МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное учреждение высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент компьютерного и математического моделирования

ДОКЛАД

Эффективная_длинная_арифметика

по образовательной программе подготовки бакалавров

по направлению 09.03.03 «Прикладная информатика»

Студент группы № Б9121-09.03.03пикд-5 _____ Ли Д. С.

(подпись)

«_____» _____ 2022г.

г. Владивосток

2022

оглавление

Введение

Известно, что арифметические действия, выполняемые компьютером в ограниченном числе разрядов, не всегда позволяют получить точный результат. Более того, существуют ограничения – размер чисел, с которыми возможно работать.

Если необходимо выполнить арифметические действия над очень большими числами, например 30! = 265252859812191058636308480000000. То в таких случаях необходимо позаботиться о представлении больших чисел в машине и о точном выполнении арифметических операций над ними.

Числа, для представления которых в стандартных компьютерных типах данных не хватает количества двоичных разрядов, называются «длинными». Реализация арифметических операций над такими «длинными» числами получила название «Длинной арифметики».

«Длинная арифметика» - в вычислительной технике операции (сложение, умножение, вычитание, деление, возведение в степень и т.д.) на числами, разрядность которых превышает длину машинного слова данной вычислительной машины. Эти операции реализуются не аппаратно, а программно, используя базовые аппаратные средства работы с числами меньших порядков.

Проблема: существует класс задач, которые нельзя решить с помощью стандартных типов данных.

Цель: изучение метода «Длинная арифметика».

Данная исследовательская работа посвящена «Длинной арифметике» и её реализации на языке С++».

Теоретическая часть

Стандартные типы данных

Рассмотрим основные целочисленные типы данных языка C++ и диапазон их значений (табл.1)

Таблица 1

short	От -32 768 до 32 767
unsigned short	От 0 до 65 535
int	От -2 147 483 648 до 2 147 483 647
unsigned int	От 0 до 4 294 967 295
long	От -2 147 483 648 до 2 147 483 647
unsigned long	От 0 до 4 294 967 295
long long	От -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807
unsigned long long	От 0 до 18 446 744 073 709 551 615

Из таблицы 1 следует, что наибольшее число, которым мы можем оперировать это 18 446 744 073 709 551 615 или просто 2^{64}-1.

Но число 2^{64} уже не помещается ни в один из представленных типов данных. Для расчёта «длинных» чисел» потребуется другой метод.

Определение

Общее

Длинная арифметика – набор алгоритмов для поразрядной работы с числами произвольной длины. Она применяется как с относительно небольшими числами, превышающими ограничения типа long long в несколько раз, так и с по-настоящему большими числами (чаще всего до 10^{100000}).

Для работы с «длинными числами их разбивают на разряды. Размер разряда может быть произвольным, но чаще всего используются следующие:

- 10 – по аналогии с цифрами числа в десятичной системе, для простоты понимания и отладки.
- 10^4 – наибольшая степень десятки, квадрат которой не превышает ограничения типа int. Используется для максимальной эффективности при хранении разрядов как чисел типа int.

- 10^9 – аналогично предыдущему пункту, но для типа long long. Позволяет достичь максимально возможной эффективности.

(Ограничения на квадрат размера разряда связанны с необходимостью перемножать между собой разряды. Если квадрат разряда превышает ограничение своего типа, при умножении возможны переполнения.)

В большинстве реализаций разряды хранятся в порядке, обратным привычному для упрощения работы с ними. Например, число 578002300 при размере разряда 10^4 представляется следующим массивом:

$$\{2300,7800,5\}$$

Количество разрядов числа может быть как ограничено, так и не ограничено, в зависимости от типа используемого контейнера: массива константной длины или вектора.

Класс

Для начала следует определиться, какие переменные в классе нам необходимы.

Из пункта «Общее» следует, что необходимо создать массив, в котором будут храниться разряды «длинного» числа. Для максимальной эффективности следует использовать _int64 для хранение в одном разряде числа с ограничением 10^9.

Далее требуется определять знак (положительный\отрицательный) у «длинного» числа. Для этого следует добавить переменную типа данных bool. Что в дальнейшем будет означать: при true – отрицательное, а при false – положительное.

Так же следует добавить константу, которая будет отмечать ограничение разряда числа. В данном случае эта константа будет равна 1 000 000 000, что равно 10^9.

Все переменные следует отнести к доступу private, для ограничения доступа вне класса.

Сложение

«Длинную арифметику» часто сравниваю с детским вычислением «в столбик». Это достаточно справедливо, так как оба метода основаны на поразрядных операциях.

Перед сложением необходимо проверить следующие условия:

- Первое число отрицательное, второе положительное: в этом случае достаточно отнять из второго числа первое, поменяв знак первого.
- Первое число положительно, второе отрицательное: в этом случае достаточно отнять из первого числа второе, поменяв знак второго.
- Оба числа отрицательные: нужно сложить модули чисел, а затем поменять знак.

Учитывая вышеописанные условия в алгоритм вычитание могут попасть числа только положительные. Для примера будем использовать массив с ограничением размера разряда 10^4 (int), обратный порядок хранения разрядов. Сложим числа:
9 4325 1235 9482 5743
7847 2837 7518 3847

Рисунок 1

5743	9482	1235	4325	9
3847	7518	2837	7847	

На рисунке 1 показан способ хранения чисел в массиве. Далее сложим первый разряд.

Рисунок 2

5743	9482	1235	4325	9
3847	7518	2837	7847	
9590				

На рисунке 2 показан результат сложения первого разряда. Сложим следующий разряд.

Рисунок 3

5743	9482	1235	4325	9
3847	7518	2837	7847	
9590	(1)7000	(+1)		

На рисунке 3 показан результат сложения второго разряда. Результат сложения оказался больше, чем допустимая вместимость 10^4, поэтому единицу мы переносим в следующий разряд. Закончим сложение.

Рисунок 4

5743	9482	1235	4325	9
3847	7518	2837	7847	
9590	7000	4073 +1	2172	10 +1

На рисунке 4 представлен полный результат сложение двух вышеописанных чисел. Он равен 10 2172 4073 7000 9590.

Таким образом в дальнейшем будет реализовано сложение.

Вычитание

Вычитание реализуется симметрично сложению. Так, как и сложение, происходит в столбик.

Перед вычитанием необходимо проверить следующие условия:

- Второе число отрицательное: достаточно сложить модули двух чисел.
- Первое число отрицательное: сложить модули двух чисел, затем поменять знак результата.
- Первое число меньше второго: отнять из второго числа первое, затем поменять знак результата.

Учитывая вышеописанные условия в алгоритм вычитание могут попасть числа только положительные и гарантированно первое число будет больше второго. Условия для примера вычитания будут аналогичны сложению. Произведем вычитание чисел:

5 4815 9845 1354 4825
3 9452 1534 5024 1523

Рисунок 5

4825	1354	9845	4815	5
1523	5024	1534	9452	3

На рисунке 5 показано расположение. Произведем вычитание первого разряда.

Рисунок 6

4825	1354	9845	4815	5
1523	5024	1534	9452	3
3302				

На рисунке 6 результат вычитания первых разрядов. Произведем вычитание второго разряда.

Рисунок 7

4825	1354	9845	4815	5
1523	5024	1534	9452	3
3302	-3670	-1		
	+10000			

Рисунок 8

4825	1354	9845	4815	5
1523	5024	1534	9452	3
3302	6330	-1		

На рисунках 7 и 8 показано подробно вычитание разрядов с вычетом 1 из следующего разряда. Завершим вычитание.

Рисунок 9

4825	1354	9845	4815	5
1523	5024	1534	9452	3
3302	6330	8310 -1	5363	1 -1

На рисунке 8 показан результат вычитания. Он равен
1 5363 8310 6330 3302.

Таким образом в дальнейшем будет реализовано вычитание.

Умножение

Реализаций умножения существует множество. Но самая эффективная – алгоритм Карацубы. Его сложность $O(N^{1.58})$. Что превосходит обычное умножение в стобик ($O(N^2)$). Алгоритм Карацубы основан на парадигме «разделяй и властвуй».

Вначале нужно проверить длину входящего числа, если она меньше или равна 256 (оптимальный вариант длинны для разделения метода карацубы и обычного умножения в столбик), то удаляем лидирующие нули и умножаем в столбик. Использование алгоритма Карацубы дает преимущество по времени на более длинных числах.

Далее, дополняем длину чисел до четной, после чего равняем. Делим числа на равные части, после чего рекурсивно перемножаем, используя алгоритм Карацубы.

Деление

Ф

Остаток

Ф

Сравнение

Для описания функций, всех операторов сравнения, достаточно описать оператор равенства и один из операторов больше или меньше.

Рассмотрим равенство: для начала проверим различие знаков. Далее проверим числа на нули. После этого проверим числа по длине, если они не равны, то числа не равны. Дальше сравним каждый разряд, если какой-то из разрядов не равен другому, то числа не равны. В ином случае числа равны.

Рассмотрим операцию меньше: для начала проверим равенство, благодаря описанному выше алгоритму. Далее проверим различие знаков «длинных» чисел. Далее сравним длину чисел и, наконец, сравним поразрядно.

Все остальные операции сравнения (!=, >, <=, >=) реализуются с помощью уже написанных операций.

Инкрементирование

Достаточно с помощью сложения вернуть значение +1. (При условии ++x).

Реализация x++: увеличивание на 1, затем возвращение значения -1.

Декрементирование

Аналогично инкрементированию, описанного выше.

Дополнительные функции

- Функция изменения знака: копирует текущее «длинное» число, меняет переменную, которая хранит в себе знак, на противоположную себе.
- Функция преобразования в строку: нужно использовать стороннюю библиотеку stringstream.
- Перегрузка функции вывода «длинного» числа: Сначала выведем знак числа, а затем само число поразрядно.
- Функция удаления ведущих нулей: простейшая функция удаление первых нулей.
- Различные конструкторы: для создания «длинного» числа. Один из конструкторов – конструктор с входным типом данных string. Проверяем первый символ строки на символ отрицания, затем по

9 цифр, в случае лучшей эффективности, сохраняем числа в массив.

Реализация

Создание класса

```
class BigInt {
private:
    static const _int64 BASE = 1000000000;
    static const _int64 BASElen = 9;
    static const _int64 DefMul = 256;
    std::vector<_int64> _digits;
    bool _isNegative;
};
```

Рассмотрим данный класс:

- BASE – константная переменная, для разделения строки на разряды.
- BASElen – длина одного разряда.
- DefMul – ограничение длины числа для использования обычного деления.
- _digits – массив, в котором хранится «длинное» число.
- _isNegative – хранит true или false в зависимости от знака числа.

```
BigInt::BigInt() {
    this->_isNegative = false;
    this->_digits.push_back(0);
}

BigInt::BigInt(const BigInt &x) {
    this->_digits = x._digits;
    this->_isNegative = x._isNegative;
};

BigInt::BigInt(std::string str) {
    if (str.length() == 0) {
        this->_isNegative = false;
    } else {
        if (str[0] == '-') {
            str = str.substr(1);
            this->_isNegative = true;
        } else {
            this->_isNegative = false;
        }
        for (long long i = str.length(); i > 0; i -= BigInt::BASElen) {
            if (i < BigInt::BASElen) this-
>_digits.push_back(atoi(str.substr(0, i).c_str()));
            else this->_digits.push_back(atoi(str.substr(i - BigInt::BASElen,
BigInt::BASElen).c_str()));
        }
    }
}

BigInt::BigInt(unsigned short l) {
    if (l < 0) {
        this->_isNegative = true;
        l = -l;
    } else this->_isNegative = false;
    do {
```

```
        this->_digits.push_back(l % BigInt::BASE);
        l /= BigInt::BASE;
    } while (l != 0);
}

BigInt::BigInt(signed short l) {
    if (l < 0) {
        this->_isNegative = true;
        l = -l;
    } else this->_isNegative = false;
    do {
        this->_digits.push_back(l % BigInt::BASE);
        l /= BigInt::BASE;
    } while (l != 0);
}

BigInt::BigInt(unsigned int l) {
    if (l < 0) {
        this->_isNegative = true;
        l = -l;
    } else this->_isNegative = false;
    do {
        this->_digits.push_back(l % BigInt::BASE);
        l /= BigInt::BASE;
    } while (l != 0);
}

BigInt::BigInt(signed int l) {
    if (l < 0) {
        this->_isNegative = true;
        l = -l;
    } else this->_isNegative = false;
    do {
        this->_digits.push_back(l % BigInt::BASE);
        l /= BigInt::BASE;
    } while (l != 0);
}

BigInt::BigInt(unsigned long l) {
    this->_isNegative = false;
    do {
        this->_digits.push_back(l % BigInt::BASE);
        l /= BigInt::BASE;
    } while (l != 0);
}

BigInt::BigInt(signed long l) {
    if (l < 0) {
        this->_isNegative = true;
        l = -l;
    } else this->_isNegative = false;
    do {
        this->_digits.push_back(l % BigInt::BASE);
        l /= BigInt::BASE;
    } while (l != 0);
}

BigInt::BigInt(unsigned long long l) {
    this->_isNegative = false;
    do {
        this->_digits.push_back(l % BigInt::BASE);
        l /= BigInt::BASE;
    } while (l != 0);
}
```

```
BigInt::BigInt(signed long long l) {
    if (l < 0) {
        this->_isNegative = true;
        l = -l;
    } else this->_isNegative = false;
    do {
        this->_digits.push_back(l % BigInt::BASE);
        l /= BigInt::BASE;
    } while (l != 0);
}

BigInt::BigInt(std::vector<_int64> v) {
    this->_digits = std::move(v);
    this->_isNegative = false;
}
```

Здесь представлены конструкторы класса, благодаря которым «длинное» число можно получить из представленных типов данных.

Служебные функции

```
std::ostream &operator<<(std::ostream &os, const BigInt &bi) {
    if (bi._digits.empty()) os << 0;
    else {
        if (bi._isNegative) os << '-';
        os << bi._digits.back();
        char oldFill = os.fill('0');
        for (long long i = static_cast<long long>(bi._digits.size()) - 2; i
>= 0; i--) {
            os << std::setw(BigInt::BASElen) << bi._digits[i];
        }
        os.fill(oldFill);
    }
    return os;
}

BigInt::operator std::string() const {
    std::stringstream ss;
    ss << *this;
    return ss.str();
}
```

Функция выше – перегрузка оператора вывода, для удобства вывода «длинного» числа. Функция ниже – приведение «длинного» числа к строке.

```
const BigInt BigInt::operator-() const {
    BigInt copy(*this);
    copy._isNegative = !copy._isNegative;
    return copy;
}
```

Функция изменения знака числа.

Сложение

```
const BigInt operator+(BigInt left, const BigInt &right) {
    if (left._isNegative) {
        if (right._isNegative) return -(-left + (-right));
        else return right - (-left);
    } else if (right._isNegative) return left - (-right);
    int carry = 0;
    for (size_t i = 0; i < std::max(left._digits.size(),
right._digits.size()) || carry != 0; ++i) {
        if (i == left._digits.size()) left._digits.push_back(0);
```

```
        left._digits[i] += carry + (i < right._digits.size() ?
right._digits[i] : 0);
        carry = left._digits[i] >= BigInt::BASE;
        if (carry != 0) left._digits[i] -= BigInt::BASE;
    }
    return left;
}
```

Сложение происходит столбиком, с учетом различных условий, описанных в теоретической части.

```
BigInt &BigInt::operator+=(const BigInt &value) {
    return *this = (*this + value);
}
```

Здесь показан перегруз оператора +=. Достаточно простая реализация, через уже готовую функцию сложения.

Вычитание

```
const BigInt operator-(BigInt left, const BigInt &right) {
    if (right._isNegative) return left + (-right);
    else if (left._isNegative) return -(-left + right);
    else if (left < right) return -(right - left);
    int carry = 0;
    for (size_t i = 0; i < right._digits.size() || carry != 0; ++i) {
        left._digits[i] -= carry + (i < right._digits.size() ?
right._digits[i] : 0);
        carry = left._digits[i] < 0;
        if (carry != 0) left._digits[i] += BigInt::BASE;
    }
    left = BigInt::_removeLeadingZero(left);
    return left;
}
```

Вычитание реализовано симметрично сложению. Так же условия описаны в теоретической части.

```
BigInt &BigInt::operator-=(const BigInt &value) {
    return *this = (*this - value);
}
```

Перегруз оператора -= реализация аналогична перегрузу оператора +=.

Умножение

Ф

Деление

Ф

Остаток

Ф

Сравнение

```
bool operator==(const BigInt &left, const BigInt &right) {
    if (left._isNegative != right._isNegative) return false;
    if (left._digits.empty()) {
        if (right._digits.empty() || (right._digits.size() == 1 &&
right._digits[0] == 0)) return true;
        else return false;
```

```
    }
    if (right._digits.empty()) {
        if (left._digits.size() == 1 && left._digits[0] == 0) return true;
        else return false;
    }
    if (left._digits.size() != right._digits.size()) return false;
    for (size_t i = 0; i < left._digits.size(); ++i) if (left._digits[i] !=
right._digits[i]) return false;
    return true;
}
```

Реализация операции == описана в теоретической части.

```
bool operator<(const BigInt &left, const BigInt &right) {
    if (left == right) return false;
    if (left._isNegative) {
        if (right._isNegative) return ((-right) < (-left));
        else return true;
    } else if (right._isNegative) return false;
    else {
        if (left._digits.size() != right._digits.size()) {
            return left._digits.size() < right._digits.size();
        } else {
            for (long long i = left._digits.size() - 1; i >= 0; --i) {
                if (left._digits[i] != right._digits[i]) return
left._digits[i] < right._digits[i];
            }
            return false;
        }
    }
}
```

Реализация операции < так же описана в теоретической части.

Остальные операции сравнения реализованы и работают от операций == и <.

```
bool operator!=(const BigInt &left, const BigInt &right) {
    return !(left == right);
}

bool operator>(const BigInt &left, const BigInt &right) {
    return !(left < right);
}

bool operator<=(const BigInt &left, const BigInt &right) {
    return (left < right || left == right);
}

bool operator>=(const BigInt &left, const BigInt &right) {
    return (left > right || left == right);
}
```

Инкрементирование

```
const BigInt BigInt::operator++() {
    return *this += (long long) 1;
}
const BigInt BigInt::operator++(int) {
    *this += 1;
    return *this - 1;
}
```

Инкрементирование чисел, простая реализация через функцию сложения.

Декрементирование

```
const BigInt BigInt::operator--() {
    return *this -= (long long) 1;
}
const BigInt BigInt::operator--(int) {
    *this -= (long long) 1;
    return *this + (long long) 1;
}
```

Декрементирование аналогично инкрементированию.

Заключение

В процессе работы над данной темой был создан класс работы с «длинными» числами, написаны функции для основных арифметических операций (сложение, вычитание, умножение, деление, взятие остатка). А так же различные функции для удобства работы с написанным классом.

Источники

1. https://brestprog.by/topics/longarithmetics/
2. https://ru.wikipedia.org/wiki/Длинная_арифметика
3. https://e-maxx.ru/algo/big_integer
4. https://habr.com/ru/post/207754/
5. https://inf2086.ru/crypto_basics/book/algo_long_arithmetic.html
6. https://megaobuchalka.ru/6/33526.html
7. https://topref.ru/referat/50385.html
8. https://intellect.icu/dlinnaya-arifmetika-s-primerami-na-si-8291
9. http://cppstudio.com/post/5036/
10. https://rg-gaming.ru/kompjutery/dlinnaja-arifmetika-c-delenie
11. http://comp-science.narod.ru/DL-AR/okulov.htm
12. https://habr.com/ru/post/124258/
13. https://studfile.net/preview/7014549/page:6/
14. https://forkettle.ru/vidioteka/programmirovanie-i-set/algoritmy-i-struktury-dannykh/73-lektsii-ot-nou-intuit/bazovye-algoritmy-dlya-shkolnikov-lektsii-ot-nou-intuit/572-lektsiya-9-dlinnaya-arifmetika
15. https://moluch.ru/archive/180/46418/
16. https://dic.academic.ru/dic.nsf/ruwiki/1299482
17. https://studassistent.ru/c/dlinnaya-arifmetika-na-si-c-si
18. https://www.pvsm.ru/algoritmy/29587
19. https://itnan.ru/post.php?c=1&p=451860
20. https://habr.com/ru/post/172285/
21. https://pro-prof.com/forums/topic/разность-чисел-длинная-арифметика-си
22. https://lisiynos.github.io/s1/long_ar.html
23. https://habr.com/ru/post/135590/
24. https://inf.1sept.ru/2000/1/art/okul1.htm
25. https://habr.com/ru/post/578718/
26. https://ru.stackoverflow.com/questions/1320123/Оптимизация-длинной-арифметики-с
27. https://www.stud24.ru/programming-computer/dlinnaya-arifmetika/22260-63531-page1.html