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: А. А. Терво
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2022

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 (/).

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 таких простейших арифметических операций для длинных чисел, как сложение, вычитание, умножение, деление, возведение в степень и операции сравнения. Идея реализации длинных чисел состоит в использовании системы счисления с основанием кратным 10. В векторе хранятся числа, не превышающие основание системы счисления, они играют роль цифр числа. Число в векторе располагается от младшего разряда к старшему для удобства оперирования.

Сложение осуществляется «столбиком». Выполняется, начиная с младших разрядов. Складываются соответствующие разряды, прибавляется остаток с предыдущего разряда, результат записывается в вектор ответа. Если получившееся число больше чем основание системы счисления, число уменьшается на основание системы счисления, а остаток переносится на следующий разряд.

Вычитание осуществляется также «в столбик», только остаток не добавляется, а вычитается из старшего разряда.

Умножение может осуществляться алгоритмом Анатолия Карацубы, но я решил всё же использовать тривиальный алгоритм.

Деление осуществляется «уголком». Ответ начинает формироваться со старшего разряда. Для поиска промежуточного делителя будем применять бинарный поиск.

Тривиальный алгоритм возведения в степень требует слишком большого количества умножений. Будем использовать бинарное возведение в степень, оно позволяет снизить количество умножений с n до $log(n)$.

Операции сравнения выполняются поразрядно.

2 Исходный код

Выполнение работы разобьём на следующие шаги:

1. Описание класса больших чисел.
2. Перегрузка простых операторов сложения, вычитания и сравнения.
3. Перегрузка операторов умножения, деления, возведения в степень.
4. Реализация ввода и работы калькулятора.

Опишем класс `TBigInt` для работы с большими числами.

BigInt.hpp	
TBigInt(std::string&)	Конструктор класса. Делает длинное число из строки.
TBigInt operator +(const TBigInt&)	Перегрузка оператора сложения.
TBigInt operator -(const TBigInt&)	Перегрузка оператора вычитания.
TBigInt operator *(const TBigInt&) const	Перегрузка оператора умножения.
TBigInt operator /(const TBigInt&)	Перегрузка оператора деления.
TBigInt Power(int r)	Функция возведения в степень.
bool operator ==(const TBigInt&) const	Перегрузка оператора ==.
bool operator <(const TBigInt&) const	Перегрузка оператора «меньше».
bool operator >(const TBigInt&) const	Перегрузка оператора «больше».
bool operator <=(const TBigInt&) const	Перегрузка оператора «меньше либо равно».
friend std::ostream& operator «(std::ostream&, const TBigInt&)	Перегрузка оператора вывода.
void DeleteZeros()	Функция удаления ведущих нулей.

В таком случае описание класса `TBigInt` выглядит следующим образом:

```
1  class TBigInt {
2    public:
3      TBigInt() {};
4      TBigInt(std::string&);
5      TBigInt(int n);
6      ~TBigInt() {};
7
8      TBigInt operator +(const TBigInt&);
9      TBigInt operator -(const TBigInt&);
10     TBigInt operator *(const TBigInt&) const;
11     TBigInt operator /(const TBigInt&);
```

```
12      TBigInt Power(int r);
13      bool operator ==(const TBigInt&) const;
14      bool operator <(const TBigInt&) const;
15      bool operator >(const TBigInt&) const;
16      bool operator <=(const TBigInt&) const;
17      friend std::ostream& operator <<(std::ostream&, const TBigInt&);
18
19    private:
20      void DeleteZeros();
21      std::vector<int> mData;
22    };
```

3 Консоль

```
[alext@alext-pc solution]$ cat tests/test.txt
38943432983521435346436
354353254328383
+
90409438473849324729384738 43
2343543
-
972323
2173937
>
2
3
-
[alext@alext-pc solution]$ ./solution <tests/test.txt
3894343333787468967 4819
90409438473849324729361303 00
false
Error
```

4 Тест производительности

В тесте производительности я сравнивал время выполнения операций в моей библиотеке и в уже существующей библиотеке для длинной арифметики GMP.

Я проводил отдельно сравнение операций сложения и вычитания, операции умножения и операции деления.

В каждом тесте по 1000 операций соответствующего вида.

```
[alext@alext-pc solution]$ ./solution <tests/+-.txt
...
My time: 0.018178 sec.
[alext@alext-pc solution]$ ./bench <tests/+-.txt
...
GMP time: 0.006496 sec.
[alext@alext-pc solution]$ ./solution <tests/mul.txt
...
My time: 0.006273 sec.
[alext@alext-pc solution]$ ./bench <tests/mul.txt
...
GMP time: 0.002323 sec.
[alext@alext-pc solution]$ ./solution <tests/div.txt
...
My time: 0.082195 sec.
[alext@alext-pc solution]$ ./bench <tests/div.txt
...
GMP time: 0.003063 sec.
```

Видно, что моя реализация проигрывает библиотечной, хоть и не слишком сильно. Вероятно, это связано с тем, что в библиотечной реализации применяются различные оптимизации, которых в моей программе нет.

5 Выводы

В ходе выполнения лабораторной работы я изучил реализацию длинной арифметики в системах счисления с основанием, кратным 10 и написал такую реализацию для основания 10^4.

Сложнее всего было понять, как правильно реализовывать опреации деления и умножения.

Также, сложным оказался алгоритм Анатолия Карацубы для умножения длинных чисел. В связи с этим, а также с тем, что он становится эффективен на относительно больших длинных числах, я не использовал его в своей программе.

Список литературы

[1] *MAXimal::algo::Бинарное возведение в степень*
URL: `https://e-maxx.ru/algo/binary_pow` (дата обращения: 16.02.2022).

[2] *Работа с очень длинными числами на C++ / Хабр*
URL: `https://habr.com/ru/post/578718/` (дата обращения: 16.02.2022).