МЕТОД ДИНАМІЧНОГО ПРОГРАМУВАННЯ У ШКОЛІ

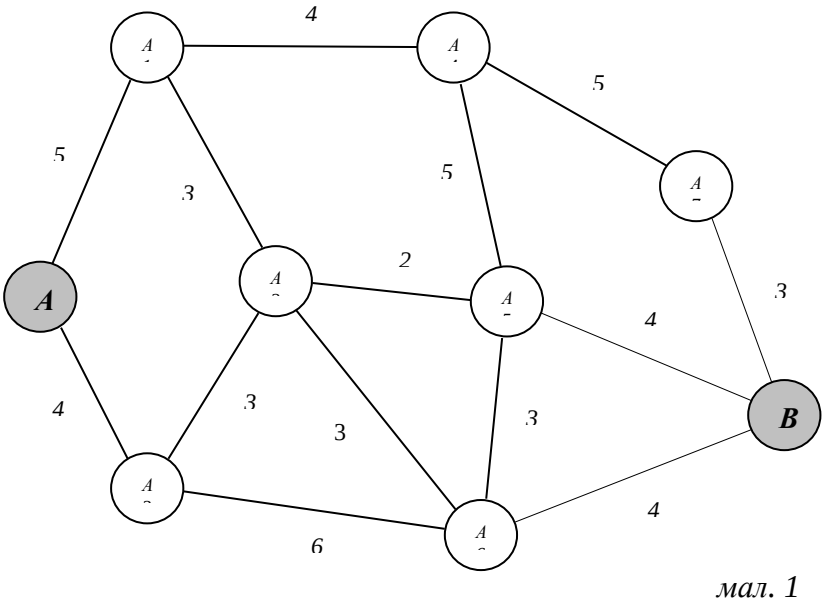
1. Ознайомлення із задачами та методом динамічного програмування.

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 і 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 крок розв'язання, інші кроки, повинні бути оптимальними по відношенню до стану, отриманому в результаті попереднього кроку розв'язання*, тобто, оптимальний розв'язок на *i*-му кроці визначається оптимальним розв'язком на (*i* - 1)-му кроці і "витратами" на перехід від (*i* - 1)-го кроку до *i* - го.

До задач оптимізації відносяться дуже багато 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 допоміжних задач, потім вони використовуються для відшукання розв'язків усе складніших допоміжних задач 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 XVII сторіччі (К.Маклорен) і навіть у древні часи (Архімед), він сформульований був порівняно недавно, у першій половині минулого століття американським математиком Р. Беллманом, а поширення одержав тільки з появою ЕОМ. Для учнів це складний метод програмування. Як зазначено в [4], годі й вимагати від них самостійно додуматись за кілька годин до того, що було останнім словом науки лише кілька десятиліть тому. Проте обійтись без методу динамічного програмування при розв'язу-

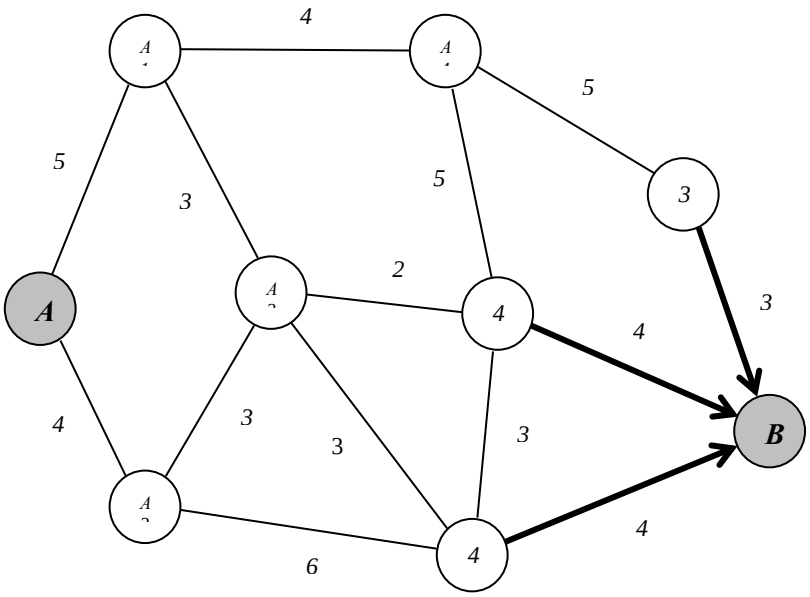


мал. 1

ванні олімпіадних задач часто буває неможливим, тому учителі шукають шляхи його викладання у школі. Можна ще раз використати думку з [4]: "…метод динамічного програмування для скінченного простору рішень справді досить простий, а його ідея зрозуміла навіть школярам з пересічною математичною підготовкою". Щоб сформувати уявлення про метод динамічного програмування, необхідно лише знайти підходящу для початкового ознайомлення задачу.

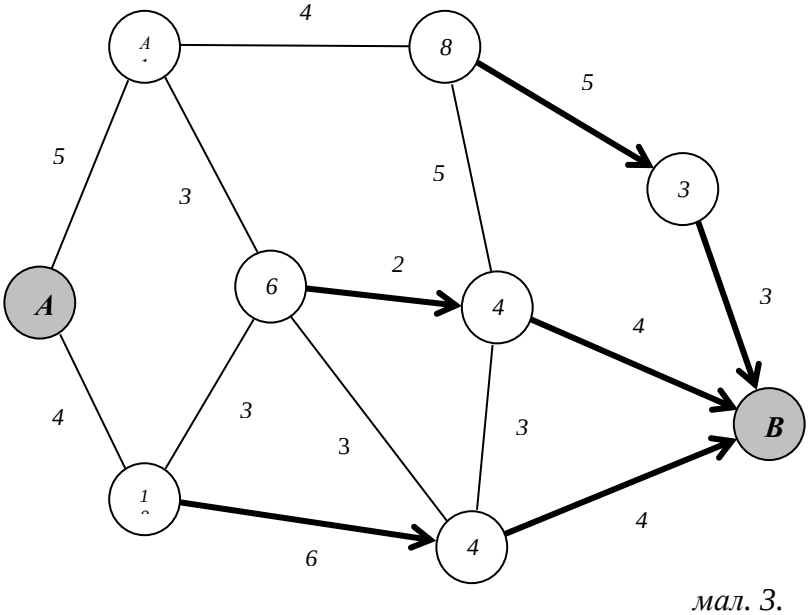
Задача 1. *На малюнку 1 зображено схему з початковим і кінцевим пунктами A та B і проміжними пунктами A_1…A_7, між якими вказані відстані. Необхідно знайти найкоротший маршрут з пункту A в пункт B.*

Це класична *задача на відшукання найкоротшого шляху*. Як легко зрозуміти, малюнок являє собою неорієнтований граф. Існує багато способів знаходження найкоротшогошляху, найпростішим з яких є повний перебір. Але, якщо кількість проміжних пунктів буде достатньо великою, необхідно скористатись більш оптимальним методом, який скоротить час пошуку найкоротшого маршруту, ми скористаємось методом динамічного програмування.



мал. 2

Розв'язання: Проаналізуємо, з яких пунктів можна потрапити в *B*. Це пункти A_5,A_6 та A_7. Поставимо стрілки з цих пунктів у пункт *B*, а в самих кружечках замість їх назв поставим відповідні числа (малюнок 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 *B* 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 вигляд, зображений на малюнку 3.



мал. 3.

Як показує наступний аналіз, в пункт A_4 можна потрапити тільки з пунк-

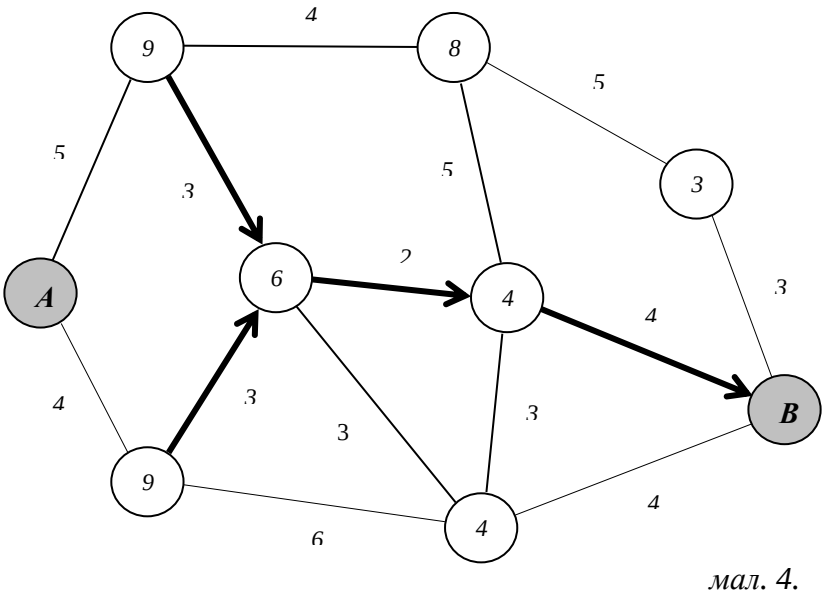
ту A_1, причому відстань буде 12, а в пункт A_2 можна потрапити з пунктів A_1 та A_3, причому відстані будуть однакові - по 9, тому ми у відповідних кружечках ставимо число 9 (зверніть увагу, що в пункті A_3 замінюємо число 10 на число 9), а також проводим стрілки з пунктів A_1 та A_3, відмінивши стрілку з пункту A_3 до пункту A_6, одержавши малюнок 4.

Отже відповідь така: з пункту A в пункт B найкоротший маршрут буде становити 13, що і відображено на останній схемі (малюнок 5).

Залишається помітити, що з пункту A коротший шлях буде в пункт A_3, а не A_1 та відобразити це на схемі.

Маршрут $A \rightarrow A_3 \rightarrow A_2 \rightarrow \rightarrow A_5 \rightarrow B$ наведемо подвійною лінією. (малюнок 5).

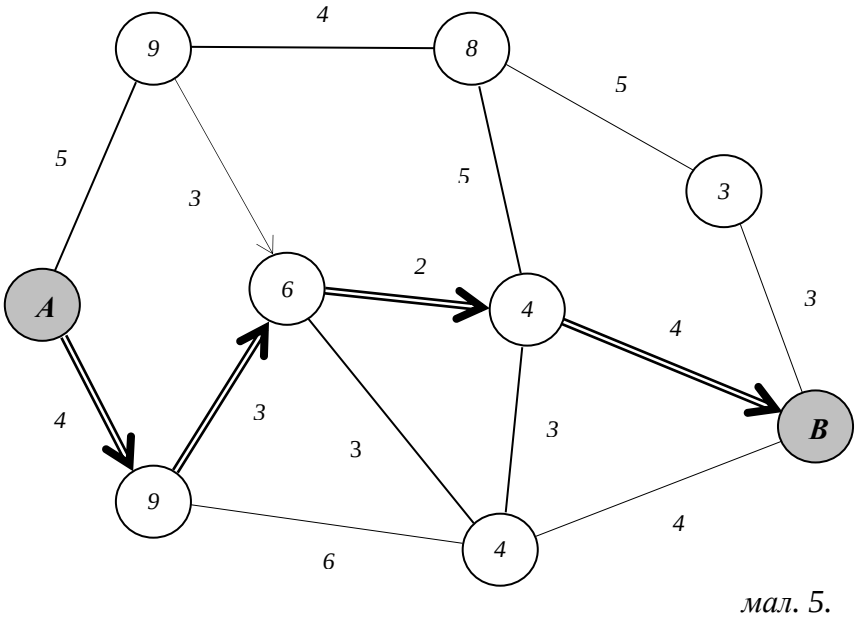
У ході розв'язання ми розглянули не всі можливі маршрути, а поступово відшукували єдиний і водночас найкоротший з них, на окремих етапах відкидаючи неоптимальні маршрути разом із їх можливими продовженнями.



мал. 4.

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

Роблячи зміни в початковій схемі (малюнок 1), ми мали можливість спостерігати оптимальні по відношенню до попередніх кроки і стани процесу пошуку найкоротшого маршруту, який відображено на малюнку 5. Цим продемонстровано принцип оптимальності. Але для складання програми пошуку найкоротшого маршруту необхідно продумати представлення даних і всіх змін у них в пам'яті комп'ютера, а також побудувати математичну модель задачі, яка містить рекурентні формули для пошуку оптимального рішення. Це буде виконано при розгляді наступних задач.



мал. 5.

2. Математична постановка задач динамічного програмування

Як зазначалось вище, 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 і достатньої мотивації.

Можна ще раз використати думку з [4]: "…метод динамічного програмування для скінченного простору рішень справді досить простий, а його ідея зрозуміла навіть школярам з пересічною математичною підготовкою".

Представлення даних задачі: Спочатку спробуємо це зробити до задачі 1. У квадратній таблиці $A[0,8]$ відведемо кожному пункту, в тому числі початковому і кінцевому, які можна позначити A_0 та A_8, один стовпчик і один рядок. Відстані між зв'язаними пунктами A_i та A_j запишемо у клітинку $A[i,j]$:

таблиця 1

i \ j	0	1	2	3	4	5	6	7	8
0	**0**	5	0	4	0	0	0	0	0
1	5	**0**	3	0	4	0	0	0	0
2	0	3	**0**	3	0	2	3	0	0
3	4	0	3	**0**	0	0	6	0	0
4	0	4	0	0	**0**	5	0	5	0
5	0	0	2	0	5	**0**	3	0	4
6	0	0	3	6	0	3	**0**	0	4
7	0	0	0	0	5	0	0	**0**	3
8	0	0	0	0	0	4	4	3	**0**

Якщо два деякі пункти A_i та A_j не зв'язані, у клітинку $A[i,j]$ запишемо 0, але це не означатиме, що відстань між цими пунктами рівна 0. Як видно з таблиці 1, дані розміщені симетрично відносно головної діагоналі (вона виділена напівжирним шрифтом), тому у таблиці можна прокласти два симетричні маршрути (зображені пунктирними ламаними стрілками). Це означатиме, що найкоротший маршрут між пунктами A та B проходить через пункти A_3, A_2 і A_5. Таке представлення даних здається найприроднішим. Але для пошуку оптимального розв'язку методом динамічного програмування потрібна інша форма представлення даних і результатів.

Слід зауважити, що для детального ознайомлення з методом динамічного програмування важливо знайти підходящу задачу. Зупинимось на подібній за змістом до № 1, але дещо іншій по формі, задачі. На наш погляд, для початківців ця задача найлегша, тому розглянемо її, не зважаючи на описи у різних варіантах у багатьох посібниках.

Задача 2. *У кожній клітинці прямокутного поля G, розміром m×n записане натуральне число. Між клітинками G[1,1] та G[m,n] можна прокладати шляхи у вигляді ламаних, щоб наступною до кожної клітинки G[i,j] була клітинка G[i+1,j]або G[i,j+1]. Визначити послідовність клітинок, на яких лежать вершини ламаної, що відображає найкоротший ступінчатий шлях від елемента G[1,1] до елемента G[n,m]. Найкоротшим будемо вважати ступінчатий шлях, що проходить через клітинки, сума елементів якого найменша.*

При розв'язуванні задач методом динамічного програмування корисно дотримуватись деяких стандартних прийомів. Перш за все слід передбачити дві робочі таблиці для вхідних даних та формування результату, надалі будемо позначати їх відповідно великими чи малими літерами *G* та *F*.

Отже, для задачі 2 введемо допоміжну таблицю *F* [1:*N*, 1:*M*] таку, що *F*[*i*, *j*] - *довжина найкоротшого шляху з клітки* [*i*, *j*] *у клітку* [*N*, *M*]. Тоді, мабуть, *L*[*N*, *M*] = *G*[*N*, *M*]. Очевидно, в останньому стовпчику можливе переміщення лише вниз, а в останньому рядку - лише вправо. Отже

$$F[i, M] = G[i, M] + F[i+1, M], \quad i = N-1, N-2, \ldots, 1. \qquad (1)$$

$$F[N, j] = G[N, j] + F[N, j+1], \quad j = M-1, M-2, \ldots, 1. \qquad (2)$$

Можна спочатку визначати перший стовпчик та перший рядок (пропонуємо написати відповідні співвідношення, подібні до (1) та (2) самостійно).
Заповнення таблиці *F* зробимо на конкретному прикладі. Нижче наведена таблиця *G* і заповнена, як описано вище, частина таблиці *F* (верхній рядок та правий стопчик):

таблиця G (1)

	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 F (1)

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

Після виконання такої підготовчої роботи, починаючи з лівого верхнього кута (самостійно попробуйте з правого нижнього кута), заповнимо решту клітинок таблиці *F*. Розглянемо незаповнені клітинки другого рядка. Щоб знайти елемент таблиці *F*[2, 2], треба до *G*[2, 2] додати менше з чисел *F*[1, 2] та *F*[2,1], тобто число 10. Тому *F*[2, 2] = 15. Подібним чином заповнимо і решту елементів другого рядка. Отримаємо таблицю *F*(2).

таблиця F (2)

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

таблиця F (3)

	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**

Після його заповнення легко подібним чином заповнити третій рядок таблиці *F*. В результаті одержимо таблицю *F*(3).

Наступний аналіз допоможе прийти до математичної моделі задачі. Три числа в останній таблиці *F* (4) виділено. Число 39 дорівнює найменшій сумі

елементів, тобто відповідає найкоротшому шляху. Числа $F[3, 1] = 27$ і $F[3, 3] = 27$ виділено для того, щоб вказати на ефективність методу динамічного програмування, адже пройшовши шлях $G[1, 1] \to G[2, 1] \to G[3, 1] \to G[3, 2] \to G[3, 3]$, наберемо не 27, а 39, що визначено найкоротшим маршрутом, а дійшовши до $F[3, 6]$, одержимо 51.

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

$$F[i, j] = G[i, j] + min\{F[i+1, j], F[i, j+1]\},$$
$$i = N - 1, N - 2, ..., 1, j = M - 1, M - 2, ..., 1. \quad (3)$$

Формула (3) 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  F[i, j+1]<F[i+1, j]
        то          F[i, j] :=G[i, j] + F[i, j+1]
        інакше   F[i, j] :=G[i, j] + F[i+1, j]
      все
   кц
кц
```

Для пошуку власне найкоротшого маршруту слід пройти зворотний шлях від $F[3, 6]$ до $F[1, 1]$, роблячи крок вліво чи вгору, вибираючи менше число (виділено пунктирною стрілкою). В остаточній таблиці F легко побачити шукану послідовність $F[1,1]$, $F[1,2]$, $F[3,2]$, $F[3,6]$. У зв'язку з обмеженим форматом статті та з метою уникнути зайвих подробиць, які відволікатимуть увагу від основної частини міркувань, при розгляді задач 1 і 2 не будемо наводити програми, адже крім описаних вище таблиць, які відображають вхідні дані та фрагменти алгоритмів, що виражають цільові функції, вони повинні містити фрагменти зв'язку з файлами вхідних та вихідних даних, запис результатів пошуку зворотних шляхів і визначення шуканої для задачі 2 послідовності. Складання та тестування програм розглянемо на наступних задачах.

таблиця F (4 - 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**

Але тут слід дати оцінку ефективності методу динамічного програмування. Він дозволяє розв'язати, наприклад, задачу 2 за $(N\text{-}1)\times(M\text{-}1)$ кроків, для таблиці 3×6 - всього за 10 кроків. Легко переконатись, що метод повного перебору вимагає $C_{N+M-2}^{N-1} = \frac{(N+M-2)!}{(N-1)!\times(M-1)!}$ умовних кроків. Для таблиці 3×6 - це $C_{3+6-2}^{3-1} = \frac{(3+6-2)!}{(3-1)!\times(6-1)!} = \frac{7!}{2!\times5!} = \frac{6\times7}{2} = \frac{42}{2} = 21$ крок, що значно більше. Якщо взяти вхідну таблицю більшої розмірності, то ця різниця може бути дуже суттєвою.

У якості альтернативи до задачі 2 можна взяти наступну задачу, яка пропонувалась на ІІІ-у етапі Всеукраїнської олімпіади з інформатики у Київській області.

Задача 3 **Pyramid**. *Відомо, що знамениті піраміди в Єгипті складаються з кам'яних кубів, покладених шарами. Недавно спритна туристська фірма, помітила кожен куб однієї з граней купленої нею піраміди числами, після чого на цій грані утворився числовий рівнобедрений трикутник (малюнок 6). Фірма встановила приз (верблюд) для туриста, який знайде найкоротший маршрут на поміченій числами грані піраміди від підніжжя до її вершини. Спочатку треба вибрати один з нижніх кубів, а потім на кожному кроці можна підніматись на один з двох суміжних із ним кубів. Написати програму, що визначає найкоротший маршрут до вершини піраміди. Найкоротший маршрут визначається найменшим значенням суми чисел на пройдених кубах.*

Технічні умови: Вхідний файл ***Pyramid.dat*** містить у кожному з n рядків ($1< n <100$) по i чисел $g[j]$, j=1.. i, де i – номер рядка, розділені пропусками, ($1< g[j] <100$);

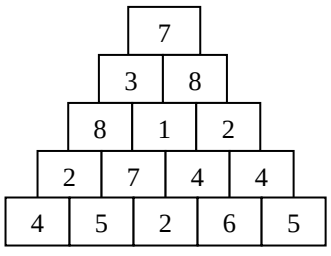
Вихідний файл ***Pyramid.res*** повинен містити в першому рядку значення суми, а в другому рядку через пропуск послідовність з n чисел, які розміщені на шляху знайденого маршруту.

Розглянемо ідею розв'язування на прикладі, наведеному в тексті задачі. Даний числовий трикутник найкраще представити у вигляді квадратної таблиці:

таблиця $g[1..n, 1..n]$

	1	*2*	*3*	*4*	*5*	.	*n*
1	7	0	0	0	0	.	.
2	3	8	0	0	0	.	.
3	8	1	2	0	0	.	.
4	2	7	4	4	0	.	.
5	4	5	2	6	5	.	.
.	.	.	.	.	.	.	.
n	.	.	.	.	.	.	.

Як видно, сам числовий трикутник розміщений *на* головній діагоналі таблиці та *під* нею. Розглянемо допоміжну таблицю $f[1..n,1..n]$. Зауважимо, що немає значення, звідки прокладати маршрут: від підніжжя до вершини, чи від вершини до підніжжя, тому шукатимемо маршрут від вершини до підніжжя. Очевидно, що можливим найкоротшим маршрутом буде перший стовпчик або головна діагональ таблиці $g[1..n]$, тому спочатку заповнимо їх згідно формул:



мал. 6

$f1,1]=g[1,1]$;

$f[i,1]=g[i,1]+f[i\text{-}1,1]$ для $i = 2..n$; $f[i, j]=g[i, j]+f[i\text{ -}1, j]$ для $j = 2..n$. (4)

Решту таблиці будемо заповнювати таким чином:

$f[i, j] = min(g[i, j]+ f[i-1, j], g[i, j]+f[i-1, j-1])$ для $i=3 .. n; j=i-1 .. n-1$ (5)

таблиця $f[1..n,1..n]$

	1	*2*	*3*	*4*	*5*	.	*n*
1	**7**					.	.
2	**10**	**15**				.	.
3	**18**	11	**17**			.	.
4	**20**	18	15	**21**		.	.
5	**24**	23	17	21	**26**	.	.
.	**.**	.	.	.	.	.	.
n	**.**	.	.	.	.	.	.

З (4) та (5) видно, що використано принцип динамічного програмування. У таблиці $f[1..n,1..n]$ результати виконання (4) виділено напівжирним шрифтом, а остаточний результат, елемент $f[5,3]$, виділено сірим кольором. Він визначається, як мінімум елементів останнього рядка таблиці $f[1..n,1..n]$. Щоб знайти саму послідовність чисел, які належать найекономнішому маршруту, слід пройти по таблиці $f[1..n,1..n]$, починаючи від $f[1,1]$ до $f[5,3]$, рухаючись вниз чи вправо і вибираючи на шляху напрям меншого елемента (маршрут виділено сірим кольором). Залишається лише вибрати елементи таблиці $g[1..n, 1..n]$, відповідні виділеному маршруту в таблиці $f[1..n,1..n]$. Це числа 7, 3, 1, 4, 2 (також виділено сірим кольором). Згідно умови маршрут визначається не вказаним, а таким набором: 2,4,1,3,7, тому для вибору маршруту краще рухатись не від $f[1,1]$ до $f[5,3]$, а від $f[5,3]$ до $f[1,1]$.

Якби була поставлена задача знайти маршрут, що визначає найбільшу суму, то таблиця $f[1..n,1..n]$ мала б такий вигляд:

таблиця $f_1[1..n,1..n]$

	1	*2*	*3*	*4*	*5*	.	*N*
1	**7**					.	.
2	**10**	**15**				.	.
3	**18**	16	**17**			.	.
4	**20**	25	21	**21**		.	.
5	**24**	30	27	27	**26**	.	.
.	**.**	.	.	.	.	.	.
n	**.**	.	.	.	.	.	.

Для цього рівність (5) довелось би замінити рівністю:

$f[i, j] = max(g[i, j]+f[i-1, j], g[i, j]+f[i-1, j-1])$ для $i=3 .. n; j=i-1 .. n-1$ (6)

Відшукання набору чисел для найдовшого маршруту (5, 7, 8, 3, 7) ілюструється таблицями $g[1..n, 1..n]$ та $f_1[1..n,1..n]$ з допомогою подібних міркувань, які були застосовані для вибору найкоротшого маршруту, але з урахуванням формули (6).

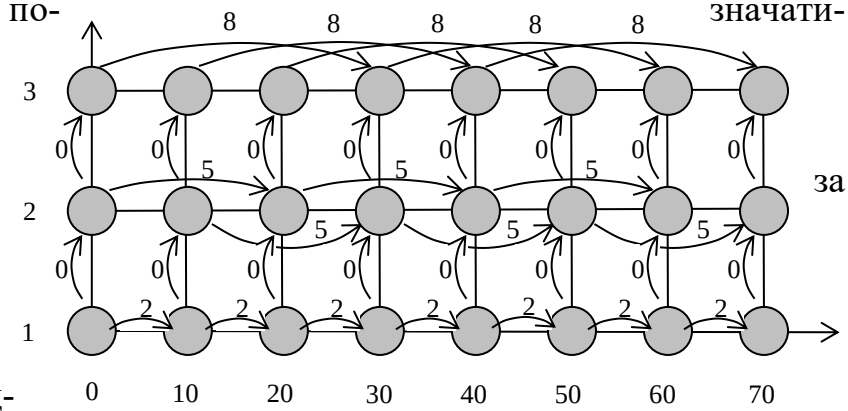
Задача 4. *Людина може підняти рюкзак, який вміщає до 70 кг предметів трьох видів, вага і ціна яких наведені в таблиці 2. Які предмети і в якій кількості слід покласти в рюкзак, щоб загальна ціна цих предметів була найбільшою і рюкзак могла підняти людина?*

таблиця 2

види предметів	Вага одного предмета, (кг)	ціна одного предмета, (грн)
I	10	2
II	20	5
III	30	8

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

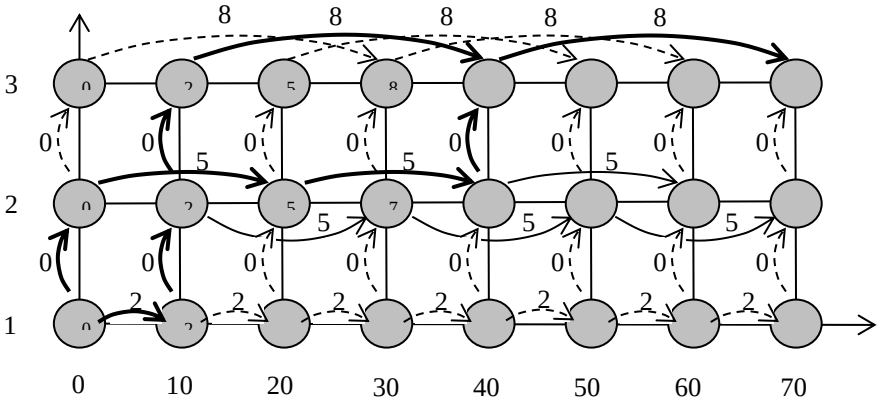
По горизонталі будемо відкладати вагу в кілограмах.Тут у нас є запас 70 одиниць. По вертикалі позначимо цифрами 1,2 та 3 різні види предметів *К*. Вибір кожного предмета позначатимемо стрілкою вправо, яка вказує на кружечок з координатою *р*(вага), на стільки одиниць більшою за координату початку стрілки, скільки важить вибраний предмет. Вибір кожного предмета відмічатимемо в тому рядку, який відповідає даному предмету. На кожній стрілці відмічатимемо ціну взятого предмета.



мал. 6

Крім горизонтальних стрілок, які означають вибір відповідного предмета, на малюнку 6 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).



мал. 7

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 почат-

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

Розглядаючи задачу 4, ми також скористались схемою, що в математиці називається графом. Її використання унаочнило умову задачі та хід її розв'язування. Ми не будемо тут показувати представлення даних та результатів у вигляді, придатному для обробки на комп'ютері, не записуватимемо алгоритм і програму. Пропонуємо самостійні вправи:

Вправа 1. *Перевірити по малюнку 7 отриману відповідь.*

Вправа 2. *Розв'язати задачу 4, змінивши обмеження 70 кг на 50 кг.*

Вправа 3. *Скориставшись поясненням до задачі 2, представити дані задачі 4 у вигляді таблиць. Скласти рекурентну формулу, яка описує цільову функцію до цієї задачі та відповідний їй фрагмент алгоритму.*

3. Ступінь складності задач динамічного програмування

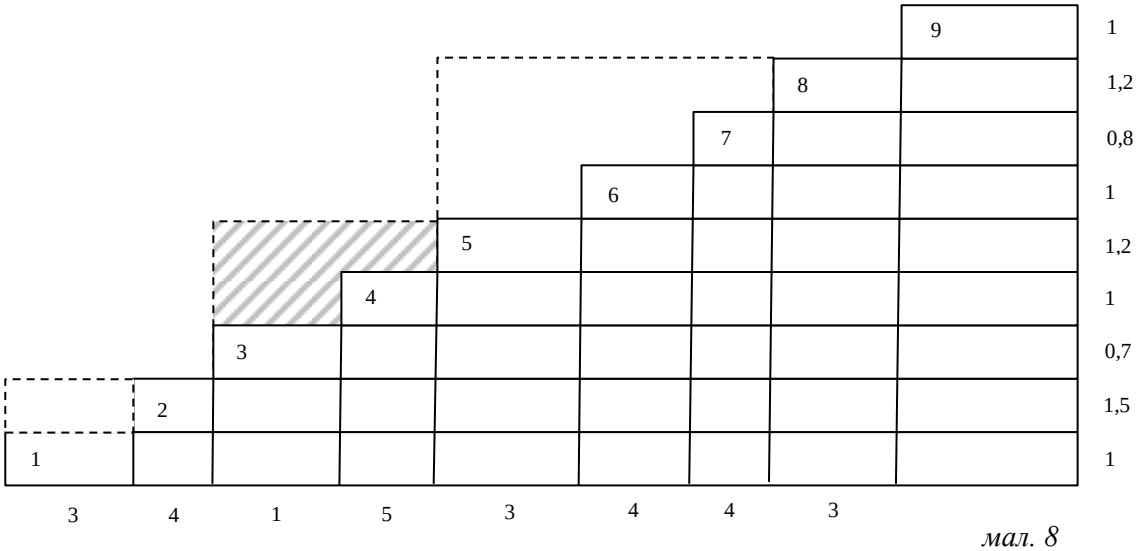
Розглянуті вище задачі порівняно нескладні, на них зручно було прослідкувати початкові етапи. Але для розв'язання задачі з допомогою комп'ютера завжди необхідна програма, що стає суто технічною проблемою при правильному виконанні математичної постановки задачі. В методі динамічного програмування найважчою ланкою є опис цільової функції, яка реалізує принцип динамічного програмування. Це не легко зробити навіть розглянувши серію готових розв'язків, адже у кожній задачі потрібні унікальні міркування. Щоб до них наблизитись, розглянемо складнішу задачу.

***Задача 5*.** **Ladder** *На малюнку 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 $дм^3$ будівельного матеріалу), 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. 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



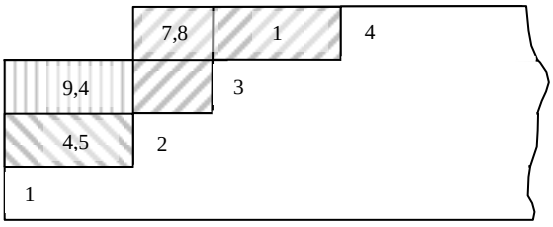
мал. 8

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

таблиця 3

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. Подібно до розглядуваної задачі, далі будемо визначальну та допоміжну таблиці позначати, як прийнято у використаній літературі, $f(i,j)$ та $g(i,j)$. Це пов'язано з тим, що елементи цих таблиць містять значення цільової функції та відповідають загальноприйнятому в математиці позначенні функцій і запису формул. Очевидно, що $f(1,j)$, де $1<j\le 9$ 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.



мал. 9

$\boldsymbol{f(1,1) = 0}$, адже нарощувати тут нічого не треба (всі $\boldsymbol{f(i,i) = 0}$);

$f(1,2) = \boldsymbol{f(1,1) + g(1,2)} = 0+1{,}5\cdot 3 = 4{,}5$;

$f(1,3) = \boldsymbol{f(1,2) + g(1,3)} = 1{,}5\cdot 3+0{,}7\cdot(3+4) = 9{,}4$;

$f(1,4) = \boldsymbol{f(1,3) + g(1,4)} = 9{,}4+1\cdot(3+4+1) = 17{,}4$;

$f(1,5) = \boldsymbol{f(1,4) + g(1,5)} = 17{,}4+1{,}2\cdot(3+4+1+5) = 33$;

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

~~$f(1,7)$~~$=\boldsymbol{f(1,6)+g(1,7)}=49+0{,}8\cdot(3+4+1+5+3+4)=65$;

~~$f(1,8)$~~$=\boldsymbol{f(1,7)+g(1,8)}=65+1{,}2\cdot(3+4+1+5+3+4+4)=93{,}8$;

~~$f(1,9)$~~$=\boldsymbol{f(1,8)+g(1,9)}=93{,}8+1\cdot(3+4+1+5+3+4+3)=120{,}8$;

Закреслені значення $f(i,j)$, хоч і обчислені, але не потрібні, вони суперечать умові, далі обчислювати їх не будемо).

$f(2,3) = \boldsymbol{min(g(2,3),f(1,2)} = min(\underline{\mathbf{2,8}};4{,}5) = 2{,}8$;

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), тобто:

$f(2,4) = \boldsymbol{min\{g(2,4), f(1,2)+g(3,4), f(1,3)\}} = min\{7{,}8;\underline{\mathbf{4,5+1}};9{,}4\} = 5{,}5$;

$f(2,5) = \boldsymbol{min\{g(2,5), f(1,2)+g(3,5), f(1,3)+g(4,5),f(1,4)\}} = min\{19{,}8;\ \underline{\mathbf{4,5+8,2}};\ 9{,}4 + +6;\ 17{,}4\} = 12{,}7$;

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

$= min\{32{,}8;4{,}5+17{,}2;9{,}4+14;\underline{\mathbf{17,4+3}};33\} = 20{,}4$;

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

$f(2,8)$ та $f(2,9)$ не обчислюємо, які, подібно до $f(1,7)$, $f(1,8)$ та $f(1,9)$, не потрібні.

Далі обмежимось словесним поясненням і формульними записами для $f(3,4)$ та $f(3,5)$. Подивимось, де може закінчуватися 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

$f(3,4) = \boldsymbol{min\{g(3,4), f(2,3)\}} = min\{\underline{\mathbf{1}};2{,}8\} = 1$;

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

$f(3,5) = \boldsymbol{min\{g(3,5), f(2,3)+g(4,5), f(2,4)\}} = min\{8{,}2;2{,}8+6;\underline{\mathbf{5,5}}\} = 5{,}5$;

Далі одержуємо:

$f(3,6) = \boldsymbol{min\{g(3,6), f(2,3)+g(4,6), f(2,4)+g(5,6), f(2,5)\}} =$

$= min\{17{,}2;2{,}8+14;\underline{\mathbf{5,5+3}};12{,}7\} = 8{,}5$;

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

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

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

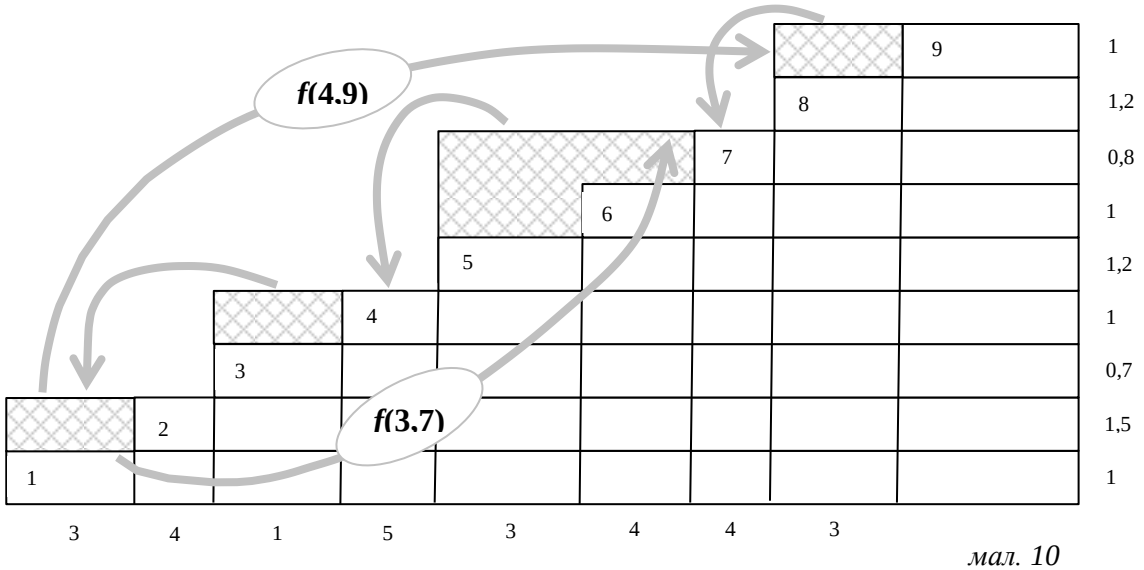
$f(3,9)$ не обчислюємо. Пропустивши $f(4,j)$, $4\le j\le 8$, знаходитимемо $f(4,9)$ як найменшу із сум $f(3,j) + g(j,9)$, де j - одне із шести чисел 4, ..., 8, тобто:

$f(4,9)= \boldsymbol{min\{g(4,9), f(3,4)+g(5,9), f(3,5)+g(6,9), f(3,6)+g(7,9),f(3,7)+g(8,9), f(3,8)\}} = min\{61{,}8;1+35{,}8;5{,}5+23{,}8;8{,}5+11{,}8;\underline{\mathbf{14,1+3}};25{,}2\} = 17{,}1$.

Якщо число 17,1 помножити на 10, то одержимо найменшу кількість будівельного матеріалу в кубічних дециметрах.

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 *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 10.

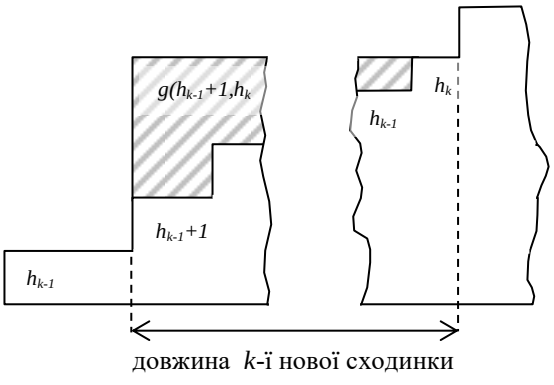


мал. 10

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 $h_1, h_2, \ldots, h_k$, при яких сума k величин

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

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



мал. 11

$$\boldsymbol{f_n(h_n) = min\{g((h_0+1,h_1) + g((h_1+1,h_2) +\ldots+g((h_{k-2}+1,\ h_{k-1}) + g((h_{k-1}+1,\ h_k)\}} \qquad (7)$$

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, але для меншої на одиницю кількості сходинок. У результаті одержуємо формулу Р. Беллмана:

$$\boldsymbol{f_k(h_k) = min\{ f_{k-1}(h_{k-1}) + g((h_{k-1}+1, h_k) \}}, \qquad (8)$$

яка дозволяє послідовно знайти числа $f_k(h_k)$ (при всіх $k \le n$, $h_k \le h_n$) на кожному кроці. 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 $h_{n-1}, h_{n-2}, \ldots, h_1$ (див стрілки на малюнку 10).

Звернемо увагу на те, що в описаному методі не накладається ніяких умов на вид функцій g. Суттєво лише, що функцію f, мінімум якої ми знаходимо, вдається представити у вигляді суми членів, в якій попередній залежить від меншої кількості змінних.

Порівняємо одержані рівності (3), (5) – (8).

У (3) присутня функція визначення мінімального значення з двох значень таблиці F: $G[i, j] + min\{F[i+1, j], F[i, j+1]$.

Формули (5) та (6) містять дещо складніші функції визначення мінімуму та максимуму:

$$min(g[i, j]+ f[i-1, j], g[i, j]+f[i-1, j-1]),$$
$$max(g[i, j]+f[i-1, j], g[i, j]+f[i-1, j-1])$$

Ускладнення, так би мовити, на цілий порядок прослідковується в наступних рівностях (7) та (8):

$$min\{g((h_0+1,h_1) + g((h_1+1,h_2) + \ldots + g((h_{k-2}+1, h_{k-1}) + g((h_{k-1}+1, h_k)\},$$
$$min\{ f_{k-1}(h_{k-1}) + g((h_{k-1}+1, h_k) \},$$

Виходить, що різні задачі на той же метод динамічного програмування тільки в загальних рисах можна об'єднати, складність їх може дуже відрізнятись, а пошуки і обґрунтування цільової функції для кожної задачі – це унікальне дослідження і доведення.

І це, безперечно, повинно накладати умови на застосування таких задач у шкільних олімпіадах з інформатики.

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

Задача 6 **Figura (***9-10 класи***).** *У квадратному масиві цілих чисел $A[n \times n]$ $(2<n<20)$ елементи, (малюнок 12), 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*.

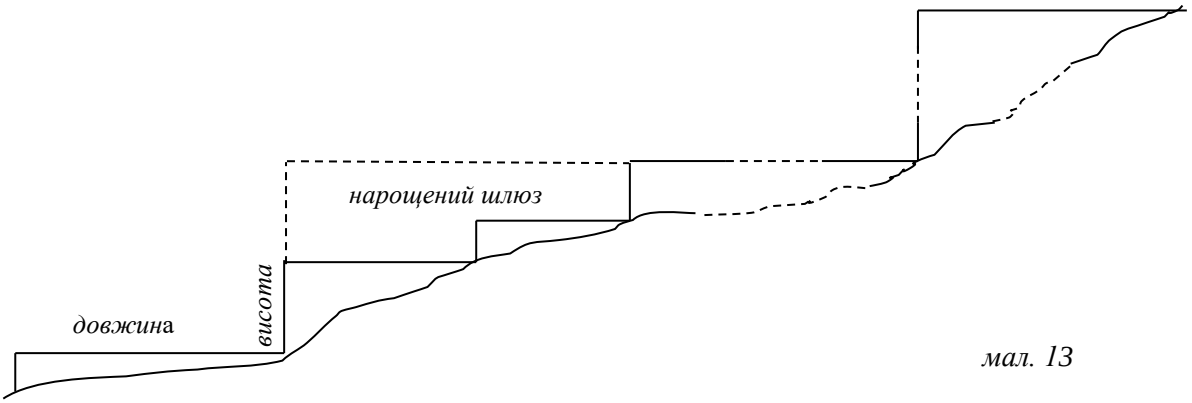
Технічні умови: Вхідний файл *Figura.dat* містить в першому рядку числа n ти m – початкову та кінцеву кількість ступенів, наступні n рядків, що містять по n чисел кожен – елементи масива $A[n \times n]$.

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.

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

мал. 12

Задача 7 **Riv (***11 клас*). *На малюнку 13 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, наростивши стінки окремих із них так, щоб об'єднати кілька шлюзів, після чого їх стало m(2<m<n<10), а будівельного матеріалу було використано найменше. 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*.



мал. 13

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.

Задачі 6-7 споріднені із задачею 5 про сходинки, тому наводити міркування про їх розв'язування немає потреби, це можна запропонувати читачам.

Задача 8 **EnterprisTurn** ***(****пропонувалась на обласному етапі олімпіади 2007 р. у Київській області) Бажаючі отримати квитки на концерт Мадонни вишикувалися у чергу із N - чоловік, кожен із яких має намір викупити лише 1 квиток. Оскільки працювала лише одна каса, продаж квитків відбувався повільно, а люди у черзі приходили у відчай. Однак,*

найкмітливіші дійшли до висновку, що декілька квитків в одні руки касирка продає набагато швидше, ніж коли ті ж квитки продаються поодинці. Тому вони запропонували тим, хто стоїть поруч, віддавати гроші першому з них, щоб той викупив квитки на всіх. Для боротьби зі спекулянтами касирка продавала не більше, ніж по 3 квитки в одні руки. Таким чином домовитися між собою мали можливість лише 2 або 3 людини. Скласти програму, яка визначатиме найменший час, за який касирка зможе продати квитки усім бажаючим. Відомо, що на продаж і-ій людині із черги одного квитка касирка витрачає A_i секунд, на продаж двох квитків - B_i секунд, а трьох квитків - C_i секунд. Зверніть увагу, що квитки на групу завжди купує перший із них, а зайвих квитків не купує ніхто.

Формат вхідних даних: Перший рядок містить єдине число N - кількість чоловік ($1 \le N \le 5000$), що утворюють чергу. Нумерація людей у черзі починається від каси. Наступні N-рядків містять трійки натуральних чисел A_i, B_i, C_i, значення кожного з яких не перевищує 3600.

Формат вихідних даних: Рядок містить одне число - найменший час, за який касирка зможе продати квитки усім бажаючим.

Як і дві попередні, ця задача також асоціюється із задачею про сходинки (**Ladder**), але обмеження на кількість квитків, що відпускаються в одні руки, робить її більш доступною для учнів. Для них важливо відчути ланцюжок: *математична модель – цільова функція – програма – тести*. Спробуємо прослідкувати його, скориставшись програмою і деякими з тестів.

```
program EnterprisTurn;
 var i,n:integer;g:array[1..3,1..5000] of integer;f:array[1..5000] of longint;
procedure Inp;
begin
  Assign(input,'t_8.tst');Reset(input); ReadLn(input,n); i:=1;
  while i<=n do                                                    begin
    Read(input,g[1,i]);Read(input,g[2,i]);Read(input,g[3,i]); i:=i+1 end;
   Close(input)
end;
procedure Outp;
begin
   Assign(output,'t_.ans');ReWrite(output);Write(output, f[n]); Close(output)
end;
function Min(x,y:longint):longint;
begin  if x>y then Min:=y else Min:=x; end;
procedure TimePurchase;
begin
   f[1]:=g[1,1]; f[2]:=Min(f[1]+g[1,2], g[2,1]);
   f[3]:=Min(f[2]+g[1,3], Min(f[1]+g[2,2], g[3,1]));
   for i:=4 to n do
    f[i]:=Min(f[i-1]+g[1,i], Min(f[i-2]+g[2,i-1], f[i-3]+g[3,i-2]))
```

```
end;
begin
  Inp; TimePurchase; Outp;
end.
```

j/i	*1*	*2*	*3*
1	**5**	**10**	**15**
2	**2**	**10**	**15**
3	**5**	**5**	**5**
4	**20**	**20**	**1**
5	**20**	**1**	**1**

таблиця g
(для тесту e t_1.tst)

таблиця f
(для тесту et_1.tst)

i	*f[i]*			
1	*g[1,1]*	*5*	*5*	*5*
2	*Min(f[1]+g[1,2], g[2,1])*	*Min(5+2, 10)*	*7*	*7*
3	*Min(f[2]+g[1,3], Min(f[1]+g[2,2], g[3,1]))*	*Min(7+5, Min(5+10, 15))*	*Min(12,Min(15,15))*	*12*
4	*Min(f[i-1]+g[1,i],Min(f[i-2]+g[2,i-1], f[i-3]+g[3,i-2]))*	*Min(12+20,Min(7+5,7+15))*	*Min(32,Min(12,22))*	*12*
5	*Min(f[i-1]+g[1,i],Min(f[i-2]+g[2,i-1], f[i-3]+g[3,i-2]))*	*Min(12+20,Min(12+20,7+5))*	*Min(32,Min(3,12))*	***12***

j/i	*1*	*2*	*3*
1	**1**	**10**	**10**
2	**2**	**3**	**4**
3	**2**	**3**	**4**
4	**2**	**3**	**4**

таблиця g
(для тесту t_2.tst)

таблиця f
(для тесту t_2.tst)

i	*f[i]*			
1	*g[1,1]*	*1*	*1*	*1*
2	*Min(f[1]+g[1,2], g[2,1])*	*Min(1+3, 10)*	*4*	*4*
3	*Min(f[2]+g[1,3], Min(f[1]+g[2,2], g[3,1]))*	*Min(4+2, Min(1+3, 10))*	*Min(6,Min(4,10))*	*4*
4	*Min(f[i-1]+g[1,i],Min(f[i-2]+g[2,i-1], f[i-3]+g[3,i-2]))*	*Min(4+2,Min(4+3,1+4))*	*Min(6,Min(7,5))*	***5***

j/i	*1*	*2*	*3*
1	**1**	**2**	**3**
2	**1**	**1**	**3**
3	**1**	**2**	**3**
4	**1**	**2**	**1**
5	**1**	**2**	**3**
6	**1**	**2**	**3**

таблиця g
(для тесту t_3.tst)

таблиця f
(для тесту et_3.tst)

i	*f[i]*			
1	*g[1,1]*	*1*	*1*	*1*
2	*Min(f[1]+g[1,2], g[2,1])*	*Min(1+1,2)*	*2*	*2*
3	*Min(f[2]+g[1,3], Min(f[1]+g[2,2], g[3,1]))*	*Min(2+1, Min(1+1, 3))*	*Min(3,Min(2,3))*	*2*
4	*Min(f[i-1]+g[1,i],Min(f[i-2]+g[2,i-1], f[i-3]+g[3,i-2]))*	*Min(2+1,Min(2+2,1+3))*	*Min(3,Min(4,4))*	*3*
5	*Min(f[i-1]+g[1,i],Min(f[i-2]+g[2,i-1], f[i-3]+g[3,i-2]))*	*Min(3+1,Min(2+2,2+3))*	*Min(4,Min(4,5))*	*4*
6	*Min(f[i-1]+g[1,i],Min(f[i-2]+g[2,i-1], f[i-3]+g[3,i-2]))*	*Min(4+1,Min(3+2,2+1))*	*Min(5,Min(5,3))*	***3***

Вище акцентувалось, що для розв'язування задачі методом динамічного програмування не достатньо навіть дуже чіткого уявлення про принцип оптимальності та цільову функцію. Основні труднощі виникають на етапі формулювання рекурентних співвідношень, які виражають цільову функцію. Вони пов'язані з особливостями умови задачі та формою представлення вхідних даних і шуканого результату. Формула, що виражає цільову функцію – це свого роду функціональне рівняння, складаючи яке, ми, іншими словами, описуємо його розв'язок, що для учнів із пересічними математичними здібностями та під-

готовкою нелегко. Щоб частково спростити цю проблему, ми, по-перше, у всіх розглянутих задачах однаково позначали таблиці: для зберігання вхідних даних – *G* або *g*, для формування оптимального рішення *F* або *f*. Застосування елементів таблиці *f*, наприклад***, f[i]****:=Min(****f[i-1]****+g[1,i], Min(****f[i-2]****+g[2,i-1], ****f[i-3]****+g[3,i-2]))* у програмах має ще й інше значення, це – функціональне рівняння, залежне від констант *g*(*i*), рекурентних звернень *f*(*i*-1), *f*(*i*-2) і т.д. у поєднанні з функціями *Max* або *Min*. де права частина рівняння виражає розв'язок рівняння. По-друге, для глибшого усвідомлення методу динамічного програмування, ми радимо користуватись готовою програмою до задачі динамічного програмування і ручним покроковим виконанням окремих тестів до цієї програми, що ми навели вище на прикладі задачі 8. При уважному спостереженні відповідних таблиць та процедури *TimePurchase* стане очевиднішим і зрозумілішим вигляд рекурентних співвідношень, а набутий досвід допоможе у розв'язуванні наступних задач на метод динамічного програмування та складанні тестів до них.

На закінчення наведемо програми до задач 5 (*Ladder*) та 6 (*Figura*) з деякими тестами.

```
program Ladder;
  type rl = array[0..50] of real;
  var i, j, u, t, l, n, m : integer; d1, d2 : array[0..50] of real;
       g : array[0..50, 0..50] of real; way1, way2 : array[0..50] of byte;t1 : real;
function Min(n:integer;b:rl):real;{Визначення мін. елемента масиву і його номера}
  var i:integer;m:real;
begin
   m:=b[1];t:=1;
   for i:=2 to n do
    if b[i]<m then begin m:=b[i];t:=i end;
   Min:=m;
end;
procedure Inp; {Читання даних з вхідного файлу}
begin
   Assign(input, 'ladder.dat');Reset(input); Read(input, n); ReadLn(input, m);
   for i:=1 to n-1 do Read(input,d1[i]);
   i:=1;ReadLn(input);
   while not EoLn(input) do begin
     Read(input,d2[i]);inc(i)end; Close(input);
end;
procedure FillG; {Генерація таблиці G}
   var s1,s2:real;
begin
   s1:=0; s2:=0; u:=0;
   for i:=1 to (n-1) do
    for j:=i{+1} to n do                                                    begin
      for u:=i to j-1 do begin s1:=s1+d1[u]; s2:=s2+d2[u+1]; end;
      g[i,j]:=s1*s2; s1:=s1-d1[i];
     for u:=i+1 to j-1 do begin g[i,j]:=g[i,j]-s1*d2[u]; s1:=s1-d1[u]; end;
```

```
  s1:=0; s2:=0       end;
end;
function MakeF(i,j:integer):real; {Обчислення елементу таблиці F}
  var u,p:integer; a:rl; x,y,gx,gy:array[0..50] of byte;
begin
  if i = 1 then MakeF:=g[1,j]
  else                                                          begin
   for u:=0 to 10 do begin a[u]:=0;x[u]:=0;y[u]:=0;gx[u]:=0;gy[u]:=0  end;
  u:=0; p:=2;
  for u:=0 to (j-i-2) do                  begin
    a[u+2]:=MakeF(i-1,i+u)+g[i+1+u,j]; x[p]:=i-1; y[p]:=i+u;
   gx[p]:=i+1+u; gy[p]:=j; inc(p) end;
  a[1]:=g[i,j]; x[1]:=i; y[1]:=j;  gx[1]:=i; gy[1]:=j; inc(p);
  if (j-i-2) = 0 then a[3]:=MakeF(i-1,j-1)
   else if (j-i-2) = -1 then a[2]:=MakeF(i-1,j-1)
        else a[u+3]:=MakeF(i-1,j-1);
  x[p-1]:=i-1; y[p-1]:=j-1;
  if (j-i-2) = 0 then MakeF:=min(3,a)
 else if (j-i-2) = -1 then MakeF:=min(2,a)
      else MakeF:=min(u+3,a);{Визнач. номерів сходинок, які треба наростити}
  if ((i=m)and(j=n))or((i=way1[l-1])and(j=way2[l-1])) then begin
    if gx[t] <> 0 then Writeln(output, gx[t], ' - ', gy[t]);
    way1[l]:=x[t]; way2[l]:=y[t]; inc(l);
    if((n-m)<>1)and((way1[l-1]<>way1[l-2])and(way2[l-1]<>way2[l-2]))then
   t1:=MakeF(way1[l-1],way2[l-1])                  end end;
end;
begin
  Inp;FillG;l:=1;
  Assign(output, 'ladder_M.res');Rewrite(output);
  if m = 1 then writeln(output, m, ' - ', n); t1:=MakeF(m,n);
  if way1[l]=0 then if way1[l-1]=1 then WriteLn(output, '1 - ', way2[l-1])
    else if way1[l]=1 then WriteLn(output, '1 - ', way2[l]);
  WriteLn(output, t1:4:2); Close(output);
end.
```

Тести до задачі *Ladder*:

№ тесту	*Ladder.dat*	*Ladder.res*
1	10 5 2 1 1 1.5 1.5 2.5 1.5 1 0.5 0.5 1.5 2 1 1.5 0.5 1 1 1.5 1.5	9 - 10 7 - 8 5 - 6 2 - 4 7.00
2	9 4 3 4 1 5 3 4 4 3 1 1.5 0.7 1 1.2 1 0.8 1.2 1	8 - 9 5 - 7 3 - 4 1 - 2 17.10
3	5 3 1.5 2.5 2 1	4 - 5 2 - 3

	1 1 0.5 1.5 0.5	1.75
4	5 3 3 1 1 1 3 1 1 1 3	2 - 4 3.00
5	15 12 5 5 5 1 5 5 5 5 1 1 5 5 5 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1	9 - 11 4 - 5 4.00
6	15 11 5 5 5 1 5 5 5 5 1 1 5 5 5 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1	14 - 15 9 - 11 4 - 5 9.00
7	15 10 1 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	14 - 15 12 - 13 10 - 11 8 - 9 6 - 7 5.00
8	15 1 1 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 - 15 157.00
9	15 2 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	6 - 15 1 - 5 71.00
10	15 3 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	9 - 15 3 - 8 1 - 2 41.00

```
program Figura;
  type q=array [1..30,1..30] of real; q1=array [1..30,1..30,1..4] of integer;
       x1=array [1..30] of real;
 var i,i1,k,j,n,m,i2,z:integer;o,f1:text;k1,k2,res:real;f,g,a1,b1,c1:q;h:q1;    x,y:x1;
function Min(r,r1:real):real;
begin
   if r<r1 then  min:=r  else  min:=r1;
end;
procedure Tabl;
 begin
   for i:=1 to n do                                    begin
    for j:=1 to n do                                 begin
      b1[i,i]:=0;
     for z:=i to j-1 do  b1[i,j]:=b1[i,j]+a1[z,j]  end   end;
 for i:=1 to n do
   for j:=1 to n do                                   begin
     c1[1,2]:=b1[1,2]; c1[i,j]:=c1[i,j-1]+b1[i,j] end;
end;
begin
  Assign(o,'figura.dat'); reset(o); read(o,n); readln(o,m);
   for i:=1 to n do
    for j:=1 to n do  read(o,a1[i,j]); close(o);
 tabl;
 for i:=1 to n do
```

```
      for j:=1 to n do  g[i,j]:=c1[i,j];
  for i:=2 to n do                                                    begin
     f[1,i]:=g[1,i]; h[1,i,3]:=1; h[1,i,4]:=i  end;
  for i:=2 to m do
    for i1:=i+1 to n do  begin
       f[i,i1]:=g[i,i1]; k2:=f[i,i1]; h[i,i1,3]:=i; h[i,i1,4]:=i1;
      for k:=i to i1-1 do                                                 begin
        f[i,i1]:=min(f[i,i1],f[i-1,k]+g[k+1,i1]);
        if k2<>f[i,i1] then                                          begin
          k2:=f[i,i1]; h[i,i1,2]:=k; h[i,i1,3]:=k+1; h[i,i1,4]:=i1  end   end   end;
  res:=f[m,n];  i1:=n;
  assign(f1,'figura.res'); rewrite(f1); writeln(f1,res:0:2);
  for i:=m downto 1 do    begin
    if h[i,i1,3]<>h[i,i1,4] then          begin
       writeln(f1,h[i,i1,3],'-',h[i,i1,4]) end;
    if h[i,i1,2]=0 then  i:=1
   else i1:=h[i,i1,2];
     i2:=i2+1                end;
  close(f1);
end.
```

Тести до задачі *Figura*:

№ тесту	*Figura.dat*	*Figura.res*
1	5 3 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	5.00 4-5 1-2
2	10 1 5 1 1 1 1 1 1 1 1 1 5 5 1 1 1 1 1 1 1 1 5 5 5 1 1 1 1 1 1 1 5 5 5 5 1 1 1 1 1 1 5 5 5 5 5 1 1 1 1 1 5 5 5 5 5 5 1 1 1 1 5 5 5 5 5 5 5 1 1 1 5 5 5 5 5 5 5 5 1 1 5 5 5 5 5 5 5 5 5 1 5 5 5 5 5 5 5 5 5 5	45.00 1-10
3	10 3 5 1 1 1 1 1 1 1 1 1 5 5 2 2 2 2 2 2 2 2 5 5 5 1 1 1 1 1 1 1 5 5 5 5 1 1 1 1 1 1 5 5 5 5 5 1 1 1 1 1 5 5 5 5 5 5 1 1 1 1 5 5 5 5 5 5 5 1 1 1 5 5 5 5 5 5 5 5 1 1 5 5 5 5 5 5 5 5 5 1 5 5 5 5 5 5 5 5 5 5	13.00 7-10 3-6 1-2
4	10 9 5 1 1 1 1 1 1 1 1 1 5 5 2 2 2 2 2 2 2 2	1.00 9-10

	5 5 5 1 1 1 1 1 1 1 5 5 5 5 1 1 1 1 1 1 5 5 5 5 5 1 1 1 1 1 5 5 5 5 5 5 1 1 1 1 5 5 5 5 5 5 5 1 1 1 5 5 5 5 5 5 5 5 1 1 5 5 5 5 5 5 5 5 5 1 5 5 5 5 5 5 5 5 5 5	
5	10 5 5 0 0 0 0 0 0 0 0 0 5 5 0 0 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 5 5 5 5 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 5 5 5 0 0 0 0 5 5 5 5 5 5 5 0 0 0 5 5 5 5 5 5 5 5 0 0 5 5 5 5 5 5 5 5 5 0 5 5 5 5 5 5 5 5 5 5	0.00 5-10
6	15 5 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 5 0 0 0 0 0 0 0 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 0 0 0 0 5 5 5 5 0 0 0 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 0 0 0 0 0 5 5 5 5 5 5 0 0 0 0 0 0 0 0 0 5 5 5 5 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 5 5 5 0 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 5 0 0 0 5 5 5 5 5 5 5 5 5 5 5 5 5 0 0 5 5 5 5 5 5 5 5 5 5 5 5 5 5 0	0.00 5-15

ЛІТЕРАТУРА:

1. І.Л. Каліхман, М.О. Войтенко, “Динамическое программирование в примерах и задачах”, Москва, ” Высшая школа” , 1979 р.”;
2. Л.М. Вивальнюк та інші “Задачі оптимізації (посібник для факультативних занять у 10-11 класах”; Київ,”Радянська школа”, 1991 р.;
3. ”Квант”, № 10, “Наука”, Москва, 1991 р., стор 2-8:
4. ”Світло”, № 2, “Поліграфіст”, Фастів,1996 р., стор. 3-41;
5. ”Інформатика”, № 1, ”Віпол”, Київ, 1999 р., стор. 6
6. В.С. Савченко “Разработка алгоритмов: от простого к сложному ”, Донецьк, 1996 р., Головне управління освіти Донецької обласної держадміністрації, Донецький обласний інститут післядипломної освіти