Бориспільський районний методичний кабінет
Щасливський навчально-виховний комплекс

В.С. ОСТАПЕЦЬ

Алгоритмічні підходи до розв'язування задач "довгої арифметики"

2004 рік

Остапець В.С.,
вчитель-методист Щасливського НВК,
Бориспільський район

I. 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: $1 \le N$, $K \le 200000$ [4], а це ще далеко не "довга арифметика".

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 [2].

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 *a*) наведемо дві задачі.

Задача **NUM**. 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*.

Як видно з умови, накладеної на вхідний файл, число *N* може бути багаторозрядним. Але застосування багаторозрядної арифметики тут не

потрібне, що стане зрозуміло в ході опису математичної моделі. Для відшукання алгоритму спочатку покладемо n = 6 (тобто N = 111111) і помітимо, що тоді число N представляється згідно умови задачі у вигляді:

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

Як видно з (II.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:

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

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 (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).

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

$$a_1 a_2 a_3 ... a_{n-2} a_{n-1} a_n \times a_n = a_n a_1 a_2 a_3 ... a_{n-2} a_{n-1} \qquad \text{(II.3)}$$

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 II.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 II.1, всі числа N, що заловольняють (II.3) 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 a - 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, властивість (3.3) 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 II.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: $a_{n-1} = \mathrm{mod}\,(a_n \times a_n, 10) = \mathrm{mod}\,(4 \times 4, 10) = 6$, а також цифра, що при цьому "помічається":

$\mathrm{div}\,(a_n \times a_n, 10) = \mathrm{div}\,(4 \times 4, 10) = 1$(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 у розділі III 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 Ректора 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 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 $K \le M$, $N \le K+100$, $1 \le K \le 10^{100}$.

II. 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 [1]. Ідея представлення багаторозрядних чисел у вигляді лінійного цілочисельного масиву реалізована в чудовій книзі С.М. Окулова "Программирование в алгоритмах" [1], 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 [1] 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 $\text{int}(\text{sqrt}(10^{250}))\approx 10^{125}$, 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 $1<n<500000$ [3]. 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, *st*2=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}
+                                   (1)
   29  {s2}
  264  {  s}
```

З (1) 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 (2).

$$\begin{array}{r} 0\quad 1\quad 0\ \{pm\} \\ 2\quad 3\quad 5\ \{s1\} \\ - \\ \underline{(2+1)\ \ 9\ \{s2\}} \\ 2\quad 0\quad 6\ \{s1\} \end{array} \qquad (2)$$

В розглянутих нижче конкретних прикладах (таблиця 1) 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 1

	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. (2)). 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 2:

таблиця 2

	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 (1), 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 (1) 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}
```
(3)

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 **LongMoDiv**, 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*

(4)

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 (5), 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
IV - й етап 3070
- 2406
V - й етап 6641
- 6015
626

(5)

Процедура **LongMoDiv**:

```
procedure LongMoDiv(st1,st2:string;var stmod,stdiv:string);
 var i,l,k, j:integer;st_1:string;bZero: boolean;
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;{-------------------------------------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 (4) в кінець stdiv нулі. Цикл {18}-{19} та рядок {20} визначають чергову цифру *stdiv*.

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.

ЛІТЕРАТУРА:

1. С. Окулов. Программирование в алгоритмах. Москва. БИНОМ. Лаборатория знаний. 2002 г.
2. Т.П. Караванова. Основи алгоритмізації та програмування. 750 задач з рекомендаціями та прикладами - С. 6
3. В. Бондаренко, С. Жук. Задачі XII Всеукраїнської олімпіади з інформатики та обчислювальної техніки. Комп'ютеру школі та сім'ї. 1999. № 3 - с. 43
4. П.В. Аксьонов, В.В. Бондаренко, С.В. Діденко, Ш.І. Ягіяєв. Задачі XIII Всеукраїнської олімпіади з інформатики та обчислювальної техніки та рекомендації щодо їхнього розв'язання. Комп'ютеру школі та сім'ї. 2000. № 3 - с. 43