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: Л. Я. Вельтман
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2019

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 стандартныи файл вывода.
Список арифметических операций:

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

  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.

1 Описание

Требуется написать реализацию простеиших арифметических операции для длинных чисел, таких как сложение, вычитание, умножение, деление, возведение в степень и операции сравнения. Идея реализации длинных чисел заключается в использовании вектора, элементами которого являются числа «короткие« (они играют роль цифр). Число в векторе расположено от младшего разряда к старшему. Размер чисел в векторе ограничен выбранным основанием системы счисления, в нашем случае 10^4.

Сложение осуществляется «столбиком». Пока оба числа не кончились, складываем числа. Начиная с младших разрядов, складываем «цифры», стоящие на данных позициях, прибавляем переносимыи со сложения на прошлом шаге остаток. Записываем итоговую «цифру» в результирующии вектор. Если она больше выбранного основания системы счисления, то запоминаем новыи остаток, уменьшаем ее на основание системы счисления.

Вычитание осуществляется «столбиком». Аналогично сложению, пока уменьшаемое число не кончилось, вычитаем из него второе число. Начиная с младших разрядов, вычитаем «цифру» и переносимыи с вычитания на прошлом шаге «занятыи разряд». Записываем итоговую «цифру» в результирующии вектор. Если она меньше 0, то запоминаем, что «заняли разряд», увеличиваем ее на основание системы счисления.

Умножение осуществляется также столбиком. Начиная с младшего разряда у первого числа, умножаем его поочередно на все разряды второго числа. Если при умножении разряда первого числа и разряда второго числа получилось число большее, чем основание системы счисления, то из этого числа вычитаем основание и получаем остаток, который запоминаем и вычитаем из полученного числа и идем умножать следующий разряд второго числа на тот же разряд первого числа, прибавляя к полученному новому числу остаток, полученный на предыдущем шаге и так далее, пока разряды первого числа не закончатся. Затем удаляем ведущие нули, если они есть.

Деление осуществляется «уголком». Количество разрядов у частного от деления не превосходит количества разрядов у делимого. Т.е. начинаем формировать ответ со старшего разряда. На каждои итерации имеем текущее значение, которое пытаемся уменьшить на максимально большое количество раз делимым. Для поиска этого числа используем бинарныи поиск.

Возведение в степень производится быстрее, чем за «умножение * степень». Ускорение достигается благодаря своиствам степени, а именно $x^{2^n} = x^{n2}$.

Операции сравнения осуществляются поразрядно.

2 Исходный код

Каждое число представлено вектором его цифр. За основание системы счисления взято 10000. Сравнение реализовано поразрядно. Операции сложения, вычитания и умножения осуществляются столбиком, по определению. Деление выполняется по алгоритму деления уголком. Возведение в степень опирается на свойство степени, которое обеспечивает сложность возведения в степень пропорционально не линии, а двоичному логарифму степени. Умножение, деление и вычитание используют функцию удаления ведущих нулей.

TLongArithm.cpp	
bool operator==(const TLongArithm&, const TLongArithm&);	Оператор «равно».
bool operator<(const TLongArithm&, const TLongArithm&);	Оператор «меньше».
bool operator>(const TLongArithm&, const TLongArithm&);	Оператор «больше».
bool operator<=(const TLongArithm&, const TLongArithm&);	Оператор «меньше или равно».
TLongArithm operator+(const TLongArithm&, const TLongArithm&);	Сложение двух чисел.
TLongArithm operator-(const TLongArithm&, const TLongArithm&);	Вычитание двух чисел.
TLongArithm operator*(const TLongArithm&, const TLongArithm&);	Умножение двух чисел.
TLongArithm operator/(const TLongArithm&, const TLongArithm&);	Деление двух чисел.
TLongArithm Power(TLongArithm&, TLongArithm&);	Возведение числа в степень.
operator bool() const;	Перегруженный булевский оператор для работы с длинными числами.
bool unEven() const;	Проверка на четность/нечетность длинного числа.
friend std::ostream& operator«(std::ostream&, const TLongArithm&);	Вывод числа на стандартный поток вывода.

```
const int base = 10000;
const int digLength = 4;

class TLongArithm {
public:
    TLongArithm() {};
    TLongArithm(std::string&);
    TLongArithm(int);
    ~TLongArithm() {};

    friend bool operator==(const TLongArithm&, const TLongArithm&);
    friend bool operator<(const TLongArithm&, const TLongArithm&);
    friend bool operator>(const TLongArithm&, const TLongArithm&);
    friend bool operator<=(const TLongArithm&, const TLongArithm&);

    operator bool() const;

    friend TLongArithm operator+(const TLongArithm&, const TLongArithm&);
    friend TLongArithm operator-(const TLongArithm&, const TLongArithm&);
    friend TLongArithm operator*(const TLongArithm&, const TLongArithm&);
    friend TLongArithm operator/(const TLongArithm&, const TLongArithm&);


    TLongArithm Power(TLongArithm&, TLongArithm&);
    bool unEven() const;

    friend std::ostream& operator<<(std::ostream&, const TLongArithm&);

private:
    void RemoveZeros(void);
    std::vector<int> digits;
};
```

3 Консоль

```
MacBook-Pro-Lina:da6 linuxoid$ make
g++ -std=c++11 -Wall -Werror -Wno-sign-compare -Wno-unused-result -O3 -o da6
main.cpp TLongArithm.cpp
MacBook-Pro-Lina:da6 linuxoid$ cat test
10000000000000000000
10000000000000000000
*
3894344587347537845456
3243532543283432211111
*
90409433343944343472938473843
234354343
-
863232
2173937
>
6
8
-
14565
1232
>
MacBook-Pro-Lina:da6 linuxoid$ cat test | ./da6
1000000000000000000000000000000000000000000000000
126314334038214277498807082007473470840616l6
90409433343944343472704119500
false
Error
true
```

4 Тест производительности

Для теста производительности я использую встроенную длинную арифметику в Python для сравнения со своей реализацией длинной арифметики. Тест производительности представляет из себя следующее: Сложение 1 тест - 100 000 образцов. 2 тест - 500 000 образцов. Вычитание 3 тест - 100 000 образцов. 4 тест - 500 000 образцов. Умножение 5 тест - 100 000 образцов. 6 тест - 500 000 образцов. Деление 7 тест - 100 000 образцов. 8 тест - 500 000 образцов.

```
MacBook-Pro-Lina:da6 linuxoid$ ./da6 <test1
C++ time: 0.317973 sec.
MacBook-Pro-Lina:da6 linuxoid$ python bench.py
Python time: 0.000423908233643 sec.

MacBook-Pro-Lina:da6 linuxoid$ ./da6 <test2
C++ time: 16.4089 sec.
MacBook-Pro-Lina:da6 linuxoid$ python bench.py
Python time: 0.00467085838318 sec.

MacBook-Pro-Lina:da6 linuxoid$ ./da6 <test3
C++ time: 14.1781 sec.
MacBook-Pro-Lina:da6 linuxoid$ python bench.py
Python time: 0.00832986831665 sec.

MacBook-Pro-Lina:da6 linuxoid$ ./da6 <test4
C++ time: 0.306754 sec.
MacBook-Pro-Lina:da6 linuxoid$ python bench.py
Python time: 0.000423908233643 sec.

MacBook-Pro-Lina:da6 linuxoid$ ./da6 <test5
C++ time: 0.358517 sec.
MacBook-Pro-Lina:da6 linuxoid$ python bench.py
Python time: 0.000584125518799 sec.

MacBook-Pro-Lina:da6 linuxoid$ ./da6 <test6
C++ time: 16.1506 sec.
MacBook-Pro-Lina:da6 linuxoid$ python bench.py
Python time: 0.00402593612671 sec.

MacBook-Pro-Lina:da6 linuxoid$ ./da6 <test7
C++ time: 0.789046 sec.
```

```
MacBook-Pro-Lina:da6 linuxoid$ python bench.py
Python time: 0.000396966934204 sec.

MacBook-Pro-Lina:da6 linuxoid$ ./da6 <test8
C++ time: 38.1655 sec.
MacBook-Pro-Lina:da6 linuxoid$ python bench.py
Python time: 0.00470304489136 sec.
```

Анализируя время выполнения данных тестов можно сделать вывод, что встроенная длинная арифметика Python выигрывает мою реализацию. Это неудивительно, наверняка создатели Python использовали различные оптимизации, которых нет в моеи программе.

5 Выводы

Длинная арифметика обладает широкой облатсью применения – используется для вычислении, требующих работы с большими числами и высокои точности. В настоящее время во многих языках программирования существуют встроенные модули для работы с длинными числами. Но, т.к. в C++ его нет, иногда требуется писать собственную библиотеку. Лабораторная работа не была особо сложной, так как для реализации математический операций достаточно школьных знаний, таких как: сложение, вычитание и умножение столбиком, деление уголком. Трудность заключалась лишь в том, чтобы применить эти знания к длинным числам, представленным необычным образом, а именно: в ячейке вектора максимальное количество хранимых цифр = 4, и вообще само длинное число хранится в обратном порядке. В моей работе присутствуют и недостатки. В частности, моя программа не может работать с отрицательными и дробными числами, но так как по заданию этого не требовалась, то в рамках этой работы, все было сделано верно. Задача реализации длинной арифметики довольно популярна среди олимпиадного программирования и неоднократно была описана во многих источниках. Было довольно интересно приобщиться к людям, уже написавшим реализацию длинной арифметики, запрограммировав такой необычный калькулятор, работающий с длинными числами.

Список литературы

[1] *Длинная арифметика .*
URL: `http://e-maxx.ru/algo/big`$_i nteger$.
(: 15.03.2019).