Московский авиационный институт
(национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №6 по курсу «Дискретный анализ»

Студент: А. А. Пустовалова
Преподаватель: А. А. Кухтичев
Группа: 8О-206
Дата:
Оценка:
Подпись:

Москва, 2017

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.

1 Описание

Требуется реализовать алгоритмы длинной арифметики, позволяющие вычислять значение арифметических операций и условий для больших чисел. Хранить большое число удобно в векторе, где каждый элемент – цифра числа в выбранной системе счисления. Цифры числа хранятся в векторе, начиная с меньшего разряда. Операции сложения, вычитания, умножения чисел, а также деления длинного числа на число из одного разряда реализовываются аналогично выполнению этих действий в столбик. Сложение и вычитание работают за $O(max(n, m))$. Деление на число из одного разряда работает за (n). Умножение – за $O(n * m)$. Деление на большое число реализуется путём быстрого подбора значений разрядов частного.
$q'_i = \lfloor (u_{j+n}b + u_{j+n-1})/v_{n-1} \rfloor$. [1] q_i не более, чем на 2 меньше q'_i. Алгоритм деления на большое число работает за $O(m * n)$.
Алгоритм быстрого возведения числа в маленькую степень – рекурсивный алгоритм. Если n – степень без остатка делится на 2, то результатом будет $m^{n/2} * m^{n/2}$. В противном случае – $m^{n-1} * m$. Таким образом, алгоритм работает за $O(\log n)$ умножений. Операции сравнения реализовываются за $O(max(n, m))$ сравнением значений разрядов числа.

2 Исходный код

В цикле считываем входные данные строками символов, затем создаём объекты класса $TBigInt$ для чисел. Согласно введённой операции обрабатываем возможные ошибки, вычисляем результат и выводим ответ или сообщение об ошибке.

1.c	
TBigInt(size_t n)	Конструктор класса, принимающий длину числа.
TBigInt(const std::string & str)	Конструктор класса, принимающий число в виде строки символов.
TBigInt()	Деструктор класса.
size_t size()	Возвращает количество разрядов в числе.
int get(size_t i)	Вовращает значение i–го разряда числа.
set(size_t i, int value)	Устанавливает значение i–го разряда числа.
friend std::ostream& operator≪	Переопределённый оператор вывода.
TBigInt operator+(const TBigInt & other) TBigInt operator-(const TBigInt & other) TBigInt operator/(const TBigInt & other) TBigInt operator/(const int & other) TBigInt operator*(const TBigInt & other) TBigInt operator^(const int & other) bool operator<(const TBigInt & other) bool operator<=(const TBigInt & other) bool operator==(const TBigInt & other)	Переопределённые операторы для арифметических операций и сравнений.
void TBigInt::DeleteZeros()	Удаление лидирующих нулей.
int GetInt(std::string s)	Преобразование строки в число.

```
1  class TBigInt {
2  public:
3      TBigInt() { }
4      TBigInt(size_t n) { mData.resize(n); }
5      TBigInt(const std::string & str);
6      ~TBigInt() {}
7      inline size_t size() const { return mData.size(); }
```

```
8     inline int get(size_t i) const { return mData[i]; }
9     void set(size_t i, int value) { mData[i] = value; }
10    friend std::ostream& operator<< (std::ostream& stream, const TBigInt & number);
11    TBigInt operator+(const TBigInt & other);
12    TBigInt operator-(const TBigInt & other);
13    TBigInt operator/(const TBigInt & other);
14    TBigInt operator/(const int & other);
15    TBigInt operator*(const TBigInt & other);
16    TBigInt operator^(const int & other);
17    bool operator<(const TBigInt & other);
18    bool operator<=(const TBigInt & other);
19    bool operator==(const TBigInt & other);
20 protected:
21    std::vector<int> mData;
22 private:
23    void DeleteZeros();
24 };
```

3 Консоль

```
1050315
7657
/
137
0
16512368543
/
0
1564651326
1265648
*
1980297821449248
13200645000000
5645321
+
13200650645321
0
165416565
<
true
18945623561
18945623561
=
true
0
0
^
Error
2
8
^
256
```

4 Тест производительности

Тест производительности представляет из себя следующее: вычисление арифметических операций для длинной арифметики сравнивается со встроенной реализацией для чисел, вместимых в long long.

Входные данные состоят из 10^4 тестов для каждой операции для чисел до 10^{15}, время измеряется в тактах процессора.

```
Длинная арифметика: 76104
Встроенная реализация: 69132
```

Тестирование показало, что длинная арифметика несколько уступает встроенной реализации на числах до 10^{15}.

5 Выводы

Выполнив пятую лабораторную работу, я научилась представлять большие числа в памяти компьютера, реализовывать вычисление арифметических операций для больших чисел с помощью алгоритмов длинной арифметики.

Список литературы

[1] Дональд Эрвин Кнут *Искусство программирования. Том 2. Получисленные алгоритмы* — Издательство «Вильямс», 2001. — 832 с. (ISBN 0-201-89684-2)