Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №6 по курсу «Дискретный анализ»

Студент: П. А. Харьков
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-19

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №6

Задача: Необходимо разработать программную библиотеку на языке C или C++, реализующую простейшие арифметические действия и проверку условий над целыми неотрицательными числами. На основании этой библиотеки нужно составить программу, выполняющую вычисления над парами десятичных чисел и выводящую результат на стандартный файл вывода.

Список арифметических операций:

- Сложение (+).
- Вычитание (-).
- Умножение (*).
- Возведение в степень (^).
- Деление (/).

В случае возникновения переполнения в результате вычислений, попытки вычесть из меньшего числа большее, деления на ноль или возведении нуля в нулевую степень, программа должна вывести на экран строку «Error».

Список условий:

- Больше (>).
- Меньше (<).
- Равно (=).

В случае выполнения условия программа должна вывести на экран строку «true», в противном случае — «false».

Количество десятичных разрядов целых чисел не превышает 100000. Основание выбранной системы счисления для внутреннего представления «длинных» чисел должно быть не меньше 10000.

1 Описание

Требуется написать реализацию арифметики над длинными числами на языке C++. Для начала необходимо ввести определение длинных чисел - это числа, которые могут быть значительно больше чисел, являющихся стандартными типами данных. Короткими числами будем называть числа, которые являются стандартными типами данных. Так как максимальным значением короткого числа является число $2^{64} - 1$, то для некоторых математических вычислений их может не хватать. Для этого и нужны длинные числа - они позволяют вычислять значения любой длины.

Длинные числа мы будем считывать, как строку, и выводить будем тоже, как строку. Но хранить их мы будем в векторе, как числа, разбив на блоки. Блоком я буду называть число, являющееся подчислом длинного числа. К примеру, число 123456789 мы можем хранить в векторе в таком виде:{1,2345,6789}. Но для того, чтобы быстрее прибавлять разряды к числу (что понадобится нам при умножении и сложении), мы будем хранить блоки в обратном порядке.

Сложение, вычитание, деление и умножение мы будем производить в «столбик». К примеру, рассмотрим алгоритм сложения в столбик. Блоки мы будем брать с конца длинного числа:

1. Берем блок из первого числа и из второго и считаем их сумму.
2. Если при предыдущем суммировании значение суммы блоков превысило максимальный размер блока, то прибавляем к текущей сумме единицу.
3. Записываем в результирующий блок полученную сумму.
4. Берем следующие блоки и повторяем, пока не дойдем до конца числа максимальной длины.

Возведение в степень же я реализовал с помощью быстрого возведения в степень. Я использовал рекурсивный способ. Допустим, что необходимо возвести число a в степень n, тогда алгоритм выглядит так:

1. Если $n == 0$, то $a = 1$.
2. Если n кратно 2, то $a^n = a^{n/2} * a^{n/2}$.
3. Если n не кратно 2 , то $a^n = a * a^{n-1}$.

2 Исходный код

long_num.cpp	
TLongNum(const std::string &str)	Функция, инициализирующая TLongNum и разбивающая число на «подчисла».
friend std::istream& operator»(std::istream &in, TLongNum &rhs)	Функция, считывающая длинное число.
friend std::ostream& operator«(std::ostream &out, const TLongNum &rhs)	Функция, выводящая длинное число в поток.
TLongNum operator+(const TLongNum &rhs) const	Функция, складывающая длинное число с длинным числом.
TLongNum operator+(const long long &rhs) const	Функция, складывающая длинное число с коротким числом.
TLongNum operator-(const TLongNum &rhs) const	Функция, вычитающая длинное число из длинного числа.
TLongNum operator-(const long long &rhs) const	Функция, вычитающая короткое число из длинного числа.
TLongNum operator*(const TLongNum &rhs) const	Функция, умножающая длинное число на длинное число.
TLongNum operator*(const long long &rhs) const	Функция, умножающая длинное число на короткое число.
TLongNum operator^(const TLongNum &rhs) const	Функция, возводящая длинное число в степень длинного числа.
TLongNum operator^(const long long &rhs) const	Функция, возводящая длинное число в степень короткого числа.
TLongNum operator/(const TLongNum &rhs) const	Функция, делящая длинное число на длинное число.
TLongNum operator/(const long long &rhs) const	Функция, делящая длинное число на короткое число.
bool operator<(const TLongNum &rhs) const	Функция, проверяющая, левое длинное число меньше ли правого длинного числа.
bool operator>(const TLongNum &rhs) const	Функция, проверяющая, левое длинное число больше ли правого длинного числа.
bool operator==(const TLongNum &rhs) const	Функция, проверяющая, равны ли длинные числа.

bool IsZero() const	Функция, проверяющая, является ли длинное число нулем.
void DeleteLeadingZeros()	Функция, удаляющая «лидирующие» нули.

Листинг без реализаций методов:

```
class TLongNum
{
    public:
        TLongNum() = default;
        TLongNum(const std::string &str);

        friend std::istream& operator>>(std::istream &in, TLongNum &rhs);
        friend std::ostream& operator<<(std::ostream &out, const TLongNum &rhs);

        TLongNum operator+(const TLongNum &rhs) const;
        TLongNum operator+(const long long &rhs) const;

        TLongNum operator-(const TLongNum &rhs) const;
        TLongNum operator-(const long long &rhs) const;

        TLongNum operator*(const TLongNum &rhs) const;
        TLongNum operator*(const long long &rhs) const;

        TLongNum operator^(const TLongNum &rhs) const;
        TLongNum operator^(const long long &rhs) const;

        TLongNum operator/(const TLongNum &rhs) const;
        TLongNum operator/(const long long &rhs) const;

        bool operator<(const TLongNum &rhs) const;
        bool operator>(const TLongNum &rhs) const;
        bool operator==(const TLongNum &rhs) const;
        bool IsZero() const;

    private:
        void DeleteLeadingZeros();
        std::vector<int> Data;
};
```

3 Консоль

```
p.kharkov$  make
g++ -std=c++14 -pedantic -Wall -Wextra -c long_num.cpp -o long_num.o
g++ -std=c++14 -pedantic -Wall -Wextra laba6.cpp long_num.o -o laba6
p.kharkov$  cat tests/main
38943432983521435346436
354353254328383
+
904094384738493247293847 3843
2343543
-
972323
2173937
>
2
3
-
p.kharkov$  ./laba6 <tests/main
38943433337874689674819
904094384738493247293613 0300
false
Error
```

4 Тест производительности

Тест производительности представляет из себя следующее: производятся 4 теста, а именно - сложения, вычитания, умножения и деления. Первый тест состоит из 100 тысяч операций сложения, каждое длинное число состоит из чисел длины 100. Результаты первого теста:

```
p.kharkov$ ./benchmark <tests/01.t
my_num time = 559 us
gmp time = 61 us
```

Второй тест состоит из 100 тысяч операций вычитания, каждое длинное число состоит из чисел длины 100. Результаты второго теста:

```
p.kharkov$./benchmark <tests/02.t
my_num time = 301 us
gmp time = 30 us
```

Третий же тест состоит из 100 тысяч операций умножения и каждое длинное число может быть длины 100. Результаты третьего теста:

```
p.kharkov$ ./benchmark <tests/03.t
my_num time = 1000428 us
gmp time = 4265 us
```

Четвертый тест состоит из 100 тысяч операций деления и каждое длинное число может быть длины от 1 до 100. Результаты четвертого теста:

```
p.kharkov$ ./benchmark <tests/04.t
my_num time = 4798252 us
gmp time = 12690 us
```

Как видно, время работы gmp значительно меньше. Это связано с тем, что моя реализация упрощенная и в ней не используются сложные алгоритмы. Как минимум, умножение столбиком значительно медленнее, чем умножение Карацубы.

5 Выводы

Выполнив шестую лабораторную работу по курсу «Дискретный анализ», я научился реализовывать простейшие арифметические операции и сравнения для длинных чисел. Длинные числа очень полезны при больших математических вычислениях, но все же сложность любой операции становится, как минимум линейной.
Также я изучил алгоритм Карацубы для умножения за $O(n^{log3})$, тогда как мой алгоритм умножения столбиком работает за $O(n^2)$. Это бы сильно уменьшило скорость работы программы при делении, так как умножение там используется часто.

Список литературы

[1] *Длинная арифметика — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Длинная_арифметика` (дата обращения: 22.03.2021).