;y:=1;x:=1;
  for i:=1 to number do                                 begin
   for i1:=1 to round(a[i,1]/2) do                      begin
     for k:=x to a[i,2]-i1+1 do
       begin b[y,k]:=p;p:=p+1 end;
     x:=k;
     for k:=y+1 to a[i,1]-i1+1 do
       begin b[k,x]:=p;p:=p+1 end;
     y:=k;
     i f  p<n+1 then
```

```
    for k:=x-1 downto i1 do
      begin b[y,k]:=p;p:=p+1 end;
  x:=k;
  if p<n+1 then
    for k:=y-1 downto i1+1 do
      begin b[k,x]:=p;p:=p+1 end;
  y:=k;x:=x+1;                                        end;
  writeln;readkey;
  for i1:=1 to a[i,1] do                    begin
    writeln;
    for k:=1 to a[i,2] do  write(b[i1,k]:4)   end;
  p:=1;y:=1;x:=1;
                                                      end;
                                                      end;
End.
```

В програмі жовтим кольором виділена процедура *Variant*, що шукає всі варіанти представлення числа *n* у вигляді добутку двох натуральних множників. В операторні дужки, виділені червоним кольором, вміщено фрагмент програми, який виводить на екранвсі варіанти згортання лінійного масиву у прямокутний масив. Для того, щоб програму було легше розібрати, окремі закінчені частини цього фрагмента обмежені також операторними дужками однакового кольору.

Програми до задач *1.11 - 3.11* та похідних від них задач підготував учень 11 класу Щасливського НВК Капелюшний Юрій.

Задача 4.11 Слід зауважити, що ця задача сформульована також не зовсім коректно, чим майстерно скористався переможець Бориспільської районної олімпіади 1999 р. учасник IV етапу XII-ї Всеукраїнської олімпіади з інформатики (м. Херсон) учень Щасливського НВК Поліно Володимир (його програма наводиться нижче). Справа в тому, що умову четвертої задачі краще було б сформулювати так:

За координатами вершин опуклого чотирикутника встановити:

а) його вид (квадрат, ромб, прямокутник, паралелограм, трапеція тощо);

б) чи можна в нього вписати коло?

в) чи можна навколо нього описати коло?

Відомо, що в чотирикутник можна вписати коло, якщо суми довжин протилежних сторін рівні. Ця умова перевіряється легко, досить визначити за формулою $d=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$ довжини сторін чотирикутника, де (x_1,y_1) і (x_2,y_2) - кінці цих сторін, задані координатами, а потім попарно їх додати і порівняти одержані числа.

А от питання про існування описаного навколо чотирикутника кола вирішується важче, адже умовою описання є рівність суми протилежних кутів чотирикутника 180°. Використовуючи координати вершин чотирикутника, можна знайти косинуси (за теоремою косинусів) та синуси (за скалярним добутком векторів), а потім тангенси протилежних кутів. У мові програмування

Turbo Pascal є стандартна функція арктангенс. Але користуючись таким методом, значення кутів можна знайти з великою похибкою, а це ускладнює перевірку можливості описання кола навколо даного чотирикутника.

Але відсутність слова "тощо" в умові задачі спростило її. Адже для перерахованих типів чотирикутників перевірка пункту в) простіша. Навколо паралелограма і ромба описати коло не можна, якщо вони не прямокутники або квадрати. Щодо трапеції, то для описання навколо неї кола досить, щоб вона була рівнобічна. Отже навколо квадрата, прямокутника та рівнобічної трапеції можна описати коло. А всі вони мають властивість - їх діагоналі рівні між собою.

Якщо ж чотирикутник не буде жодного із перерахованих типів, то він може бути якогось іншого типу, про що програма, природньо, також повинна повідомити.

```
program MultiPageWork;
  uses Crt,Graph;
  var d1,d2,d3,d4,d5,d6:real; i,d,r,j:integer;
     vp,op,ans:string; A:array[1..4,1..2] of real; rt:text;
 begin
  ClrScr;
  Assign(rt,'dan_z4.pas'); {Читання даних з зовнішнього файлу}
  reset(rt);
  for i:=1 to 4 do              begin
   for j:=1 to 2 do   begin
     read(rt,A[i,j])   end       end
  close(rt);
  for i:=1 to 4 do                  begin {Вивод даних на екран}
     for j:=1 to 2 do       begin
     write(A[i,j]:4:2,' ')  end;
   writeln                          end;
  {Знаходження довжин сторін чотирикутника за методом координат}
  d1:=sqrt(sqr(A[1,1]-A[2,1])+sqr(A[1,2]-A[2,2]));
  d2:=sqrt(sqr(A[2,1]-A[3,1])+sqr(A[2,2]-A[3,2]));
  d3:=sqrt(sqr(A[3,1]-A[4,1])+sqr(A[3,2]-A[4,2]));
  d4:=sqrt(sqr(A[4,1]-A[1,1])+sqr(A[4,2]-A[1,2]));
  {Знаходження довжин діагоналей чотирикутника за методом координат}
  d5:=sqrt(sqr(A[1,1]-A[3,1])+sqr(A[1,2]-A[3,2]));
  d6:=sqrt(sqr(A[2,1]-A[4,1])+sqr(A[2,2]-A[4,2]));
  {Нехай спочатку буде трапеція}
  ans:='трапеція';vp:='NO';op:='NO';r:=0;
  {Якщо сума довжин протилежних сторін чотирикутника рівна,
     то можна вписати коло}
  if (d1+d3)=(d2+d4) then vp:='YES';
   {Якщо довжини діагоналей чотирикутника рівні, то можна описати коло}
  if d5=d6 then op:='YES';
  {Якщо всі сторони чотирикутника рівні, то це ромб}
```

```
   if (d1=d2) and (d2=d3) and (d3=d4) then     begin
      ans:='ромб';r:=1;
   {Якщо подвоєний квадрат сторони дорівнює квадрату діагоналі,  то цей ромб, ще й квадрат }
      if (d1*d1+d2*d2)=(sqr(A[1,1]-A[3,1])+sqr(A[1,2]-A[3,2])) then
       ans:='квадрат';                                  end;
     {Якщо протилежні сторони чотирикутника рівні,то це паралелограм}
     if (d1=d3) and (d2=d4) and (r=0) then        begin
        ans:='паралелограм';
     {Якщо сума квадратів суміжних сторін дорівнює квадрату діагоналі чотирикутника,  то цей паралелограм, ще й прямокутник}
     if (d1*d1+d2*d2)=(sqr(A[1,1]-A[3,1])+sqr(A[1,2]-A[3,2])) then
      ans:='прямокутник'                              end;
     {Вивод результатів на екран}
      writeln(ans);  writeln('Чи можна описати:  ',op);
     writeln('Чи можна вписати:  ',vp);
     readkey;
end.
```

Розглянуті вище задачі теоретичного та практичного турів районного етапу олімпіади з інформатики наведені, як цікаві при пошуках відповідей на питання, поставлені в розділах II-IV.

V. ПИСЬМОВИЙ ОПИС АЛГОРИТМІВ – НЕВІД'ЄМНА СКЛАДОВА РОЗВ'ЯЗУВАННЯ ЗАДАЧ З ПРОГРАМУВАННЯ В ШКОЛІ

Вище наголошувалась важливість вміння робити письмовий опис алгоритмів при виконанні завдань з програмування. Тут розглянемо це питання детальніше.

На жаль ми вже звикли до тенденції зміщення акцентів шкільної інформатики від алгоритмізації та програмування у бік "користувацької" психології. Мовляв, сучасні комп'ютери не вимагають спеціальної підготовки, а в програмістах необхідність і зовсім невелика. В зв'язку з цим хочеться звернути увагу на те, що курс інформатики покликаний не тільки забезпечувати реалізацію дуже важливої функції цементування в єдине ціле всієї сукупності сучасних базових знань, а й формувати окремі компоненти загальної освіти - логічне, алгоритмічне і структурне мислення, необхідні тепер будь-якій пересічній людині. Навчати користуванню комп'ютером можна по мірі необхідності, попутно із виконанням цих основних завдань.

Якщо прийняти вищесказане, доведеться погодитись, що опанування інформатикою вимагає дуже серйозних і специфічних вмінь і навичок письмово формулювати думки. На жаль цьому не сприяють ні сучасні технології навчання, які роблять акцент на усному, а не на письмовому мовленні в молодших класах, на уроках рідної та іноземних мов, ні зниження вимог до оформлення

записів при розв'язуванні математичних задач. Свого часу масовий приплив у школи через відсутність кваліфікованих вчителів інформатики інженерів-електронників, викладачів вузів та вчителів фізики був вимушеним. На жаль ці спеціалісти, не маючи якісної педагогічної освіти, поряд із корисними прийомами та технологіями навчання принесли в шкільну інформатику традицію недооцінки методики навчання. Учитель інформатики і сьогодні цінується переважно з огляду на знання техніки, прикладних програм та програмування, а знання ним загальної і спеціальної методики, дидактики та психології учнів враховуються недостатньо.

Особливо важливе володіння методикою викладання тем, що стосуються алгоритмізації і програмування. Перш за все, всі нові поняття, які тут зустрічаються, необхідно чітко пояснювати і закріплювати на достатній кількості прикладів. Треба враховувати, що в алгоритмізацію не можна механічно переносити відомі учням з інших дисциплін поняття числа, величини, функції і т.д.

Все вищесказане приведено, як аргумент проти загальноприйнятої сьогодні схеми оцінки роботи юного програміста, коли до уваги береться тільки програма, яка перевіряється виключно на комп'ютері. Проте така схема оправдана лише при тій умові, що програма *діюча* і повністю, а не частково, розв'язує поставлену задачу. Очікуваний результат можна не одержати, як через наявність непринципової помилки в програмі, так і через некоректну роботу тестуючої програми, можливі помилки в тестах або збої в роботі комп'ютера. В кожному випадку виставлені бали не будуть об'єктивними. Уникнути необ'єктивності в оцінюванні можна, перевіривши попередні етапи роботи учня, тобто *письмовий опис* аргументів, результатів, математичної моделі та алгоритму. Це так же природно, як перевірка в математиці опису розв'язування задачі поряд із отриманою відповіддю.

Ці міркування стосуються не тільки роботи на уроках, а й олімпіад та інших змагань з інформатики. Свого часу офіційною відмовою від "ручної" перевірки текстів алгоритмів та програм стала її велика трудоємкість, яка була наслідком відсутності єдиних вимог до оформлення письмової частини роботи та опису алгоритмів. На жаль таке становище збереглось до сьогодні, бо і тепер в дуже багатьох викладачів інформатики домінує снобістська позиція програміста над позиціями методиста.

Тоді можливо є сенс у виробленні цих єдиних і обгрунтованих вимог до письмового оформлення робіт з інформатики та доведенні їх до широкого загалу вчителів? В результаті через деякий час може з'явитись достатня грамотність і в учнів, що приведе до більш високої їх культури в програмуванні, а отже і вищої результативності роботи.

Якщо на олімпіадах з інформатики на третьому та четвертому етапах все це не так актуально, то на уроках, позакласних заняттях, шкільних і районних олімпіадах навпаки, на наш погляд, надто важливо, тому спробуємо спростувати думку про складність і безперспективність вироблення згадуваних вище єдиних вимог. Наголосимо, що мета зробити відкриття не ставилась, на це у звичайного вчителя нема жодних підстав. Нижче ми спробуємо показати, що

достатньо прості, наочні і доступні вимоги до оформлення і опису алгоритмів, як і розв'язань задач вже давно є і можуть успішно використовуватись.

Перш за все уточнимо, що повинно обов'язково входити до письмової частини роботи юного програміста [10]. Назвемо та опишемо орієнтовні етапи:

Дослідження умови задачі та оголошення й опис величин;
Опис математичної моделі;
Загальний опис і доведення алгоритму;
Уточнення алгоритму та оцінка його складності;

Для ілюстрації візьмемо конкретну задачу.

Задача ***Num***[6]. *(видозмінений у бік ускладнення варіант пропонувався у 2000-2001 н.р. на обласній олімпіаді в Київській області)* Дано натуральне число *N*, записане *n* одиницями. Визначити число *k* - найменшу кількість чисел, в записі яких є лише цифри "3" і які в сумі дорівнюють числу *N*.

Технічні умови: вхідний файл *Num.dat* містить натуральне число *n* (1<*n*<*255*).
вихідний файл *Num.res* повинен містити число *k*.

a) ***Дослідження умови задачі та оголошення й опис величин.***

1) Згідно умови задачі аргументом повинне бути число *n* цілого типу (1<*n*<*255*), результатом - число *k* (найменш можлива кількість доданків, також цілого типу).

2) Згідно ознаки подільності натурального числа на 3 робимо висновок: *дане число N представляється у вигляді суми, кожен доданок якої містить лише цифри "3" тоді і тільки тоді, коли кількість цифр числа N ділиться без остачі на число 3*. Отже, якщо *n* ділиться без остачі на 3, то представлення його згідно умови задачі можливе, інакше – ні.

3) Повний перебір всіх варіантів суми чисел, що мають у записі лише цифри "3" через обмеження 1<*n*<*255* неефективний.

b) ***Опис математичної моделі.***

Для відшукання алгоритму спочатку покладемо *n* = 6 (тобто *N* = 111111) і помітимо, що тоді число *N* представляється згідно умови задачі у вигляді:

$$\underbrace{\underbrace{(33333+33333+33333)}_{99999}+\underbrace{(3333+3333+3333)}_{9999}+\underbrace{(333+333+333)}_{999}+\underbrace{(33+33+33)}_{99}+}_{основна\ частина\ суми}$$
$$+\ \underbrace{\underbrace{(3+3+3)}_{9}}_{основна\ частина\ суми}\ +\underbrace{3+3}_{остача} \qquad (V.1)$$

[6] Складніший варіант задачі пропонувався на IV етапі Київської обласної олімпіади з інформатики у 2000-2001 н.р.

Як видно з (1), число N складається з 17 доданків, тобто k = 17. *Основна_частина_суми* дорівнює числу 15 = 35 (кількість груп доданків, що мають у записі цифру "3").

Наведена формула може бути узагальнена, і матиме вигляд:

$$N = \underbrace{(33...3 + 33...3 + 33...3)}_{\text{число з } n\text{-1 цифр "9"}} + ... + \underbrace{(33333 + 33333 + 33333)}_{99999} + \underbrace{(3333 + 3333 + 3333)}_{9999} + \underbrace{(333 + 333 + 333)}_{999} + \underbrace{(33 + 33 + 33)}_{99} + \underbrace{(3 + 3 + 3)}_{9} + \textit{остача.} \quad \text{(V.2)}$$

Остача – це число, яке може бути, як менше, так і більше за 9, причому в своєму записі не обов'язково матиме лише цифри "9", воно може ділитись на 3 без остачі, або мати остачу 1 чи 2. В першому випадку це число додасть до числа *основна_частина_суми* певне число k_1, у другому приведе до відповіді: k = 0. Очевидно, що формула (V.2) при умові визначення числа k_1 дозволяє отримати найменш можливе значення числа k = *основна_частина_суми* + k_1, тобто цілком відповідає умові задачі і може бути її математичною моделлю.

Щодо визначення числа k_1, то це окрема задача, вона вимагає представлення натурального числа, записаного *довільними цифрами* у вигляді доданків, що мають у записі лише цифри "3". Але легко помітити, що *остача* при обмеженні в задачі ***NUM*** (1<*n*<*255*) дорівнює числу *n-1*, а представити число, менше за 255 у вигляді доданків, записаних лише "трійками" неважко. Для цього слід вибрати k_2, початкову кількість цифр ”3” (k_2 = 1 або k_2 = 2, наприклад, для *остача* = 255 k_2 = 2), і, в залежності від цього обчислити *div*(*остача*, 33) + *div* (*mod* (*остача*, 33), 3) або *div* (*остача*, 3).

*c) **Загальний опис і доведення алгоритму**.*

Алгоритм розв’язування задачі у загальному вигляді (для форми його опису зупинимось на навчальній алгоритмічній мові, що вивчається в школі) буде такий:

```
алг NUM
   арг ціл n
   рез ціл k
поч ціл основна_частина_суми, остача, k1
| основна_частина_суми
|                остача   } згідно формул (1) та
| ввод n
| якщо mod(n, 3) <> 0
| | то k:=0
| | інакше  основна_частина_суми := 3(n - 1)
| | остача := n - 1
| | ВИЗНАЧИТИ_К1(остача, k1)  | допоміжний алгоритм
| | k := основна_частина_суми + k1
| все
| вивод k
кін
```

Прокоментуємо форму запису алгоритму. Вона може бути довільною, в тому числі словесною чи у вигляді блок-схеми. Головне, щоб вибраний спосіб опису алгоритму був використаний доцільно й грамотно. Нам здається, що краще дотримуватись правил запису на алгоритмічній мові. При цьому слід привчати учнів дотримуватись позначень, введених у *a)* та *b)*, що в коментарях (на наш погляд, обов'язкових) допоможе уникнути багатослів'я. Коментарі слід використовувати, як для опису величин, так і в тілі алгоритму.

При оголошенні величин бажано використовувати словесні імена, вживаючи знак "_" для з'єднання кількох слів в одне ім'я, наприклад: *основна_частина_суми*.

Під основним алгоритмом слід повністю за такими ж правилами навести допоміжні алгоритми, записані у вигляді *процедур* чи *функцій* (у наведеному прикладі ми цього робити не будемо через очевидність допоміжного алгоритму **ВИЗНАЧИТИ_К1***(остача, k_1)*). Для скорочення письмової частини роботи можна це порекомендувати й учням, якщо допоміжні алгоритми достатньо прості і очевидні з пункту *b)*.

Щодо *існування та єдиності алгоритму* – то це *теорема*, її доведення слід записувати згідно традиційних правил записів доведень теорем. Але спочатку слід чітко сформулювати умову й висновок, які важливо не сплутати з аргументами й результатами задачі. Для нашої задачі теорему існування її розв'язання можна сформулювати так:

> ***Теорема*** **NUM**. Якщо *дано натуральне число N, записане n одиницями (1<n<255)*, то *існує алгоритм, який визначає число k - найменшу кількість чисел, в записі яких є лише цифри "3" і які в сумі дорівнюють числу N*. (умова й висновок виділені курсивом)

Доведення випливає з *a)*, п.2) і формули (V.2). Дійсно, ця формула представляє число *N*, як суму найменшої кількості доданків, записаних дев'ятками, а кожен доданок, у свою чергу, виділяє найменш можливу кількість чисел, записаних трійками.

Доведення алгоритму **ВИЗНАЧИТИ_К1***(остача, k_1)* приводити не будемо з тої ж причини, що й запис самого алгоритма. Нарешті, процес відніманння від числа *N* доданків, записаних лише з допомогою цифр "3" скінченний. Ці міркування повністю доводять теорему NUM.

Отже, загальні принципи, яких слід дотримуватись при письмовому описі алгоритмів можуть бути такі:

- Виконати письмово всі попередні етапи розв'язування задачі на ПК (включно опис алгоритму), додавши доведення алгоритму;
- Оголошення і опис аргументів та результатів провести ще на першому етапі, надаючи зручні і зрозумілі імена і дотримуючись їх на всіх наступних етапах;
- Математичну модель бажано побудувати, узагальнивши конкретні і прості приклади. Строго дотримуватись лаконічності міркувань, застосовуючи прийнятих в математиці позначень, нумеруючи формули і посилаючись при необхідності на ці номери;

- Записувати алгоритм слід як можна простішим способом, найдоцільніше навчальною алгоритмічною мовою, суворо дотримуючись загальноприйнятих і загальновідомих правил, широко застосовуючи коментарі і метод послідовного уточнення;
- Доведення алгоритму проводити схематично, не повторюючи встановлених на попередніх етапах фактів.

VI. ПРО ЗАОЧНИЙ ТУР РАЙОННОЇ ОЛІМПІАДИ З ІНФОРМАТИКИ

Підводячи підсумки розділів II – V, розглянувши питання про письмовий опис алгоритмів, додамо ще дещо, на наш погляд, суттєве до методики підготовки та проведення районних турів олімпіад. Зокрема опишемо проведення заочних турів районних олімпіад на прикладі проведеного у Бориспільському районі в 2000-2001 н.р.

Серед завдань відсутня розбивка на класи, але є по два завдання з однаковим номером, одне - нижчого рівня складності, друге - вищого. Можна завдання першого рівня вважати завданнями для 9-10 класів, а завдання другого рівня - для 11 класу. Але, як свідчить практика, розбивка на класи неефективна, вміння складати алгоритми більше залежить від здібностей та досвіду, ніж від віку учасника.

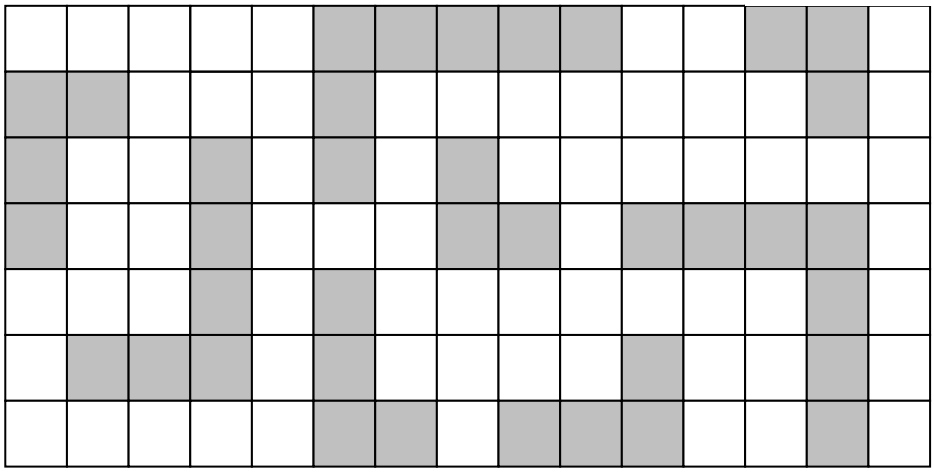
В зв'язку з тим, що пропоновані задачі розраховані як на теоретичний тур, так і на практичний тури, до них обов'язково потрібно описати математичну модель та алгоритм словесно, у вигляді блок-схем або навчальною алгоритмічною мовою у звичайному учнівському зошиті. Опис повинен бути зрозумілим, вичерпним і стислим. Бажано використати для цього відомості з розділу V.

Можна додати ГМД (3,5дюйма) із діючими програмами на Турбо Паскаль (версії 6 - 7) або Бейсику (інтерпретатор для IBM-сумісних машин, бажано Basica чи GW). В такому випадку програма повинна зчитувати дані із зовнішнього файлу і виводити на екран відповідь (пояснення в дужках до кожної задачі). За коректно працюючу програму при оцінюванні бали подвоюються, але за непрацюючу - знімаються зовсім.

Завдання заочного туру посилаються один раз в середині вересня. Перед цим слід провести методоб’єднання учителів інформатики району, на якому обговорити умови проведення заочного туру, затвердити оргкомітет, журі та Положення про заочний тур районної олімпіади з інформатики. Завдання розсилаються у всі школи району. Останній термін здачі розв'язаків запропонованих завдань – за один тиждень до дати проведення офіційного туру районної олімпіади, до цього часу слід оголосити результати заочного туру та згідно них визначити додаткових учасників офіційного туру районної олімпіади.

Задача 1(перший рівень). У прямокутному полі m×n, розбитому на клітки, міститься ряд "кутиків" - зігнутих на 90 градусів у довільному місці смужок шириною одна клітка, наприклад: (див. мал. VI.1). Довжини сторін "кутиків" та напрям згину сторін довільні. "Кутики" ніде не накладаються і не дотикаються. Інформація про розташування кутиків задана цілочисельною таблицею *K[1:m, 1:n]*, у якій значення *K[i,j] = 1*, якщо клітка входить у деякий "кутик", і *K[i,j] = 0*, якщо не входить. Визначити кількість "кутиків" у заданому прямокутному полі [11].

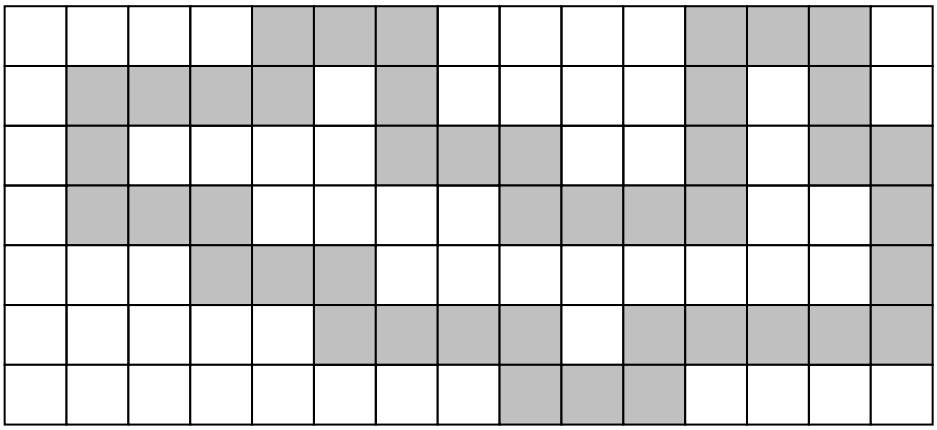
малюнок VI.1



(Вхідний файл KUT.dat має вигляд: перший рядок містить два натуральні числа m та n, наступні n рядків містять по m символів (0 чи 1), відділених пропуском. Нулями позначені пусті, а одиницями - зафарбовані клітинки)

Задача 1(другий рівень). У прямокутному полі m × n, розбитому на клітки, міститься "змія" - неперервна ламана лінія шириною в одну клітку яка може згинатися лише на 90 градусів "змія" може утворювати замкнений або розімкнений контур, наприклад: (див. мал. VI.2). "Змія" себе не перетинає і ніде не дотикається різними частинами крім, можливо, "голови" до кінця "хвоста" Інформація про розташування "змії" задана таблицею ціл таб *K[1: n, 1:m]* Зна-

малюнок VI. 2



чення *K[i, j] = 1*, якщо клітка належить змії, і *K[i, j] = 0*, якщо не належить. Визначити, чи утворює "змія" замкнений контур [11].

(Вхідний файл ZMIY.dat має вигляд: перший рядок містить два натуральні числа m та n, наступні n рядків містять по m символів (0 чи 1), відділених пропуском. Нулями позначені пусті, а одиницями - зафарбовані клітинки)

Задача 2(перший рівень). Ціле додатне число можна зобразити сумою його частин, що називається розбиттям. Наприклад, число 4 можна зобразити так: 4; 3+1; 2+1+1; 2+2; 2+2; 1+1+1+1. *P(n)* - кількість розбиттів, наприклад: *P(4)=5*. Для заданого числа *n* знайти всі його розбиття та *P(n)* [12].
(Вхідний файл NUMB.dat має вигляд: єдиний рядок містить натуральне число n (n<20) дане число, яке треба зобразити у вигляді розбиттів.

Задача 2(другий рівень). Дано *N* пар круглих дужок. Необхідно перебрати всі варіанти розміщення цих дужок так, щоб виконувалась відповідність відкритих і закритих дужок, прийнята в інформатиці (при перегляді виразу зліва направо в будь-який момент кількість закритих дужок не повинна перевищувати кількість відкритих). Наприклад, для *N=3* є п'ять таких дужкових виразів:

((( ))), (( ) ( )), (( )) ( ) , ( ) (( )), ( ) ( ) ( ).

Вирази ())((),)(())(і тому подібні є помилковими [12].
(Вхідний файл SKOB.dat має вигляд: єдиний рядок містить парне натуральне число n - кількість дужок (n<20).

Задача 3(перший рівень). На деякій ділянці землі ростуть кущі ягід. Координати площини для кожного куща задані парою чисел *(X,Y), i = 1, 2, ...,n*, причому числа додатні. Ділянку з кущами необхідно обгородити парканом мінімальної довжини у вигляді прямокутника, сторони якого паралельні осям координат. Якщо паркан проходить по центру деякого куща, то кущ вважати обгородженим. Знайти координати кутів паркану та його загальну довжину.
(Вхідний файл DIL1.dat має вигляд: перший рядок містить натуральне число n, (n<20), два наступні рядки містять по n чисел, розділених пропуском - координати кущів.

Задача 3(другий рівень). На деякій ділянці землі ростуть кущі ягід. Координати площини для кожного куща задані парою чисел (*X,Y*), *i = 1, 2, ...,n*, причому числа додатні. Ділянку з кущами необхідно обгородити парканом мінімальної довжини. Якщо паркан проходить по центру деякого куща, то кущ вважати обгородженим. Знайти координати кутів паркану та його загальну довжину.
(Вхідний файл DIL2.dat має вигляд: перший рядок містить натуральне число n, (n<20), два наступні рядки містять по n чисел, розділених пропуском - координати кущів.

Порад щодо розв'язування задач заочного туру районної олімпіади ми не будемо наводити, адже для деяких з них вказані літературні джереза, які загальновідомі. Крім того, читачам може бути цікаво справитись із завданнями заочного туру самостійно. Натомість проілюструємо заочний тур ще однією задачею, яка пропонувалась разом із рештою задач, але поза конкурсом і може бути цікава своєю рекурсивною реалізацією.

Задача 4. **В квадратний масив *N*5*N* (*N*<11), що заповнено нулями з допомогою одиниць введено замкнутий контур без самоперетинів. Написати програму, яка заповнює внутрішню область контура одиницями.**

Технічні умови:

вхідний файл MAS.*dat* містить *N* рядків, заповнених нулями і одиницями, згідно умови.

вихідний файл MAS.*res* повинен містити *N* рядків, заповнених нулями і одиницями, згідно умови, де внутрішня область заповнена одиницями.

Завдання, які передбачають опрацювання прямокутних масивів цілих чисел, заповнених таким чином, наприклад, нулями та одиницями, що одиниці розміщені у вигляді замнненого контура, ламаної лінії, "змійки" і т.д., зустрічаються часто. Серед них задачі 1 (першого та другого рівнів).

Як уже згадувалось, ці задачі взяті із [11] (№ 4, теоретичний тур, 10 клас, 1990 р, стор. 122, № 4, теоретичний тур,10 клас, 1991 р, стор.129), куди і відсилаємо читача для ознайомлення з ідеями алгоритмів розв'язування наведених задач.

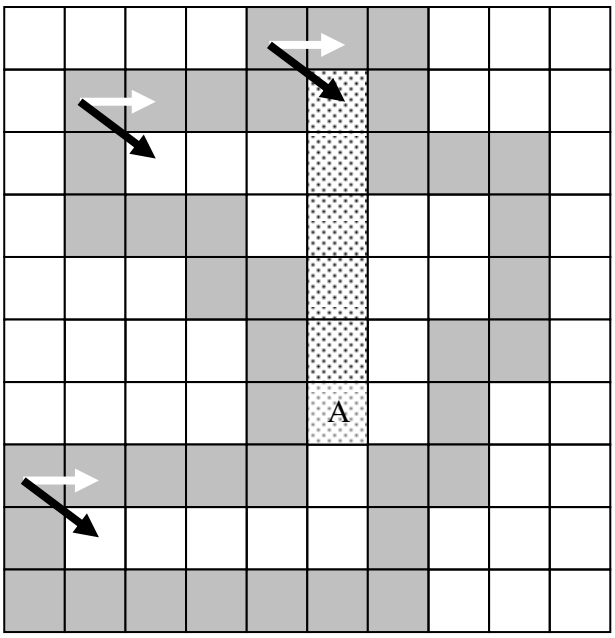
Як правило, такі задачі вимагають деяких навичок роботи з циклами при опрацюванні масивів та вміння підмітити ідею алгоритму. Проте відносити такі задачі до одного типу не слід. Часто вони схожі лише малюнком-ілюстрацією. Це стосується і задачі 4. Про-ілюструємо її малюнком VI.3. Домовимось білі клітинки ототожнювати із числами 0, а зафарбовані в сірий колір - із числами 1. Тоді в квадратному масиві A[10,10] одиниці устворять замкнений контур без самоперетинів. Очевидно, на першому етапі, використовуючи замість елементів даного масива A[*N*, *N*] клітинки малюнка VI.3, можна отримати таке наближення шуканого алгоритма:

алгоритмVI.1

```
алг ЗАПОВНЕННЯ_КОНТУРА
поч
¦ пц для і від 1 до N
¦ ¦ пц для j від 1 до N
¦ ¦ ¦ якщо клітинка належить внутрішній частині області контура
¦ ¦ ¦ ¦ то зафарбувати клітинку сірим кольором
¦ ¦ ¦ все
¦ ¦ кц
¦ кц
кін
```

Але цей алгоритм легко було б реалізувати, якби клітинки, що розміщені зовні контура, мали колір, відмінний від білого. А зафарбувати їх іншим кольором не легше, ніж клітинки, розміщені у внутрішній області контура.

малюнок VI.3



Інтуїція підказує, що клітинки, розміщені у внутрішній області контура, чимось об'єднані, точніше, мають таку властивість: *кожна з них сусідніми має або клітинки, що належать контуру, або клітинки, розміщені у внутрішній його області.* (VI.1)

Отже треба шукати не для кожної клітинки спосіб визначення, чи розміще-на вона всередині контура чи ні.

Досить знайти одну клітинку, що має властивість (VI.1), а потім для всіх сусідніх з нею клітинок перевіряти, чи володіють вони також властивістю (VI.1). Нехай визначити одну із клітинок, розміщених у внутрішній області контура можна досягти з допомогою допоміжного алгоритма **В_КОНТУР**, а визначити решту клітинок, що володіють властивістю (1) можна з допомогою допоміжного алгоритма **ЗАПОВНИТИ_КОНТУР**. Тоді замість алгоритма VI.1 одержимо значно коротший алгоритм VI.2:

алгоритм VI.2

алг **ЗАПОВНЕННЯ_КОНТУРА**
поч
¦ **В_КОНТУР**; **ЗАПОВНИТИ_КОНТУР**
кін

Залишається тільки описати ці допоміжні алгоритми.

Щоб сформулювати алгоритм **В_КОНТУР**, досить підмітити (див. малюнок 3, білі та чорні стрілки), що *кожна "кутова" клітинка контура, яка зліва і зверху обмежена клітинками, що лежать зовні контура, обов'язково має безпосередньо від себе справа клітинку, яка також належить контуру, а на "південному заході"від неї обов'язково розміщена клітинка, що належить внутрішній області контура.* (VI.2)

Отже алгоритм **В_КОНТУР** попередньо можна записати так:

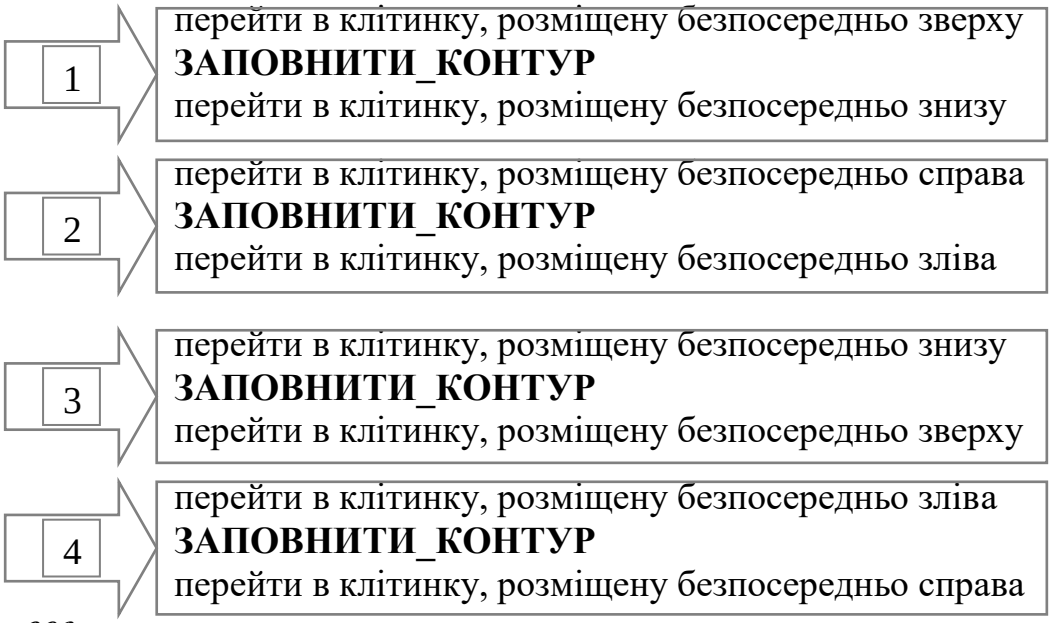
алг **В_КОНТУР** *алгоритм VI.3*
поч
¦ ***пц покі*** клітинка, що належить внутрішній області контура не знайдена
¦ ¦ ¦ шукати клітинку, помічену білою та чорною стрілками
¦ ¦ ¦ ***якщо*** знайдена клітинка, помічена білою та чорною стрілками
¦ ¦ ¦ ¦ ***то*** клітинку, розміщену безпосередньо правіше і нижче від поміченої білою та чорною стрілками вважати клітинкою, що належить внутрішній області контура
¦ ¦ ¦ ***все***

кц
кін

Алгоритм **ЗАПОВНИТИ_КОНТУР** описується несподівано просто для тих, хто добре знає, що таке рекурсія. Він "працює" в середовищі клітинок, розміщених у внутрішній області замкнутого контура без самоперетинів і має такий вигляд:

алг **ЗАПОВНИТИ_КОНТУР** *алгоритм VI.4*
поч

якщо клітинка належить внутрішній області контура і не зафарбована сірим кольором
то зафарбувати її сірим кольором



все
кін

Спробуйте самостійно виконати цей алгоритм для невеликого поля. Щоб зрозуміти, як він виконується, зверніть увагу на чотири серії команд, на які вказують стрілки. Кожна з них містить команду "**ЗАПОВНИТИ_КОНТУР**", яка є рекурсивним викликом самого алгоритма VI.4. Тому досить уявити, що буде в результаті виконання, наприклад, серії 1. Якщо виконати серію 1 взявши за стартову клітинку, помічену літерою "А", то в результаті будуть зафарбовані клітинки, виділені на малюнку 3 світло-сірим кольором.

Але через складність "ручного" виконання рекурсивних алгоритмів краще, коли ви його реалізуєте в середовищах Кумир, ALGO чи PascalABC, які є інтерпретаторами відомої в школах навчальної алгоритмічної мови, завантаживши виконавця "Робот".

Нижче приводимо алгоритм розв'язування задачі 4, записаний мовою Кумир, із використанням допоміжних алгоритмів-процедур **В_КОНТУР***(n,A,x,y)* та **ЗАПОВНИТИ_КОНТУР***(n,x,y,A)*.

Щодо мови пакета "Кумир" зауважимо: вона є досить потужною мовою програмування, з одного боку, дуже схожа до навчальної алгоритмічної мови, але із великою кількістю додаткових можливостей, що дозволяє ставити її поряд, наприклад, з мовою програмування Паскаль. Спостереження переконують, що шлях "Кумир" → Паскаль в процесі оволодіння методами програмування є значно легшим і коротшим порівняно із навчанням основам програмування безпосередньо на мові Паскаль, особливо, якщо мати на увазі учнів середніх класів. Дуже суттєво, що вона крім достатньо потужних можливостей менш формалізована і більш наочна, ніж мова Паскаль На наш погляд у сказаному вище читач переконається, розбираючи наведений нижче алгоритм, який зовсім легко "перевести" на мову Паскаль:

алгоритм VI. 5

```
алг ЗАПОВНЕННЯ_КОНТУРА(арг ціл n, арг рез ціл таб A[1:n, 1:n])
поч ціл x, y
| В_КОНТУР(n, A, x, y);  ЗАПОВНИТИ_КОНТУР(n, x, y, A)
кін
алг В_КОНТУР (арг ціл n, арг рез ціл таб A[1:n, 1:n], рез ціл i, j)
поч ціл k, j1
| k:=0; i:=1
| пц поки i<n і k<2
| | j1:=1
| | пц поки j1<n
| | | якщо A[i,j1]=1 то k:=k+1 все
| | | якщо k<2 то j1:=j1+1 інакше j:=j1; j1:=n  все
| | кц
| | i:=i+1
| кц
кін
алг ЗАПОВНИТИ_КОНТУР (арг ціл n,l,p, арг рез ціл таб A[1:n, 1:n])
поч ціл i, j
| i:=l; j:=p
| якщо A[i, j]=0
| |то A[i, j]:=1
| | i:=i -1;  Заповнити_Контур(n, i, j, A); i:=i+1
| | j:=j -1;  Заповнити_Контур(n, i, j, A); j:=j+1
| | i:=i+1; Заповнити_Контур(n, i, j, A); i:=i -1
| | j:=j+1; Заповнити_Контур(n, i, j, A); j:=j -1
| все
кін
```

Для логічного завершення розгляду задачі 4 пропонуємо програму її розв’язання мовою Turbo Pascal:

```
program Fill_Contour;
type bin = array[1..10, 1..10] of 0..1;
  var A : bin;  N, x, y : byte;
procedure Inp;{-------------------------------------------}
```

```
  var i, j : integer;
begin
  assign(input, 'Mas.dat'); reset(input);
  N:=1;
  while not EoLn do       begin
   read(A[1,N]); inc(N)  end;
  dec(N);
  for i:=2 to N do
   for j:=1 to N do
     read(input, A[i,j]);
  close(input);
end;{-------------------------------------------------------------}
procedure Outp;
  var i, j : integer;
begin
  assign(output, 'Mas.res'); rewrite(output);
  for i:=1 to N do     begin
    for j:=1 to N do
      write(output, A[i,j], ' ');
     writeln(output)  end;
   close(output);
end;{-------------------------------------------------------------}
procedure In_Cont(A : bin; N : byte; var k,l : byte);
   var i, j : integer;
begin
   for i:=1 to N do
     for j:=1 to N do
       if A[i,j] = 1 then              begin
    k:=i+1; l:=j+1;  i:=N; j:=N end;
end;{-------------------------------------------------------------}
procedure Fill_Cont(N, k, l : byte; var A : bin);
   var i, j : integer;
begin
   i:=k; j:=l;
   if A[i,j] = 0 then                          begin
     A[i,j]:=1;
     i:=i-1; Fill_Cont(N, i, j, A); i:=i+1;
     j:=j+1; Fill_Cont(N, i, j, A); j:=j-1;
     i:=i+1; Fill_Cont(N, i, j, A); i:=i-1;
     j:=j-1; Fill_Cont(N, i, j, A); j:=j+1  end;
end;{-----------------------------------------------------}
begin
   Inp;
   In_Cont(A, N, x, y);
   Fill_Cont(N, x, y, A);
```

```
  Outp;
end.
```

Програма *Fill_Contour* зчитує вхідні дані з файлу Mas.dat, зразок якого наведено нижче, а виводить результуючий масив, що відображає заповнений контур у вихідний файл Mas.res.

Зразок вхідного файлу Mas.dat:

```
0 0 0 0 1 1 1 0 0 0
0 1 1 1 1 0 1 0 0 0
0 1 0 0 0 0 1 1 1 0
0 1 1 1 0 0 0 0 1 0
0 0 0 1 1 0 0 0 1 0
0 0 0 0 1 0 0 1 1 0
0 0 0 0 1 0 0 1 0 0
1 1 1 1 1 0 1 1 0 0
1 0 0 0 0 0 1 0 0 0
1 1 1 1 1 1 1 0 0 0
```

VII. ЗАСТОСУВАННЯ МЕТОДІВ ОПРАЦЮВАННЯ БАГАТОРОЗРЯДНИХ ЧИСЕЛ ПРИ ВИВЧЕННІ ПРОГРАМУВАННЯ У ШКОЛІ

Серед найважливіших тем з алгоритмізації та програмування, які слід опанувати учням загальноосвітніх шкіл, що хочуть досягти певних успіхів у програмуванні слід назвати:

- математична логіка та множини;
- цілі числа;
- комбінаторика;
- дійсні числа;
- многочлени;
- координатний метод;
- теорія ігор;
- довга арифметика;
- перебори;
- сортування;
- метод динамічного програмування;
- алгоритми на графах та інші.

У цьому розділі розглянемо підходи до розв’язування олімпіадних задач з використанням так званої ″довгої арифметики″, тобто базові алгоритми опрацювання багаторозрядних чисел та їх застосування в олімпіад них задачах з програмуваппя.

VII.1 Роль задач багаторозрядної арифметики у формуванні алгоритмічної культури учнів. Класифікація та зразки задач багаторозрядної арифметики.

Звичайну цікавість, властиву багатьом видам “братів наших менших” – тваринам, людина в ході еволюції перетворила у феноменальну ознаку свого виду, трансформувавши її в потужний двигун будь-якого прогресу, стосовно ж школи - це надзвичайно важливий стимулятор підвищення якості знань. В підтвердження вищесказаного можна пригадати загальновідомі слова А. Ейнштейна: ”Здається майже чудом, що сучасні методи навчання ще не зовсім задушили святу допитливість”. Отже *одним із найважливіших завдань вчителів є не саме*

навчання, навіть за найпрогресивнішими технологіями, а розвиток свідомого інтересу учнів до предмету вивчення.

У цьому плані при вивченні програмування в школі дуже корисною може бути так звана *довга арифметика*, точніше, - *арифметика багаторозрядних чисел*, адже будь-який програміст-початківець швидко помічає, що обчислення часто виходять за межі розрядів стандартних числових типів. Важливо спрямувати його увагу на причини відсутності навіть у таких потужних мовах програмування, як Pascal, процедур опрацювання "довгих" чисел і на способи уникнення такої незручності. Розібравши алгоритми довгої арифметики, учень значно підвищить свій рівень програміста.

а) Спробуємо відповісти для себе, чому мови програмування високого рівня не мають можливостей опрацювання багаторозрядних чисел, що дуже затрудняє визначення і збереження в пам'яті комп'ютера, наприклад, 100! чи a^k, навіть при не таких вже й великих значеннях a і k, наприклад 2003^{2004}?

Напрошуються п'ять аргументів. По-перше, "скільки нитка не протягується, а кінець в неї обов'язково є", тобто обмеження розрядності все одно доведеться зробити, тому очевидно, розроблювачі мов програмування високого рівня зупинились на оптимальному співвідношенні між кількістю розрядів найпотужніших числових типів з одного боку та завантаженням пам'яті комп'ютера і швидкістю виконання програм з іншого. По-друге, дуже довгі стандартні числові типи і процедури їх обробки в мові Pascal відсутні, мабуть, по тій же причині, що й звичайна в математиці операція піднесення натурального числа до натурального степеня, іншими словами, якщо можливо скласти компактні алгоритми виконання не так часто вживаних в практиці операцій, то не доцільно загромаджувати ними транслятор. По-третє, як правило, в задачах виникають локалізовані потреби у виконанні операцій над багаторозрядними числами, найчастіше достатньо або лише операцій додавання та множення, або самі числові дані мають спеціальні обмеження. По-четверте, навіть введення дуже довгих числових типів та окремий стандартний модуль із процедурами виконання операцій порівняння, додавання, віднімання, множення та цілочисельного ділення над ними проблему не вирішать, адже на черзі стануть від'ємні та дробові числа, операції ділення, визначення радикалів і т.д. І нарешті, по-п'яте, практична діяльність людей майже не передбачає обробки "астрономічних" чисел, наприклад, число π з кількома сотнями значущих цифр або 5000! нам можуть знадобитись, хіба в плані "спортивного інтересу". Інакше, як відноситись, наприклад, до задачі "Bus" (IV етап XIII Всеукраїнської олімпіади з інформатики), в якій обмеження на K та N - число працівників заводу, які збираються на зупинці службового автобусу та кількість зупинок відповідно такі: $1 \le N$, $K \le 200000$, а це ще далеко не "довга арифметика"[17].

Але поряд з деякими незручностями, викликаними обмеженістю операцій над багаторозрядними числами, ми маємо чудову нагоду отримати задоволення від "чистого" програмування, доступного навіть пересічним школярам, не переобтяженого складними математичними моделями, тобто, нагоду і стимул реалізації, за академіком А.П. Єршовим, едісонівського таланту створювати все, що завгодно з нуля й одиниці [14].

Відмітимо, що створення алгоритмів обробки багаторозрядних чисел не вимагає ніяких спеціальних знань, крім загальних арифметичних понять та правил і стійких навичок оперування з циклами, масивами, рядками та файлами. Поряд з цим потрібні неабияка зосередженість, настирливість і вправність. Тому задачі опрацювання багаторозрядних чисел можна віднести до *задач на техніку програмування*, які принесуть велику практичну користь всім програмістам-початківцям, їх слід грунтовно вивчати вже на перших етапах опанування програмуванням. Це й визначає місце "довгої арифметики". Задачі обробки багаторозрядних чисел самі по собі носять олімпіадний характер. Але разом з тим вони використовуються в якості опорних при розв'язуванні великої кількості складніших олімпіадних задач з програмування.

b) Якщо обмежитись цілочисельною арифметикою, то слід виділити такі *базові алгоритми опрацювання багаторозрядних чисел*:

1. **Введення (читання) багаторозрядного числа**;
2. **Збереження багаторозрядного числа**;
3. **Визначення модуля (абсолютної величини) багаторозрядного числа**;
4. **Цілі додатні (натуральні) багаторозрядні числа**:

4.1 порівняння натуральних багаторозрядних чисел;
4.2 додавання натуральних багаторозрядних чисел;
4.3 віднімання натуральних багаторозрядних чисел (меншого від більшого);
4.4 множення натурального багаторозрядного числа на натуральне одноцифрове число;
4.5 множення натуральних багаторозрядних чисел;
4.6 визначення остачі при діленні двох натуральних багаторозрядних чисел;
4.7 визначення неповної частки при діленні двох натуральних багаторозрядних чисел;

5. **Цілі багаторозрядні числа**:

5.1 порівняння цілих багаторозрядних чисел;
5.2 додавання цілих багаторозрядних чисел;
5.3 віднімання цілих багаторозрядних чисел;
5.4 множення цілих багаторозрядних чисел.

Вище наведена орієнтовна класифікація задач на багаторозрядні числа з точки зору реалізації арифметичних операцій та числових типів. З відомих причин нижче ми обмежимось розглядом лише задач класу 4. Використовуючи перераховані алгоритми, можна розв'язувати велику кількість задач "довгої арифметики", зокрема, визначення найбільшого спільного дільника та найменшого спільного кратного двох натуральних цілих чисел, скорочення звичайного дробу з багатоцифровими чисельником і знаменником, перевірку взаємної кратності двох багатоцифрових натуральних чисел, перевірку натурального багаторозрядного числа "на простоту", визначення чисел Фібоначчі, факторіалів та членів інших відомих числових послідовностей з дуже великими номерами, обчислення ланцюгових дробів і ін.

Дамо змістовну класифікацію цих задач на багаторозрядні числа:

- задачі на відшукання багатоцифрових чисел без застосування арифметики багаторозрядних чисел;

- задачі на застосування базових алгоритмів 1-5.

c) Для ілюстрації типу *а*) наведемо кілька задач.

Одна з них задача **NUM** вже описана в п'ятому розділі. Отже. розглянемо другу.

Задача **ОБЕРТОВЕ ЧИСЛО**.Знайти натуральне число $N = \overline{a_1 a_2 a_3 ... a_{n-2} a_{n-1} a_n}$, де a_i (*i=0,1,2,3,4,5,6,7,8,9*)– його цифри, коли відомо, що число *N*, помножене на свою цифру найнижчого розряду a_n, дорівнює числу $N_1 = \overline{a_n a_1 a_2 a_3 ... a_{n-2} a_{n-1}}$, у якого цифри a_i співпадають із цифрами числа *N* з такими ж індексами, тобто

$$a_1 a_2 a_3 ... a_{n-2} a_{n-1} a_n \times a_n = a_n a_1 a_2 a_3 ... a_{n-2} a_{n-1} \qquad \text{(VII.1)}$$

Такі числа в математиці називають *обертовими*. Перш за все зауважимо, що винятками серед цифр a_i є цифри a_1 та a_n, які не можуть дорівнювати нулю. Можна не розглядати і випадок, коли a_n= = 1, адже тоді відповідь очевидна, число $N = \underbrace{1111\ldots1}_{\text{довільне число одиниць}}$.

Таким чином, "полем" дії шуканого алгоритма будемо вважати числа, що закінчуються цифрами "2", "3", "4", "5", "6", "7", "8", "9", тобто, якщо дотримуватись позначень умови завдання, то $2 \le a_n \le 9$, де a_n - остання справа цифра числа *N*. Це означає, що нам потрібно знайти не більш, як вісім чисел. Але чи всі вони існують? Як виявляється, існують всі вісім таких чисел, хоча деякі з них, що видно з таблиці II.1, мають досить значну кількість цифр.

таблиця VII.1

Число	*к-ть цифр*
105263 157894 736842	18
1034 482758 620689 655172 413793	28
102564	6
102040 816326 530612 244897 959183 673469 387755	42
1016 949152 542372 881355 932203 389830 508474 576271 186440 677966	58
1014 492753 623188 405797	22
1 012658 227848	13
10 112359 550561 797752 808988 764044 943820 224719	44

Щоб переконатись, що при їх знаходженні не застосовувались базові задачі 1.-5. , пояснимо алгоритм відшукання чисел.

Як видно з таблиці VII.1, всі числа N, що заловольняють (VII.1) мають у найвищому розряді цифру "1", а наступну цифру - "0". Легко зрозуміти, що інакше бути не може. Адже в найвищому розряді числа N_1 повинна бути цифра a_n ($a_n = a_1 \times a_n + a$, де *а* - цифра, що "помічається" при множенні попередньої цифри на a_n). Це можливо при a_1= 1 і a_2 = 0. Для підтвердження цього візьмемо згадане вище число №3 (має найменшу кількість цифр). Утворимо нове число $N^1 = N \times 1000000 + N$ = 102564102564 і помножимо його на 4 (остання цифра): 102564102564 × 4 = 410256410256.

З цієї таблиці видно, що більшість із знайдених чисел багатоцифрові. Крім того, в таблиці наведено лише, так би мовити, найменші періоди обертових чисел, насправді їх безліч, а розрядність не обмежена. Дійсно, ця властивість виконується, наприклад, не тільки для числа 102564, а й для чисел:

102564102564,

102564102564102564,
102564102564102564102564,
102564... 102564, і т.д., тобто 102564, повторене довільне число разів, тобто обертових чисел існує безліч.
Тепер перейдемо до опису алгоритму. Розглянемо наступну таблицю (виконання множення числа № 3 на 4 (див. таблицю II.1)):

таблиця VII.2

Цифри, що "помічаються"	01221
Цифри числа n	102564
Остання цифра числа n (число 4)	× 4
Результат	410256

Якщо відома цифра найменшого розряду a_n = 4 шуканого числа *N*, то відома і цифра найменшого розряду числа N_1: a_{n-1} = mod ($a_n \times a_n$, 10) = mod (4 × 4, 10) = 6, а також цифра, що при цьому "помічається":
div (a_n X a_n, 10) = div (4 X 4, 10) = 1(в таблиці 2 - підкреслені). Але a_{n-1} не тільки остання в числі N_1, а й передостання в числі *N*. Отже на першому кроці знайдено цифру, що стоїть зліва від початково-відомої цифри числа *N*. На наступному кроці знайдемо дві останні цифри числа N_1: $\overline{a_{n-2}a_{n-1}} = \mathrm{mod}(\overline{a_{n-1}a_n} \times a_n, 10^2) = \mathrm{mod}(64 \times 4, 10^2) = \mathrm{mod}(256, 10^2) = 56$ Отже визначено три цифри числа *N*: $\overline{a_{n-2}a_{n-1}a_n} = 564$ та наступна цифра, що помічається (цифра 2). Цей процес повторюватимемо, знаходячи на кожному кроці ще одну цифру числа *N*. Але виникає питання: коли зупинитись? Очевидно тоді, коли на деякому кроці нова знайдена цифра числа N_1 буде рівною a_n = 4, а цифра, що "помічається" буде рівною 0 (в таблиці II.2 - виділені сірим фоном). Виразимо наведені вище міркування у вигляді алгоритму:

```
алг літ Обертове_Число(арг ціл цифра_n)
поч ціл цифра, число, div, літ результат
¦ div:=0; цифра:=цифра_n;число:=цифра*цифра_n+div;знач:=""
¦ пц
¦ ¦ знач:=Переведення(цифра)+знач;  div:=div(число,10)
¦ ¦ цифра:=mod(число,10)
¦ ¦ число:=цифра*цифра_n+div
¦ кц_при цифра = цифра_n і div=0
кін
```

Переведення*(цифра)* - це допоміжний алгоритм-функція, який перетворює величину *цифра*, наприклад 9 у відповідний символ, в нашому випадку “9”.
Клас задач *b*) проілюструємо також двома задачами (пропонувались на обласній олімпіаді в Київській області у 2002-2003 н.р.)
Задача **SEQUENCE**. Послідовність чисел {*S*(*n*)} утворюється за рекурентними формулами: *S*(1) = 1, *S*(2) = 2, ..., *S*(*k*) = *k*, а при *n*>*k* *S*(*n*) = *S*(*n* - 1) + *S*(*n* - 2) + … + *S*(*n* - *k*) і має вигляд, наприклад, для *k* = 5: 1, 2, 3, 4, 5, 15, 29, 56, 109, 214, ... Якщо числа цієї послідовності виписати одне за одним без пропусків, то утвориться деяка нескінченна послідовність цифр. Написати програму, що визначає, якою буде *m*-та цифра в такій послідовності.

Технічні умови:
вхідний файл ***Sequence.dat*** містить числа k і m ($1<k<m<2003$);
вихідний файл ***Sequence.res*** повинен містити одну з цифр 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
Приклад файлу ***Sequence.dat:*** 5 15.
Приклад файлу ***Sequence.res***: 2
Ідея розв'язання задачі випливає з таких міркувань. Уявимо, що m-та цифра знаходиться в l-му члені послідовності і виписана підпослідовність
$S(1)$, $S(2)$, $S(3)$, …, $S(l-1)$, $S(l)$ без пропусків. Очевидно, що зберігати всі числа послідовності немає потреби. На кожному етапі слід пам'ятати лише п'ять останніх чисел послідовності. Якщо при цьому знати номер p останньої цифри числа $S(l-1)$, то залишиться визначити цифру в числі $S(l)$ під номером $(m-p)$. На деякому етапі доданки стануть багатоцифровими, тому доведеться користуватись наведеними нижче програмами опрацювання багаторозрядних чисел, а також наведеною там же програмою **Fib_N**.
Задача **SINGER**. Аллу Пугачову називають жінкою, яка співає. Одному ДиРектору також захотілось мати звання "співаючого". І через деякий час він став популярним виконавцем. Але немає нічого вічного під Місяцем. Популярність Співаючого ДиРектора різко спала через те, що він, повіривши у свої виняткові вокальні здібності, став зловживати виконанням пісень "вживу". ДиРектором володіли два взаємно виключаючі бажання – бути популярним і радувати слухачів "живим" голосом. Після тривалих роздумів він звернувся за порадою до вічної і мудрої своєї студентки Віри Сердючко. Та, розкинувши карти Таро нагадала, що популярність ДиРектора буде продовжуватись до того часу, доки він дотримуватиметься правила: на кожному концерті не слід співати "живим" голосом дві пісні підряд і всі концерти повинні бути різні за послідовністю "живих" і "фанерних" номерів (тоді слухачі не помітять). Окрилений надією ДиРектор тут же захотів узнати вік своєї популярності і оголосив: того, хто складе програму, що підраховує скільки концертів буде продовжуватись популярність ДиРектора, негайно внесе до списків студентів економічного факультету на довічне навчання. Спробуйте написати таку програму.
Технічні умови:
вхідний файл ***Signer.dat*** містить єдине число n ($1< n <100$);
вихідний файл ***Signer.res*** повинен містити шукане число концертів, що задовольняють вимогам, вказаним в умові задачі.
Приклад файлу ***Signer.dat***: 4
Приклад файлу ***Signer.res***: 8
Позначимо пісню, виконану "живим" голосом цифрою 1, а пісню, виконану "під фанеру"- цифрою 0. Моделлю одного концерту з n пісень буде послідовність з цифр 0 і 1. Назвемо невдалим концертом послідовність з 0 і 1 , у якій поряд стоять хоча б дві одиниці підряд. Задача зводиться до підрахунку числа послідовностей довжини n без двох сусідніх цифр 1. Наведемо приклад при n= 4. Таких послідовностей існує 8.

0000 0010 0101 1001

0001 0100 1000 1010

Як підрахувати кількість таких послідовностей (позначимо через $P(n)$)? Нехай є послідовність довжини n. На першому місці в послідовності записана цифра 0. Число таких послідовностей $P(n\text{-}1)$. Розглянемо випадок, коли на першому місці записана цифра 1. Тоді на другому місці обов'язково повинна бути цифра 0. Число таких послідовностей $P(n\text{-}2)$. Отже тримуємо формулу: $P(n) = P(n\text{-}1) + P(n\text{-}2)$, - це не що інше, як формула обчислення чисел Фібоначчі. Таким чином, число послідовностей довжини n, у яких немає двох цифр 1, що стоять поряд дорівнює n-му числу Фібоначчі. Типу даних Longint вистачить тільки для обчислень при $n < 44$. Для обчислення кількості послідовностей при великих значеннях n потрібно використовувати "довгу" арифметику, а саме, процедури додавання і виводу "довгих" чисел, можна знову скористатись програмою **Fib_N**, переписавши її у вигляді функції.

Пропонуємо для самостійного розв'язання ще кілька складніших задач.

Задача **ЗВИЧАЙНІ ДРОБИ**. Написати програму додавання двох звичайних дробів

$$\frac{a}{b}+\frac{c}{d}=\frac{a\cdot\frac{нск(a,b)}{b}+c\cdot\frac{нск(a,b)}{d}}{нск(a,b)},$$

скорочення результату та, при потребі, виділення цілої частини, якщо на a, b, c, d накладено обмеження: $1 \le a, b, c, d \le 10^{100}$.

Задача **ПРОСТІ ЧИСЛА**. Написати програму визначення діапазону простих чисел, розміщених у проміжку [M . . N], де $K \le M$, $N \le K+100$, $1 \le K \le 10^{100}$.

VII.2. Опрацювання багаторозрядних чисел на базі їх текстового способу представлення в пам'яті комп'ютера

Існує кілька підходів до розв'язування задач довгої арифметики, кожен з яких визначається способом представлення багаторозрядних чисел в пам'яті комп'ютера. Серед них найбільш поширені табличний і рядковий.

а) Табличний підхід до опрацювання багаторозрядних чисел, що полягає в їх розбитті на окремі числа стандартної розрядності, занесенні їх у лінійні масиви і виконанні арифметичних операцій над елементами цих масивів [15]. Ідея представлення багаторозрядних чисел у вигляді лінійного цілочисельного масиву реалізована в чудовій книзі С.М. Окулова "Программирование в алгоритмах" [15], стор.9-24, тому повторюватись ми не будемо. Але зауважимо, що алгоритми довгої арифметики, засновані на табличному представленні багаторозрядних чисел, можна написати принципово відмінні, від запропонованих у названій книзі, звичайно, взявши на озброєння основну ідею – використання систем числення з досить великою основою.

Проте, як правило, практично не користуються системами числення з основою більшою за 16 тому, що існує обмежений набір для позначення цифр у цих системах. Але якщо кожну "цифру" такого числа зберігати у вигляді елемента лінійного масиву, то можна буде використовувати цю "цифру", не виходячи з десяткової системи числення, тобто, не думаючи про її позначення. Наприклад, якщо взяти за основу числення число 100, то 2004 можна представити: $2004 = 20*100^1 + 4*100^0$, а якщо позначити 20 = @, то отримаємо: 2004 = @4.

Підмітимо цікаву деталь -чотирьохцифрове десяткове число 2004 при переведенні у систему числення з основою 100 стане двоцифровим числом, тобто, використавши загальноприйняті позначення, матимемо: 2004_{10}= @4_{100}. Ситуація подібна з двійковою системою числення, коли $137_{10}=10001001_2$, тобто при суттєвому збільшенні основи числення суттєво зменшується кількість цифр переведеного числа. Отже, зробимо три очевидні висновки:

1) найпростіше переводити десяткові числа в системи з основою, що виражається у вигляді одиниці з нулями, наприклад, $123456789_{10}=xyz_{1000}$, де *x* - цифра, що дорівнює 123, *y* – цифра, що дорівнює 456, а *z* – цифра, що відповідає 789 у системі числення з основою 1000 ;

2) за основу числення доцільно взяти 10^9, адже це число "вміщається" у тип longint. Тоді число $123456789_{10}=\alpha_{1000000000}$, тобто стане одноцифровим;

3) Використання лінійного цілочисельного масиву позбавляє від необхідності вигадувати "цифри" для таких систем числення.

Звернемо увагу ще на одну особливість, навівши приклад. Нехай дано число 126345 067473986 003489605 307653806 543964545_{10}. Після його переведення в число з основою 10^9, одержимо:

1000000000^4	1000000000^3	1000000000^2	1000000000^1	1000000000^0
126345	67473986	3489605	307653806	543964545

У першому рядку таблиці вказано степінь основи числення, у другому "цифри" відповідних розрядів, записані в десятковій системі числення. Як бачимо, у кожному розряді нулі, що стоять зліва не записуються, тому при записуванні результату це доведеться обов'язково враховувати.

Щодо самих алгоритмів арифметичних операцій, запропонованих у [15] можна сказати наступне. Вони дуже ефективні, можливо за рахунок пониження наочності, тому учням можна пропонувати більш наочні алгоритми, які спираються на загальновідомі правила додавання, віднімання, множення і т.д., свідомо допускаючи деяку громіздкість програмних кодів.

На закінчення обговоримо ідею збільшення розрядності елементів масиву і вибору основи числення, що перевищує 10^9. Це також стане можливим, якщо елементи масиву матимуть не цілочисельний, а рядковий тип, наприклад, нехай дано масив A:array[1..3] of string[20]:

A[2]	A[1]	A[0]
10000000000000000000^2	10000000000000000000^1	10000000000000000000^0
3489605	34876534678762307638	00000000000543964545

Як видно з наведеної таблиці, у наймолодшому розряді зліва розміщено 11 нулів, що свідчить про стрінговий характер елементів масиву. Наймолодший розряд числа записано в A[0], степені основи числення співпадають з індексами відповідних розрядів. Але тепер стало неможливим виконання арифметичних операцій над елементами масиву A. Проте ця проблема вирішується з допомогою запропонованого нижче модуля **Long_Arytm** опрацювання багаторозрядних чисел, представлених у вигляді рядкових величин.

Вирішимо також питання про оптимальну основу числення. Рядковий тип дозволяє за розрядну одиницю взяти, наприклад, 10^{250}, але для уникнення аварійних зупинок при виконанні операції множення краще взяти за основу числення

число int(sqrt(10^{250}))≈10^{125}, зупинимось на 10^{100}. Система числення з такою основою буде цілком достатньою для представлення багаторозрядни чисел будь-якої довжини, з одного боку, і не приведе до непередбачених наслідків при обчисленнях, з іншого.

Отже, об'єднавши рядковий і табличний підходи до представлення багаторозрядних чисел та застосувавши описані вище процедури і функції, можна одержати достатньо ефективний спосіб розв'язування задач опрацювання багаторозрядних чисел.

b) Рядковий підхід до опрацювання багаторозрядних чисел заснований на тому, що рядкові (стрінгові, чи текстові) величини в мові Pascal можна розглядати, як лінійні масиви символів, тобто *array[byte] of char*, та на наявності в цій мові зручних процедур і функцій опрацювання рядків, внаслідок чого можна обійтися без додаткового використання масивів, що є однією з найважливіших переваг цього підходу над табличним. Недоліком є обмеженість кількості розрядів чисел (не більше 255 цифр), але він не надто суттєвий, адже числа з більшою розрядністю практично не використовуються, навіть додаткові умови до деяких олімпіадних задач, що вимагають опрацювання чисел з кількістю розрядів, що перевищує назване обмеження, виглядять неприродно і надумано. На користь абсурдності надвеликої кількості цифр в числах, що зустрічаються в олімпіадних задачах можна привести задачу "Лото", що пропонувалась на першому турі XII Всеукраїнської олімпіади, в якій пропонувалось визначити *k* у рівнянні $C_n^k = C$, при 1<*n*<500000 [16]. Під час розбору завдань було повідомлено, що *k* може містити до 2500 знаків, тому жодний із доступних методів, включаючи довгу арифметику не дасть результату швидше, ніж за 48 годин.

Розглянемо детальніше рядковий підхід. Його додатковою перевагою можна вважати відсутність необхідності написання базових алгоритмів 1 – 2, які повністю реалізовані стандартними засобами мови Pascal (процедури опрацювання файлів і рядкових величин). Очевидними також є базові алгоритми 3 та 5.1-5.4, наприклад, алгоритм 3 (визначення модуля цілого числа) реалізується, якщо запозичити термінологію мови Basic, такою однорядковою функцією:

```
function Abs(st:string):string;
begin
  if st[1]='-' then Delete(st,1,1);Abs:=st
end;
```

З цієї причини нижче зосередимось тільки на базових алгоритмах 4.1- 4.7. Спочатку наведемо базовий алгоритм 4.1 у вигляді функції **Long_Comp**:

```
function Long_Comp(st1,st2:string):boolean;
  var i,l1,l2:byte;st:string;
begin
 l1:=Length(st1);l2:=Length(st2);
 if l1<l2 then for i:=l1-l2 downto 1 do st1:='0'+st1{*}
         else for i:=l2-l1 downto 1 do st2:='0'+st2;{*}
  if st1>=st2 then Long_Comp:=true
  else Long_Comp:=false;
end;
```

Вона порівнює два багаторозрядні числа, записані рядками *st*1 та *st*2 і повертає *true*, якщо *st*1 ≥ *st*2 та *false*, якщо *st*1 ≥ *st*2. В якості коментаря достатньо звернути увагу на рядки, позначені {*****}(далі подібним чином будемо позначати рядки програмних кодів, що потребують коментарів). Цикли `for i:=l1-l2 downto 1 do st1:='0'+st1` та
`for i:=l2-l1 downto 1 do st2:='0'+st2`
доповнюють коротший рядок символів зліва символами ‘0’ до довжини довшого рядка. Це допоможе уникнути помилки у випадку, коли, наприклад, *st*1=432, *st2*=2345, адже без перетворення ‘432’ у ‘0432’ Pascal фактично порівняє ‘4320’ і ’2345’, що приведе до хибної відповіді.
Базовий алгоритм 4.2 реалізований в наведеній нижче функції **Long_Add** :

```
function Long_Add(st1,st2:string):string;                    {1}
  var i,l1,l2,pm,s1,s2,s,cod:word;st:string;                 {2}
begin
  if st1[0]<st2[0] then
  begin st:=st1;st1:=st2;st2:=st end;                        {3}
  st1:='0'+st1;                                              {4}
  l1:=Length(st1);l2:=Length(st2);pm:=0;                     {5}
  for i:=l1-l2 downto 1 do st2:='0'+st2;                     {6}
  for i:=l1 downto 1 do                          begin       {7}
    Val(st1[i],s1,cod);Val(st2[i],s2,cod);                   {8}
    s:=s1+s2+pm;                                             {9}
    st1[i]:=Char((s mod 10)+48);pm:=s div 10 end;           {10}
  if st1[1]='0' then Delete (st1,1,1);                      {11}
  Long_Add:=st1
end;
```

Для зручності коментування функції **Long_Add** вставлено номери її вузлових рядків у вигляді коментарів. З {1} випливає, що багаторозрядні числа вводяться з основної програми, як рядки *st*1 і *st*2, а функція повертає також у вигляді рядка результат додавання двох багаторозрядних чисел. Всі числові локальні змінні{2}, за винятком *cod*, яка згідно формату стандартної функції **Val** повинна мати тип integer або word, для економії пам’яті можуть бути оголошені, як byte. {3} переставляє на місце першого доданка число з більшою кількістю цифр, а {4} забезпечує ще один старший розряд, рівний 0 на випадок, коли сума буде довшою на одну цифру, ніж *st*1. {5} носить підготовчий характер для циклу {6}, де зліва від старшого розряду *st*2 добавляються нулі, щоб *st*1 та *st*2 мали однакову довжину. Основний смисл функції **Long_Add** реалізований в рядках{7}-{10}, спочатку з допомогою функції **Val** символи перетворюються в числа *s*1 та *s*2{8}, потім ці числа разом із числом *pm* утворюють число *s*.
Для роз’яснення значень величин *pm* та *s* пригадаємо алгоритм додавання натуральних чисел у стовпчик на прикладі:

```
  010  {pm}
  235  { s1}
+                                          (VII.3)
   29  {s2}
  264  {  s}
```

З (VII.3) видно, що *pm* – це цифри, які запам'ятовуються для додавання в старшому відносно поточного розряді, якщо сума цифр поточного розряду більша за 9, а для найменшого розряду pm надається значення 0. Число *s* – це результат додавання *s*1 та *s*2.

У {8} отримується нове значення *st*1[*i*] з допомогою функції **Char**, як остача від ділення *s* на 10. Таким чином, *pm* = *s* div 10, а *st*1[i] дорівнює **Char**((*s* mod 10)+48). Доданок 48 необхідний для того, щоб уникнути невідповідності коду цифри, як символа, та самої цифри.

Нарешті в {11} з допомогою процедури Delete видаляється лівий крайній символ *st*1, якщо він рівний 0.

Базовий алгоритм 4.3 реалізований в наведеній нижче функції **Long_Sub**:

```
function Long_Sub(st1,st2:string):string;
  var i,l1,l2,pm,s1,s2,s:byte;cod:integer;st:string;
begin
 l1:=Length(st1);l2:=Length(st2);pm:=0;
 for i:=l1-l2 downto 1 do st2:='0'+st2;
 for i:=l1 downto 1 do                              begin
    Val(st1[i],s1,cod);Val(st2[i],s2,cod);
    if s1>=s2+pm then begin s:=s1-(s2+pm);pm:=0 end     {12}
    else         begin s:=(s1+10)-(s2+pm);pm:=1 end;    {13}
    st1[i]:=Char(s+48)                                 end;
 i:=1;
 while st1[1]='0' do begin Delete (st1,1,1);i:=i+1 end;
 LongSub:=st1
end;
```

Для усвідомлення цієї функції, яка зовнішньо дуже подібна до функції **Long_Add**, цілком досить зосередити увагу на {12}, {13} та (VII.3).

```
  0   1   0  {pm}
  2   3   5  { s1}
–                                          (VII.4)
 (2+1) 9  {s2}
  2   0   6  {s1}
```

В розглянутих нижче конкретних прикладах (таблиця VII.2) видно, що алгоритм обчислення різниці для першого значно простіший, ніж для другого, він зводиться до порозрядного віднімання, адже всі цифри числа *b* не перевищують відповідні цифри числа *a* і не треба "позичати" одиницю в старшому розряді.

таблиця VII.2

	приклад 1						***приклад 2***							
позички	0	0	0	0		*позички*	9	9	9	9	9	9	10	
a	**2**	**9**	**7**	**3**	**4**	***a***	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
—						**—**								

b		**5**	**6**	**2**	**1**		***b***		**1**	**9**	**9**	**9**	**9**	**9**
	2	**4**	**1**	**1**	**3**			**9**	**8**	**0**	**0**	**0**	**0**	**1**

У другому прикладі показано ситуацію, де, навпаки, за “позичкою” потрібно звертатись аж до найстаршого розряду. В зв’язку з тим, що у вибраному нами рядковому способі представлення даних не застосовуються масиви (нагадаємо, що рядок - це лінійний масив символів, а не одноцифрових чисел), окремо опрацьовується кожен розряд, як символ, алгоритм для другого прикладу постійно вимагатиме циклічних змін у першому числі *a* (*st*1). Помітимо, що для відповіді не суттєво, позичати одиницю в старшому розряді *st*1, чи додавати її до відповідного розряду *st*2. Таким чином ми уникаємо циклічних змін величини *st*1, запам’ятовуючи щоразу збільшення старшого розряду *s*2 на *pm* (див. (VII.3)). Операція порозрядного віднімання виконується з допомогою команд `s:=s1-(s2+pm)` або `s:=(s1+10)-(s2+pm)`, залежно від значення *pm*, що показано в `{12}` та `{13}`. Описаний процес проілюстровано в таблиці VII.3:

таблиця VII.3

	приклад 1						***приклад 2***							
a	**2**	**9**	**7**	**3**	**4**		***a***	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
—							**—**								
b		**5**	**6**	**2**	**1**		***b***	**0+1**	**0+1**	**1+1**	**9+1**	**9+1**	**9+1**	**9+1**	**9**
	2	**4**	**1**	**1**	**3**				**9**	**8**	**0**	**0**	**0**	**0**	**1**

Базовий алгоритм 4.4 реалізований в наведеній нижче функції **Long_Mult_1**:

```
function Long_Mult_1(st1,st2:string):string;
  var i,l,pm,s1,s2,cod,s:word;st:string;
begin
  st1:='0'+st1;l:=Length(st1);pm:=0;
  for i:=l1 downto 1 do                          begin
    Val(st1[i],s1,cod);Val(st2,s2,cod);s:=s1*s2+pm;
    st1[i]:=Char((s mod 10)+48);pm:=s div 10     end;
    if st1[1]='0' then Delete(st1,1,1);
    Long_Mult_1:=st1
end;
```

Потреби в її детальному коментуванні немає, адже алгоритм множення багатоцифрового натурального числа на одноцифрове практично аналогічний до (VII.3), з тою різницею, що замість додавання одноцифрових чисел застосовується їх множення, а pm може бути не тільки нулем чи одиницею, а й цифрами 2, 3, ..., 8. Необхідність цієї функції в тому, щоб, як буде видно нижче, забезпечити множення багатоцифрових чисел, що проілюструємо функцією **Long_Mult**:

```
function Long_Mult(st1,st2:string):string;
  var i,j,l:word;st0,st:string;
begin
  st0:='';l:=Length(st2);
  for i:=l downto 1 do                  begin
    st:=Long_Mult_1(st1,Copy(st2,i,1));
    for j:=1 to l-i do st:=st+'0';
    st0:=Long_Add(st,st0)                end;
```

```
  Long_Mult:=st0;
end;
```

Вона дуже коротка і проста, тому також можна обійтись без детального коментування. Звернемо увагу лише на один момент, для чого, подібно до (VII .3) наведем ілюстрацію алгоритму множення в стовпчик:

```
    2 3 5  {s1}
×
      2 9  {s2}
 2 1 1 5   {st0}
+
 4 7 0     { st}
 6 8 1 5   {st0}
```

(VII .5)

Звідси видно, що в процесі множення величина *st*0 накопичує в собі результат, тому є початковою і кінцевою, а допоміжна величина *st* - це доданок, який щоразу зсувається на одну позицію вліво після чого додається до *st*0. У функції **Long_Mult** використовуються раніше розглянуті функції **Long_Mult_1** та **Long_Add**, тому всі вони можуть бути віднесені до єдиного модуля, названо вище **Long_Arytm**.

Для логічної завершеності цього модуля доповнимо його процедурою **Long-MoDiv**, яка реалізує базові алгоритми 4.6 - 4.7. Зауважимо, що у нашому випадку краще створити не дві функції, аналогічні операціям цілочисельного ділення *a mod b* та *a div b* в мові Pascal, а одну процедуру, адже відомий з арифметики алгоритм ділення з остачею в стовпчик (4) приводить одночасно до двох результатів – неповної частки та остачі (проілюструємо на прикладі):

```
ділене → 32405 | 297 ← дільник
       - 297     109 ← неповна частка
         2705
       - 2673
           32 ← остача від ділення
```

(VII .6)

Хоча уникненню надмірної громіздкості та складності цієї процедури сприяє застосування в ній описаних вище функцій **Long_Comp** та **Long_Sub,** проте створити її в такому ж компактному вигляді, як вищенаведені функції, не вдається, якщо взяти за основу алгоритм (4). Труднощі компактності процедури на Паскалі виникають через багаторазове використання процедур і функцій обробки рядків.

Зауважимо також, що ми свідомо не використовуємо функцію **Long_Mult_1**, замінивши її на кожному етапі циклічним відніманням. Як бачимо з наступного прикладу (VII .7), на I-V етапах віднімання слід зробити відповідно 8, 3, 1, 2 та 5 разів. На швидкодію програми заміна множення на віднімання також суттєво не вплине.

```
I - й етап   100000001|1203
           - 9624      83125
II - й етап  3760
           - 3609
III - й етап  1510
            - 1203                                  (VII .7)
IV - й етап    3070
             - 2406
V - й етап      6641
              - 6015
                626
```

Процедура **LongMoDiv**:

```
procedure LongMoDiv(st1,st2:string;var stmod,stdiv:string);
 var i,l,k, j:integer;st_1:string;bZero: boolean;
begin
 if Comp(st1, st2) then    begin                                    {!}
 st_1:='';stdiv:='';i:=1;l:=Length(st1);                            {14}
 repeat                                                             {15}
  k:=0;if st2='' then break;
  while not Comp(st_1,st2) do                        begin          {16}
   st_1:=st_1+st1[i];i:=i+1;stdiv:=stdiv+'0';
   if i>Length(st2) then break;bZero := true;
   if Length(st_1)>Length(st2) then          begin
    For j := 1 to Length(st_1)-1 do
     if st_1[j]<>'0' then bZero := false;
     if (st_1[Length(st_1)]<>str2[Length(str2)])and bZero
      then                                  begin
       Delete(st_1,Length(st_1),1);break end end end;               {17}
  Delete (stdiv,Length(stdiv),1);
  while st_1[1]='0' do                           begin
   if st_1 = '' then break; Delete (st_1,1,1) end;
  while Comp(st_1,st2) do     begin                                 {18}
   st_1:=Sub(st_1,st2);k:=k+1 end;                                  {19}
  stmod:=st_1;
  if st_1='' then stmod := '0';stdiv:=stdiv+Chr(k+48);              {20}
  while Length(stdiv)>Length(st1) do
   Delete(stdiv,Length(stdiv),1);
 until i>l;                                                         {21}
 while stdiv[1]='0' do Delete(stdiv,1,1);
end
  else                    begin                                     {!}
   stdiv:='0';  stmod:=st1 end;                                     {!}
end;{---------------------------------------end LongMoDiv}
```

Щоб уникнути зайвої деталізації, дамо лише найважливіші коментарі до цієї процедури. Після зрозумілих підготовчих команд {14} записано основний

цикл {15}–{21}. Він виконується, доки ділене *st*1 ≥ *st*2 (перевіряється з допомогою функції **Comp**). Цикл {16}–{17} накопичує *st_1*, доки воно не перевищить *st*2, та добавляє згідно алгоритму (VII.6) в кінець stdiv нулі. Цикл {18}-{19} та рядок {20} визначають чергову цифру *stdiv*. Звертаємо увагу на рядки з коментарем: **{!}**. Вони забезпечують правильні значення неповної частки та остачі при цілочисельному діленні багаторозрядних чисел у випадку, коли ділене менше за абсолютною величиною від дільника.

Закінчуючи розгляд опрацювання багаторозрядних чисел з допомогою рядкових величин, наведемо два традиційні приклади.

Приклад 1. Написати програму обчислення *n*-го члена послідовності Фібоначчі (при $n \le 1000$).

```
program Fib_N;
  uses Long_Arytm;                                          {22}
  var i,n:integer;str,str1,str2:string;
begin
  ReadLn(n);
  if (n=1)or(n=2) then WriteLn('1')
  else                          begin
  str1:='1';str2:='1';
  for i:=3 to n do        begin
    str:=Long_Add(str1,str2);                               {23}
    str1:=str2;str2:=str end end;
  WriteLn(str)
end.
```

Програма не потребує коментарів, хіба що можна наголосити на {22}- оголошення згаданого вище модуля **Long_Arytm**, при його відсутності слід описати в програмі **Fib_N** функцію **Long_Add**, яку викликається в {23}.

Приклад 2. Написати програму визначення числа *n*! ($0 \le n \le 500$).

```
program Factorial_N;
  uses Long_Arytm;
  var i,n:integer;str,str1,str2:string;
begin
  ReadLn(n);
  if (n=0)or(n=1) then WriteLn('1')
  else                          begin
  str:='1';
  for i:=2 to n do str:=Long_Mult(str,Char(i+48));
  WriteLn(str)
end.
```

Коментарі до програми **Factorial_N** повністю аналогічні попереднім.

VII.3 Розв'язування складних олімпіадних задач із програмування з допомогою комбінування методів довгої арифметики та інших методів програмування

(комбінаторна задача із використанням багаторозрядної арифметики)

Перед тренерами учасників учнівських олімпіад з інформатики, а точніше, з програмування, постійно виникають питання, *якими повинні бути задачі* та *як навчати їх розв'язуванню*. Питання зрозумілі і незрозумілі водночас, якщо враховувати, що програмування – це нетрадиційний для школи предмет, який вимагає вільного володіння математичним апаратом, а задачі важко розділяти за віковими категоріями. Перш за все треба розібратись, які розділи математики та в якому порядку і поєднанні доцільно розглядати у процесі підготовки юного програміста, на яких типових задачах слід починати шліфовку знань, умінь і практичних навичок? На наш погляд, відповідь на це питання дуже вдало дано у [16], де після розгляду арифметики багаторозрядних чисел поставлено комбінаторні алгоритми, а вже потім розглянуто теми перебору і методів його скорочення, алгоритми на графах, обчислювальну геометрію і т.д. Тому і ми тут, зовсім не випадково, зосередимось на розгляді комбінаторних алгоритмів, взявши у якості робочого матеріалу дуже цікаву задачу з [16]. Наступна задача пропонувалась на ІІІ-у етапі Всеукраїнської олімпіади-2005 з інформатики у Київській області.

Задача 15. ([16], стор. 75) На смужці клітинкового паперу висотою в одну і довжиною *N* клітинок деякі клітинки зафарбовані в чорний і білий кольори.

Кодом клітинки є послідовність чисел – кількості ідучих підряд чорних клітинок зліва направо. Наприклад, для наведеної смужки кодом буде послідовність 2, 3, 2, 8, 1. При цьому кількість білих клітинок, які розділяють групи чорних ніде не враховуються (головне, щоб дві сусідні групи розділені принаймні однією білою клітинкою). Одному й тому ж коду можуть відповідати кілька смужок, наприклад наведеному коду відповідає й така смужка:

Задача полягає в тому, щоб знайти кількість смужок довжини *N*, які від-

повідають заданому коду.

Вхідні дані. У єдиному рядку файлу файла *input.txt* записано число *N* – довжина смужки ($1 \le N \le 200$), потім число *K* ($0 \le K \le (N+1)/2$) – кількість чисел у коді і далі *K* чисел, що визначають код.

Вихідні дані. У вихідний файл *output.txt* записується кількість смужок довжини *N*, які відповідають заданому коду.

а) ***Дослідження умови та виділення параметрів***. З умови видно, що задача має комбінаторний характер, причому на підрахунок смужок, а не на їх генерацію, отже слід знайти потрібні формули. На користь цього також свідчить можливість надто великого числа смужок, генерація яких може виходити за межі стандартних розрядів та допустимих проміжків часу.

Зробимо позначення: *s* - сума чисел коду та числа *K* - 1, *Ns* = *N* – *s* - кількість незафарбованих клітинок, які слід розмістити серед *K*1 = *K* + 1 місць. Кількість варіантів такого розподілу, яку позначимо *KS* і буде шуканою відповіддю. З огляду на це, запитання задачі 3 можна сформулювати інакше: *знайти число KS - кількість способів розміщення Ns пропусків на K*1 *місцях, щоб їх сума дорівнювала N*s. (*)

Зауваження. Слід звернути увагу на те, що у [16] за s помилково взято суму чисел коду та числа *K*. Там же, у вказівці до розв'язування, помилково стверджується, що залишається підрахувати *число розміщень N – s по відомій кількості місць – K* + 1. Але жодна із формул - $A_N^k = = N^k$ (розміщень з повтореннями) та $A_N^M = N! / (N - M)!$ (розміщень без повторень) у нашому випадку не підходить. Очевидно, задача відноситься до категорії складних (не випадково вона наведена під №15 із 17, включених у якості додаткових до розділу "Комбінаторні алгоритми"), тому, на нашу думку, короткої вказівки (причому з указаними неточностями та обмеженнями, що потребують застосування багаторозрядної арифметики) зовсім не достатньо. Поставимо перед собою завдання: *знайти ефективний алгоритм пошуку числа KS, яке за рахунок умов 1 ≤ N ≤ 200 та 0 ≤ K ≤ (N+1)/2 може бути багаторозрядним числом*. Забігаючи наперед, зауважимо, що в процесі розв'язання цього завдання доведеться шукати відповіді й на інші, які виникнуть по ходу і з яких випливатимуть важливі наслідки.

b) ***Пошуки математичної моделі***. Як уже згадувалось вище, формули числа розміщень застосувати до задачі не можна. Але при визначенні кількості варіантів завжди виникає спокуса використати одну з комбінаторних формул. Нескладний аналіз допоможе представити шукане число у вигляді числа комбінацій без повторень, але з огляду на наступні міркування, це буде зроблено значно нижче.

Справа в тому, що математичну модель задачі вибирають здебільшого залежно від кількох обставин: обмежень на вхідні та вихідні дані, віку виконавців або стану їх математичної підготовки. Ми поки що не будемо звертати увагу на формат даних, обмежившись стандартними типами цілих чисел, зокрема типом *longint*[7] .

Дві інші обставини проігнорувати ніяк не можна. За браком достатньої математичної підготовки, учні змушені користуватись спрощеною моделлю задачі, що приводить до неповного чи неефективного розв'язку. Виходячи з цього, можна стверджувати, що учні, розв'язуючи подібну задачу, майже обов'язково, підуть не шляхом застосування комбінаторики, яка вивчається, на жаль тільки в 11 класі, причому без достатнього закріплення та застосування.

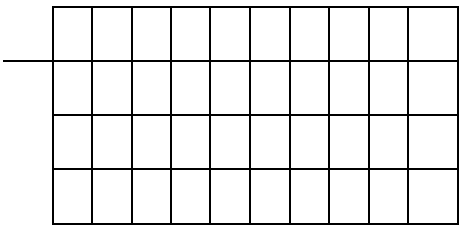
Отже, прослідкуємо за емпіричним ходом пошуку робочих формул. Зупинимось на прикладі смужки при *N* = 8, *K* = 3, код 1, 2, 1. Як початкову, візьмемо смужку (малюнок VII.1):

мал. VII..1

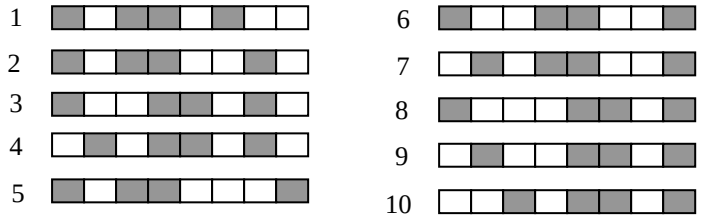
Зробимо підрахунок, попередньо згенерованих варіантів смужок. Як видно з малюнка VII.2, таких смужок 10. Як уникнути генерування смужок? З (*) випливає, що задача для смужки, зображеної на малюнку VII.1, зводиться до варіантів, які видно з наступної таблиці:

0

[7] Маємо на увазі тип *longint* для мови Turbo Pascal.



таблиця VII.4



мал. VII..2

Очевидно, що *KS* буде функцією від двох величин – *Ns* та *K*+1, тобто *KS*(*Ns*,*K*+1), у нашому прикладі *KS*(2,4). Спробуємо визначити формулу для знаходження *KS*(2, 4), а на її основі формулу для визначення *KS*(*i*, *j*). Очевидно, що *KS*(1,*j*) = *j*, *KS*(*i*,1)=1, *KS*(0,*j*)=1. (VII.III.1)

Отже: *KS*(2, 1) = 1;

KS(2, 2) = *KS*(2, 1) + *KS*(1, 1) + *KS*(0, 1) = 1 +1 + 1 = 3;

KS(2, 3) = *KS*(2, 2) + *KS*(1, 2) + *KS*(0, 2) = 3 + 2 + 1 = 6; (VII.III.2)

KS(2, 4) = *KS*(2, 3) + *KS*(1, 3) + *KS*(0, 3) = 6 + 3 + 1 = 10

Міркуючи подібним чином, визначимо *KS*(3, 4) і одержимо (VII.III.3), а потім, узагальнивши (VII.III.1), (VII.III.2) та (VII.III.3), запишемо рекурентні співвідношення для визначення *KS*(*i*, *j*), які пропонуємо довести самостійно. Зауважимо, що кількість доданків у сумі дорівнює *i*+1.

KS(3, 1) = 1;

KS(3, 2)=*KS*(3, 1)+*KS*(2, 1)+ *KS*(1, 1)+*KS*(0, 1)=1 +1 + 1 + 1 = 4;

KS(3, 3)=*KS*(3, 2)+*KS*(2, 2)+ *KS*(1, 2)+*KS*(0, 2)=4 + 3 + 2 + 1 = 10; (VII.III.3)

KS(3, 4)=*KS*(3, 3)+*KS*(2, 3)+*KS*(1, 3)+*KS*(0, 3)=10 + 6 + 3 + 1 = 20

Отже, остаточно маємо:

KS(1, *j*) = *j*, *KS*(*i*, 1) = 1, *KS*(0, *j*) = 1,

KS(*i*,*j*) = *KS*(*i*,*j*-1)+*KS*(*i*-1,*j*-1)+*KS*(*i*-2,*j*-1)+…+*KS*(1,*j*-1)+*KS*(0,*j*-1) (VII.III.4)

Останні рекурентні співвідношення дають змогу скласти алгоритм розв'язування задачі, але для його спрощення проведемо додаткове дослідження з допомогою Microsoft Excel (див. таблицю VII.5). Проаналізувавши знайдені в ній результати (2) та (3), помічаємо, наприклад, що *H*9 = *H*8 + *G*8 + *F*8 + *E*8 + *D*8 + *C*8 = 252, тобто значення вибраної комірки *X*(*i*, *j*) дорівнює сумі значень комірок діапазону *X*(*i* - 1, 0):X(*i* - 1, *j*). Але тоді *H*9 = *H*8 + *G*9, тобто *X*(*i*, *j*) = *X*(*i*, *j*-1) : X(*i* - 1, *j*).

A	B	C	D	E	F	G	H	I	J	K	L	M
кількість пропусків, які треба розмістити												
	j \ i	0	1	2	3	4	5	6	7	8	9	10
для розміщення пропусків	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	2	3	4	5	6	7	8	9	10	11
	3	1	3	6	10	15	21	28	36	45	55	66
	4	1	4	10	20	35	56	84	120	165	220	286
	5	1	5	15	35	70	126	210	330	495	715	1001
	6	1	6	21	56	126	252	462	792	1287	2002	3003
	7	1	7	28	84	210	462	924	1716	3003	5005	8008
	8	1	8	36	120	330	792	1716	3432	6435	11440	19448
	9	1	9	45	165	495	1287	3003	6435	12870	24310	43758
	10	1	10	55	220	715	2002	5005	11440	24310	48620	92378
	11	1	11	66	286	1001	3003	8008	19448	43758	92378	184756
	12	1	12	78	364	1365	4368	12376	31824	75582	167960	352716

місця												
	48	1	48	1176	19600	249900	2598960	22957480	177100560	1217566350	7575968400	43183019880
	49	1	49	1225	20825	270725	2869685	25827165	202927725	1420494075	8996462475	52179482355
	50	1	50	1257	22100	292825	3162510	28989675	231917400	1652411475	10648873950	62828356305
	51	1	51	1326	23426	316251	3478761	32468436	264385836	1916797311	12565671261	75394027566
	52	1	52	1378	24804	341055	3819816	36288252	300674088	2217471399	14783142660	90177170226
	53	1	53	1431	26235	367290	4187106	40475358	341149446	2558620845	17341763505	1,07519E+11

таблиця VII.5

Виходячи з цього, з (VII.III.4) одержуємо остаточні робочі формули:
$KS(1, j) = j$, $KS(i, 1) = 1$, $KS(0, j) = 1$, $KS(i,j) = KS(i,j\text{-}1)+KS(i\text{-}1,j)$ (VII.III.5)

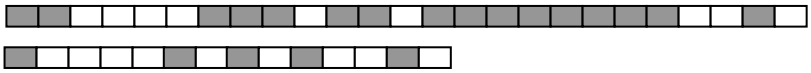
Для перевірки та закріплення формул (VII.III.5) пропонуємо:

1. Обчислити з її допомогою та перевірити генеруванням кількість смужок, взявши за початкові:

input.txt :	*4 0*	*4 2 2 2*	*5 2 1 2*	*10 4 1 2 1 1*
output.txt:	*1*	*0*	*3*	*15*

мал.VII. 3

2. Обома способами визначити *KS* для смужок, наведених на малюнку VII.4.



мал. VII..4

Наголосимо на деяких цікавих особливостях задачі. Врахувавши (*), не вплинемо на результат, якщо перетворимо початкову смужку, наведену в умові задачі і зображену на малюнку 4 (N = 25, K = 5, код 2, 3, 2, 8, 1). Визначивши для обох смужок s_1 = 20, Ns_1 = 5, $K_1 + 1 = 6$ та s_2 = 14, Ns_2 = 5, $K_2 + 1 = 6$, бачимо, що $Ns_1 = Ns_2$ і $K_1 + 1 = K_2 + 1$, отже для цих смужок KS буде однакове. Назвемо смужки з однаковими Ns та $K + 1$ *еквівалентними* і будемо першим кроком алгоритму виконувати заміну початкової смужки на її *мінімальну еквівалентну*, назвавши цю дію *мінімізацією* смужки. Це принесе значний ефект при визначенні KS генеруванням нових смужок. Наведемо алгоритм мінімізації:

1) Початок. п. 2
2) Визначити суму коду (s'). п. 3
3) Одержати новий код (s''), замінити кожне його число на 1. п. 3
4) N := s' - s''. п. 4
5) Кінець.

с) ***Пошуки ефективного алгоритму розв'язування задачі***[8]. Рекурсивний алгоритм, який не потребує доведення і оцінки на складність, легко одержати, використавши (VII.II.5). Але необхідно дослідити, в яких межах він ефективно працюватиме без використання багаторозрядної арифметики, а також, чи можна його застосувати для багаторозрядних чисел. Серед проблем можуть бути переповнення через надто велике заглиблення в рекурсію та не допустимий час виконання.

```
program Strip _REC; {рекурсивна програма}
   var n,k,s,i,j,b:integer;f:text;
       a:array[1..50][9] of longint; p:array[1..50,1..50]of longint;
   function KS(i,j:longint):longint;
   begin
      if i=1 then KS:=j
      else if j=1 then KS:=1 else KS:=KS(i,j-1)+KS(i-1,j)
    end;
begin
   assign(f,'n3.dat');reset(f); read(f,n,k);
   for i:=1 to k do begin read(f,a[i]);s:=s+a[i] end;
   close(f);
   i:=n-(s+(k-1));j:=k+1;
   assign(f,'n3.res');rewrite(f); writeln(f,KS(i,j));close(f);
end.
```

[8] Алгоритми будемо наводити у вигляді програм на мові Turbo Pascal.
[9] Тут і далі обмеження для масивів взято довільно.

Значно ефективнішим є циклічний алгоритм, який передбачає з допомогою вкладених циклів для обчислення *KS*(5, 6) послідовно, рядок за рядком, визначити таблицю 6×6:

```
program Strip_For; {циклічна програма}
    var n,k,s,i,j,b:longint;f:text;
    a:array[1..256]of longint; p:array[1..100,1..100]of longint;
begin
    assign(f,'n3.dat');reset(f);read(f,n,k);
   for i:=1 to k do begin read(f,a[i]);s:=s+a[i] end;
   close(f);
   b:=n-(s+(k-1));
   for i:=1 to b+1 do p[1,i]:=1;  for j:=1 to k+1 do p[j,1]:=1;
   for j:=2 to k+1 do
     for i:=1 to b+1 do  p[i,j]:=p[i-1,j]+p[i,j-1];
   assign(f,'n3.res');rewrite(f); writeln(f,p[b+1,k+1]);close(f);
end.
```

Крім того, в циклічному алгоритмі простіше використати ”довгу арифметику”. Але слід врахувати, що, наприклад, при застосуванні текстового представлення багатоцифрових чисел, коли кожен елемент таблиці матиме тип *string* (256 байт), виникнуть проблеми переповнення пам’яті, тому краще замість одночасного обчислення прямокутної таблиці багаторазово обчислювати лінійну таблицю. Можна знайти інші виходи, наприклад, застосування динамічної пам’яті. Цим багато учнів і закінчать роботу над подібною задачею, хоча одержану програму ще не можна назвати ефективною.

Для пошуку справді ефекивного алгоритму ще раз скористаємось таблицею 2. З неї перш за все можна побачити межі дії алгоритму без використання багаторозрядних чисел. Права та нижня межі таблиці відсутні, що вказує на можливе продовження її в цих напрямках. Пунктирні межі між рядками №13-47 вказують на розрив таблиці. Нижній фрагмент (рядки 48-53) наведено лише для того, щоб указати одне із значень, що виходить за межі типу *longint* - *KS*(10, 53) = 1,07519E+11. У лівому верхньому куті таблиці виділено прямокутну область, яка ілюструє обчислення *KS*(5, 6) = 252, що дорівнює кількості смужок наведеного в умові задачі зразка (мал. 4). Як виявляється, таблиця відображає відомий числовий трикутник біноміальних коефіцієнтів Паскаля, якщо за його вершину взяти її лівий верхній кут. У цій частині таблиці заштрихованими та білими клітинками показано ”поверхи” трикутника Паскаля. На перший погляд, обчислювати значення таблиці з допомогою формули числа комбінацій $C_n^m = n!/(m!(n-m)!)$ не зрозуміло як, адже, наприклад, *KS*(5, 6) = C_{10}^5 . Але, представивши частину таблиці 2, в дещо іншому вигляді, де для вирівнювання ширини стовпчиків всі числа замінено відповідними записами зразка C_j^i і сірими лініями виділено рядки трикутника Паскаля (табл. VII.6), легко встановити залежність між *i, j* з одного боку та *n, m* з іншого.

C	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*
0	1	2	3	4	5	6	7	8	9	10

C_0^0	C_1^1	C_2^2	C_3^3	C_4^4	C_5^5	C_6^6	C_7^7	C_8^8	C_9^9	C_{10}^{10}
C_1^0	C_2^1	C_3^2	C_4^3	C_5^4	C_6^5	C_7^6	C_8^7	C_9^8	C_{10}^9	C_{10}^{10}
C_2^0	C_3^1	C_4^2	C_5^3	C_6^4	C_7^5	C_8	C_9^7	C_{10}^8	C_{11}^9	C_{10}^{10}
C_3^0	C_4^1	C_5^2	C_6^3	C_7^4	C_8^5	C_9^6	C_{10}^7	C_{11}^8	C_{12}^9	C_{10}^{10}
C_4^0	C_5^1	C_6^2	C_7^3	C_8^4	C_9^5	C_{10}^6	C_{11}^7	C_{12}^8	C_{13}^9	C_{10}^{10}
C_5^0	C_6^1	C_7^2	C_8^3	C_9^4	C_{10}^5	C_{11}^6	C_{12}^7	C_{13}^8	C_{14}^9	C_{10}^{10}
C_6^0	C_7^1	C_8^2	C_9^3	C_{10}^4	C_{11}^5	C_{12}^6	C_{13}^7	C_{14}^8	C_{15}^9	C_{10}^{10}
C_7^0	C_8^1	C_9^2	C_{10}^3	C_{11}^4	C_{12}^5	C_{13}^6	C_{14}^7	C_{15}^8	C_{16}^9	C_{10}^{10}
C_8^0	C_9^1	C_{10}^2	C_{11}^3	C_{12}^4	C_{13}^5	C_{14}^6	C_{15}^7	C_{16}^8	C_{17}^9	C_{10}^{10}
C_9^0	C_{10}^1	C_{11}^2	C_{12}^3	C_{13}^4	C_{14}^5	C_{15}^6	C_{16}^7	C_{17}^8	C_{18}^9	C_{10}^{10}
C_{10}^0	C_{11}^1	C_{12}^2	C_{13}^3	C_{14}^4	C_{15}^5	C_{16}^6	C_{17}^7	C_{18}^8	C_{19}^9	C_{10}^{10}
C_{11}^0	C_{12}^1	C_{13}^2	C_{14}^3	C_{15}^4	C_{16}^5	C_{17}^6	C_{18}^7	C_{19}^8	C_{20}^9	C_{10}^{10}

таблиця VII.6

Очевидно, що $m = i$, $n = i + j - 1$, тобто $KS(i, j) = C_{i+j-1}^{\ i}$, а, врахувавши позначення (*), остаточно одержимо: $KS = C_{i+j-1}^{\ i}$ (VII.III.6)

Без значної і систематичної практики застосування комбінаторики, що характерно для більшості учнів, (VII.III.6) встановити аналітично не легко, а з допомогою таблиці 3 вона стає очевидною.

Покажемо, як можна одержати цю формулу без наведених вище міркувань. Зрозуміло, що загальна кількість білих клітинок у смужці після усіх вставок дорівнюватиме $j + i$. Якщо прийняти, що вставка кожної з i додаткових білих клітинок можлива тільки між двома іншими білими клітинками, то першу з i білих клітинок можна розмістити j - 1 способами, наступну - j способами і т.д., а останню можна розмістити $i + j$ - 1 способами. Отже кількість таких розміщень дорівнюватиме $A_{\,i+j-1}^{\ i} = (i + j - 1)! /(i + j - 1 - i)! = (i + j - 1)! /(j - 1)!$. Але тут враховано всі розміщення білих клітинок, а в нашій задачі білі клітинки, які вставляються - рівноправні, тому одержане число слід ще поділити на кількість перестановок з i елементів, тобто на $i!$, у результаті чого отримаємо рівність (VII.III.6).

Цю формулу важко побачити учневі, який з відомих причин навіть не обов'язково знайомий з комбінаторикою, тому, для нього більш звична математична модель, яка використовує (VII.III.5). Поза сумнівом, вищеописана схема розв'язування даної задачі, тобто: рекурсивна програма ***Strip_REC*** $\Rightarrow$ циклічна програма ***Strip_FOR*** $\Rightarrow$ програма на базі (VII.III.6) - це найбільш природний шлях пошуку повного і ефективного розв'язання, тому можна порадити учителям саме такої схеми дотримуватись при роботі з учнями над подібною задачею.

Необхідно також зауважити, що розглянуту задачу слід давати на олімпіаді диференційовано, відповідно до віку учасників. Це можна зробити за рахунок зміни формату вхідних і вихідних даних, а також тестів, для 8-9 класів достатньо обмежитись використанням стандартних числових типів, що дозволить обмежитись рекурсивним чи циклічним алгоритмами, для 10-11 класів можна дати наведені технічні умови, що вимагатиме опрацювання багаторозрядних чисел і може привести до застосування комбінаторики.

Отже, доведення формули (VII.III.6), хоч і доступне далеко не всім учасникам олімпіади, але радикально спрощує алгоритм та забезпечує одержання повного розв'язку даної задачі, в тому числі на множині багаторозрядних чисел. Наведемо відповідну програму:

```
program Strip;[10]
   uses Long_Arytm;
  var n,k,i,s,j,m:integer;a:array[1..101] of integer;f,f1:text;b, ks:string;
begin{Strip}
  Assign(f,'input.txt');  Reset(f); Assign(f1,'output.txt'); ReWrite(f1);
  Read(f,n); Read(f,k);
  for m:=1 to k do  Read(f,a[m]);
  if (n<1) or (n>200) or (k<0) or (k>(n+1)/2) then
     begin WriteLn(f1,'ERROR'); Close(f); Close(f1) end
 else  for m:=0 to k do s:=s+a[m+1];
  j:=k+1; i:=n-s-k+1;
 MoDiv(Fact(i+j-1), Mult(Fact(i), Fact(j-1)),b, ks);
 WriteLn(f1,ks); Close(f); Close(f1);
end.{end  Strip}
```

Зауважимо, що програма ***Strip*** не у всіх випадках повертає правильне значення *ks*. Проаналізуємо кілька вхідних файлів і результати занесемо до таблиці VII.5. (вихідні файли наведено у додатку).

input1.txt: 8 3 1 2 1

input2.txt: 25 5 2 3 2 8 1

input3.txt:

111 52 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

input4.txt:

200 50 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

input5.txt:

200 60 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

input6.txt:

200 62 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

input7.txt:

200 65 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

input8.txt:

200 75 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

input9.txt:

200 90 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

[10] Програма враховує застосування "довгої арифметики" з представленням чисел у вигляді рядкових величин, необхідні процедури і функції для реалізації базових алгоритмів опрацювання багаторозрядних чисел, зокрема порівняння чисел, додавання, віднімання, множення багацифрового числа на одноцифрове, множення багатоцифрових чисел, визначення факторіалу та цілочисельне ділення, включені у модуль Long_Arytm, який оголошений у програмі, наведено в попередньому розділі..

input10.txt:
200 20 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 11

таблиця VII.5

Згідно (VII.III.6), наприклад, у тесті №10 необхідно обчислити 20!, 151! та 171!, але представлення багаторозрядних чисел у рядковому вигляді ставить під сумнів правильність визначення факторіалів таких великих чисел та значення неповної частки при їх цілочисельному діленні. Для уникнення цієї про-

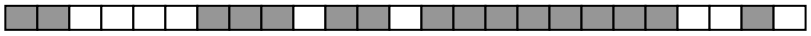
$$C_{171}^{151} = \frac{171!}{151!\cdot 20!} = \frac{152\cdot 153\cdot ...170\cdot 171}{20!}$$

блеми слід модернізувати програму ***Strip***, використавши перетворення:

З огляду на елементарність при наявності модуля ***Long_Arytm*** пропонуємо виконати це перетворення самостійно та одержати результати до наведених тестів.

На цьому дослідження задачі можна вважати цілком вичерпаним. Проте деякі виявлені факти дозволяють зробити додаткові висновки. Наприклад, ми поки-що не побачили практичної потреби у згаданій вище мінімізації смужки. Але можна запропонувати нову задачу:

Задача 15′. На смужці клітинкового паперу висотою в одну і довжиною *N* клітинок деякі клітинки зафарбовані в чорний і білий кольори. Кодом клітин-



ки є непуста множина чисел – кількості ідучих підряд чорних клітинок зліва направо. Наприклад, для наведеної смужки кодом буде множина {2, 3, 2, 8, 1}. При цьому кількість білих клітинок, які розділяють групи чорних ніде не враховуються (головне, щоб дві сусідні групи розділені принаймні однією білою клітинкою). Задача полягає в тому, щоб знайти кількість смужок довжини *N*, які відповідають заданому коду.

Вхідні дані. У єдиному рядку файлу файла *input.txt* записано число *N* – довжина смужки ($1 \le N \le 25$), потім число *K* ($0 \le K \le (N+1)/2$) – кількість чисел у коді і далі *K* чисел, що визначають код.

Вихідні дані. У вихідний файл *output.txt* записується кількість смужок довжини *N*, які відповідають заданому коду.

Задача 15′ лише зовнішньо дуже схожа на задачу 15. Насправді ж з умови випливає, що слід враховувати всі смужки з послідовностями чорних клітинок, що є перестановками з повторенням у коді. Звідси легко вивести формулу для визначення *Ks* у задачі 15′. У зв'язку з цим у файлі *input.txt* взято обмеження $1 \le N \le 25$. Очевидно, що перша задача легша від другої, більше того, вона є її частинним випадком, а побачити це допоможе введене нами поняття мінімізації смужки.

Не будемо вказувати на інші виявлені факти. По-перше, це дасть можливість уважним читачам зробити власні маленькі відкриття, по-друге, нема потреби розкривати таємниці ще кількох нових задач, які можуть бути представлені у якості олімпіадних. Зауважимо лише, що ми намагались прослідкувати, як,

шукаючи розв’язок однієї задачі можна прийти до інших, можливо не менш цікавих, а то й зовсім несподіваних, тобто продемонструвати шлях *від задачі до задачі*.

Надзвичайно цінною якістю особистості є вміння швидко, за екстремальних обставин, як буває на олімпіадах, шукати розв’язки задач, та не менш важливе вміння без поспіху, але грунтовно, проводити повне дослідження поставленої проблеми, що часто приводить до нових проблем.

додаток

output1.txt: 10
output2.txt: 252
output3.txt: 2558620845
output4.txt:
22294864007509202019319906 27
output5.txt:
39349707693707417946654498477149309918680
output6.txt:
21045998620349722340045216678707617713760
output7.txt:
52164357285294491100042489748515630828 00
output8.txt:
61643832300695848134415168385026326 0
output9.txt:
23225278051098322489290
output10.txt: 0

VIII. МЕТОД ДИНАМІЧНОГО ПРОГРАМУВАННЯ

VIII. 1 Поняття про задачі динамічного програмування.

Людина в своїй діяльності часто зустрічається з потребою вибрати найбільше чи найменше, найкраще чи найгірше при управлінні певним процесом. Математичні задачі такого характеру називаються *задачами оптимізації*. Для їх розв’язування розроблено багато методів, проте значна частина таких задач потребує не тільки знаходження *оптимального розв’язку*, а й для його відшукання використання *оптимальної стратегії*, тобто оптимального керування процесом розв’язування задачі, в іншому випадку розв’язування не буде ефективним. *Оптимальна стратегія володіє тією властивістю, що якими б не були початковий стан і початковий розв’язок, інші розв’язки, повинні бути оптимальними по відношенню до стану, отриманому в результаті початкового розв’язку.* Тобто, оптимальний розв’язок на 1-му кроці визначається оптимальным розв’язком на (i - 1) -му кроці і "витратами" на перехід від (i - 1) - го кроку до i - го.

До задач оптимізації відносяться й безліч комбінаторних та переборних задач, породжених практичною і науковою діяльністю людини. Існує наукова дисципліна - *дискретна математика* - одним із завдань якої є саме розробка алгоритмів розв’язання таких задач.

Застосування методу до розв'язування комбінаторних задач власне кажучи означає використання принципу декомпозиції: спочатку знаходяться розв'язки найпростіших підзадач, потім вони використовуються для відшукання розв'язків усе більш складних підзадач і, нарешті, для розв'язку вихідної задачі.

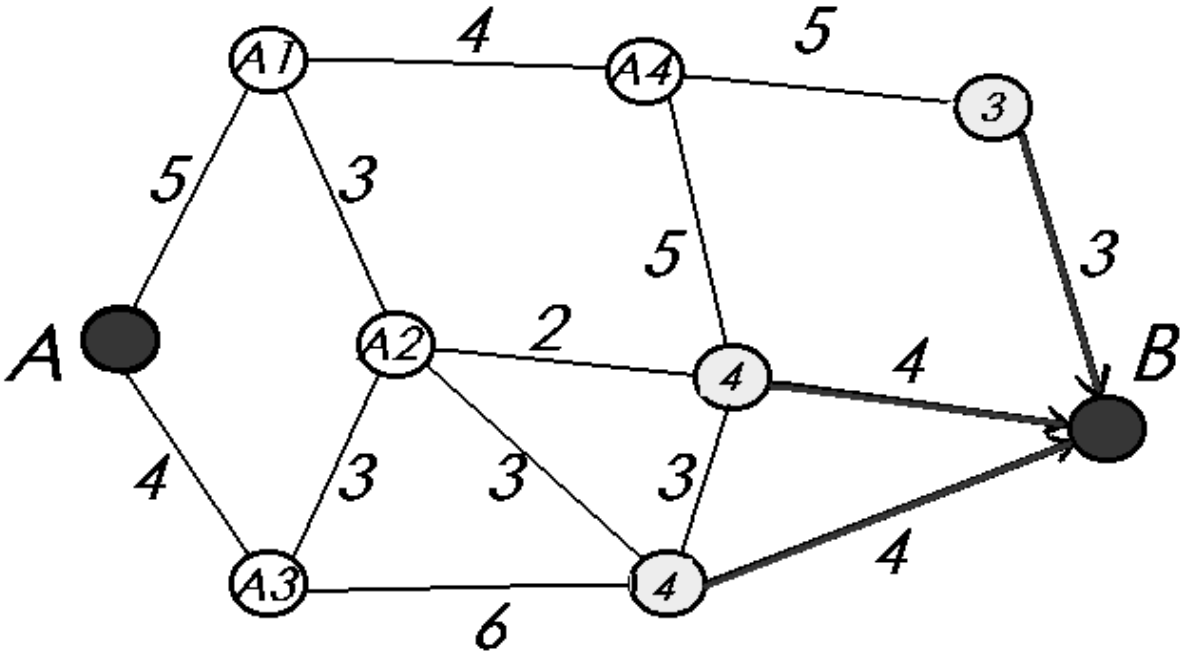
Такий метод розв'язування задач оптимізації називається *методом динамічного програмування*, а відповідні задачі - *задачами динамічного програмування*. Хоча в неявному вигляді цей метод застосовували ще К.Маклорен (1698-1746) і навіть древньогрецький математик Архімед, він сформульований був порівняно недавно, менш як півстоліття тому американським математиком Р. Беллманом, а поширення одержав тільки з появою ЕОМ.

Щоб сформувати уявлення про задачі та метод динамічного програмування, спочатку розглянемо дві задачі.

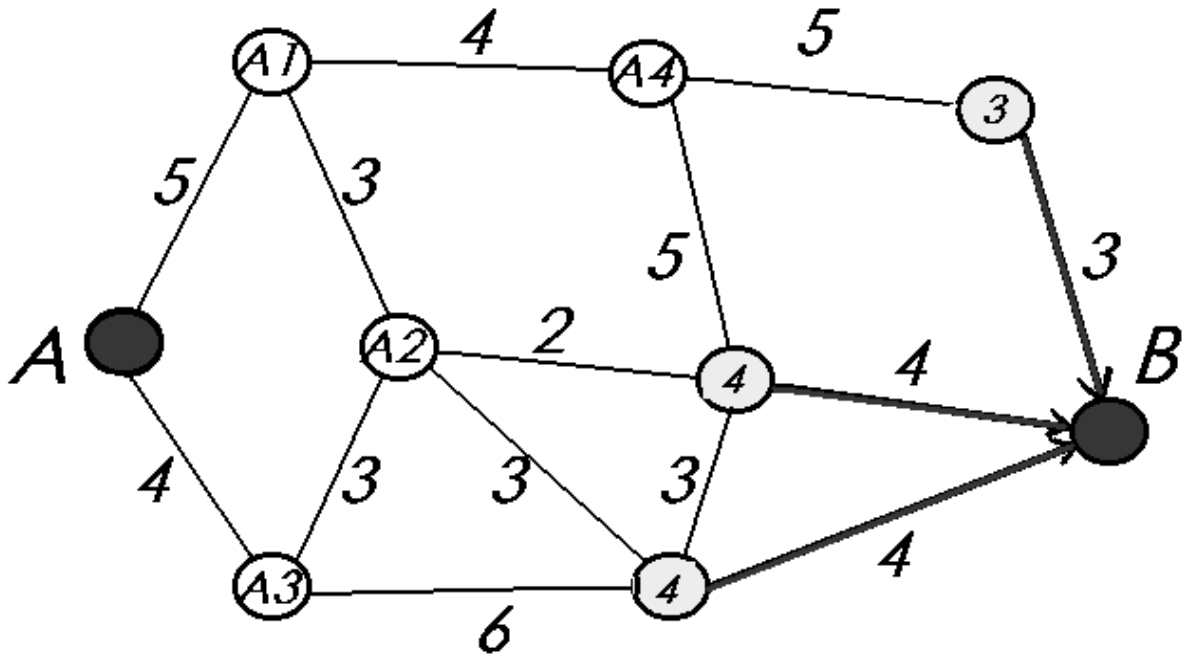
Задача 1. **Уявіть собі, що ви начальник космічної рятівної служби. На малюнку VIII.1 зображено схему астероїдів, на яких розміщено заправні станції. Перельоти, для яких вистачає пального, позначені на схемі відрізками. Числа біля відрізків показують кількість днів, потрібних для перельоту між станціями. Ви перебуваєте в пункті А, а в пункті В трапилась аварія. Як найшвидше дістатися в пункт В?**

Сформульована задача є задачею на відшукання *найкоротшого шляху*. Хоча цей найкоротший шлях можна відшукати, перебравши всі можливі маршрути перельоту з А в В, її, із зрозумілих причин,зручно розв'язувати методом динамічного програмування.

Розв'язання: Проаналізуємо, з яких пунктів можна потрапити в В. Це пункти A_7,A_5 та A_6. Поставимо стрілки з цих пунктів у пункт В, а в самих кружечках замість їх назв поставим відповідні числа(малюнок VIII.2)

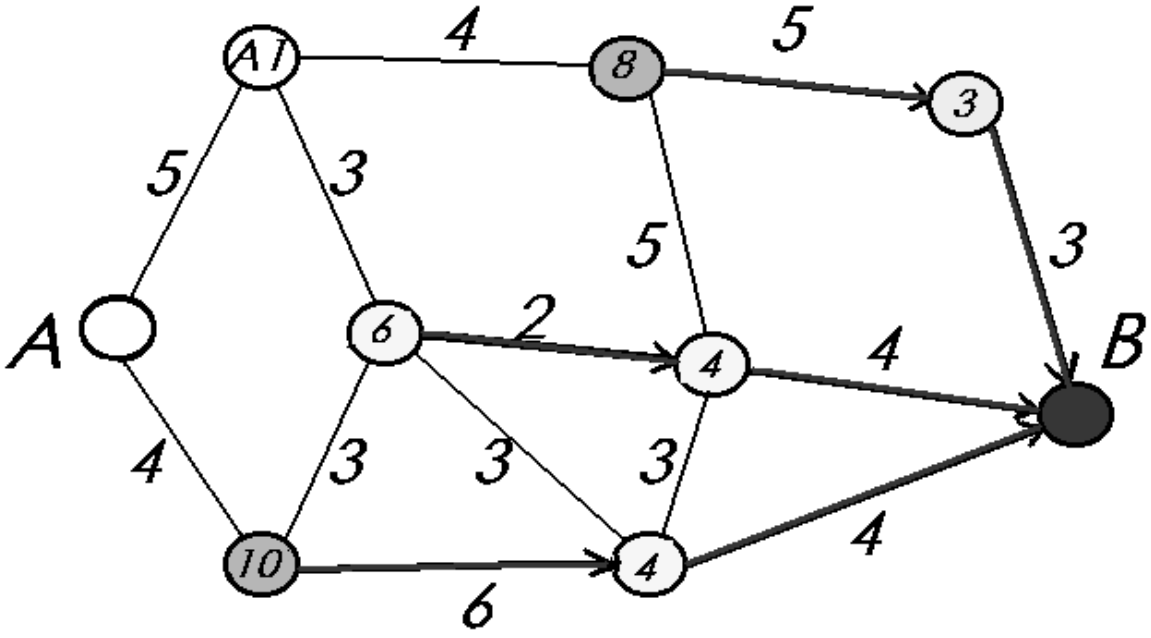


мал. VIII. 1.



мал. VIII. 2.

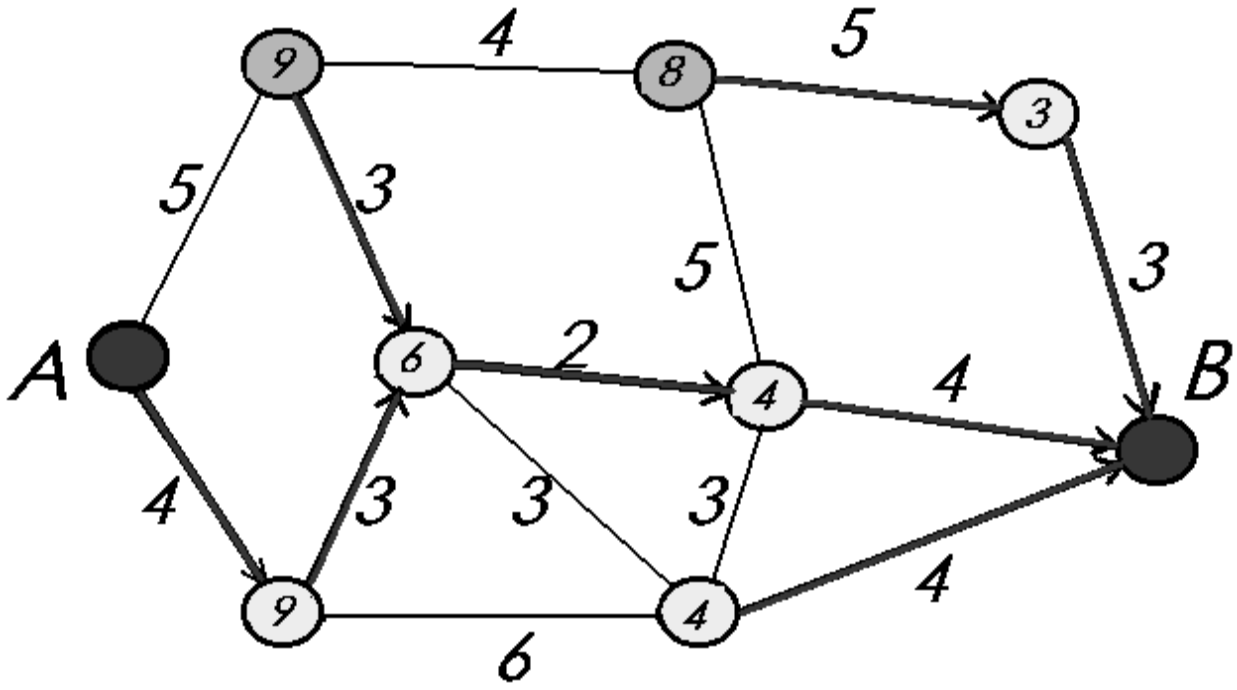
Тепер проаналізуємо, з яких пунктів можна потрапити у пункти A_7, A_5 та A_6. Це можна зробити з пунктів A_4 , A_2 та A_3. Проаналізуємо, через який пункт шлях до пункту В з пункту A_4 коротший. Очевидно, це через пункт A_7(відстань 3+5=8 проти 4+5=9, що через пункт A_5). Тому замість A_4 ставим число 8 і проводим стрілку від пункту A_4 до пункту A_7. Подібне проводим і відносно пунктів A_2 та A_3, після чого схема матиме такий вигляд(малюнок VIII.3):



мал. VIII. 3

Як показує наступний аналіз, в пункт A_4 можна потрапити тільки з пункту A_1, причому відстань буде 12, а в пункт A_2 можна потрапити з пунктів A_1 та

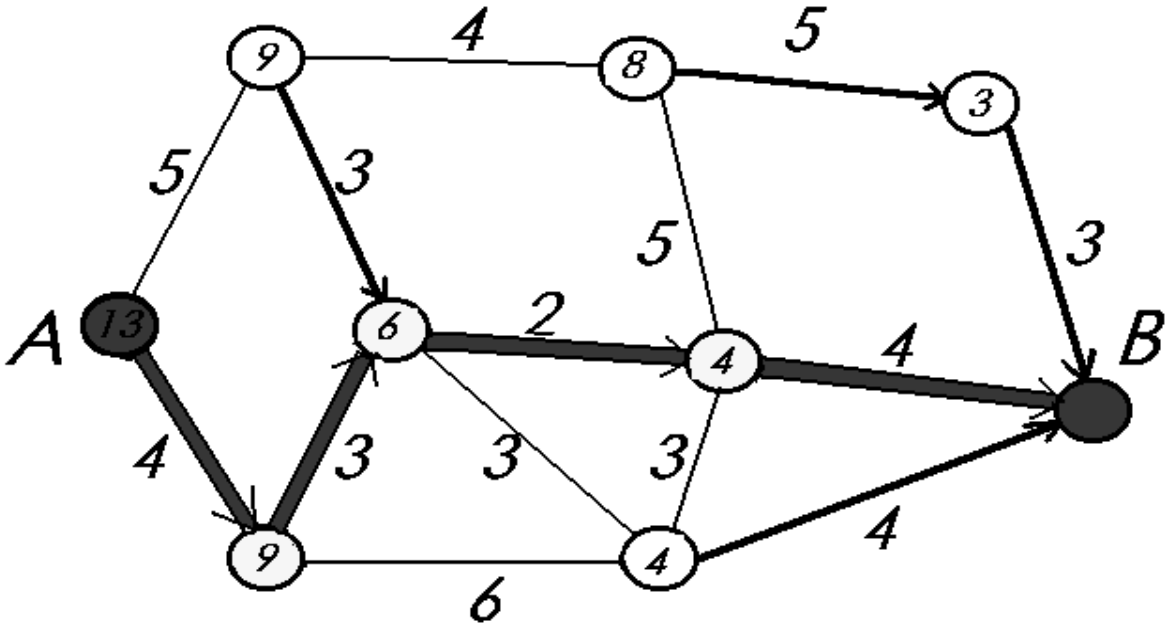
A_3, причому відстані будуть однакові - по 9, тому ми у відповідних кружечках ставимо число 9(зверніть увагу, що в пункті A_3 замінюємо число 10 на число 9), а також проводим стрілки з пунктів A_1 та A_3, відмінивши стрілку з пункту A_3 до пункту A_6. Одержимо схему(малюнок VIII.4):



мал. VIII. 4

Залишається помітити, що з пункту А коротший шлях буде в пункт $_{A3}$, а не A_1 та відобра-зити це на схемі. Ми маршрут А - A_3 - A_2 - A_5 - В наведем подвійною лінією.

Отже відповідь така: **з пункту А в пункт В найкоротший маршрут буде становити 13 днів ходу**, що і відображено на останній схемі (малюнок VIII.5).



мал. VIII. 5

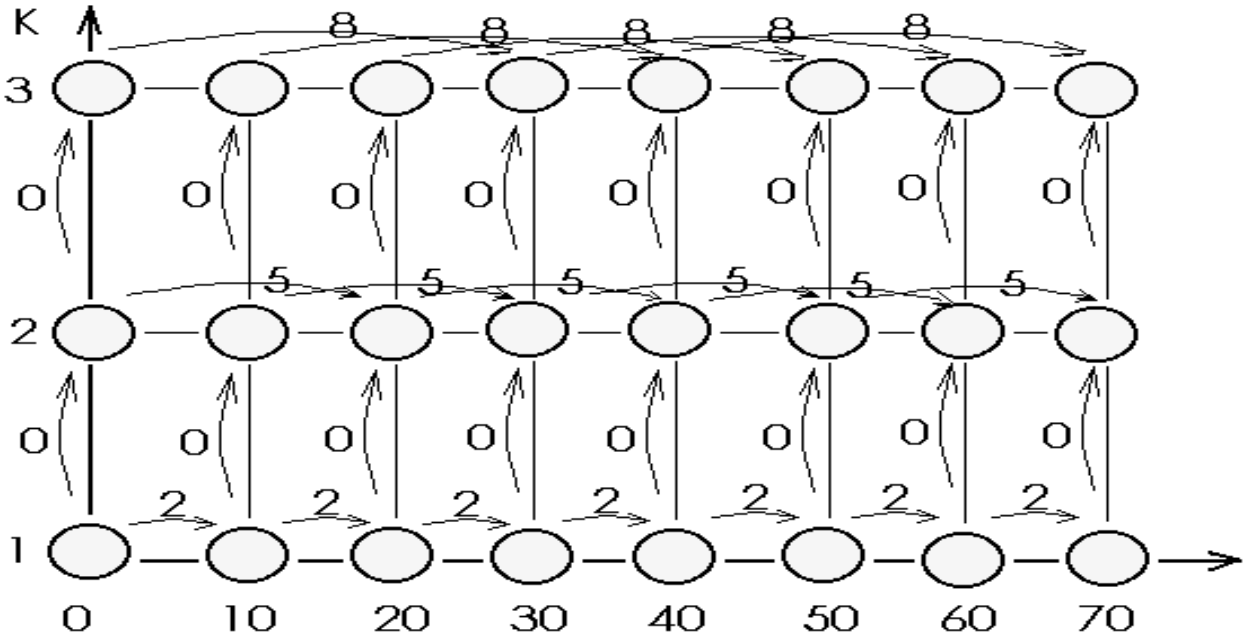
В ході розв’язання ми розглянули не всі можливі маршрути, потім порівнявши їх, а поступово відшукували єдиний і водночас найкоротший з них, на окремитх етапах відкидаючи неоптимальні маршрути разом із їх можливими продовженнями.

Зрозуміло, що коли найкоротший маршрут єдиний, то його можна прослідкувати, прямуючи і з пункту В в пункт А. Пропонуємо зробити це самостійно.

Задача 2. **Людина може підняти рюкзак, який вміщає до 70 кг предметів трьох видів, вага і ціна яких наведені в таблиці 1. Які предмети і в якій кількості слід покласти в рюкзак, щоб загальна ціна цих предметів була найбільшою і рюкзак могла підняти людина?**

таблиця VIII.1

Види Предметів	**Вага одного предмета, (кг)**	**Ціна одного предмета, (грн)**
I	10	2
II	20	5
III	30	8



мал. VIII. 6

Розв’язання. Для наочності проілюструємо умову задачі таким малюнком VIII.6:

Горизонтально по осі Ор будемо відкладати вагу в кілограмах.Тут у нас є запас 70 одиниць. По вертикалі позначимо цифрами 1,2 та 3 різні види предметів *К*. Вибір кожного предмета позначатимемо стрілкою вправо, яка вказує на кружечок з координатою *р*(вага), на стільки одиниць більшою за координату початку стрілки, скільки важить вибраний предмет. Вибір кожного предмета

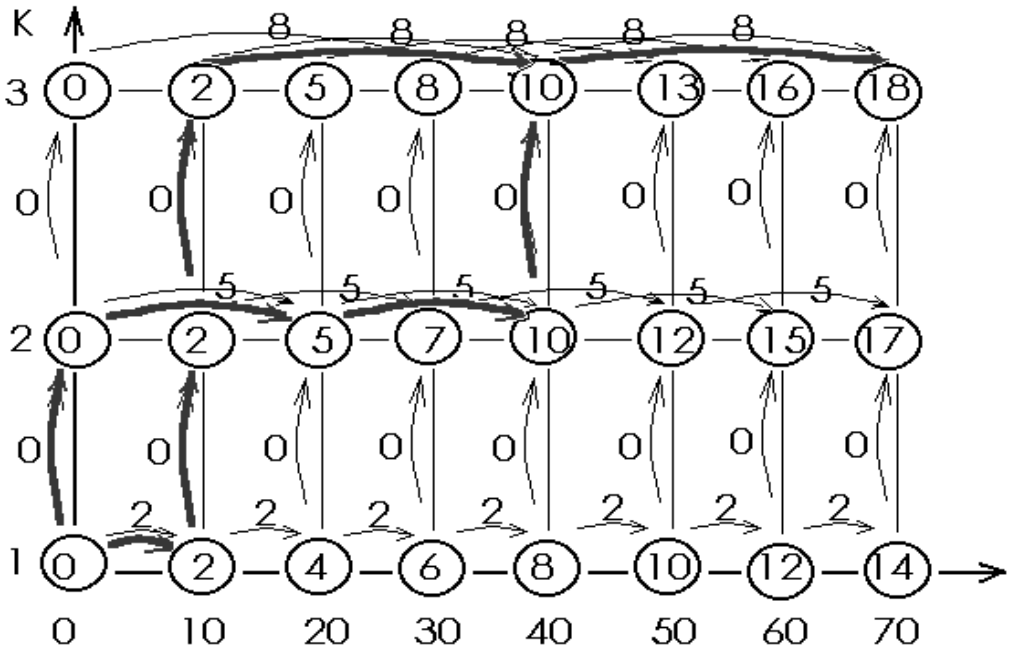
відмічатимемо в тому рядку, який відповідає даному предмету. На кожній стрілці відмічатимемо ціну взятого предмета.

Крім горизонтальних стрілок, які означають вибір відповідного предмета, на малюнку проставлено вертикальні стрілки, які означають перехід до вибору предметів іншого виду. Біля них проставлено нулі, ці стрілки не змінюють вартості взятих предметів. Предмети можна брати в будь-якому порядку, але ми для спрощення будем брати їх в зростаючому порядку номерів.

Ми отримали задачу визначення найдовшого шляху, якщо числа біля стрілок означатимуть відстані, тобто задачу, принципово аналогічну попередній. Тому її теж розв'язуватимем методом динамічного програмування. Нам залишиться тільки проставляти послідовно в кружечках на малюнку числа, що означають довжину найдовшого шляху, який веде в даний кружечок з початкового кружка (0:1).

У початковий кружок запишемо 0. З нього ведуть дві стрілки; одна вправо довжиною 2, тому в кружок (10;1) записуємо 2, і одна вгору довжиною 0, тому в кружок (0;2) записуємо 0. Тепер розглянемо стрілки, що ведуть з останніх заповнених кружків. Так з (10;1) в (20;1) веде стрілка з цифрою 2, а в (10;1) стоїть теж 2. Тому в (20;1) записуємо 4 - це найдовша відстань від початку до кружка (20;1). І так далі. Якщо в кружок ведуть два і більше шляхів, то вибираємо найдовший і його довжину записуємо в цей кружок. Так в (20;2) веде стрілка зліва(довжина шляху 5) і стрілка знизу(довжина шляху 4). Вибираємо 5 = max{5,4} і ліву стрілку відмічаємо жирною лінією(малюнок VIII.7).

Кінець шляху - це кружок (70;3). Якщо рухатись від нього назустріч жирним стрілкам, то виявляється, що існує два рівноцінних способи вибору: два предмети третього виду і один першого або один предмет третього виду і



мал. VIII. 7

два другого. Загальна ціна 18 грн. однакова для обох випадків: $2 \cdot 8 + 0 \cdot 5 + 1 \times \times 2 = 1 \cdot 8 + 2 \cdot 5 + 0 \cdot 2 = 18$.

Вправа. **Розв'язати задачу 2 змінивши обмеження 70 кг на 50 кг.**

Розглядаючи наведені задачі, ми користувались схемами, що в математиці називаються графами. Їх використання унаочнило умови задач та хід їх розв'язування. Поки що ми не ставили питання про представлення даних та результатів у вигляді, придатному для обробки на комп'ютері, а самі алгоритми не записували у вигляді програм. Це вже дещо інше завдання, тому воно буде описане нижче.

VIII. 2 Математична постановка та програми розв'язування задач динамічного програмування.

Цей процес проілюструємо на прикладі наступних задач.

Задача 3. **Дані послідовності $x_1, x_2, ..., x_n$ та $y_1, y_2, ..., y_m$. Знайти їх спільну підпослідовність набільшої довжини.**

Розв'язання. Нехай X[1 : n], Y[1 : m] - масиви, в яких записані дані послідовності. Означимо масив R[1 : n , 1 : m] як таблицю, в кожній клітинці R[i , j] якої записано число членів найдовшої спільної підпослідовності X[i], Y[j].

Зрозуміло, що елементи першого рядка цього масива R[1 , k] дорівнюють 1, якщо для i ∈ [1 , k] виконується : X[1]=Y[i], і R[1 , k] дорівнюють 0 в протилежному випадку. Пер-ший стовпчик заповнюється аналогічно. Розглянемо тепер елемент R[i , j](для менших значень індексів таблиця нехай вже заповнена). **Якщо X[i] ≠ Y[j]**, то додавання до послідовнос-тей X[1], ..., X[i - 1] та Y[1],..., Y[j - 1] нових елементів не впливає на довжину їх найбіль-шої спільної підпослідовності, тобто **R[i , j] = max (R[i - 1 , j] , R[i , j - 1])**. Якщо ж X[i] = Y[j], то довжина шуканої підпослідовності може зрости на 1 у випадку, коли
R[i - 1 , j - 1] = R[i - 1 , j] = R[i , j - 1]. (Простіше кажучи, якщо крайні елементи не входили в підпослідовність.) Таким чином, **якщо X[i] = Y[j],**

R[i , j] = max (R[i - 1 , j] , R[i , j - 1], R[i - 1 , j - 1] + 1).

Ми одержали наступний алгоритм заповнення масиву R:

```
алг Мах_Довжина_Підпослідовності(арг цел n,m,целтаб X[1:n],Y[1:m],
                                 рез целтаб R[1:n,1:m])
нач цел i, j
  нц для i от 1 до n
     нц для j от 1 до m  R[i,j] : = 0  кц
  кц
  нц для i от 1 до m
        нц для j от 1 до i
              если X[1] = Y[j] то R [ 1 , i ] : = 1  все
        кц
  кц
  нц для i от 1 до n
        нц для j от 1 до i  если Y[1]=X[j]  то R[i,1]:=1 все  кц
  кц
  нц для i от 2 до n
        нц для j от 2 до m
```

```
            если X[i]=Y[j]
                то R[i,j]:=max(max(R[i-1, j ], R[i, j-1]), R[i-1,j-1]+1)
                иначе R[i,j]:=max(R[i-1,j],R[i,j-1])
            все
        кц
    кц
кон
```

Описаний алгоритм знаходить не саму найдовшу підпослідовність даних послідов-ностей, як вимагається в задачі, а лише її довжину. Проте це не змінило ідеї шуканого алгоритма, а лише трохи його скоротило. Всі, хто зрозуміє запропонований алгоритм, легко зможуть самі його удосконалити. Ми ж наведемо контрольний приклад.

Дані масиви (таблиці VIII.2 –VIII. 4):

X[1 : n] , при n = 7:

Y[1 : m], при m = 9:

таблиця VIII.2

1	2	3	4	5	6	7
3	7	9	5	12	14	8

таблиця VIII.3

1	2	3	4	5	6	7	8	9
4	3	2	1	25	26	8	7	3

Одержаний масив R[1 : n , 1 : m] :

таблиця VIII. 4

	1	2	3	4	5	6	7	8	9
1	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
2	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**2**	**2**
3	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**2**	**2**
4	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**2**	**2**
5	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**2**	**2**
6	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**2**	**2**
7	**0**	**1**	**1**	**1**	**1**	**1**	**2**	**2**	**2**

Згідно методу побудови масива R[1 : n , 1 : m] після виконання алгоритма елемент

R[n , m], тобто R[7 , 9], є результатом. Як видно, справді, найдовша спільна підпослідовність даних послдовностей містить два елементи, це числа 3 та 7.

Задача 4 . **Завод одержав заказ на виготовлення n автоматичних ліній протягом m місяців. Вартість виготовлення j (0≤ j ≤ n) ліній в і - тому місяці (враховуючи накладні витрати, зарплату і т.д.) задається масивом C[1:m, 1:n]. Якщо таку кількість ліній виготовити**

не можна, запишемо в таблицю 5 деяке значення, нехай -1. Як спланувати виробництво, щоб сумарна вартість n ліній за m місяців була міні мальною?

таблиця VIII.5

рядки - кількість ліній, стовпчики - місяці	**0**	**1**	**2**	**3**	**4**
1	10	**12**	15	20	-1
2	10	14	**16**	19	23
3	10	**10**	17	19	-1

Розв'язання. Нехай S [i , j] - мінімальна вартість виготовлення j ліній на кінець i - го місяця. Зрозуміло, що S [1 , j] = C [1 , j], ($0 \le j \le n$). Очевидно також: щоб виготовити j ліній до кінця i - го місяця потрібно виготовити j_1 ліній до кінця (i - 1) -го і j_2 ліній в i - му місяці, причому, $j_1 + j_2 = j$. Нам потрібно знайти мінімальну вартість, тобто

$$\mathbf{S [i , j] = \min\{ S [i - 1 , j_1] + C [i , j_2] \} \text{ при } j_1 + j_2 = j} .$$

При розшифровці попереднього рядка слід звернути увагу на зміст фігурних дужок. Видно, що S [m , n] - шукана мінімальна вартість виконання заказу. Оптимальна стратегія може бути одержана оберненим переглядом масиву C [i , j].

Розглянемо контрольний приклад. Нехай нам потрібно виготовити 4 лінії за 3 місяці, як відображено у таблиці VIII.6.

Масив S [3 , 4]:

таблиця VIII.6

	0	**1**	**2**	**3**	**4**
1	10	12	15	20	-1
2	20	22	25	28	31
3	-	-	-	-	38

Зрозуміло, що нас не цікавлять інші значення S [i , j] при i = 3, адже ми обов'язково повинні виготовити 4 лінії. Тому в таблиці №3 у відповідних місцях поставлені прочерки.

Роз'яснимо ще, як знаходиться S [2 , 1]. Як можна виготовити одну лінію на кінець другого місяця? Можна виготовити її в першому місяці (це коштуватиме 12 умовних одиниць) і нічого не робити в другому місяці(це коштуватиме 10 у.о.). А можна в першому місяці нічого не робити (10 у.о.), а одну лінію виготовити в другому місяці (це коштуватиме14 у.о.). Щоб мінімізувати вартість, потрібно із цих варіантів вибрати кращий (в даному випадку це - перший).

Описане буде виглядати так:

S [1 , 0] = 10; S [1 , 1] = 12;

S [1 , 2] = 15; S [1 , 3] = 20;

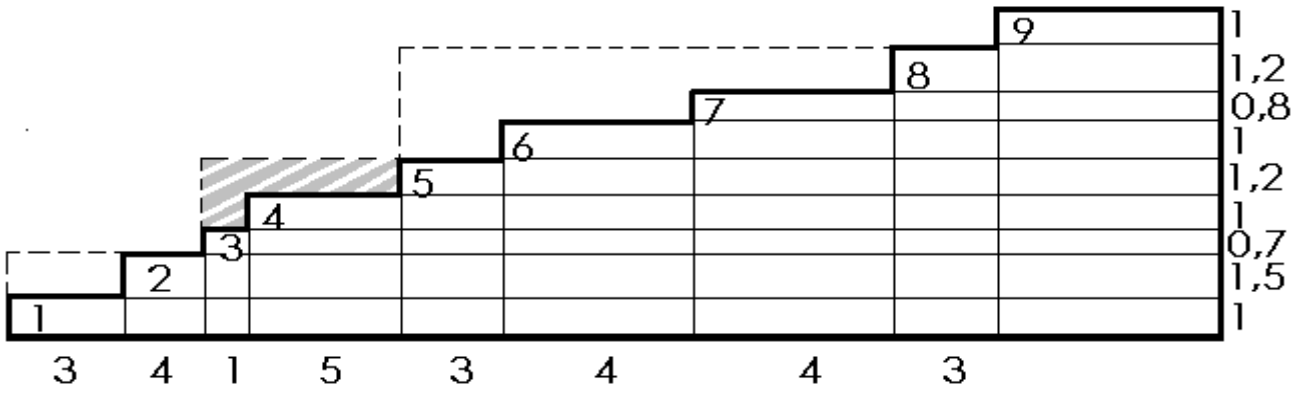
S [1 , 4] - не визначено; S [2 , 0] = 10 + 10 = 20;
S [2 , 1] = min(12 + 10, 10 + 14) = 22;
S [2 , 2] = min(15 + 10, 12 + 14, 10 + 16) = 25;
S [2 , 3] = min(20 + 10, 15 + 14, 12 + 16, 10 + 19) = 28;
S [2 , 4] = min(20 + 10, 15 + 14, 12 + 16, 10 + 19) = 31;
S [3 , 0] = не потрібно визначати;
S [3 , 1] = не потрібно визначати;
S [3 , 2] = не потрібно визначати;
S [3 , 3] = не потрібно визначати;
S [3 , 4] = min(20 + 10, 15 + 14, 12 + 16, 10 + 19) = 38;

Отже відповідь така: мінімальна вартість чотирьох ліній, виготовлених за три місяці буде становити 38 умовних одиниць, причому за перший місяць треба виготовити 1 лінію, за другий місяць треба виготовити 2 лінії, а в третьму місяці слід виготовити також одну лінію. Обернений перегляд масиву C[1 : m , 1 : n] проілюстровано виділеними жирним підкресленим шрифтом його відповідними елементами таблиці 5.

Задача 5. **На малюнку VIII.8 зображено східці в розрізі, що містять дев'ять сходинок однакової ширини, рівної 10 дм, а також вказано розміри кожної з них також в дециметрах. При реконструкції цих східців вирішили наростити окремі сходинки так, щоб їх стало лише чотири, а будівельного матеріалу було використано найменше. Які сходинки слід наростити і на скільки? Вказати також мінімальну кількість будівельного матеріалу, потрібного для реконструкції.**

Розв'язання. Не зважаючи на прості варіанти реконструкції, один з яких (звісно,не найкращий) показано на малюнку 8 пунктиром (як легко підрахувати, в цьому випадку для реконструкції необхідно 345 дм^3 будматеріалу), самих варіантів очевидно так багато, що облишимо ідею їх повного перебору з метою відшукання найоптимальнішого. Набутий вже нами досвід підказує, що задачу слід розв'язувати методом динамічного програмування. Зупинимось на математичній постановці задачі.

малюнок VIII.8



Позначимо через g(i,j) площу добудови пунктиром на малюнку однієї із сходинок, наприклад, площа добудови сходинки, яка починається на третій і

закінчується на п’ятій сходинках (на малюнку 8 заштриховано) g(3,5) = 8,2 дм^2 . Результати обчислень значень g(i,j) занесем у таблицю 7 (курсивом виділені значення, що не підходять за умовою, бо інакше після нарощення сходинок буде менше, ніж чотири):

таблиця VIII.7

i / j	1	2	3	4	5	6	7	8
9	*120,8*	*95,6*	*68*	**61,8**	**35,8**	**23,8**	**11,8**	**3**
8	*93,8*	*71,6*	**48**	**42,8**	**21,8**	**12,8**	**4,8**	-
7	*65*	**46,4**	**27,6**	**23,6**	**8,6**	**3,2**	-	-
6	**49**	**32,8**	**17,2**	**14**	**3**	-	-	-
5	**33**	**19,8**	**8,2**	**6**	-	-	-	-
4	**17,4**	**7,8**	**1**	-	-	-	-	-
3	**9,4**	**2,8**	-	-	-	-	-	-
2	**4,5**	-	-	-	-	-	-	-

Тепер позначимо через f(i,j) мінімальні площі, одержані в результаті реконструкцій частин східців, де i - кількість одержаних сходинок, j - номер сходинки, де закінчилась на даному етапі реконструкція. Очевидно, що f(1,j), де 1< j ≤ 9 співпадатимуть із відповідними значеннями g(1,j), а f(4,9) буде розв’язком задачі. Наведем значення f(i,j) і пояснимо, як вони одержувались.

f(1,1) = 0, адже нарощувати тут нічого не треба (всі **f(i, i) = 0)**;

f(1,2) = **f(1,1) + g(1,2)** = 0+1,5*3 = 4,5;

f(1,3) = **f(1,2) + g(1,3)** = 1,5*3+0,7*(3+4) = 9,4;

f(1,4) = **f(1,3) + g(1,4)** = 9,4+1*(3+4+1) = 17,4 ;

f(1,5) = **f(1,4) + g(1,5)** = 17,4+1,2*(3+4+1+5) = 33;

f(1,6) = **f(1,5) + g(1,6)** = 33+1*(3+4+1+5+3) = 49; (наступні f(1,j) - не потрібні, бо тоді остаточно сходинок буде менше 4). Але для повноти таблиці приведемо їх. Отже,

f(1,7) = **f(1,6) + g(1,7)** = 49+0,8*(3+4+1+5+3+4) = 65;

f(1,8) = **f(1,7) + g(1,8)** = 65+1,2*(3+4+1+5+3+4+4) = 93,8;

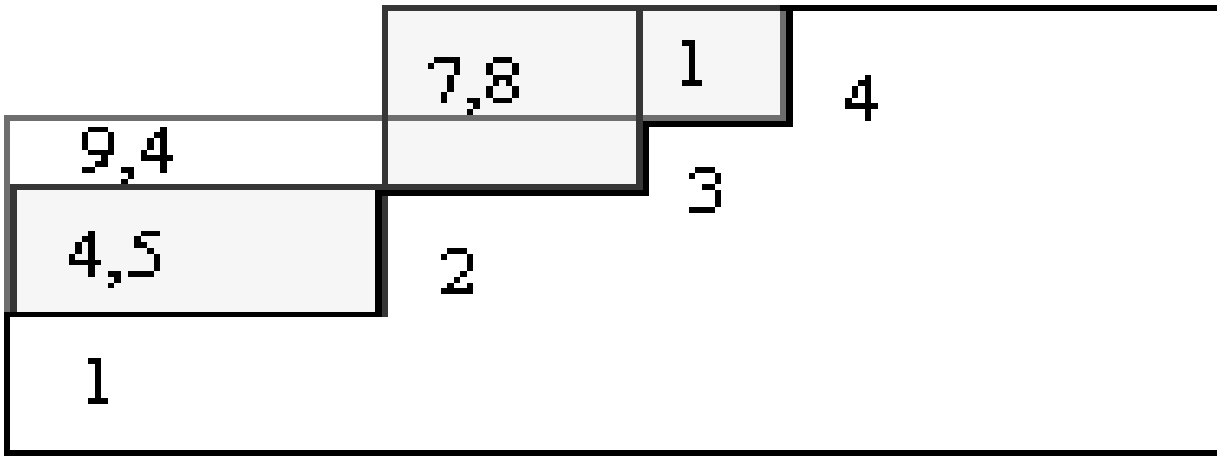
f(1,9) = **f(1,8) + g(1,9)** = 93,8+1*(3+4+1+5+3+4+3) = 120,8;

(f(i,j), виділені сірим кольором не потрібні, вони суперечать умові, далі обчислювати їх не будемо)

f(2,3) = **min(g(2,3),f(1,2) =** min(**2,8**;4,5) = 2,8 ;

Пояснимо детальніше, як шукати f(2,3) і наступні значення f(i,j). На відміну від f(1,j), тут потрібно вибрати оптимальний із двох варіантів: щоб замінити перші три сходинки двома, ми повинні наростити або другу сходинку, або першу, і f(2,3) дорівнює найменшій із відповідних площ.

Розглянувши малюнок 8, бачимо, що для обчислення f(2,4) потрібно знайти менше з трьох чисел 7,8;4,5+1;9,4(малюнок 9),



мал. VIII.9

тобто:

f(2,4) = **min{g(2,4),f(1,2)+g(3,4),f(1,3)}** =

= min{7,8**;4,5+1**;9,4 } = 5,5;

f(2,5) = **min{g(2,5),f(1,2)+g(3,5),f(1,3)+g(4,5),f(1,4)}** =

= min{19,8;**4,5+8,2**;9,4+6;17,4 } = 12,7;

f(2,6) = **min{g(2,6),f(1,2)+g(3,6),f(1,3)+g(4,6),f(1,4)+g(5,6),f(1,5)} =**

= min{32,8;4,5+17,2;9,4+14;**17,4+3**;33 } = 20,4;

f(2,7)=**min{g(2,7),f(1,2)+g(3,7),f(1,3)+g(4,7),f(1,4)+g(5,7),f(1,5)+g(6,7), f(1,6)} =** min{46,4;4,5+27,6;9,4+23,6;**17,4+8,6**;33+3,2;49} = 20,4;

f(2,8) та f(2,9) не обчислюємо.

Далі обмежимось словесним поясненням і формульними записами для f(3,4) та f(3,5). Подивимось, де може кінчатися друга сходинка східців. Звичайно ж, на 2-й або на 3-й старій. Але ми уже знаємо, які двохсходинкові східці закінчуються на цьому рівні: f(2,2) = 0, а площу найкращих двохсходинкових східців f(2,3) = 2,8 ми уже знайшли, так що

f(3,4) = **min{g(3,4),f(2,3)}** = min{**1**;2,8} = 1;

Подібними міркуваннями визначаємоf(3,5): із трьох сходинок, що кінчаються на 5-й сходинці, друга може закінчиться на 2-й, 3-й або 4-й. Отже:

f(3,5) = **min{g(3,5),f(2,3)+g(4,5),f(2,4)}** = min{8,2;2,8+6;**5,5**} = 5,5;

Далі одержуємо:

f(3,6) = **min{g(3,6),f(2,3)+g(4,6), f(2,4)+g(5,6),f(2,5)}** =

= min{17,2;2,8+14;**5,5+3**;12,7} = 8,5;

f(3,7)=**min{g(3,7),f(2,3)+g(4,7),f(2,4)+g(5,7),f(2,5)+g(6,7),f(2,6)}**=

= min{27,6;2,8+23,6;**5,5+8,6**;12,7+3,2;20,4} = 14,1;

f(3,8)=**min{g(3,8),f(2,3)+g(4,8),f(2,4)+g(5,8),f(2,5)+g(6,8),f(2,6+g(7,8), f(2,7)} =** min{48;2,8+42,8;5,5+21,8;12,7+12,8;**20,4+4,8**;26} = 25,2;

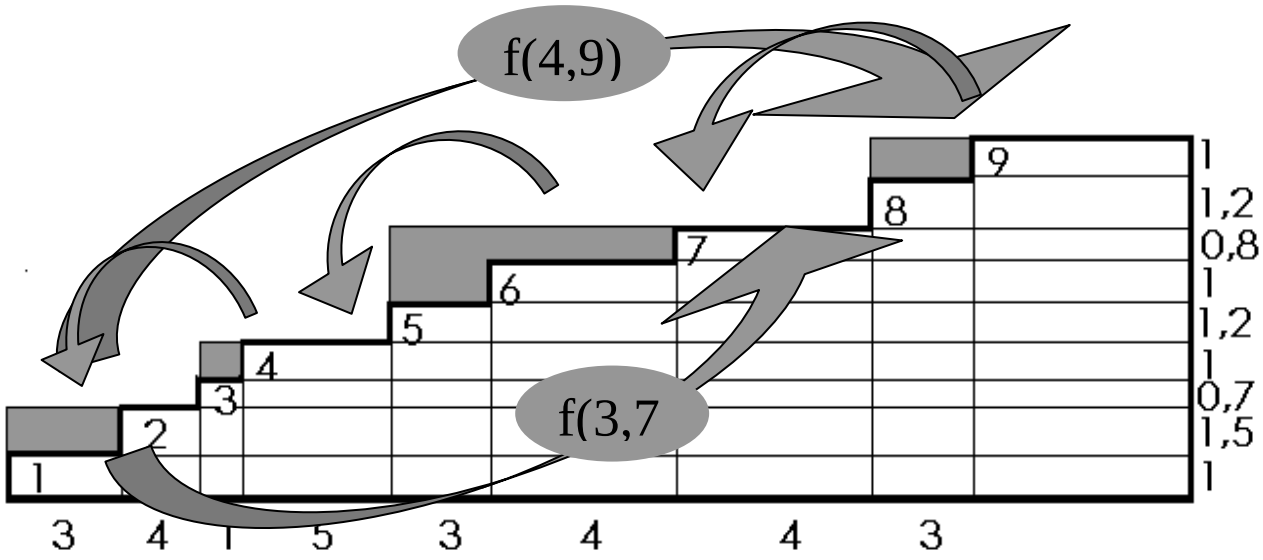
f(3,9) не обчислюємо. Пропустивши f(4,j), 4≤j≤8, знаходитимемо f(4,9) як найменшу із сум f(3,j) + g(j,9), де j - одне із шести чисел 4, ..., 8, тобто:

f(4,9)=**min{g(4,9),f(3,4)+g(5,9),f(3,5)+g(6,9),f(3,6)+g(7,9),f(3,7)+g(8,9),**

f(3,8)} = min{61,8;1+35,8;5,5+23,8;8,5+11,8;**14,1+3**;25,2} = 17,1.

Якщо число 17,1 помножити на 10, то одержимо найменшу кількість будівельного матеріалу в дм^3.

Постає питання, де повинні розміщуватися нові сходинки. Помітимо, що оптимальні варіанти всюди підкреслені і виділені жирним шрифтом. В останній оптимальний варіант ввійшла кількість **f(3,7) = 14,1**. Отже третя сходинка повинна закінчуватись на 7-й старій. Тепер повернемось крок назад до визначення f(3,7) і побачимо, що в f(3,7) входить площа перших двох сходинок f(2,4) = 5,5. Отже, друга сходинка повинна закінчуватись на 4-й старій. І, нарешті, у вираз



мал. VIII.10

f(2,4) входить f(1,2). Це означає, що перша нова сходинка закінчується на 2-й старій, тобто остаточну картину реконструкції можна побачити на малюнку VIII.10.

Щоб закінчити математичну постановку цієї задачі, зробимо узагальнені міркування. Нехай потрібно наростити східці, що мають N сходинок, так, щоб вони їх мали n, де n<<N, тобто n набагато менше N. Нехай нам уже відомо оптимальний розподіл сходинок, коли їх в нових східцях менше n. Нехай h_k – номер старої сходинки, на якій закінчується k-та нова (малюнок 11). Очевидно, починається ця k-та сходинка на (h_{k-1}+1) –й старій. Додаткову площу, виділену на малюнку 11, позначимо через g((h_{k-1}+1, h_k). Загальна задача полягає в тому, щоб знайти значення h_1, h_2, …, h_k, при яких сума k величин

$$\mathbf{g((h_0+1, h_1) + g((h_1+1, h_2) + \dots + g((h_{k-2}+1, h_{k-1}) + g((h_{k-1}+1, h_k)}$$

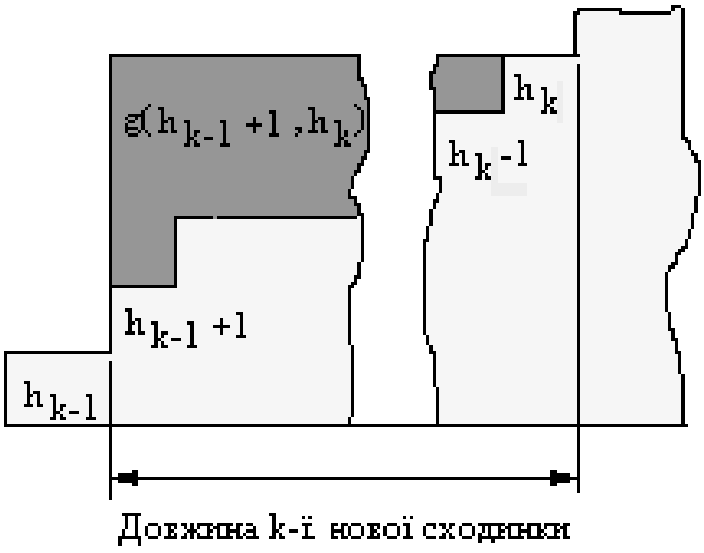
(при даному k=n і $h_k=h_n$) найменша, позначимо це найменше значення через $f_n(h_n)$, тобто:

$$\mathbf{f_n(h_n) = min\{g((h_0+1,h_1) + g((h_1+1,h_2) +\dots+g((h_{k-2}+1, h_{k-1}) + g((h_{k-1}+1, h_k)\}} \text{ (VIII.1)}$$

Зауважимо, що сума перших n-1 доданків, в які h_n не входить, приймає найменше значення $f_{n-1}(h_{n-1})$ – це та ж задача для на 1 меншої кількості сходинок. В результаті одержуємо формулу Р. Беллмана:

$$\mathbf{f_k(h_k) = min \{ f_{k-1}(h_{k-1}) + g((h_{k-1}+1, h_k) \}}, \qquad \text{(VIII.2)}$$

яка дозволяє послідовно знайти числа $f_k(h_k)$ (при всіх $k \le n$, $h_k \le h_n$) на кожному кроці (на малюнку 3 (VIII.1) та (VIII.2) виділено сірим кольором). Її потрібно розуміти так: для знаходження мінімуму слід надавати k всі можливі значення, починаючи з 1, і для кожного знайти і запам'ятати значення h_{k-1}, яке становить найменшу величину $f_k(h_k)$. А потім, дійшовши до останнього значення k=n, потрібно, "задкуючи", повернутися назад і знайти всі оптимальні значення h_{n-1}, h_{n-2}, …,h_1 (див стрілки на малюнку VIII.10).



мал. VIII.11

Звернемо увагу на те, що в описаному методі не накладається ніяких умов на вид функцій g (цим динамічне програмування вигідно відрізняється від лінійного, в якому вимагається, щоб всі розглядувані залежності були лінійними і змінні змінювались неперервно). Суттєво лише, що функцію f, мінімум якої ми знаходимо, вдається представити у вигляді суми членів, в якій попередній залежить від меншої кількості змінних.

Узагальнимо уявлення про метод динамічного програмування. Динамічне програмування – це метод оптимізації, пристосований до операцій, в яких процес прийняття рішень може бути розбитий на окремі етапи (кроки). Такі операції називаються багатокроковими.

В основі методу динамічного програмування лежить, як вже згадувалось вище, *принцип оптимальності*, сформульований Р. Беллманом. Цей принцип та ідея включення конкретної задачі оптимізації в сім'ю аналогічних багатокрокових задач приводять до рекурентних співвідношень – функціональних рівнянь – відносно оптимального значення *цільової функції*. Їх розв'язання дозволяє послідовно одержати оптимальне управління для вихідної задачі оптимізації.

Дамо загальний опис моделі динамічного програмування. Розглядається керована система, яка під впливом управління переходить із початкового стану $\overline{\xi_0}$ в кінцевий стан $\overline{\xi_n}$.

Нехай процес управління системою можна розбити на *n* кроків. Нехай - $\overline{\xi_1}, \overline{\xi_{22}}, ..., \overline{\xi_{1n}}$, стани системи після першого, другого, … , *n*-го кроку.

$$\underline{\xi_0} \xrightarrow{\overline{u_1}} \underline{\xi_1} \xrightarrow{\overline{u_2}} \dots \xrightarrow{\overline{u_{k-1}}} \underline{\xi_{k-1}} \xrightarrow{\overline{u_{k+1}}} \dots \xrightarrow{\overline{u_n}} \underline{\xi_n}$$

Стан $\overline{\xi_k}$ системи після* k-*го кроку* (k=1,2,3,…,n) *характеризується параметрами

$$\xi_k^{(1)}, \xi_k^{(2)}, \dots, \xi_k^{(s)},$$

які називаються *фазовими координатами*. Стан $\overline{\xi_k}$ можна зобразити точкою s-вимірного простору, називається фазовим простором. Послідовне перетворення системи (по крокам) досягається з допомогою деяких заходів $\overline{u_1}, \overline{u_{12}}, ..., \overline{u_{1n}}$, які складають управління системою $U = (\overline{u_1}, \overline{u_{12}}, ..., \overline{u_{1n}})$, де $\overline{u_k}$ - управління на *k*-му кроці, яке переводить систему із стану $\overline{\xi_{л-1}}$
в стан $\overline{\xi_k}$. Управління $\overline{u_k}$ на *k*-му кроці полягає у виборі значень визначених управляючих змінних $\overline{u_k^{(1)}}, \overline{u_k^{(2)}}, ..., \overline{u_k^{(n)}}$.

Будемо надалі припускати, що стан системи в кінці *k*-го кроку залежить тільки від попереднього стану системи $\overline{\xi_{k-1}}$ та управління $\overline{u_k}$ на даному кроці. Така властивість одержала назву *відсутності наслідку*. Прзначимо цю залежність у вигляді $\overline{\xi_k} = F_k(\overline{\xi_{k-1}}, \overline{uk})$. Ця рівність має назву *рівняння станів*. Функції $F_k(\overline{\xi_{k-1}}, \overline{u_k})$ будемо вважати заданими.

Варіюючи управління *U,* одержимо різну ”ефективність” процесу, яку будемо оцінювати кількісно цільовою функцією *Z,* яка залежить выд початкового стану системи $\overline{\xi_0}$ та від вибраного управління *U*.

$$Z = \Phi(\overline{\xi_0} б U).$$

Показник ефективності *k*-го кроку процесу управління, який залежить від стану $\overline{\xi_{k-1}}$ на початку цього кроку і управління $\overline{u_k}$, вибраного на цьому кроці, позначимо через $f_k(\overline{\xi_{k-1}}, \overline{u_k})$. В розглядуваніфй задачі покрокової оптимізації цільова функція повинна бути адитивна, тобто $Z = \sum_{k=1}^{n} f_k(\overline{\xi_{k-1}}, \overline{u_k})$.

Звичайно умовами процесу на управління на кожному кроці $\overline{u_k}$ накладаються деякі обмеження. Управління, що задовольняють цим обмеженням, називаються *допустимими*.

Задачу покрокової оптимізації можна сформулювати так: визначити сукупність допустимих управлінь $\overline{u_1}, \overline{u_{12}}, ..., \overline{u_n}$, що переводять систему із початкового стану в кінцевий і максимізуючий або мінімізуючий показник ефективності.

Управління, при якому досягається максимум (мінімум) цільової функції називається *оптимальним управлінням*.

Якщо змінні керування приймають дискретні значення, то модель динамічного програмування називається *дискретною.* Якщо ж зазначені змінні змінюються безперервно, те модель ДП називається *неперервною.* У залежності від кількості параметрів станів (s) і кількості керуючих змінних на кожному кроці (г) розрізняють одновимірні і багатовимірні моделі динамічного програмування. Число кроків у задачі може бути або скінченним, або нескінченним.

У деяких задачах, що розв'язуються методом динамічного програмування, процес керування природно розбивається на кроки. Наприклад, при розподілі на кілька років ресурсів діяльності підприємства кроком природно вважати часовий період; при розподілі засобів між *n* підприємствами номером кроку природно вважати номер чергового підприємства. У інших задачах розбивка на кроки вводиться штучно. Наприклад, неперервний керований процес можна розглядати як дискретний, умовно розбивши його на деякі часові відрізки - кроки. Виходячи з умов кожної конкретної задачі, довжину кроку вибирають таким чином, щоб на кожному кроці одержати просту задачу оптимізації і забезпечити необхідну точність обчислень.

VIII. 3 Як пояснювати учням метод динамічного програмування.

Слід визнати, що метод динамічного програмування важкодоступний для учнів. Вони ще не мають достатнього рівня математичної свідомості для сприйняття таких понять, як методи оптимізації, цільова функція, оптимальна стратегія і т.д. Тому добір задач для учнів може бути дуже обмеженим. Слід обмежитись дискретними моделями. Дуже зручно ознайомлювати учнів з методом динамічного програмування під час розбору комбінаторних задач після розгляду методів повного та скороченого переборів. Необхідно попередньо дуже добре опрацювати рекурсивні алгоритми. Важливо дати поняття про ефективність алгоритмів, адже розгляд методу динамічного програмування вимагає зрозумілої ідостатньої мотивації.

Для ознайомлення з методом важливо знайти підходящу задачу, наприклад таку:

Дано цілочисельну прямокутну таблицю А з *n* рядків та *m* стовпчиків. Необхідно перебрати всі ступінчаті шляхи з лівого верхнього кута в правий нижній і знайти *найменшу суму елементів*, через які проходить такий шлях та сам цей шлях. Шлях, що визначає найменшу суму елементів, називається *найкоротшим маршрутом*. Ступінчатим називатимемо такий шлях, коли з будь-якої клітинки дозволяється рух в сусідню клітку, розміщену нижче або правіше від неї.

Розв'язання.

ПІДГОТОВЧА РОБОТА: Введемо допоміжну таблицю L [1:N, 1:M] таку, що L[i, j] - *довжина найкоротшого шляху з клітки* [i, j] *у клітку* [N, M]. Тоді, мабуть, L[N, M] = A[N, M]. Тому що рух може бути лише вправо і вниз, то в останньому стовпчику можливе переміщення лише вниз. Отже

$$\mathbf{L[i, M] = A [i, M] + L [i +1, M], \quad i = N - 1, N - 2, ..., 1.} \qquad \text{(VIII.3)}$$

Аналогічно визначаються значення елементів останнього рядка таблиці L:

$$L[N, j] = A[N, j] + L[N, j+1], \quad j = M - 1, M - 2, ..., 1. \qquad \text{(VIII.4)}$$

Можна спочатку визначати перший стовпчик та перший рядок (пропонуємо написати відповідні співвідношення, подібні до (VIII.3) та (VIII.4) самостійно).

ЗАПОВНЕННЯ ТАБЛИЦІ: Для наступного опису зручно взяти конкретний приклад. Нижче наведена таблиця A і заповнена, як описано вище, частина таблиці L (верхній рядок та правий стопчик):

таблиця A (VIII.8)

	1	2	3	4	5	6
1	7	3	9	15	18	4
2	14	5	16	7	11	5
3	6	8	4	2	7	3

таблиця L (VIII.9)

	1	2	3	4	5	6
1	7	10	19	34	52	56
2	21					
3	27					

Виконана підготовча робота. Тепер, починаючи з лівого верхнього кута (самостійно попробуйте з правого нижнього кута), заповнимо решту таблиці L. Розглянемо незаповнені клітинки другого рядка. Щоб знайти елемент таблиці L[2, 2], треба до A[2, 2] додати менше з чисел L[1, 2] та L[2,1], тобто число 10. Тому L[2, 2] = 15. Подібним чином заповнимо і решту елементів другого рядка. Отримаємо:

таблиця L (VIII.10)

	1	2	3	4	5	6
1	7	10	19	34	52	56
2	21	**15**	**31**	**38**	**49**	**54**
3	27					

Після його заповнення легко подібним чином заповнити третій рядок таблиці L. В результаті одержимо:

таблиця L (VIII.11)

	1	2	3	4	5	6
1	7	10	19	34	52	56
2	21	15	31	38	49	54
3	**27**	~~23~~	**27**	29	36	**39**

АНАЛІЗ: Три числа в останній таблиці L (VIII.11) виділено червоним кольором. Число 39 дорівнює найменшій сумі елементів, тобто відповідає найкоротшому маршруту. Числа L[3, 1] = 27 і L[3, 3] = 27 виділено для того, щоб вказати на ефективність методу динамічного програмування, адже пройшовши червоний маршрут, наберемо не 27, а 39, що визначено найкоротшим маршрутом, а дійшовши до L[3, 6], одержимо 51.

Найголовнішим у розв'язуванні задачі методом динамічного програмування є відшукання *рекурентної формули*, що виражає *цільову функцію*. В нашому випадку вона має такий вигляд:

L[i, j] = A [i, j] + min{L[i+1, j], L[i, j+1],
i = N - 1, N - 2, ..., 1, j = M - 1, M - 2, ..., 1. (VIII.5)

Формула (VIII.5) в алгоритмі представляється фрагментом:

```
пц для j від  M-1 до 1 крок –1
   пц для i від  N-1 до 1 крок –1
      якщо  L[i, j+1]< L[i+1, j]
        то        L[i, j] := A [i, j] + L[i, j+1]
        інакше  L[i, j] := A [i, j] + L[i+1, j]
      все
   кц
кц
```

Для пошуку власне найкоротшого маршруту слід пройти зворотний шлях від L[3, 6] до L[1, 1], роблячи крок вліво чи вгору, вибираючи менше число (виділено синім кольором).

таблиця L (VIII.12 - остаточна)

	1	*2*	*3*	*4*	*5*	*6*
1	**7**	**10**	19	34	52	56
2	21	**15**	31	38	49	54
3	27	**23**	**27**	**29**	**36**	**39**

Слід підкреслити, що метод динамічного програмування дозволяє розв'язати цю задачу за (N-1)×(M-1) кроків, для таблиці 3×6 - всього за 10 кроків. Метод повного перебору вимагає $C_{N+M-2}^{N-1} = \frac{(N+M-2)!}{(N-1)!\times(M-1)!}$ кроків, для таблиці 3×6 - $C_{3+6-2}^{3-1} = \frac{(3+6-2)!}{(3-1)!\times(6-1)!} = \frac{7!}{2!\times5!} = \frac{6\times7}{2} = \frac{42}{2} = 21$ крок, що значно більше.

На закінчення наведемо дві задачі на метод динамічного програмування, що пропонувались на III (обласному) етапі Всеукраїнської олімпіади з інформатики у Київській області.

1. **(***9 - 10 класи***) FIGURA**(*30 балів*) У квадратному масиві цілих чисел A[$n\times n$] (2<n<20)елементи, що лежать на головній діагоналі і нижче від неї (на малюнку заштриховані), утворюють ступінчасту "фігуру". Суму цих чисел назвемо "площею" фігури.

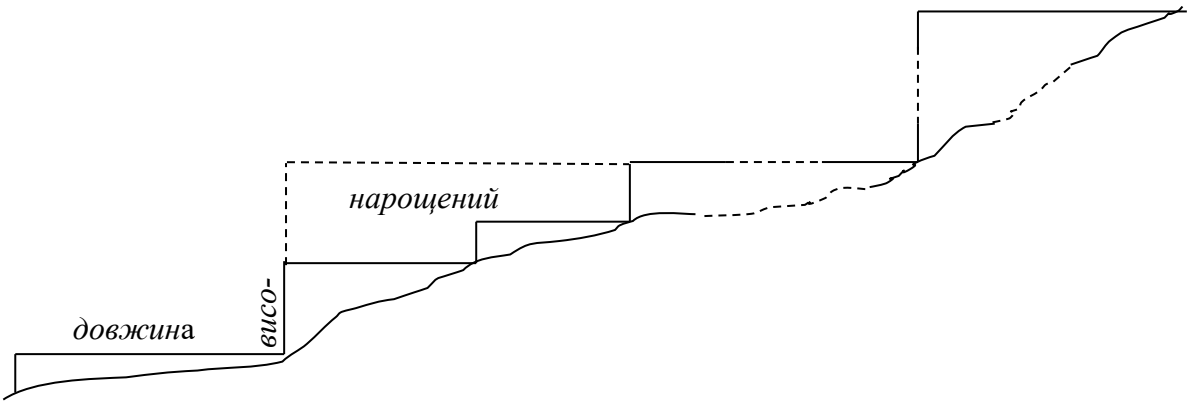
   Написати програму *Figura.** яка зменшує кількість ступенів фігури до m (1<m<n), добавляючи до неї деякі клітинки, розміщені над головною діагоналлю (на малюнку виділено пунктиром) і збільшуючи площу фігури мінімально.

2	3	7	5	9
6	2	8	1	5
8	9	4	4	6
8	7	5	7	2
6	6	7	3	9

Технічні умови: Вхідний файл *Figura.dat* містить в першому рядку числа *n* ти *m* – початкову та кінцеву кількість ступенів, наступні *n* рядків, що містять по *n* чисел кожен – елементи масива A[*n*×*n*].

Вихідний файл *Figura.res* повинен містити в першому рядку число, на яке збільшилась площа фігури, в кожному з наступних рядків по два числа - номери рядків і стовпчиків доданих до фігури елементів.

1. **(***11 клас***) RIV**(*30 балів*). На малюнку зображено в розрізі похиле русло річки й систему із *n* шлюзів, довжина *l* й висота *h* яких відома, а ширина однакова. При реконструкції системи шлюзів вирішили зменшити їх кількість, нарастивши стінки окремих із них так, щоб об'єднати кілька шлюзів, після чого їх стало *m*(2<*m*<*n*<10), а будівельного матеріалу було використано наймен-



ше.

Написати програму *Riv.**, що визначає, які стінки слід наростити і на скільки, якщо загальна площа нарощених під час реконструкції частин стін у розрізі повинна бути мінімальною.

Технічні умови: Вхідний файл *Riv.dat* в першому рядку містить два числа *n* і *m* – початкову і кінцеву кількість шлюзів, наступні *n* рядків – по два числа: l_i - довжину *i*-го шлюзу та h_i (1<*i*<*n*) – його висоту.

Вихідний файл *Riv.res* повинен містити в кожному з *m* рядків по два числа: k_i – початкові номери шлюзів, де починається нарощення *i*-го шлюзу та p_i – початкові номери шлюзів, де кінчається нарощення *i*-го шлюзу (1<*i*<*m*). Якщо шлюз не нарощувався, то k_i та p_i – співпадають.

IX. ЗАСТОСУВАННЯ ГРАФІВ У ПРОГРАМУВАННІ

IX.1 Графи, їх види та найпростіші властивості

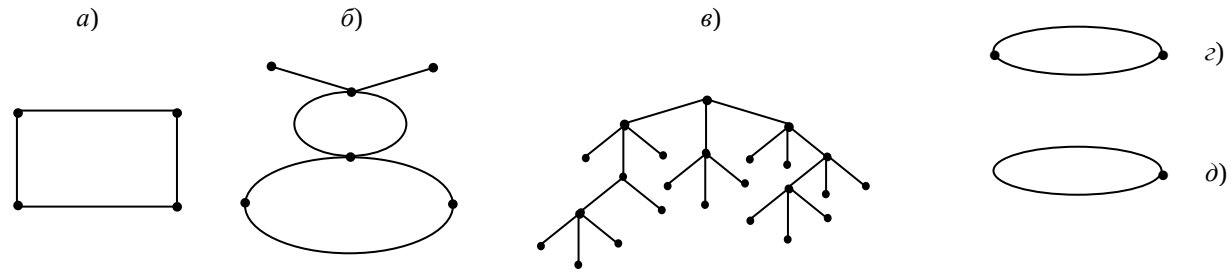
При розв'язуванні багатьох практичних задач, зокрема в галузі прикладної математики, техніки та економіки, часто використовують *графи,* які описані у відповідній галузі математики – *теорії графів*.

Існує декілька причин зростання інтересу до теорії графів. Безсумнівним є той факт, що теорія графів застосовується в таких областях, як фізика, хімія, теорія зв'язку, проектування обчислювальних машин, електротехніка, машинобудування, архітектура, дослідження операцій, генетика, психологія, соціологія, антропологія і лінгвістика. Ця теорія тісно пов'язана також і з багатьма розділами математики, серед яких - теорія груп, теорія матриць, числовий аналіз, теорія ймовірностей, топологія і комбінаторний аналіз.

За допомогою графів спрощується розв'язування задач автоматики, електроніки, фізики, хімії та інших наук. За допомогою графів зображуються схеми доріг, газопроводів, тепло- і електромереж і т.д.

Перша робота з теорії графів належить ще Леонарду Ейлеру, хоча сам термін "граф" уперше введено лише в 1936 році угорським математиком Денешем Кенігом.

Графами називають *схеми, що складаються із точок(вершин) і відрізків кривих (ребер або дуг), які з'єднують ці точки* (приклади графів зображені на малюнку 2). Якщо намагатись дати строге означення поняття граф, необхідно

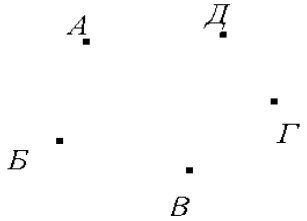


мал. IX.1

скористатись поняттями *множини* та *інцидентності (відповідності)*. Отже можна говорити, що *графом називається сукупність двох множин V (точок) та E (ліній), між елементами яких встановлено відношення інцидентності, причому, кожен елемент* $e \in E$ *інцидентний двом елементам* $v1, v2 \in V$.

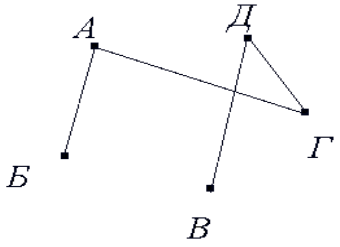
Познайомимося з основними поняттями теорії графів на прикладі.

Задача: Андрій, Богдан, Василь, Григорій і Дмитро при зустрічі потисли один одному руки (кожен потиснув руку кожному один раз). Скільки усього рукостискань було зроблено?



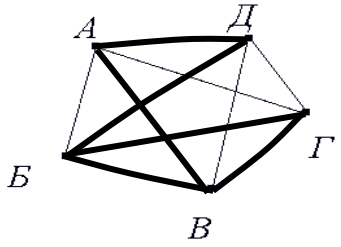
Нульовий граф із п'ятьма вершинами

мал. IX.2



Неповний граф із п'ятьма вершинами

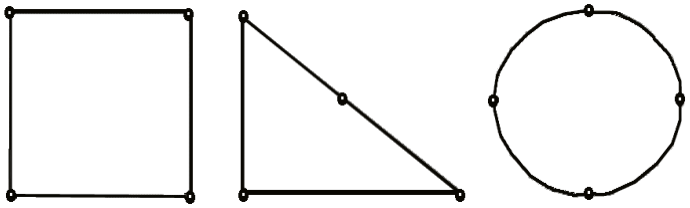
мал. IX.3



Повний граф

мал. IX.4

Нехай кожному з п'яти молодих людей відповідає визначена точка на площині, названа першою буквою його імені (мал. IX.2), а рукостисканню - від-



мал IX.5

різок частини кривої, що з’єднує конкретні точки-імена. Точки А, Б, У, Г, Д називаються *вершинами* графа, а відрізки ліній, що з'єднують ці точки - *ребрами* графа. При зображенні графів на малюнках-схемах відрізки можуть бути прямолінійними або криволінійними.

Довжини відрізків-ребер і розташування точок-вершин можуть бути довільними. Наприклад, усі три фігури на малюнку IX.5 зображують той самий граф. Якщо розглядати об’єкти, зображені на малюнку IX.5, як графи, що описують той самий процес, то вони міститимуть одну і ту ж інформацію. Такі графи називаються *ізоморфними*. Взагалі, ізоморфними називають два графи, які відрізняються тільки порядком вершин.

Розглянемо процес з’єднання точок А, Б, У, Г, Д ребрами. Ситуація, що відповідає моменту, коли рукостискання ще не відбувалися, являє собою схему, яка складається із точок (мал. IX.2). Така схема, що складається з «ізольованих» вершин, називається *нульовим (пустим) графом.* Ситуація, коли виконані ще не всі рукостискання, може схематично бути зображена, наприклад, за допомогою малюнка IX.3: потисли руки А і Б, А і Г, Д і Г, В і Д. Уявимо наступний момент, коли додадуться, наприклад, потискування рук А і В, Г і Б. Графи, у яких не побудовані всі можливі ребра, називаються *неповними графами.* На малюнку IX.4 зображений граф, що відповідає всім виконаним рукостисканням. Граф, що не є повним, можна доповнити до повного з тими ж вершинами, додавши відсутні ребра. Так, наприклад, на малюнку IX.3 зображений неповний граф з

п'ятьма вершинами. На малюнку IX.4 ребра, які перетворюють цей граф у повний граф, зображені товстішими лініями. Сукупність вершин графа на цьому малюнку лише з ребрами, зображеними товстішими лініями, називається доповненням графа.

Існують графи, пари вершин яких мають по кілька ребер (мал. IX.1*г*), або коли деяка вершина зв'зана сама з собою ребром (мал. IX.1*д*). В першому випадку говорять, що пара вершин *кратна*, а в другому ребро називають *петлею*. Якщо множини вершин *V* та ребер *E* скінченні, то відповідні графи також називаються *скінченними*.. Всі вищенаведені графи можна назвати неорієнтованими, бо ребра не вказують напрямок зв'язку між вершинами. Якщо ребра зображені у вигляді однонапрямлених стрілок, то такі графи називаються *орієнтованими*, вершини з яких виходять стрілки та вершини, на які вказують стрілки називаються *початком* і *кінцем* відповідного ребра. Далі, якщо не вказано окремо, будемо говорити лише про скінченні неорієнтовані графи.

Властивість 1. Неважко довести, що *повний граф з n вершинами має кількість ребер n(n-1)/2*.

Дійсно, у повному графі кожна вершина зв'язана окремим ребром із будь-якою з інших вершин, отже від кожної вершини проведено *n*-1 ребро, таким чином ребер повинно бути *n(n-1)*. Але тут кожне ребро враховано двічі, адже воно з'єднує дві вершини. Звідси випливає, що повний граф з *n* вершинами, має рівно *n(n-1)/2* ребер, тобто, висловлюючись термінами комбінаторики[11], це число сполучень з *n* елементів по 2.

Можна запропонувати завдання на закріплення поняття графа, перевірку та застосування знань про його види і найпростіші властивості:

IX.2 Степінь вершини графа

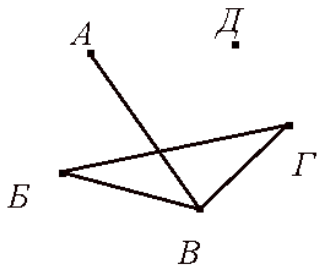
Степенем вершини графа називається кількість його ребер, що виходять з цієї вершини.

На малюнку IX.6 зображений граф, який має 5 вершин. Степінь вершини А позначимо Ст.$_А$. На малюнку : Ст.$_А$ = 1, Ст.$_Б$ = 2, Ст.$_В$ = 3, Ст.$_Г$= 2, Ст.$_Д$= 0. Вершина називається *непарною*, якщо степінь цієї вершини – число непарне, *парною* - якщо степінь цієї вершини – число парне. Для визначених графів справедливі такі закономірності, або властивості:

Властивість 2. *Степені вершин повного графа однакові, і кожна з них на 1 менше числа вершин цього графа.* Якщо степені усіх вершин графа рівні, то граф називається *однорідним*. Таким чином, будь-який повний граф - однорідний.

---

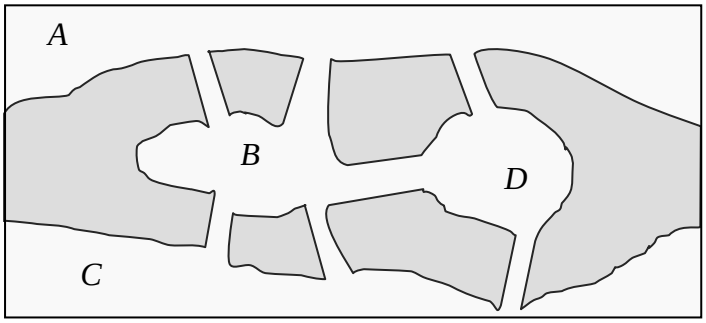
[11] Комбінаторика – це окрема галузь математики, що вивчає питання про розташування елементів деяких скінченних множин відповідно до точно заданих правил і про те, скількома різними способами таке розташування можна здійснити. Одним із найважливіших понять комбінаторики є поняття *комбінації* – будь-якої підмножини M_1 з *k* елементів, деякої множини M_0 з *n* елементів. Число комбінацій з n елементів по *k* елементів обчислюється за формулою $C_n^k = \frac{n!}{k!\cdot(n-k)!}$. Теорія графів тісно пов'язана з комбінаторикою.



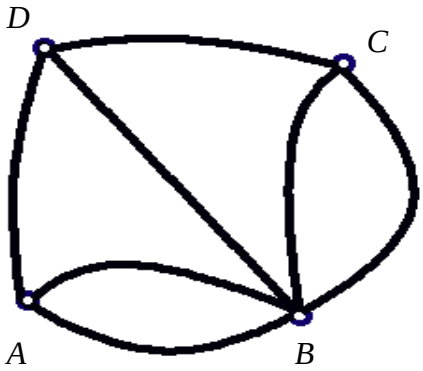
мал.IX.6

Властивість 3. Сума степенів вершин графа число парне, рівне подвоєному числу ребер графа. Ця закономірність справедлива не тільки для повного, але і для будь-якого графа.

Властивість 4. Число непарних вершин будь-якого графа число парне.



мал. IX.7



мал. IX.8

Проблема, що привела до написання Л. Ейлером першої наукової праці з теорії графів відома в історії, як *задача про Кенігсбергзькі мости*. ДО XVIII століття через річку, на якій стояло місто Кенігсберг у Східній Прусії (нині м. Калінінград у Російській Федерації), було побудовано 7 мостів, що зв'язували з берегами й один з одним два острови, розташовані в межах міста (див. мал. IX.7). Задача полягає в тому, щоб пройти (якщо це можливо) по всіх семи мостах так, щоб на кожному з них побувати лише один раз і повернутися до того місця, звідки почався маршрут. Розв’язати цю задачу вдалося в 1736 р. Л. Эйлеру. У ході розв’язування (після інтерпретації умови задачі у вигляді графа, у якому вершини - острови й береги, а ребра - мости, представлені на малюнку IX.7) видатний математик ввів багато вузлових понять та встановив, крім уже розглянутих нами вище, ще цілий ряд важливих закономірностей, що й лягли в основу теорії графів. Означимо поняття *зв’язного графа*. Вершини графа v_1 і v_2 називаються зв’заними, якщо існує маршрут з початком у вершині v_1 і кінцем у вершині v_2. Граф називається зв’язним, якщо всі його вершини зв’язані між собою.

Розв’язання задачі про Кенігсбергзькі мости досить просте. Проходження по всіх мостах за умови, що потрібно на кожному побувати один раз і повернутися на початок подорожі, мовою теорії графів виглядає як задача зображення одним розчерком графа, представленого на малюнку IX.8, іншими словами, степені всіх вершин повинні дорівнювати 2, тобто бути парними. Але, оскільки граф на цьому малюнку має чотири вершини, причому всі непарні, то такий

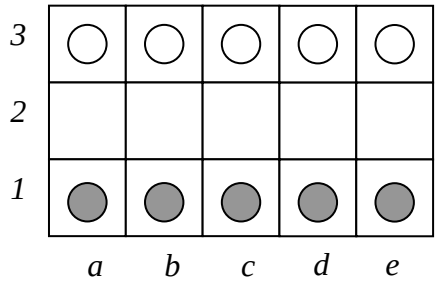
граф накреслити одним розчерком неможливо. Отже, пройти по кенігсбергских мостах, дотримуючись заданих умов, неможливо.

Додамо, що Л. Ейлер також ввів у теорію графів поняття *шляху* та *циклу*, в зв'язку з чим існують так звані Ейлерові шляхи та Ейлерові цикли, про які буде йти мова нижче.

IX.3. Найпростіші задачі, які розв'язуються з допомогою графів. Метод "гудзиків і ниток"

Але поки що зупинимось на кількох цікавих задачах, пов'язаних із поняттям степеня вершин та відповідними властивостями графів. Ці задачі розв'язуються методом "гудзиків і ниток".

Задача про коней. На першій та третій горизонталях шахової дошки розмірами 3Х5 розміщено по п'ятеро коней (чорні та білі відповідно, на *мал. 9* зображено кружечками). Визначити мінімальну кількість ходів, що міняють місцями білих і чорних коней.



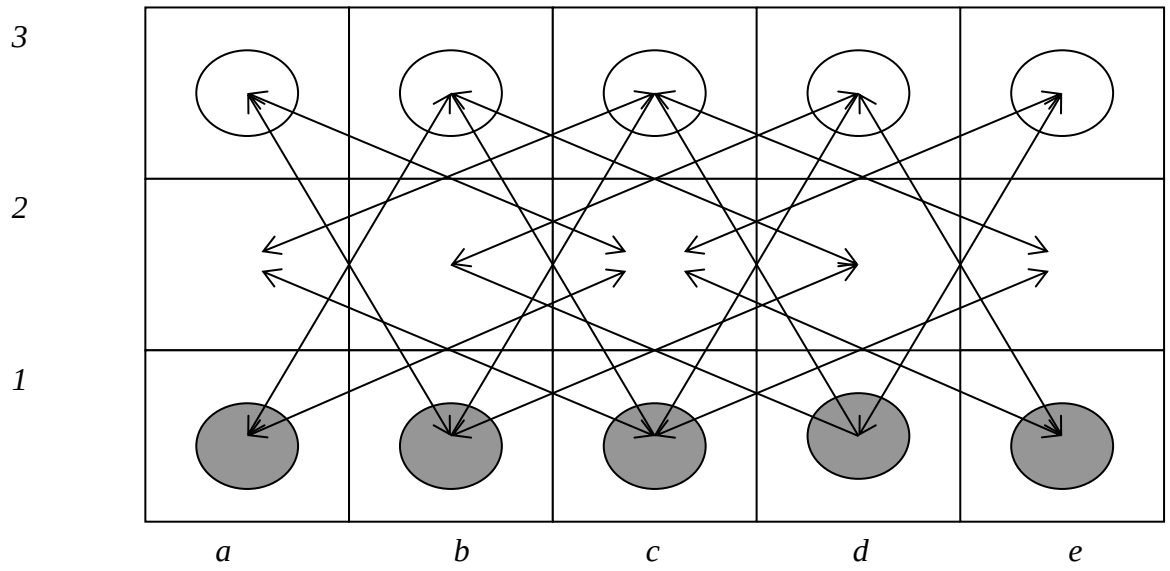
мал.IX.9

Це головоломка, дещо подібні їй (зокрема на полі 3Х3) можна зустріти у різних популярних збірниках логічних задач. Цікаво розглянути варіант цієї задачі у [8], яка уже згадувалась вище в зв'язку з поняттям ізоморфності представлення задачі. Проте завдання цілком оригінальне, адже вказані комбінації коней у літературі не зустрічаються. Для розв'язування задачі з використанням повного перебору неефективне навіть з допомогою комп'ютера, тому слід змоделювати граф і "розплутати" його достатньо відомим методом "гудзиків і ниток".

Завдання вимагає дуже цінного вміння досліджувати задачу, що не часто зустрічається серед учнів.

Вище (*див. мал. IX.10*) наведено хід міркувань, що приводять до розв'язку. Спочатку слід встановити з допомогою стрілок зв'язки між клітинками, пов'язаними між собою ходом конем і побудувати відповідний граф (мал. IX.11). Наступне його "розплутування" приведе до малюнка IX.12, на якому стрілками показано замкнуті маршрути, що дозволять поміняти місцями білих і чорних коней.

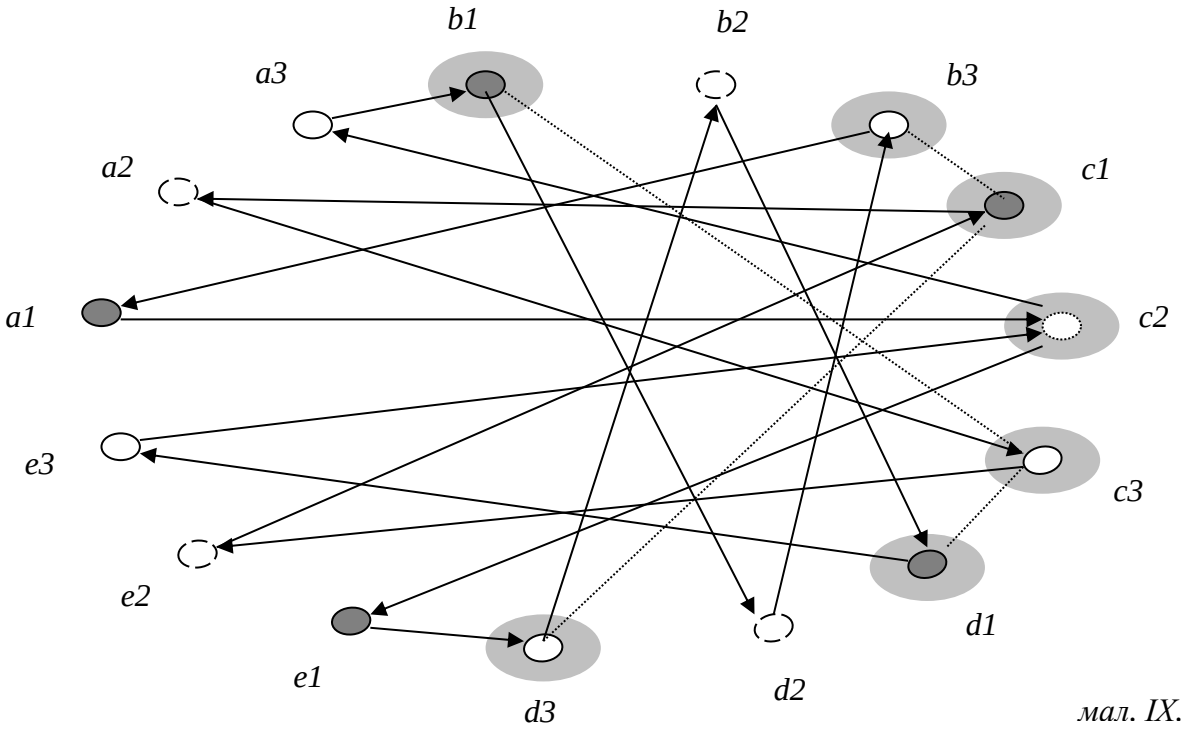
Початковий граф слід перетворити, видаливши ребра, які зв'язують між собою вершини, степені яких непарні (на мал.IX.11 ці вершини оточені сірими ореолами, а ребра - пунктиром).



мал. IX.10

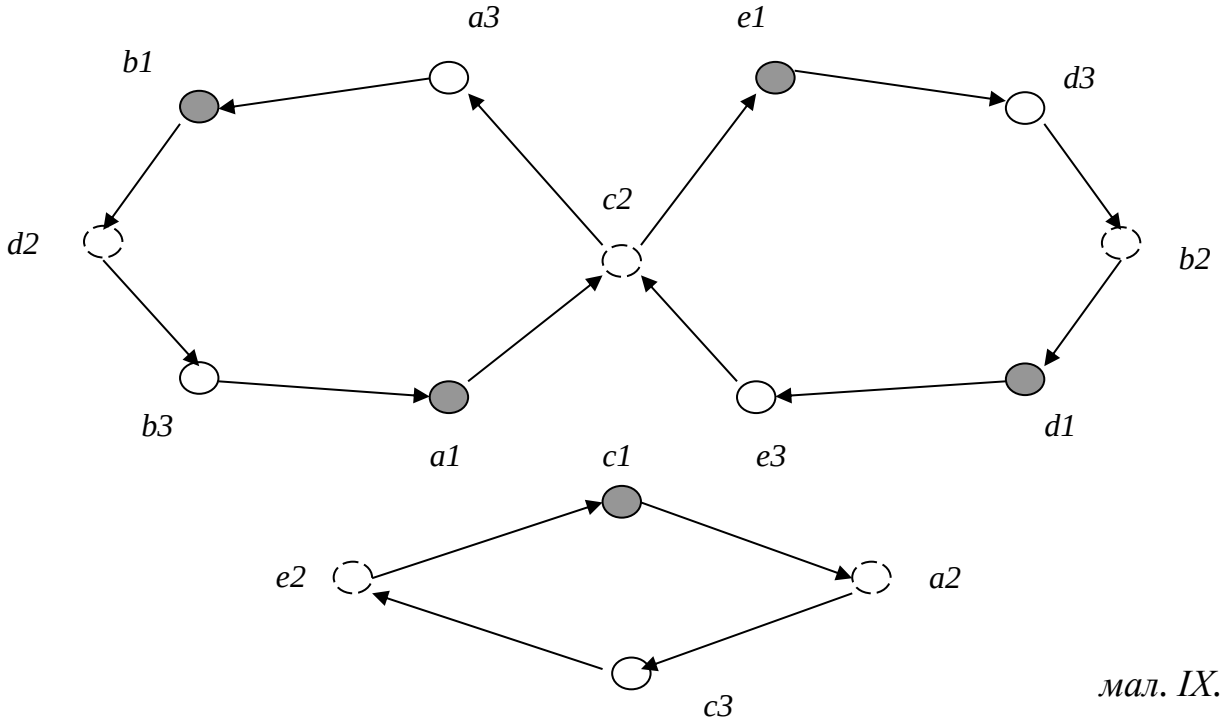
Існування і оптимальність розв'язку випливає із парності всіх вершин у кінцевому графі (мал.IX.12). Наведемо відповідь: найменша кількість ходів, що міняють місцями білих і чорних коней дорівнює 16:

1) a1-c2; 2) b1-d2; 3) b3-a1; 4) a3-b1; 5) c2-a3; 6) d2-b3;
7) e1-c2; 8) d1-b2; 9) e3-d1; 10) d3-e1; 11) c2-e3; 12) b2-d3;
13) c1-e2; 14) c3-a2; 15) e2-c3; 16) a2-c1.



мал. IX.11

Цікаво, що результуючий граф, як видно з малюнка IX.12, складається з трьох частин, на яких можна проілюструвати поняття *зв'язності* графа, *шляхів* і *циклів* у ньому. Але до цього поняття звернемось трохи нижче.



мал. IX.12

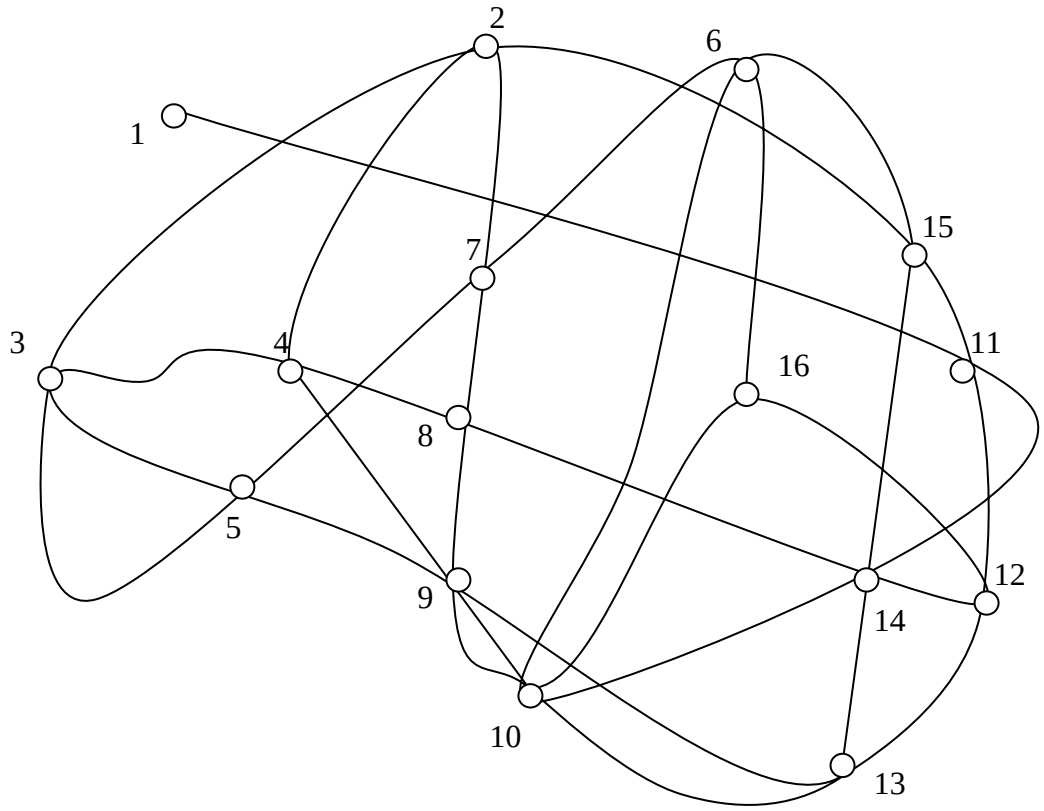
Розглянемо ще цікаву задачу.

Задача про зайця. На невеликій засніженій лісовій галявині ростуть кущі ліщини, калини та ожини. Заєць вибіг із нірки, розміщеної під одним із кущів, але відчувши поряд мисливців, деякий час плутав сліди, перебігаючи від одного куща до іншого, врешті причаївся під одним із кущів. Перед мисливцями стоїть завдання: визначити, якщо це можливо, де сховався заєць.

Для розв'язання цієї задачі слід побудувати граф, вершинами якого будуть кущі, а ребрами – сліди зайця, прокладені між кущами. У задачі сказано, що заєць сховався не в нірці, а під одним із кущів, отже кущ, де знаходиться нірка є непарною вершиною. Але, згідно з властивістю 4, число непарних вершин у будь-якому графі – парне, більше того, у нашому випадку таких вершин повинно бути лише дві (це легко зрозуміти). Отже заєць сховався під кущем, зв'язаним непарним числом слідів.

Як видно з малюнка IX.13, серед вершин під номерами 1-16 лише перша та остання непарні. Отже заєць може переховуватись тільки під одним із кущів, позначених номерами 1 або 16. Зауважимо, що у випадку парності всіх вершин заєць може знаходитись під будь-яким кущем, що зробить його пошуки практично неможливими.

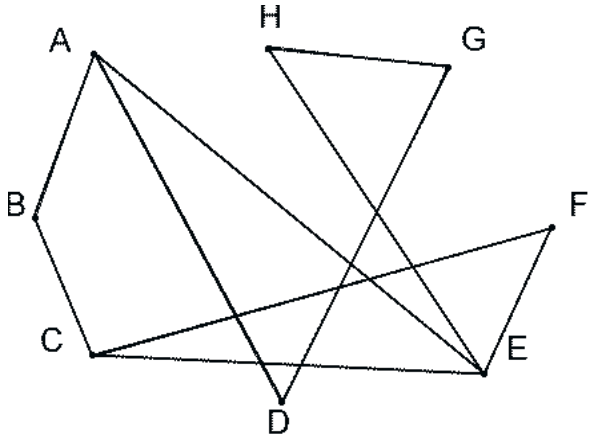
Вершини на малюнку пронумеровані довільно, хоча можна пронумерувати їх у порядку відвідання зайцем, але це окрема задача, пов'язана з поняттям шляху в графі.



мал. IX.13

IX.4 Шлях на графі. Цикли

На малюнку IX.14 за допомогою графа зображена схема доріг між населеними пунктами. Наприклад, із пункту A (вершина графа) у пункт H можна дістатись різними маршрутами: ADGH, AEH, AEFCEH, ABCEH .



мал. IX.14

Чим відрізняється маршрут AEH від маршруту AEFCEH? Тим, що в другому маршруті на перехресті в точці E ми побували двічі. Цей маршрут довший, ніж AEH. Маршрут AEH можна отримати з маршруту AEFCEH, викресливши з останнього маршрут FCE. Маршрут AEH є *шляхом* у графі, а маршрут AEFCEH *не є* шляхом.

Отже, *послідовність різних ребер графу, кожне наступне з яких починається в кінці попереднього, називається шляхом*. Шлях, у якому ніяке ребро не

зустрічається двічі, називається *простим*. Вершина, від якої прокладений маршрут, називається початком шляху, вершина наприкінці маршруту - кінець шляху.

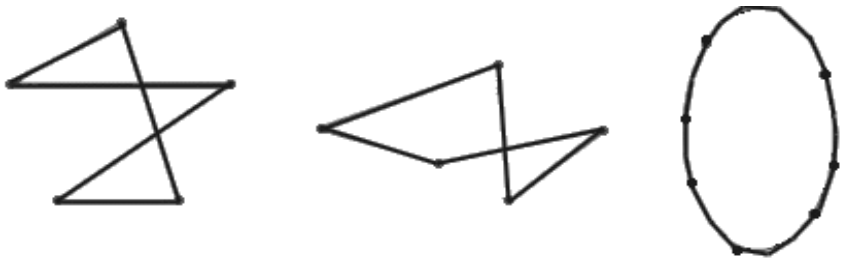
Існують різні маршрути, по яких можна добратися, наприклад, з вершини А у вершину Н.

Маршрути CFEC, ADGHECBA, ADFCEA будуть *циклами*, CFEC - *елементарний цикл*.

Циклом називається шлях, у якому збігаються початок з кінцем.

Якщо усі вершини циклу різні, то такий цикл називається *елементарним* (чи простим) циклом. Якщо ж цикл містить у собі всі ребра графа один раз, такий цикл називається *Ейлеровим циклом*, або *Ейлеровою лінією*. Зауважимо, що існують ще так звані *Гамільтонові цикли*, тобто цикли, що містять кожну вершину тільки один раз.

Гамільтон Вільям (1805 - 1865) - ірландський математик і астроном - один з творців теорії векторного числення, теорії комплексних чисел та теорії квантеріонів. Зробив внесок у створення теорії графів. Назва гамільтонів граф виникула у зв'язку з тим, що в 1859 році він випустив до продажу своєрідну іграшкову ломи головку. Її основною частиною бук правильний додекаедр, зроблений з дерева. Це один з так званих многогранників: його гранями є 12 правильних п'ятикутників, в кожній з його вершин сходиться 3 ребра. Кожна з вершин гамільтонового додекаедра була позначена назвою одного з великих міст Земної кулі – Брюссель, Кантон, Делі, Лондон і т.д. Задача полягала в знаходженні шляху вздовж ребер додекаедра, який проходить через кожне місто і тільки один раз. Гамільтонів цикл на додекаедрі не покриває, звичайно всіх ребер додекаедра, бо в кожній вершині він проходить в точності по двох ребрах.



мал. IX.15

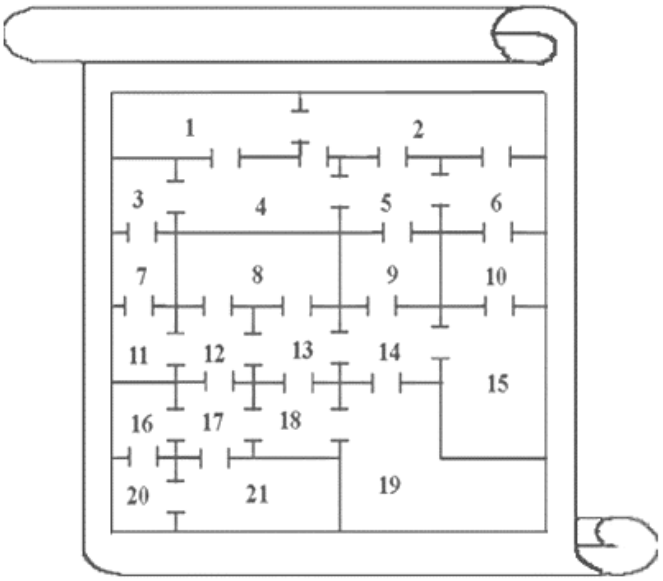
Додамо також, що *скінченні неорієнтовані графи* називаються *Ейлеревими*, тоді і тільки тоді, коли вони зв'язні і степені всіх вершин парні. На малюнку IX.15 наведено приклади Ейлеровых ліній. Очевидно, що шлях, прокладений по сторонах будь-якого багатокутника, який починається і закінчується в одній вершині, є Ейлеровою лінією.

Розглянемо задачу, яка проілюструє пошук у так званих лабіринтах[12] з допомогою графів.

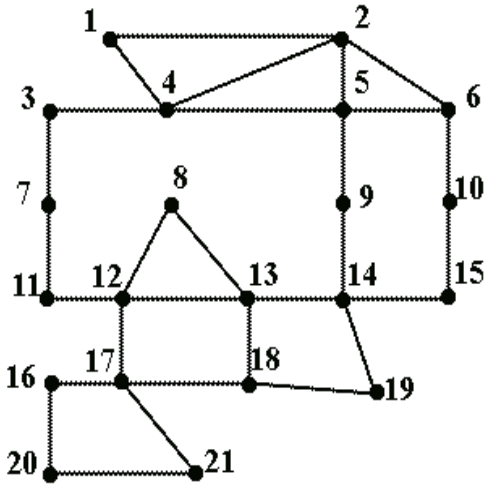
[12] Лабіринтами (від грецького λαβιριγοξ) античні автори називали споруди з багатьма складно з'єднаними кімнатами, з яких важко знайти вихід.

Задача про підземелля. На малюнку IX.16 представлено план підземелля, в одній із кімнат якого сховано ключа від скарбу.

Для того, щоб відшукати цього ключа, достатньо війти в одну із крайніх кімнат підземелля, пройти крізь усі двері, причому одного разу через кожні. Ключ сховано за дверима, які відкриються останніми. Визначити, в якій кімнаті сховано ключ.



мал. IX.16

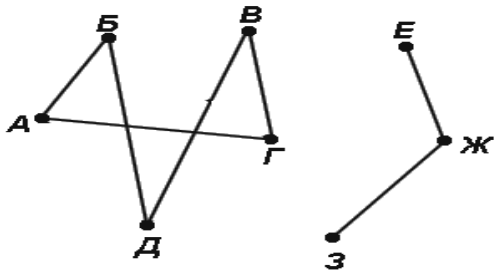


мал. IX.17

Підказка. У графі, який відповідає плану підземелля (мал. IX.17), вершини - кімнати, а ребра з'єднують лише ті вершини, які відповідають кімнатам, поєднаним дверима. У створеному відповідно схемі підземелля графі (мал.IX.17) лише дві непарні вершини. Знайдіть ці вершини і пригадайте властивість Ейлерового шляху.

IX.5 Зв'язні графи

Дві вершини графа називаються зв'язаними, якщо в графі існує шлях із кінцями у цих вершинах. Якщо такого шляху не існує, вершини називаються не зв'язаними.



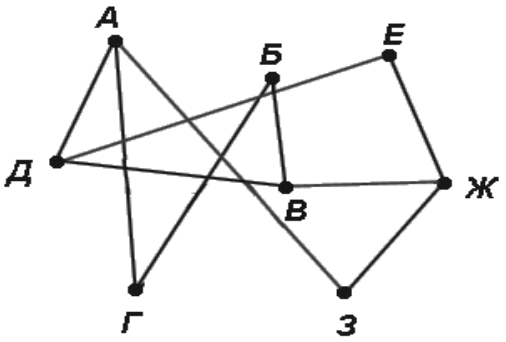
мал. IX.18

Так, на малюнку IX.18 будь-яка пара вершин із набору А,Б,В,Г,Д дає зв'язаний граф, оскільки від кожної з них до будь-якої можна "пройти" по ребрах графа.

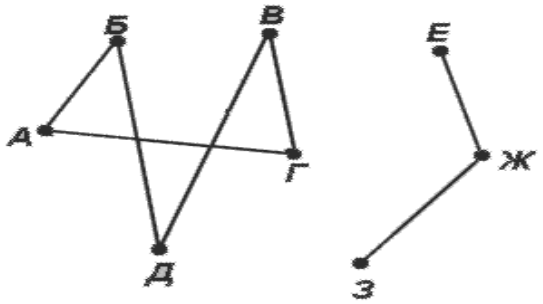
Пари вершин, одна із яких А,Б,В,Г,Д, а інша Е,Ж,З не зв'язані, тому що від одної до іншої "пройти" по ребрах неможливо.

Граф називається *зв'язаним*, якщо будь-яка пара його вершин зв'язана. Граф називається *незв'язаним*, якщо він має хоча б одну незв'язну пару вершин. На малюнку IX.18 зображено незв'язаний граф.

Якщо, наприклад, між вершинами Д и Е провести ребро, то граф стане зв'язаним. *Ребро*, в теорії графів, *після видалення якого граф зі зв'язаного перетворюється у незв'язаний*, *називається мостом*. Прикладами мостів на малюн-



мал. IX.19



мал.IX.20

ку IX.18 могли б служити ребра ДЕ, А3, ВЖ та інші, кожні з яких з'єднували б вершини ізольованих частин графа (див. мал. IX.19).

Завдання для самостійного опрацювання:

1. Визначити, зв’язаний чи незв’язаний граф представлено на малюнку IX.20**:**

2. Вказати, чи є зв’язаними на малюнку 20 вершини:

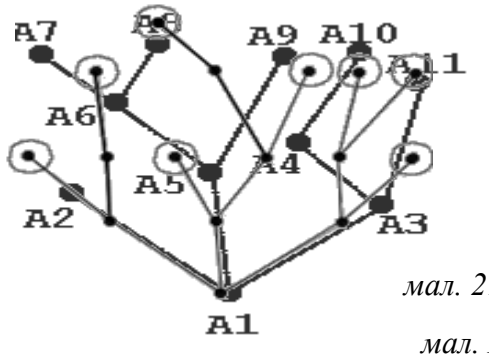
А) Б і В; Б) Г і Ж; В) А і Г.

Скільки мостів можна побудувати для заданого графа?

А) 15; Б) 10 В) 12 Г) 18.

IX.6 Дерева

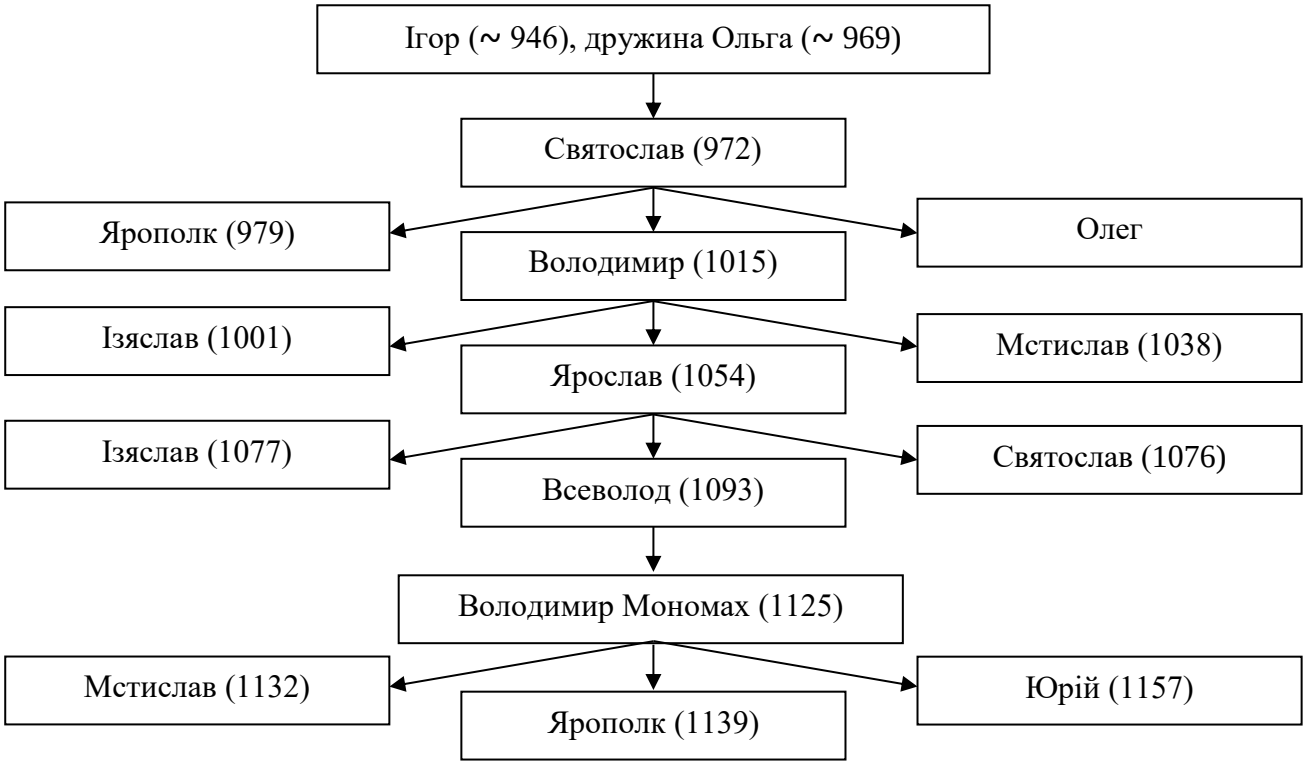
Деревом називається будь-який зв'язний граф, що не має циклів. Будемо вважати деревом і всякий граф, що складається з однієї (ізольованої) вершини. На малюнку IX.21 наведено приклад графа-дерева. Вершина дерева, що має степінь 1, називається висячою вершиною (на мал. IX.21 вони позначені кружком).



мал. 21

мал. IX.23

Властивість 5. Для кожної пари вершин дерева існує єдиний шлях, що їх з'єднує. Цією властивістю користуються для визначення всіх предків, наприклад, по чоловічій лінії, або будь-якої людини, чий родовід представлено генеологічним деревом, що є деревом і у теорії графів.

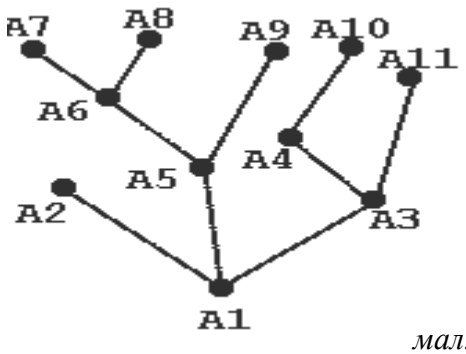


мал IX.22

Наприклад, на малюнку-дереві, вершини якого зображені прямокутниками з інформацією, представлене генеалогічне дерево – родовід київських князів (мал.. IX.22).

Властивість 6. Усяке ребро в дереві є мостом.

Дійсно, після видалення будь-якого ребра дерева, воно розпадається на два дерева. Це легко побачити на малюнку IX.23, видаливши, наприклад, ребро A5A6, або ребро A1A3, або ребро A4A10 (залишивши другим деревом вершину A10).



мал. IX.23

Властивість 7. Дерево із n-вершинами має n-1 ребро.

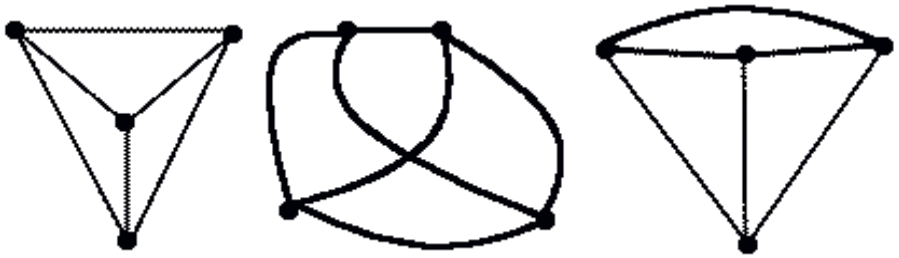
Доведення: Для дерева - ізольованої вершини n=1 і ребер 0, що відповідає властивості (n-1 = 1-1 = 0). Якщо із графа-дерева, що не є ізольованою вершиною, видалити одне ребро, то отримаємо два дерева із тими ж вершинами. А щоб отримати три дерева, потрібно видалити два ребра; для одержання чотирьох дерев - три ребра і т.д. Найбільша кількість дерев із графа з n-вершинами може дорівнювати n (n ізольованих вершин), чого можна досягти, видаливши n-1 ребро. І т.д.

IX.7 Ізоморфні графи Поняття плоского графа. Формула Эйлера.

Для підрахунку числа рукостискань, які роблять чотири товариші А, Б, В і Г при зустрічі, можна скористатися , якщо побудувати повний граф із чотирма вершинами.

На малюнку IX.24 представлено кілька варіантів зображення повного графа, що має 4 вершини.

Графи, зображені на цьому малюнку, підтверджують інформацію про можливі рукостискання чотирьох товаришів. *Такі графи називають ізоморфними (однаковими)*.

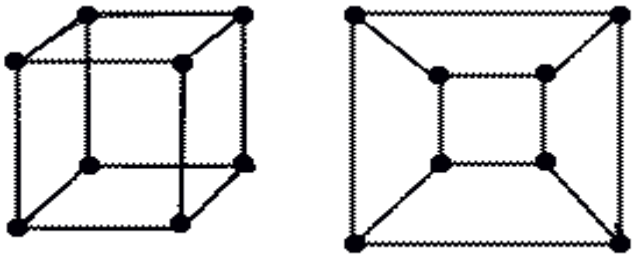


мал. IX.24

Для того, щоб з'ясувати, чи є ізоморфними дві графи, потрібно переконатися в тому, що у них:

- однакова кількість вершин ;
- якщо вершини одного графа з'єднані ребром, то й відповідні їм вершини іншого графа також з'єднані ребром.

На другому зображенні ребра графу перетинаються, а на першому і третьому зображеннях графи не мають пересічних ребер. Граф, який можна накреслити так, щоб його ребра перетиналися тільки у вершинах, називається *плоским графом* (мал. IX.25).



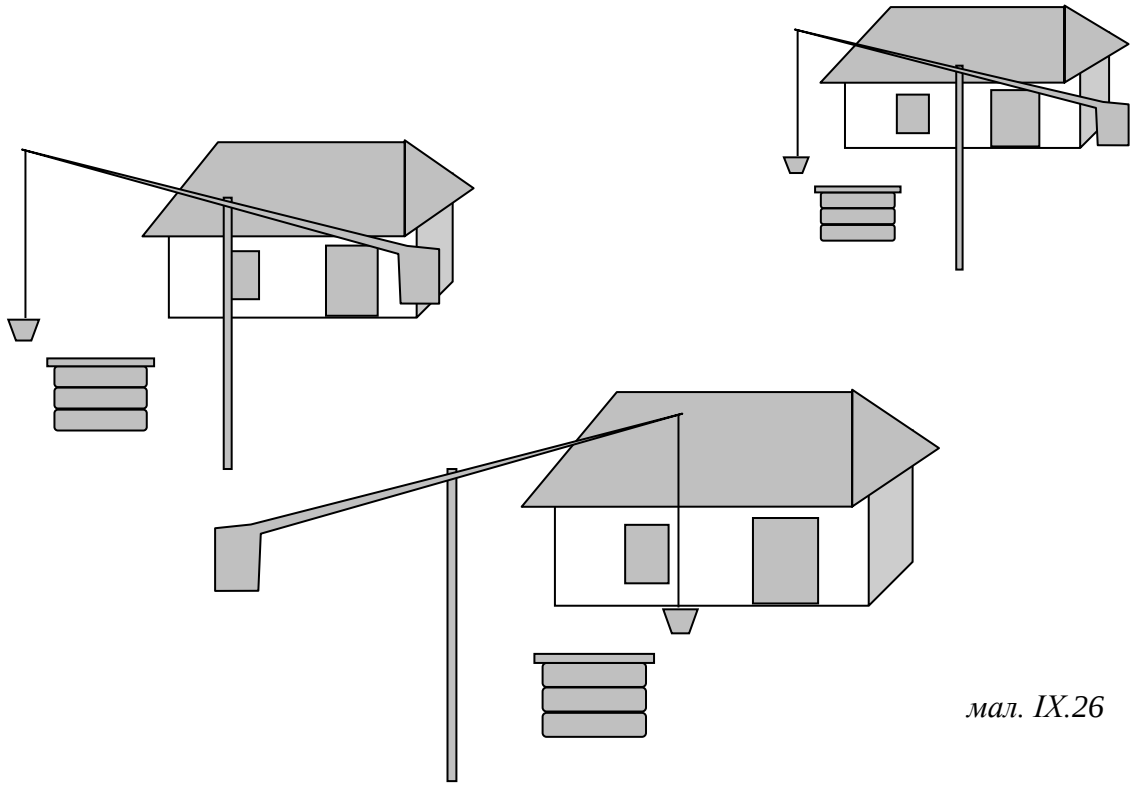
мал.IX. 25

Прості цикли і дерева є наочними прикладами плоских графів.

Розв’яжемо стародавню задачу про три будинки і три колодязі, суть якої зводиться до з'ясування питання, чи є розглянутий граф плоским.

Задача про три будинки і три колодязі. У трьох різних будинках живуть три сусіди, які між собою посварилися (мал.. IX.26). Недалеко від їхніх будинків розташовані три колодязі. Чи можна від кожного будинку прокласти до кожного з колодязів стежину так, щоб ніякі дві з них не перетиналися?

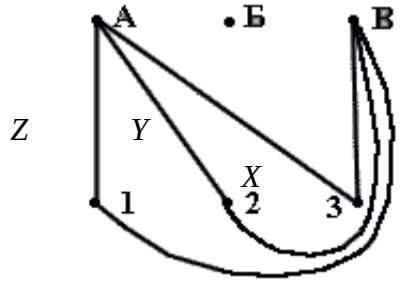
Розв'язання: Після проведення 8-х стежин можна переконатися, що провести дев'яту стежину, не пересічну ні з якою із раніше проведених , не вдається. Побудуємо граф, вершини якого А, Б, В, 1, 2, 3 відповідають будинкам і колодязям умови задачі (мал. IX.27), і спробуємо довести, що дев'яту стежину - ребро графа, що не перетинає інші ребра, провести неможливо.



мал. IX.26

Проведені в графі на малюнку IX.27 ребра А1, А2, А3 і В1,В2,В3 відповідають стежинам, які проведені від будинків А і В до всіх колодязів.

Побудований граф розбив площину на три області: X, У, Z.



мал.IX. 27

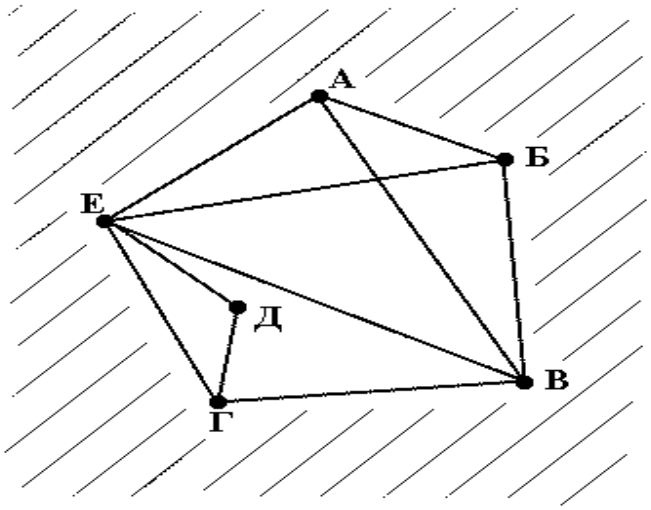
Вершина Б, у залежності від її розташування на площині, попадає в одну із цих трьох областей. Якщо ми розглянемо кожен із трьох випадків попадання вершини Б в одну з областей X, Y чи Z, то переконаємося, що кожного разу одна з вершин графа 1, 2 чи 3 (один із колодязів) буде недоступною для вершини Б (тобто неможливо провести одне із ребер Б1, Б2 чи Б3, яке не перетинало б уже наявних у графі ребер).

Таким чином, відповідь задачі буде такою: НЕМОЖЛИВО!

Плоский граф, окрім вершин і ребер, характеризують ще й гранями.

Гранню плоского графа називається частина площини, обмежена простим циклом і та, що не містить усередині циклів.

На малюнку IX.28 зображений плоский граф, який має 4 грані: АВЕА, ЕБВГДЕ, ЕДГЕ і зовнішня стосовно циклу АБВГЕА область, яку називають також нескінченною гранню (на малюнку вона позначена штрихуванням).



мал.IX. 28

Дерево має одну грань- зовнішню (нескінченну грань).

Розглянемо довільний граф - дерево, що має В-вершин і Р-ребер.

За властивістю 7, дерево має число ребер на 1 менше, ніж число вершин, тобто В-Р=1. Якщо будемо перетворювати вихідне дерево, додаючи до нього непересічні ребра (щоб графи, які виходять, залишалися плоскими), утворюватимуться нові грані (крім однієї нескінченної на початку). Простежимо, якими стануть співвідношення між кількістю вершин, ребер і граней отриманих графів.

Очевидно, що при додаванні одного ребра до графа число граней збільшується на 1 (з'являється простий цикл). При наступних додаваннях із кожним новим ребром або з'являється новий цикл, або існуючий розпадається на два (тобто в будь-якому випадку при додаванні одного ребра число граней також збільшується на одну).

Таким чином, при даному перетворенні різниця Р-Г постійна (тут Р - число ребер, а Г- число граней отриманого в результаті перетворень графа). Оскільки у дерева одна грань (Г=1), то Р-Г = Р-1.

Розглянуте перетворення не змінює заданого числа вершин В. Скориставшись формулами і рівністю Р = В - 1 одержимо, що Р – Г = В - 2. Остання формула В – Р + Г = 2 називається *формулою Ейлера*. Цю формулу називають також формулою Эйлера для багатогранників, тому що це співвідношення справедливе для вершин, ребер і граней усіх просторових багатогранників.

IX.8 Представлення графів.

Графи зручно представляти одним із наведених нижче способів.

1. Матриця інцидентності (таблиця IX.1). Вона представляє собою прямокутну таблицю, у якій рядки відображають ребра графа, а стовпчики – вершини. Одиниці у кожному рядку вказують на те, що дві вершини інцидентні ребру, номер якого співпадає з номером відповідного рядка. Цей спосіб представлення не економний, бо в кожному рядку міститься лише дві одиниці.

2. Матриця суміжності (таблиця IX.2). На перетині кожного рядка і стовпчика поставлено одиницю, якщо вершини з такими номерами суміжні (з'єднані ребром).

3. Використовують також представлення графа з допомогою переліку вершин (таблиця IX.3).

таблиця IX.1

	I	II	III	IV	V	VI	VII
1	1	1	0	0	0	0	0
2	1	0	1	0	0	0	0
3	0	1	0	1	0	0	0
4	1	0	0	0	1	0	0
5	0	1	0	0	0	1	0
6	0	0	1	1	0	0	0
7	0	0	1	0	1	0	0
8	0	0	0	1	0	1	0
9	0	0	0	0	1	0	1
10	0	0	0	0	0	1	1

таблиця IX.2

	I	II	III	IV	V	VI	VII
I	0	1	1	0	1	0	0
II	1	0	0	1	0	1	0
III	1	0	0	1	1	0	0
IV	0	1	1	0	0	1	0
V	1	0	1	0	0	0	1
VI	0	1	0	1	0	0	1
VII	0	0	0	0	1	1	0

таблиця IX.3

ребра	Вершини
1	I,II
2	I, III
3	II, IV
4	I, V
5	II, VI
6	III, IV
7	III, V
8	IV, VI
9	V, VII
10	VI, VII

Кожен із наведених способів має свої недоліки і переваги, тому використовується, залежно від обставин.

Наведені таблиці описують один і той же граф. Пропонується самостійно відтворити його за кожним способом представлення. Цікавим і корисним буде завдання написати програму, яка за однією з таблиць генерує дві інші.

IX.9 Задачі

Задача 1. У трьох вершинах правильного п'ятикутника розмістили по фішці. Дозволяється пересувати їх по діагоналі в будь-яку вільну вершину. Чи можна таким способом досягти того, щоб одна з фішок повернулась на своє місце, а дві інші помінялись місцями?

Розв'язання. Послідовно занумеруємо вершини п'ятикутної зірки (отриманої в результаті можливих переміщень) числами 1, 2, 3, 4, 5. Бачимо, що з вершини 1 можна за один хід попасти або у вершину 1 або у вершину 4; з вершини 2 - або у вершину 4, або у вершину 5; З вершини 3 - або у вершину 5, або у вершину 1; з вершини 4 - або у вершину 1, або у вершину 2; з вершини 5 - або у вершину 2, або вершину 3. Ці переміщення "стають більш закономірними", якщо вершини такої п'ятикутної зірки "розвернути" на п'ятикутнику в такому порядку: 1, 3, 5, 2, 4 Тоді переміщенням по сторонах зірки відповідають переміщення фішок по п'ятикутнику в сусідню вершину. (якщо вона вільна). Очевидно, що за допомогою таких переміщеньпо п'ятикутнику не можна змінити порядок розміщення фішок, а отже ні можна досягти того, щоб одна з фішок повернулась на своє місце, а дві інші помінялись місцями.

Задача 2. Чи можна накреслити на площині 25 відрізків так, щоб кожен з них перетинав рівно 5 інших відрізків?

Розв'язання. Поставимо у взаємно однозначну відповідність кожному відрізку деяку точку на площині. Довільну пару таких точок з'єднаємо ребром тоді і тільки тоді, коли відповідні їм відрізки перетинаються. В отриманого

графа кількість непарних вершин - парне число, а тому відповідь: ні.

Задача 3. Розбита на клітинки площина пофарбована десятьма фарбами так, що сусідні (тобто ті, які мають спільну сторону клітинки мають різний колір, причому використано всі 10 фарб. Назвемо сусідніми дві фарби, якими пофарбовані дві сусідні клітинки Яка мінімальна можлива кількість пар сусідніх фарб?

Розв'язання. Кожному розфарбуванню площини поставимо у від повідність граф таким способом. Зобразимо фарби точками з номери ми 1, 2, 3,..., 10. Якщо фарби сусідні, то з'єднуємо відповідні точки ребром. Тоді кожній парі сусідніх фарб відповідатиме ребро графа. Очевидно, що з кожної вершини графа виходить хоча б одне ребро Отриманий граф має бути зв'язним, бо інакше площина розіб'ється ним на кілька відокремлених одна від одної частин, що неможливо. Найменша кількість ребер у зв'язних графів з однаковою кількістю вершин - у дерев. Отже, кількість пар сусідніх фарб не може бути меншою за 9. Залишається вказати таке розфарбування площини, при якому кількість пар сусідніх фарб дорівнює точно 9.

Задача 4. У тридев'ятому королівстві кожні два міста з'єднані дорогою з одностороннім рухом. Довести, що існує місто, з якого в будь-яке інше можна проїхати не більш як двома дорогами.

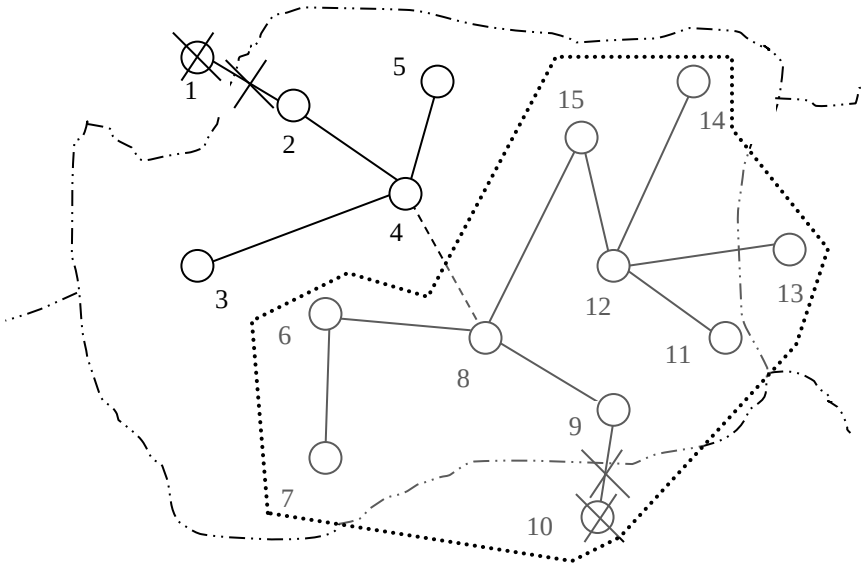
Розв'язання. Позначимо за А місто, з якого виходить найбільша кількість доріг. Доведемо методом математичної індукції по кількості міст королівства, що з міста А можна проїхати в будь-яке інше місто більш як двома дорогами.

Задача при n - 1, 2, 3 очевидна. Припустимо, що при кількості міст n при будь-якому виборі напрямків доріг між містами з міста А можна проїхати не більш як двома дорогами в будь-яке інше місто королівства.

Нехай тепер кількість міст дорівнює $n + 1$. Якщо з А є пряма дорога в усі міста королівства, то нічого доводити не треба. У протилежному випадку нехай В - місто, дорога з якого веде в А. Тоді, за припущеним, з міста А можна проїхати не більш як двома дорогами в будь-яке інше місто королівства, крім міста В. Можливі випадки: 1) існує місто С, відмінне від містА, В, дорога з якого веде в місто А. Тоді, відкинувши з розгляду місто С, отримуємо за припущенням, що з міста А в місто В можна проїхати не більш як двома дорогами; 2) такого міста С не існує. Тоді в місто А входить лише дорога з міста В, а тому, навіть відкинувши з розгляду довільне місто С, залишається справедливим твердження про те, що з міста А виходить найбільша кількість доріг. Тоді за припущенням з міста А в місто В можна проїхати не більш як двома дорогами.

Задача № 5 (пропонувалась на Київській обласній заочній Інтернет-олімпіаді в 2006 році) Газотранспортна система країни, розрахована на постачання природного газу з кількох сусідніх країн, складається з n вузлів та m трубопроводів, що їх сполучають (приклад дивись на малюнку IX.29). У зв'язку із переорієнтацією імпортної політики країни окремі вузли, через які надходив природний газ, та з'єднані з ними трубопроводи необхідно демонтувати (на малюнку перекреслено). Написати програму, яка визначає число k - найменшу кількість додаткових трубопроводів, які треба добудувати (на малюнку показано

пунктиром), щоб не порушити забезпечення газом споживачів.



мал. IX.29

Технічні вимоги: вхідний файл gas.dat, вихідний файл gas.sol.

Формат вхідних даних: у першому рядку через пропуск записано номери вузлів, через які надходить природний газ, ті з них, які слід демонтувати, взято в дужки, у кожному з m наступних ($1<m\leq 100$) рядків через пропуск записано номери вузлів, з'єднаних m-им трубопроводом.

Формат вихідних даних: вихідний файл повинен містити число k.

Коректність вхідних даних не перевіряти.

Приклад (відповідно малюнку) :

Вхідний файл*:* gas.dat

(1) (10) 13
1 2
2 4
3 4
4 5
6 7
6 8
8 9
8 15
9 10
11 12
12 13
12 14
12 15

вихідний файл gas.sol

1

Розв'язання. Розв'язання зводиться до відшукання зв'язних компонентів графа (нехай G), побудованого згідно умови задачі. Скористаємось вхідними даними і малюнком прикладу, з якого видно, що число k залежить від: *а*) кількості зв'язних компонент графа G; *б*) розміщення вузлів типу 13 (таких, через які газ поступає в газотранспортну систему і які не підлягають демонтуванню). Очевидно, що вилучення вершин типу 1, 10 та відповідних ребер (на малюнку перекреслено) на розв'язок задачі не впливає, тому для скорочення програмного коду відповідний фрагмент не будемо писати. Підготуємо масив $M1[1..n]$, у якому

натуральними числами позначено вершини, що входять у зв'язні компоненти:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
M[i]	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2

На малюнку один із двох для даного прикладу зв'язних компонентів виділено контуром, обмеженим точками, а у таблиці позначено цифрою "2". Після цього залишається запам'ятати в іншому масиві (наприклад *M*2) вершини, записані у першому рядку вхідного файлу без дужок (у прикладі вершина 13) і підрахувати кількість зв'язних компонент графа G, які не містять жодної вершини, записаної у *M*2. Це й буде шукане число *k*.

```
program GAS;
 const m=100;
 type arr1=array[1..m,1..m] of byte; arr2=array[1..2,1..m] of byte; lin = array[1..m]
of byte;
 var  i,j,n:byte;
       fv:text;
      C:arr1;D:arr2; use : lin; t,res: integer;

procedure INPUT(var cont:arr1);
 var i,j,m : byte; st1,st2 : string;
begin
 Assign(fv, 'gas.dat'); Reset(fv);
 n:=0; m:=0; i:=1; readln(fv, st1);
 while pos('(', st1) <> 0 do
  if st1[pos('(', st1)-1]=' ' then
   delete(st1,pos('(', st1)-1,(pos(')', st1)-pos('(', st1)+2))
  else delete(st1,pos('(', st1),(pos(')', st1)-pos('(', st1)+1));
  if st1[1]=' ' then delete(st1,1,1);
 i:=1;
 if pos(' ', st1) <> 0 then
  while pos(' ', st1) <> 0 do                                  begin
    st2:=copy(st1, 1, pos(' ', st1)-1);
    delete(st1, 1, pos(' ', st1)); val(st2, use[i], t); inc(i)  end
  else val(st1,use[1], t);
 if st1 <> '' then val(st1, use[i], t);
 while not EoF(fv) do   begin
   read(fv, i); readln(fv, j);
   if i > n then n:=i; if j > n then n:=j;
   cont[i,j]:=1; cont[j,i]:=1;
   inc(m)                    end;
 Close(fv)
end;
procedure CONNECT(A:arr1;n:byte;var B:arr2);
 var i,j,k,l,m,f,i1,j1:byte; check : lin;
begin
 for i:=1 to 2 do for j:=1 to n do B[i,j]:=0;
```

```
f:=0;l:=1;
while f=0 do                          begin
 i:=1;j:=1;
 while B[1,i]<>0 do i:=i+1;B[1,i]:=l;
 while j<=n do
  if (B[1,j]=l) and (B[2,j]=0) then begin
  for k:=1 to n do
    if A[j,k]=1 then B[1,k]:=l;
   B[2,j]:=1;j:=1                     end
  else j:=j+1;
 f:=1;for m:=1 to n do
       if B[1,m]=0 then f:=0;l:=l+1 end;
for i:=1 to n do check[i]:=0;
 t:=1;
 for i:=1 to n do                                 begin
   for j:=1 to n do
    if D[1,j] = i then    begin
     check[t]:=j; inc(t) end;
   for i1:=1 to n do
    for j1:=1 to n do
     if (use[i1] = check[j1]) and (use[i1]<>0) then  begin
      i1:=n; j1:=n; inc(res)                        end;
    t:=1; for i1:=1 to n do check[i1]:=0       end;
  t:=0;
  for i:=1 to n do
   if d[1,i]>t then t:=d[1,i];
Assign(fv,'gas.sol');ReWrite(fv);
writeln(fv,t-res);
Close(fv)
end;
begin
 Input(C); Connect(C,n,D)
end.
```

У наведеній нижче програмі **GAS** процедура **INPUT** відповідає за читання вхідних даних та створення матриці суміжності графа G і списку вузлів-постачальників газу після модернізації газотранспортної системи. Процедура **CONNECT** визначає зв'язні компоненти графа G, зберігає інформацію про них і обчислює число *k*.

Програма перевірена на п'яти тестах:

Gas.dat	1	2	3	4	5
	(1) (10) 13	1 (4) 7	(3) 7 (8)	1 (3) 5 (8) 10	4 (9) (12) 15
	1 2	1 2	1 11	1 2	3 4
	2 4	2 5	11 10	3 4	3 2
	3 4	2 6	9 10	5 6	1 2
	4 5	2 7	9 15	7 8	1 19
	6 7	3 5	15 8	9 10	3 20

	6 8	4 5	15 12	6 5
	8 9		11 12	6 7
	8 15		2 3	5 20
	9 10		2 4	7 19
	11 12		7 14	8 18
	12 13		6 14	18 16
	12 14		6 5	15 16
	12 15		5 13	10 15
				9 10
				14 17
				13 17
				12 13
				11 17
Gas.sol	1	2	3	4 5
	1	0	2	2 1

X. ВИСНОВКИ. ТЕОРЕТИЧНЕ, АПАРАТНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ОЛІМПІАД ІЗ ІНФОРМАТИКИ.

З огляду на те, що олімпіади з інформатики, як наголошувалось вище, не відповідають віку учасників по змісту та складності, потребують додатково значних спеціальних знань та здібностей з математики, вивчення інформатики починається переважно в 10-х класах, а вивчення розділу “Основи алгоритмізації та програмування” згідно діючої програми передбачається лише на другому році навчання і в дуже обмеженому обсязі (12-15 навчальних годин), *підготовку до олімпіад не слід пов’язувати з уроками інформатики*, її треба починати у 7-8 класах, при наявності факультативних курсів. *Факультативні курси необхідні не тільки з метою більш раннього ознайомлення з предметом, а й для того, щоб забезпечити теоретичний, в першу чергу математичний, супровід*. Без нього неможливо виробити свідомі уміння та навички в цій галузі. Дійсно, вже на початкових етапах оволодіння технікою програмування учні повинні розібратись з поняттями й типами величин, з представленням чисел, з процедурами, функціями. Інформатика всі ці поняття розглядає хоч і в близькому до загальноприйнятого, але в суттєво відмінному, розширеному смислі. Тому паралельно з вивченням програмування необхідно вчасно закріплювати й узагальнювати поняття величини, числа, функції, послідовності і т.д. Це можливо тільки за умови, що вчитель інформатики комбінуватиме курси програмування та прикладної математики. Тут можна скористатись програмою спецкурсу “Прикладна математика” для учнів 8-11 класів з поглибленим вивченням математики канд. фіз.-мат. наук О. Б. Рудика [1]. Ця програма передбачає вивчення елементів математичної логіки, комбінаторики, цілих чисел, кільця многочленів, подільності, оптимізації та методів перебору, координатного методу в планіметрії та стереометрії, графів і теорії ігор. Крім цих, досить нетрадиційних для учнів загальноосвітніх шкіл тем, протягом вивчення курсу програмування необхідно оволодіти також рекурсивними алгоритмами, методами пошуку, сортування, переборів, методом динамічного програмування, евристичними методами і т.д.

В різних джерелах надруковано дуже багато олімпіадних задач з інформатики з поясненнями, розв'язаннями, тестами і лістингами програм. Проте, задачі частіше класифікуються по роках чи турнірах, без системи.

Окремим питанням стоїть вибір і вивчення засобів опису алгоритмів і програм. Хоча тут немає єдиної вимоги, але слід мати на увазі, що на III-IV етапах приймаються програми, написані лише на мовах Pascal та C. Тому, готуючись до олімпіад, не слід вивчати різні версії мови Basic, а також Visual Basic і Delphi. Можливості Basic недостатні для програмування олімпіадних задач, а Visual Basic та Delphi, навпаки, переобтяжені можливостями, що не використовуються в цих програмах.

Перехід від алгоритмічної до професійної мови не викличе труднощів у розумінні формування понять і навичок структурного програмування, чого не скажеш про Basic. Але дидактичний ефект буде значний, адже на початкових етапах досить непрості нюанси програмування в середовищах Pascal та C будуть відтягувати значні інтелектуальні зусилля учнів, що ускладнить процес навчання і приведе до суттєвої втрати часу.

Для проведення олімпіад з інформатики сьогодні необхідні тестуючі програми, причому їх застосування бажано запроваджувати на всіх етапах, починаючи від І-го (шкільного), вони принесуть велику користь і під час тренувальних турнірів. Тестуючу програму при крайній необхідності можна створити самому, запрограмувавши необхідну кількість функцій, потрібний рівень аналізу, зручну інструментальну оболонку та універсальність. Перш за все, чим зручніша й простіша оболонка, тим оперативнішою буде робота тестуючої програми. Що повинна мінімально "вміти" тестуюча програма? Універсальність забезпечать дві можливості: використання для читання даних і виводу результатів текстових файлів стандартного формату; запуск на виконання відкомпільованих програм (exe-файлів). Мінімально доцільний набір функцій повинен забезпечити випробування одного exe-файлу всіма тестами, генерацію відповідних файлів результатів. Порівняння одержаних результатів із контрольними можна робити вручну. Але на це піде багато часу і можливі помилки. Тому обов'язковою повинна бути функція порівняння результатів, переведення їх у бали та виводу в окремий файл. Якщо тестуюча програма "вміє" це робити, можна доповнити її додатковими функціями й можливостями. Дуже важливою функцією є забезпечення безперервної роботи тестуючої програми до повного аналізу всіх exe-файлів. Ця функція дасть можливість автоматично занести одержані результати до бази даних та визначити рейтинг учасників олімпіади. Але створити тестуючу програму, передбачити й подолати всі недоліки її роботи важко. Тестуюча програма не повинна допускати, щоб exe-файл аварійно перервав її роботу, вона мусить правильно реагувати на зациклення чи "зависання" тестованого файлу. Нарешті, важливою функцією тестуючої програми є реагування на час роботи exe-файлу, адже це повинно суттєво впливати на оцінку тестованої програми.

Але вищенаведені міркування спонукають до деяких важливих і цілком правомірних висновків:

1. тестуюча програма повинна бути досконалою й професійною;

2. У зв’язку з тим, що умови проведення Всеукраїнської олімпіади з інформатики повинні бути однакові на всіх турах і у всіх регіонах, бажано, щоб журі I-IV етапів мали можливість користуватись однаковими тестуючими програмами;
3. Принципи використання тестуючих програм повинні бути демократичними, вони повинні мати ліцензію і відкриті для користувачів.

На жаль через непрозорість і низьку ефективність єдиної політики в олімпійському русі з інформатики такі міркування при всій їх логічності доводиться віднести до розряду майже утопічних ідей.

ЛІТЕРАТУРА:

1. Програми для загальноосвітніх навчальних закладів. Навчальні програми для профільного навчання. Програми факультативів, спецкурсів, пропедевтичних курсів, гуртків. Інформатика, Запоріжжя, Прем'єр, 2003 р.
2. Програма курсу "Інформатика" для базової школи 7-9 класи (проект), Київ, Інформатика, № 11, 2003 р.
3. Г. Науменко, Навіщо нам формула Хартлі?, "Освіта", 25.05.1993р.
4. О.Б. Рудик, Динамічне програмування, Світло, №2, 1996 р.
5. О.Л. Хижа, Розв'язування задач підвищеної складності з інформатики, Київ, Інформатика, №№ 37-43, 1999 р.
6. Ж. Арсак, "Программирование игр и головоломок", Москва, "Наука", 1990р.
7. А. Г. Кушниренко та інші, "Основы информатики и вычислительной техники", Москва, "Просвещение", 1990р.
8. М.З. Грузман, Эвристика в информатике, Винница, Арбат, 1998 р.
9. Світло, № 1, 2001р., стор. 33.
10. Комп'ютер у школі та сім'ї, № 7. 2003 р., стор. 31.
11. В.В. Черняхівський, Збірник задач з основ алгоритмізації, Львів, Видавництво науково-технічної літератури,1997 р.
12. В.С. Савченко, Разработка алгоритмов: от простого к сложному, Донецк, 1996 р.
13. Комп'ютер у школі та сім'ї. 2000. № 3. стор.43
14. Т.П. Караванова. Основи алгоритмізації та програмування. 750 задач з рекомендаціями та прикладами - С. 6
15. С. Окулов, Основы программирования, Москва, Лаборатория Базовых Знаний, 2002 р.
16. С. Окулов, Программирование в алгоритмах, Москва, Лаборатория Базовых Знаний, 2002 р.
17. Комп'ютер у школі та сім'ї. 1999. № 3 – стор. 43
18. О.П. Кузнецов, Г.М. Адельсон-Вельский, Дискретная мавтематика для инженера, Москва, "Энергоатомиздат", 1988 г.
19. Динамическое программирование в примерах и задачах, Москва, Высшая школа, 1979 р.
20. С.А. Абрамов та ін., Задачи по программированию, Москва, Наука, 1988 р.
21. Каліхман, М.О. Войтенко, Динамическое программирование в примерах и задачах", Москва, Высшая школа , 1979 р.
22. Квант, № 10, Наука, Москва, 1991 р., стор 2-8.

ЗМІСТ