UZUN SONLI ARIFMETIKA ALGORITMLARINI C++ DASTURLASH TILIDA YOZISH

Mahmudov Muhamadxon Nuralixon o‘g‘li
Andijon davlat universiteti, Axborot texnologiyalari va kompyuter injiniringi fakulteti, Axborot texnologiyalari kafedrasi o‘qituvchisi

Annotatsiya: *Ba’zi berilgan olimpiada masalalari dasturlash tili taqdim etgan butun sonli o‘zgaruvchilarning maksimal qiymatigan katta sonlarni talab etadi, aynan shunday sonlar bilan ishlash imkonini taqdim etuvchi qo‘shish, ayrish, ko‘paytirish, bo‘lish, taqqoslash amallarini foydalanish yoritilgan.*

Kalit so‘zlar: *C++, matematik ustunlik amallar, vector, uzun son, qisqa son.*

Barchamiz maktabda ikkita ko‘p xonali sonlarni qo‘shish, ayrib, ko‘paytirib chiqanmiz. Misol uchun $6541654 + 46421654 = ?$ natijasi nechchi ekanligini topaylik. Buni qo‘shishni biladigan odam ham hech qayerga yozmasdan uning javobi 52963308 (52 963 308) ekanligini aytishi mumkin. Oddiy kalkulyator yordamida ham hisoblash mumkin. Biz bolalikda maktabda o‘qiganimizda bunday sonlarni boshqacha usulda ya’ni , ustunli hisoblash usuli bilan o‘rgatishgan. Ustunli hisoblash usuli o‘zi nima ?

```
  46421654
+  6541654
----------
  52963308
```

(1-rasm)¶

Bu – birinchi eng katta sonni yozib uning tagidan keyingi sonni o‘ng tartiblagan xolda raqama-raqam yozishda iborat. Bunday usulni foydasi hisoblashda sonni raqamlarga ajratilgan holda hisoblab chiqamiz (1 - rasm). $4 + 4 = 8$. Bu rasmdagi misol ikki sonni qo‘shish edi. Bu sonlarni ko‘payritish esa ancha murakkab, ustunli ko‘paytirishda birinchi sonni ikkinchi son raqamiga ko‘paytirib chiqiladi, xar bir ko‘paytirilgan raqamning pozitsiyasidan boshlab yoziladi. Misol uchun $11990987 * 11190957$ ko‘paytirib ko‘ring. Uning ustunli usulda hisoblashdagi natijasi 2 - rasmda berilgan. Bo‘lish esa eng qiyin amallardan biridir. Bo‘lishga misol 3 – rasmda berilgan.

$$
\begin{array}{r}
11990987 \\
* \quad 11190957 \\
\hline
83936909 \\
+ \quad 59954935 \\
+ \quad 107918883 \\
+ \quad 00000000 \\
+ \quad 107918883 \\
+ \quad 11990987 \\
+ \quad 11990987 \\
+ \quad 11990987 \\
= \quad 134190619904559
\end{array}
$$

(2 - rasm)

$$
\begin{array}{r|l}
9042019 & 80 \\
- \quad 9042000 & 113025 \\
\hline
= \quad 19 &
\end{array}
$$

(3 - rasm)

Bizni maktabda o‘rgatgan matematik usullar dasturlashda nima aloqasi bor ? Ko‘p xonali sonlarni ustun shaklda dasturda yozib hisoblab chiqamizmi ? Dasturda ustunli ko‘rinishda ma’lumotlarni yozib bo‘lmaydiku? Bu kabi savollar qiynayotgan bo‘lsa kerak. Bu ustunli ko‘rinishda hisoblash biz uchun qulay hisoblanadi va siz bunday ko‘rinishdagi ma’lumot siz uchun tanish. Chunki siz bularni maktabda o‘rgangansiz. Yangi ma’lumotlarni eski ma’lumotlarga tayangan holda o‘rganish aslo zarar qilmaydi. Demak biz ko‘p xonali butun sonlarni qanday usulda hisoblashni bilib oldik, endi bunday usuldagi hisoblashni C++ tilida qanday yozamiz. Ko‘p xonali sonning raqamlarini saqlash uchun **vector** tipidan (inglizcha template container) foydalanamiz. Vektor tipi xususiyati shundan iboratki u bir tomonlama o‘suvchi massivdir. Biz sonlar ustida amal bajarganimizda necha xonali bo‘lib ketishini bilmaymiz, shu sababli vektor konteyneridan foydalanganmizmiz maqul. Shuningdek qaysi ma’lumot tipini ham ifodalashni belgilan olamiz. C++ tilida 4 ta butun sonlar tiplarining xarakteristikasini ko‘rib chiqamiz ularning qisqartirilgan shaklidan foydalanganmiz.

Nomi	Bayt	Eng katta son	xonlar (digits)	bo‘luvchisi (devisor)
uint8_t	1	$2^8 - 1$	2	100
uint16_t	2	$2^{16} - 1$	4	10 000

uint32_t	3	$2^{32}-1$	9	1 000 000 000
uint64_t	4	$2^{64}-1$	9	1 000 000 000

(1- jadval)

Ko‘p xonali butun sonlar bilan ishlaganda ma’lumotlarni saqlashda qaysi tipdan foydalanish katta ahamiyatga ega. Shuning uchun biz turli xil parameter orqali (1 - jadval) dagi ma’lumotlar tipidan foydalanib turli xil testlarni o‘tkazdik.

Uzun sonli arifmetika - bu standart ma'lumotlar turlaridan ancha katta raqamlar bilan ishlash imkonini beruvchi dasturiy vositalar (ma'lumotlar tuzilmalari va algoritmlari) to‘plami. Butun sonli uzun arifmetikaning turlari Umuman olganda, faqat olimpiada masalalarida ham berilgan sonlar to‘plami etarlicha katta, shuning uchun biz uzun sonli arifmetikaning har xil turlarini tasniflaymiz.

Asosiy g‘oya shundan iboratki, raqam uning raqamlari qatori sifatida saqlanadi. U yoki bu sanoq sistemasidan raqamlardan foydalanish mumkin, odatda o‘nlik sanoq sistemasi va uning darajalari (o‘n ming, milliard) yoki ikkilik sanoq sistemasi qo‘llaniladi. Bunday uzun arifmetikada raqamlar ustida amallar qo‘shish, ayirish, ko‘paytirish, ustunga bo‘lish uchun “maktab” algoritmlari yordamida bajariladi. Biroq, ular uchun tez ko‘paytirish algoritmlari ham qo‘llaniladi.

Bu faqat manfiy bo‘lmagan uzun raqamlar bilan moslashgan. Manfiy raqamlarni qo‘llab-quvvatlash uchun raqamning "manfiyligi" ning qo‘shimcha bayrog‘ini kiritish va qo‘llab-quvvatlash yoki boshqa kodlarda ishlash kerak.

$(a > b, a > 0, b > 0)$

Umumiy qoidalar

Ushbu algoritmlar **Python** dasturlash tilida yozish talab etilmaydi, buning sababi dasturlash tili o‘zi uzun sonlar bilan ishlashni biladi. Ammo buning uchun xotirasidan ko‘p joy egallaydi. Barcha algrotimlarni **C++** dasturlash tilida ifodalanadi, shuningdek dastur kodlari ishlashi uchun **C++14** standartini talab qiladi.

Biz uzun raqamlarni vektor sifatida saqlaymiz, vektor asosi esa **int** tipi deb olamiz, bu yerda har bir element raqamning bir raqamidir. Samaradorlikni oshirish uchun biz sonoq tizimda milliard asosida ishlaymiz, ya'ni vektorning har bir elementi bitta raqam emas, balki 9 ta raqamlarni o‘z ichiga oladi:

C++

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <algorithm>
using namespace std;
// bu bizning uzun sonimiz bo‘ladi
using longint = vector<int>;
// bu esa xar bir razryadimiz nechta raqam saqlashini bildiradi
constexpr int base = 1e9;
constexpr auto digits = 9;
```

1 - kod

E’tibor bering $base = 1e9 = 1 * 10^9 = 1\ 000\ 000\ 000$. Raqamlar vektorda shunday tartibda saqlanadiki, eng kam ahamiyatli raqamlar (ya'ni, birliklar, o‘nliklar, yuzliklar va boshqalar) birinchi bo‘lib keladi. Bundan tashqari, barcha operatsiyalar shunday amalga

oshiriladiki, ulardan birortasini bajargandan so‘ng, oldingi nollar (ya'ni, sonning boshida qo‘shimcha nollar) bo‘lmaydi (albatta, har biridan oldin hech qanday nol yo‘q deb hisoblasak). operatsiya). Shuni ta'kidlash kerakki, nol raqami uchun taqdim etilgan dasturda bir vaqtning o‘zida ikkita ko‘rinish to‘g‘ri qo‘llab-quvvatlanadi: raqamlarning bo‘sh vektori va bitta elementni o‘z ichiga olgan raqamlar vektori - nol.

Uzun sonli vektorimizni ekranga chiqarish

Birinchidan, biz vektorning oxirgi elementini chiqaramiz (yoki 0 vektor bo‘sh bo‘lsa), so‘ngra vektorning qolgan barcha elementlarini 9 ta belgi uzunligiga tenglaymiz bo‘sh qismlarni nol bilan to‘ldirib chiqamiz:

C++

```
ostream& operator <<(ostream& os, longint& v) {
if (!v.empty()) {
auto i = v.crbegin();
os << *i;
for (++i; i != v.crend(); ++i) {
os << setw(digits) << setfill('0') << *i;
}
} else {
os << 0;
}
return os;
}
```

```
void print(const longint& v) {
if (!v.empty()) {
auto i = v.crbegin();
cout << *i;
for (++i; i != v.crend(); ++i) {
cout << setw(digits) << setfill('0') << *i;
}
} else {
cout << 0;
}
cout << endl;
}
```

2 - kod

Ekranga chiqarish jarayonini ikki usulda amalga oshirdik, birinchi usulda biz chiqarish oqimi operatorini bizning **longint** tipimiz bilan ishlashi uchun ko‘rsatmalar berdik, bu usulning qulayligi to‘g‘ridan-to‘g‘ri dasturning istalgan yerida biz **cout** buyrug‘idan foydalanib uzun sonimizni ekranga chiqarishimiz mumkin. Ikkinchi usul funksiyadan foydalangan holda ekranga chiqarish. E’tibor bering biz **setw** va **setfill** o‘qim funksiyalaridan foydalandik, **setw** funksiyasi o‘qim bufferida berilgan butun songa teng bo‘shliq hosil qiladi, **setfill** funksiyasi esa bufferdagi bo‘shliqni berilgan belgi bilan to‘ldiradi. Natijada bizning razryadagi sonimiz 9 ta belgidan kam bo‘lsa oldiga 0 belgisi qoyiladi. Matematikadan ma’lumki butun sonning oldida joylashgan 0 raqamlarini tushirib qoldirish mumkin, shu sababdan birinchi razryad sonimizni qanday bo‘lsa shunday chiqarish maqul, bizning holatda bu oxirgi element.

Kiritilgan ma’lumotni uzon sonli vektorga saqlash

Berilgan ma'lumot **string** tip bo'lishi maqsadga muvofiq, buning sababi shundaki **cin** yoki boshqa ko'rinishdagi kiritish o'qimi yordamida berilgan sonlar ketma-ketligini biz uzun sonli vektorimizga aylantirib olishimiz oson bo'ladi. Faqat sonlardan iborat ketma-ketlikni teskari tartibda 9 ta belgidan qilib qirqib olib, keyin ularni **int** tipiga o'tkazib saqlashimiz kerak bo'ladi. Dastur kodi quyidagicha:

C++

```
istream& operator >>(istream &is, longint &v) {
string s;
getline(is, s);
auto len = s.length();
v.clear();
v.reserve(len / digits + (len % digits == 0 ? 0 : 1));
while (len > digits) {
v.push_back(stoi(s.substr(len-digits, digits).c_str()));
len -= digits;
}
v.push_back(stoi(s.substr(0, len).c_str()));
return is;
}
```

```
longint input(const string &s) {
auto len = s.length();
longint v;
v.reserve(len / digits + (len % digits == 0 ? 0 : 1));
while (len > digits) {
v.push_back(stoi(s.substr(len-digits, digits).c_str()));
len -= digits;
}
v.push_back(stoi(s.substr(0, len).c_str()));
return v;
}
```

3 - kod

v.reserve(len / digits + (len % digits == 0 ? 0 : 1)) – bu kod yordamida berilgan satrdagi sonlar ketma-ketligi uchun nechta sonlar razryadi kerak bo'lishini aniqlab olamiz va bu yo'l bilan vektorning majburiy kengayish jarayonini bartaraf etib dastur ishlashini berilgan satr qancha katta bo'lsa shuncha marta tezlashtirgan bo'lamiz.

Agar kiritilgan raqamda oldingi nollar bo'lishi mumkin bo'lsa, ularni o'qib chiqqandan so'ng ularni shu tarzda olib tashlash mumkin:

C++

```
auto clean = [](longint &v) {
while (v.size() > 1 && v.back() == 0)
v.pop_back();
};
```

```
void clean(longint &v) {
while (v.size() > 1 && v.back() == 0)
v.pop_back();
}
```

4 - kod

Uzun sonlarni taqqoslash

Ikki uzun sonni taqqoslash uchun birinchi ularning razryadlar sonini taqqoslash yetarli agar biri ikkinchisidan katta bo‘lsa razryadi katta bo‘lgan uzun son katta hisoblanadi. Agar razryadlar soni teng bo‘lsa unda ketma-ket katta razryaddan boshlab raqamlarni taqqoslaymiz birinchi farqli bo‘lgan razradning raqamlarini kattasini topish yetarlik. Dastur kodi quyidagicha:

C++

```
int compare(const longint &a, const longint &b) {
if (a.size() == b.size()) {
for (auto i = a.size()-1; i >= 0; --i) {
if (a[i] == b[i])
continue;
return a[i] > b[i] ? 1 : -1;
}
return 0;
}
return a.size() > b.size() ? 1 : -1;
}
```

```
int compare(const string &a, const string &b) {
if (a.size() == b.size()) {
for (auto i = a.size()-1; i >= 0; --i) {
if (a[i] == b[i])
continue;
return a[i] > b[i] ? 1 : -1;
}
return 0;
}
return a.size() > b.size() ? 1 : -1;
}
```

5 - kod

Berilgan kodlarning birinchisi bizning **longint** tipimizdan kattasini topish uchun ko‘rsatmalar yozilgan. Ikkinchi kodda esa **string** tipidagi sonning kattasini topish uchun ko‘rsatmalar yozilgan. Ba’zi bir masalalarda berilgan uzun sonlarni kattasini topish kerak bo‘ladi, shu hollar uchun **string** tipidagi ikki sonni farqini topishni o‘zi kifoya. **Compare** – funksiyasi natijasida $[1; 0; -1]$ qiymatlardan birini beradi, 1 – bu chap tarafdagi o‘zgaruvchi kattaligini bildiradi, -1 esa o‘ng tarafdagi o‘zgaruvchini anglatadi, 0 esa ikkalasi teng ekanligini bildiradi.

Uzun sonlarni qo‘shish

Berilgan a_n, b_m uzun sonlarini natijasini a o‘zgaruvchiga saqlaydi va javob tariqasida qaytaradi. $a_{\max(n,m)}$ o‘zgaruvchiga xotira hajmi va dastur ishlash tezligidan yutish maqsadida mavjud bo‘lgan sonda natijani saqlash maquldir. Bu yerda n, m razryadlar soni $(n, m < 2^{32} - 1)$.

C++

```
longint& operator +(longint &a, const longint &b) {
int carry = 0;
auto an = max(a.size(), b.size());
auto bn = b.size();
if (a.size() < an) {
```

```
a.reserve(an+1);
}
for (size_t i = 0; i < an; ++i) {
a[i] += carry + (i < bn ? b[i] : 0);
carry = a[i] >= base;
if (carry)
a[i] -= base;
}
if (carry)
a.push_back(1);
return a;
}
```

```
longint& add(longint &a, const longint &b) {
int carry = 0;
auto an = max(a.size(), b.size());
auto bn = b.size();
if (a.size() < an) {
a.reserve(an+1);
}
for (size_t i = 0; i < an; ++i) {
a[i] += carry + (i < bn ? b[i] : 0);
carry = a[i] >= base;
if (carry)
a[i] -= base;
}
if (carry)
a.push_back(1);
return a;
}
```

6 - kod

Uzun sonlarni qo‘shish

Berilgan a_n, b_m uzun sonlarini natijasini a_n o‘zgaruvchiga saqlaydi va javob tariqasida qaytaradi. Ushbu algoritm kamchiligi manfiy son bo‘lish holati inobta olinmagan, shu sababdan $a_n, b_m, n > m$ shartni qanoatlantirishi kerak. Shuningdek aytishdan so‘ng oldingi o‘rinda turgan nollarni o‘chirib yuboramiz. Bu yerda n, m razryadlar soni $(n, m < 2^{32} - 1)$.

C++

```
longint& operator -(longint &a, const longint &b) {
int carry = 0;
auto bn = b.size();
for (size_t i = 0; i < bn; ++i) {
a[i] -= carry + b[i];
carry = a[i] < 0;
if (carry)
a[i] += base;
}
// Ayrishdan so‘ng bo‘sh bo‘lganlarini tozalaymiz
while (a.size() > 1 && a.back() == 0)
a.pop_back();
return a;
```

```
}
longint& sub(longint &a, const longint &b) {
int carry = 0;
auto bn = b.size();
for (size_t i = 0; i < bn; ++i) {
a[i] -= carry + b[i];
carry = a[i] < 0;
if (carry)
a[i] += base;
}
// Ayrishdan so‘ng bo‘sh bo‘lganlarini tozalaymiz
while (a.size() > 1 && a.back() == 0)
a.pop_back();
return a;
}
```

7 - kod

Uzun sonni qisqa songa ko‘paytirish

a_n uzun butun sonli vektorni $b, (b < base)$ qisqa butun songa ko‘paytiradi va natijani a uzun butun sonli vektorda saqlaydi. Bu yerda n razryadlar soni $(n < 2^{32} - 1)$.

C++

```
longint& operator *(longint &a, const int b) {
int carry = 0;
auto an = a.size();
for (size_t i = 0; i < an; ++i) {
long long cur = carry + a[i] * 1LL * b;
a[i] = cur % base;
carry = cur / base;
}
if (carry)
a.push_back(carry);
return a;
}
longint& mul(longint &a, const int b) {
int carry = 0;
auto an = a.size();
for (size_t i = 0; i < an; ++i) {
long long cur = carry + a[i] * 1LL * b;
a[i] = cur % base;
carry = cur / base;
}
if (carry)
a.push_back(carry);
return a;
}
```

8 - kod

Uzun sonlarni ko‘paytirish

a_n uzun butun sonni b_m uzun butun songa ustunli ko‘paytiradi va natijani c_{n+m} uzun sonli vektorga saqlaydi. Bu yerda n, m xar bir vektordagi razryadlar soni. Ushbu algoritm $(n + m < 2^{32} - 1)$ miqdarogacha bo‘lgan razryadagi sonlar bilan ishlash imkoniga ega.

C++

```
longint operator *(const longint &a, const longint &b) {
auto an = a.size();
auto bn = b.size();
longint c(an+bn);
for (size_t i = 0; i < an; ++i) {
int carry = 0;
for(size_t j = 0; j < bn; ++j) {
int cc = c[i+j];
long long cur = c[i+j] + a[i] * 1LL * b[j] + carry;
c[i+j] = cur % base;
carry = cur / base;
}
if (carry)
c[i+bn] += carry;
}
// 0 ga teng bo‘lganlarini tozalaymiz
while (c.size() > 1 && c.back() == 0)
c.pop_back();
return c;
}
```

```
longint mul(const longint &a, const longint &b) {
auto an = a.size();
auto bn = b.size();
longint c(an+bn);
for (size_t i = 0; i < an; ++i) {
int carry = 0;
for(size_t j = 0; j < bn; ++j) {
int cc = c[i+j];
long long cur = c[i+j] + a[i] * 1LL * b[j] + carry;
c[i+j] = cur % base;
carry = cur / base;
}
if (carry)
c[i+bn] += carry;
}
// 0 ga teng bo‘lganlarini tozalaymiz
while (c.size() > 1 && c.back() == 0)
c.pop_back();
return c;
}
```

9 - kod

Bu yerda c_{n+m} vektor uchun ajratilgan $n+m$ uzunlikdagi razryad ko‘plik qilishi mumkin, shuning o‘shiqcha bo‘lgan razryadlarni olib tashlaymiz. n, m bu a_n, b_m sonlarning razryadlar soni.

Uzun sonni qisqa songa bo‘lish

Berilgan a uzun sonni b qisqa songa bo‘lish, ushbu algoritm to‘g‘ri ishlashi uchun berilgan b soni $b < base$ shartni qanoatlantiruvchi bo‘lishi kerak.

C++

```
longint& operator /(longint &a, const int b) {
int carry = 0;
auto an = a.size();
for (long i = an-1; i >= 0; --i) {
long long cur = a[i] + carry  * 1LL * base;
a[i] = cur / b;
carry = cur % b;
}

// 0 ga teng bo‘lganlarini tozalaymiz
while (a.size() > 1 && a.back() == 0)
a.pop_back();
return a;
}
```

```
longint& div(longint &a, const int b) {
int carry = 0;
auto an = a.size();
for (long i = an-1; i >= 0; --i) {
long long cur = a[i] + carry  * 1LL * base;
a[i] = cur / b;
carry = cur % b;
}

// 0 ga teng bo‘lganlarini tozalaymiz
while (a.size() > 1 && a.back() == 0)
a.pop_back();
return a;
}
```

10 - kod

FOYDALANILGAN ADABIYOTLAR:

1. Ivor Horton,Peter Van Weert, "Beginning C++17: From Novice to Professional", Kessel-Lo, Belgium, 788-pages.

2. Сиддхартха Рао, Освой самостоятельно C++ за 21 день, СЕДЬМОЕ ИЗДАНИЕ, Москва • Санкт-Петербург • Киев 2013, 651 ст.

3. https://emaxx-algo.ru