УДК 378.14

Броницька Н. А., Дармосюк В. М., Бережецька В.
Миколаївський національний університет
імені В. О. Сухомлинського

ДОВГА АРИФМЕТИКА В КОМБІНАТОРНИХ ЗАДАЧАХ

В статті розглядаються питання застосуванням алгоритмів довгої арифметики до комбінаторних задач. Досліджується необхідність вивчення таких алгоритмів для програміста-початківця. Аналізуються різні алгоритми та доцільність їх використання. Наводяться приклади конкретних алгоритмів довгої арифметики.

Ключові слова: *комбінаторика, довга арифметика, комбінаторні методи, алгоритми, програмування, мови програмування, типи даних.*

Комбінаторика – розділ математики, присвячений розв'язанню задач вибору та розміщення елементів деякої, зазвичай, скінченної множини відповідно до заданих правил. Кожне таке правило визначає спосіб побудови деякої конструкції із елементів початкової множини, що зветься комбінаторною конфігурацією. Характерними властивостями цих об'єктів є те, що вони відповідають деяким обмеженням щодо них. На меті комбінаторного аналізу стоїть дослідження комбінаторних конфігурацій, алгоритмів їх побудови, оптимізація таких алгоритмів, а також розв'язання задач переліку.

Комбінаторика має широке практичне застосування. Її методами користуються хіміки при вивченні різних можливих типів зв'язків атомів у молекулах; біологи, наприклад, у процесі знаходження послідовностей амінокислот у білкових сполуках; кібернетики при розв'язанні задач кодування й побудові обчислювальних пристроїв, математики – при розв'язанні задач, особливо в теорії ймовірності. Також комбінаторику використовують у своїх моделях фізики, архітектори, економісти й представники багатьох інших наук. Цим зумовлене підвищення інтересу до комбінаторики в наш час. Цей напрямок активно розвивається і у програмуванні.

Досить часто комбінаторні обчислення передбачають досить громіздкі розрахунки, що зумовило інтерес до цього розділу саме в програмуванні, оскільки за допомогою комп'ютера можна швидко і безпомилково отримати відповідь. Але в таких розрахунках виникають наскільки великі результати, що вони не можуть бути записані в жоден числовий тип даних для їх подальшого опрацювання. В зв'язку з цим виникає необхідність вивчення та використання такого розділу програмування як "Довга арифметика", який дає змогу працювати з надвеликими числами без обмежень.

При навчанні програмістів теми "Комбінаторика" та "Довга арифметика" розглядаються поряд і навіть паралельно, оскільки вони нерозривно пов'язані.

Ми хотіли б детально розглянути питання довгої арифметики, тому що її програмування охоплює великий пласт знань з різних галузей, дозволяє розвивати логічне мислення, удосконалювати вміння застосування структур даних для найбільш оптимальної роботи алгоритмів.

Мови програмування мають вбудовані типи даних, розмір яких, в основному, не перевищує 64 біта (близько 10^{19}). Десяткова довга арифметика була реалізована в мові програмування АЛМІР-65 на ЕОМ МИР-1 і в мові програмування АНАЛІТИК на ЕОМ МИР-2. Для роботи з великими числами, однак, існує досить багато готових оптимізованих бібліотек для довгої арифметики. Вбудовані бібліотеки роботи з великими числами є в *Python і Java*. Однак, зауважимо, що для інших мов завжди є можливість написати ці бібліотеки самостійно. Особливо це рекомендується робити програмістам-початківцям, оскільки самостійне написання функцій довгого множення, сумування і т. д. систематизує і упорядковує знання про алгоритмізацію та програмування, про структури

даних, про представлення чисел, розвиває логіку. Так само основні поняття роботи з довгими числами допомагають при вирішенні питань переведення чисел в різні системи числення.

Комбінаторика спонукає програміста-початківця максимально оптимізувати обчислення (застосування і написання формул), використовувати максимально відповідні типи даних і стежити за можливістю їх переповнення і при необхідності переходити від стандартних типів до використання алгоритмів довгої арифметики.

Існує безліч алгоритмів і методів організації таких обчислень.

Починаючи розглядати довгі числа, першою проблемою є зберігання таких чисел в пам'яті комп'ютера. Природно, що їх можна розміщувати або як рядок типу string, або як масив. Рядок, як і масив, в сучасних мовах програмування обмежується лише розміром оперативної пам'яті комп'ютера. Для кращого розуміння основ програмування, розгляду різних алгоритмічних структур та структур даних, а також для глибшого розуміння цих понять, порівняння часу виконання різних алгоритмів та задля розвитку логічного мислення доцільним є розгляд всіх варіантів такого розміщення та виконання арифметичних дій при різних способах.

Всі приклади будемо надавати на мові програмування С++, оскільки вона є однією з найбільш поширених серед програмістів на сьогоднішній день і надає широкі можливості для порівняння застосування різних структур даних та методів. Крім того практично всі сучасні мови програмування є С-подібними, тобто засвоївши саме цю мову, легко потім здійснити перехід до інших мов.

Основна ідея класичних алгоритмів полягає в тому, що число зберігається у вигляді масиву його цифр або груп цифр.

Цифри можуть використовуватися з тієї чи іншої системи числення, зазвичай застосовуються десяткова система числення і її степені (десять, сто, тисяча і т.д.), або двійкова система числення.

Операції над числами в цьому виді довгої арифметики проводяться за допомогою "шкільних" алгоритмів додавання, віднімання, множення, ділення стовпчиком. Втім, до них також застосовні алгоритми швидкого множення: наприклад, швидке перетворення Фур'є і алгоритм Карацуби.

Для прикладу розглянемо спосіб зберігання довгого числа по цифрам і опишемо для нього алгоритм додавання. Для вивчення більш широкого кола питань мови програмування С++ і застосування бібліотеки STL, розглянемо розміщення числа у векторі (vector), оскільки вектор надає більше можливостей його опрацювання, ніж масив.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
// функція зчитування числа у вектор
void read(vector<int> &a) {
char c;
for(;;){
cin.get(c);
if ((c==' ')||(c=='\n'))break; // ознака закінчення потоку зчитування
a.push_back(c-'0'); // (c-'0') – знаходження цифри із символу
}
}
```

/* У результаті виконання функції зчитування задане число буде поцифрово записано у вектор у зворотному порядку */

```
// функція виведення вмісту вектора на екран
void write(vector<int> a) {
for (size_t i=0; i< a.size();i++) cout << a[i];
```

```
}
/* функція перевірки довжин заданих чисел, запису довшого числа у перший вектор і доповнення коротшого вектора нулями з початку до довжини другого */
void obmin(vector<int> &a, vector<int> &b) {
if (a.size()==b.size()) return;
if (a.size()>b.size()) swap(a,b);
for (size_t i=0; i< a.size()-b.size(); i++) {
a.insert(a.begin(),0);
}
}
// функція додавання двох довгих чисел
vector<int> sum(vector<int> a, vector<int> b) {
vector<int> c;
short p=0;
obmin(a,b);
c.assign(a.size(),0);
for (short i = a.size()-1; i >= 0; i--) {
c[i]=a[i]+b[i]+p;
p=0;
if (c[i]>9) {c[i]-=10; p=1;}
}
if (p) c.insert(c.begin(),1);
return c;
}
// головна функція
int main()
{
vector<int> a,b,c,rez;
read(a);
read(b);
write(sum(a,b));
}
```

У комбінаторних задачах зазвичай робота проводиться з цілими невід'ємними довгими числами. Для підтримки від'ємних чисел необхідно ввести і підтримувати додаткову змінну-прапор "заперечності" числа, або ж працювати в доповнюючих кодах.

Підвищення ефективності можна досягти роботою в системі по підстановці різних степенів десятки, тобто кожен елемент масиву містить не одну, а відразу кілька (зазвичай до 9) цифр. Назвемо кількість цифр у групи *основою групи*.

При такому підході змінюється розташування числа у пам'яті комп'ютера та, відповідно, його зчитування і виведення, які мають свої особливості. Для уніфікації програми можна внести директиву препроцесора #define osn 9, що дасть змогу надалі використовувати різні основи. Для такої форми представлення довгого числа функції введення/виведення великих чисел можна записати наступним чином:

```
//функція введення довгого числа по групам цифр
void read(vector<int> &a){
string s;
cin>>s;
for (int i=s.length(); i>0; i-=osn)
if (i<osn){
string t=s.substr(0,i);
int g=0,p=1;
for(int k=t.length()-1;k>=0;k--)
{g+=(t[k]-'0')*p; p*=10; }
a.push_back (g);
}
else {
string t=s.substr(i-osn, osn);
```

```
int g=0,p=1;
for(int k=osn-1;k>=0;k--)
{g+=(t[k]-'0')*p; p*=10; }
a.push_back(g);
}
}
/* функція виведення довгих чисел з перетворенням кожної групи в задану основою кількість цифр */
void output(vector<int> &k){
printf ("%d", k.empty()? 0 : k.back());
for (int i=(int)k.size()-2; i>=0; --i)
cout<<setw(osn)<<setfill('0')<<k[i];
}
```

Обробка даних чисел відрізнятиметься лише процесом пошуку цілої частини від ділення: у груповому методі цифра переносу визначатиметься діленням на 10^{osn}.

Широке застосування алгоритми довгої арифметики знаходять саме під час розв'язування комбінаторних задач.

Наприклад, наступна задача вимагає великої кількості непростих обчислень, які вручну виконувати вкрай незручно і важко, причому при програмуванні таких розрахунків виникають дуже великі числа, які вимагають застосування швидких алгоритмів довгої арифметики.

Нехай задано натуральні числа n, m, s, причому $m \le s \le n$. Якщо можливими значеннями дискретної випадкової величини є 0, 1, 2, … m, а відповідні їм ймовірності виражаються за формулою $p_k = P(X = k) = \frac{C_m^k \cdot C_{n-m}^{s-k}}{C_n^s}$, $k = 0,1,2,...,m$, то говорять, що випадкова величина X має гіпергеометричний розподіл. Цей закон розподілу є важливим і застосовується в задачах статистичного контролю якості та в суміжних галузях.

Потрібно записати закон розподілу випадкової величини X – кількість бракованих деталей серед взятих s деталей із партії з *n* деталей, якщо відомо, що в партії всього m бракованих деталей.

Як бачимо, в цій задачі застосовується комбінаторна формула обчислення комбінацій. Для обчислення потрібно неодноразове її застосування і в процесі обчислень виникнуть довгі числа, до яких ми і будемо застосовувати алгоритми довгої арифметики за наведеним зразком або можна застосовувати їх більш швидкі аналоги, про які мова йтиме в наступних статтях даної тематики.

Таким чином вивчення алгоритмів довгої арифметики дає можливість глибоко і предметно навчати майбутніх програмістів логіці зберігання чисел і роботи з ними; коректній роботі з елементами масиву і контролю ймовірності виходу за його межі; роботі з нетривіальними структурами мов програмування. Такий підхід закладає фундамент для розвитку впевнених навичок програмування. А також робота з довгими числами дозволяє розв'язувати великий клас комбінаторних задач і спрощувати їх математичні розрахунки.

Використана література:

1. Алгоритмы на C++ (олимпиадный подход): длинная арифметика. – [Электронный ресурс]. – Режим доступа : http://cppalgo.blogspot.com/2010/05/blog-post.html
2. Википедия – Свободная энциклопедия: длинная арифметика. – [Электронный ресурс]. – Режим доступа. – URL: http://ru.wikipedia.org/wiki/Длинная_арифметика
3. Школа программистов: длинная арифметика – [Электронный ресурс]. – Режим доступа : http://acmp.ru/article.asp?id_text=1329
4. MAXimal: длинная арифметика. – [Электронный ресурс]. – Режим доступа : http://e-maxx.ru/algo /big_integer
5. *Гольдштейн В.* Длинная арифметика (лекция). – [Электронный ресурс]. – Режим доступа : http://informatics.mccme.ru/moodle/mod/resource/ view.php?id=449
6. Хабрахабр: Алгоритм Карацубы для умножения двух чисел. – [Электронный ресурс]. – Режим

доступа : http://habrahabr.ru/post/121950/

7. *Рейнгольд Э.* Комбинаторные алгоритмы. Теория и практика / Э. Рейнгольд, Ю. Нивергельт, Н. Део. – М. : Мир, 1980. – 476 с.

Броницкая Н. А., Дармосюк В. Н., Бережецкая В. Длинная арифметика в комбинаторных задачах.

В статье рассматриваются вопросы применения алгоритмов длинной арифметики в комбинаторных задачах. Исследуется необходимость изучения таких алгоритмов для начинающего программиста. Анализируются различные алгоритмы и целесообразность их использования. Приводятся примеры конкретных алгоритмов длинной арифметики.

***Ключевые слова**: комбинаторика, длинная арифметика, комбинаторные методы, алгоритмы, программирование, языки программирования, типы данных.*

Bronitska N. A., Darmosyuk V. M., Berezhetska V. Long arithmetic in combinatorial problems.

In this paper use of the questions long arithmetic algorithms to combinatorial problems. We investigate the need to study these algorithms for beginner programmers. Analyzed various algorithms and the feasibility of their use. Examples of specific algorithms for long arithmetic.

***Keywords**: combinatorics, long arithmetic, combinatorial methods, algorithms, programming, programming language, data types.*

УДК 378.371:53

Василенко С. Л.
Національний педагогічний університет імені М. П. Драгоманова

РОЛЬ СЕМІНАРСЬКИХ ЗАНЯТЬ У ПРОЦЕСІ НАБУТТЯ СТУДЕНТАМИ ІНТЕЛЕКТУАЛЬНОГО І ПРАКТИЧНОГО ДОСВІДУ ДІЯЛЬНОСТІ

У статті проаналізовано процес проведення семінарського заняття з курсу загальної фізики у його функціональних елементах та цілісності. Визначено оптимальні педагогічні дії, спрямовані на реалізацію навчальних цілей. Показано, що семінарські заняття сприяють як становленню інтелектуального и практичного досвіду діяльності студентів, так і розв'язанню проблеми оптимізації навчального процесу.

***Ключові слова**: семінарське заняття, інтелектуальний і практичний досвід діяльності, педагогічне моделювання навчального процесу.*

Важливою рисою сучасного освітнього процесу є те, що студент одержує величезну кількість інформації з певної дисципліни не лише безпосередньо у процесі її вивчення в обсязі, який визначений навчальною програмою, але й з безлічі різних джерел. Проте для того, щоб засвоєння цієї інформації здійснювалось успішно і забезпечувало майбутніх фахівців необхідною сумою знань, вона повинна бути певним чином класифікована і систематизована. При цьому викладач може прослідкувати лише за тією часткою навчальної інформації, яку він безпосередньо виклав у процесі різних видів аудиторних занять. Очевидно, що забезпечити становлення системи знань у галузі тієї чи іншої науки вищий навчальний заклад не в змозі. Що ж стосується інформації, одержаної студентом в рамках даної дисципліни, але ззовні, то він має освоїти її самостійно. І саме у процесі самостійної діяльності у студента закладаються основи навчальної праці, формуються внутрішні передумови, мотивація, відношення до пізнавальних потреб і, що головне – прагнення до самоосвіти і психологічна настанова на майбутню професійну діяльність. Очевидно, що самостійна діяльність студента буде продуктивною лише за умови його постійної психологічної готовності до розв'язання навчальних проблем, активності, творчого відношення до поставлених завдань. На нашу думку, формування у студентів