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: И. П. Моисеенков
Преподаватель: Н. С. Капралов
Группа: М8О-208Б-19

Дата: 09.03.2021
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

Согласно [1], длинная арифметика — это набор программных средств (структуры данных и алгоритмы), которые позволяют работать с числами гораздо больших величин, чем это позволяют стандартные типы данных.

Будем рассматривать числа в системе счисления с основанием 10000. Каждый разряд такого числа будет храниться в отдельной ячейке массива. Разряды нумеруются справа налево.

Операции сравнения выполняются следующим образом. Сначала сравним размеры массивов с числами. Если они равны, то будем сравнивать разряды, начиная со старшего, пока не найдем несовпадение. Сложность сравнения - $O(n)$, где n - размер массива с числом.

Операции сложения и вычитания выполняются «в столбик». Для этого необходимо пройтись по разрядам, начиная с младших, и сложить/вычесть их, не забывая о возможном переполнении рязрядов. По заданию библиотека должна поддерживать работу с целыми неотрицательными числами, поэтому при попытке вычесть из меньшего числа большее программа выдаст ошибку. Сложность алгоритма сложения - $O(max(n, m))$, где n, m - размеры массива с первым и вторым числом соответственно. Слжоность алгоритмы вычитания - $O(n)$.

Умножение выполняется тоже «в столбик». Сложность такого алгоритма - $O(n * m)$.

Деление будет только целочисленным. Делить числа будем «уголком». Будем сносить по одному разряду из делимого, начиная со старшего, и бинарным поиском определять разряды частного. Деление на ноль вызовет ошибку. Сложность такого алгоритма $O(n * (n + log(BASE) * m)) \sim O(n * (n + m))$, где n, m - размеры массива с первым и вторым числом соответственно, $BASE$ - основание системы счисления.

Для возведения в степень воспользуемся бинарным алгоритмом, работа которого основывается на двух равенствах:
$a^k = (a^{k/2})^2$, если n - чётное; $a^k = a^{k-1} * a$, если n - нечётное [2].

Таким образом, возведение в степень выполняется за $O(log(k) * n^2)$, где k - показатель степени, n - размер массива с числом.

2 Исходный код

Для работы с длинными числами создадим класс TBigInt. Единственный его атрибут - это вектор целых чисел - разрядов длинного числа. Основание системы счисления и максимальную длину разряда объявим как глобальные константы.

Для класса переопределим операции сложения, вычитания, умножения, деления, сравнения на больше, сравнения на меньше, проверки на равенство согласно алгоритмам, описанным выше. Дополнительно определим метод для возведения в степень и несколько вспомогательных методов - для удаления незначащих нулей, получения i-ого разряда, получения длины числа. Для удобства переопределен оператор[].

В классе реализовано два конструктора: один из них преобразует число типа int в длинное число, другой преобразует строку в длинное число.

```
1  const int BASE = 10000;
2  const int DIGIT_LENGTH = ceil(log10(BASE));
3
4  class TBigInt {
5  private:
6      std::vector<int> digits;
7      void RemoveZeroes();
8  public:
9      TBigInt(int value = 0);
10     TBigInt(const std::string &value);
11     int &operator[](int id);
12     int GetDigit(int id) const;
13     int GetNumSize() const;
14     void Resize(int size);
15     TBigInt operator+(const TBigInt &other) const;
16     TBigInt operator-(const TBigInt &other) const;
17     TBigInt operator*(const TBigInt &other) const;
18     TBigInt operator/(const TBigInt &other) const;
19     TBigInt Pow(TBigInt& other);
20     bool operator<(TBigInt &other) const;
21     bool operator>(TBigInt &other) const;
22     bool operator==(TBigInt &other) const;
23     friend std::ostream &operator<<(std::ostream &out, TBigInt num);
24 };
```

3 Консоль

```
mosik@LAPTOP-69S778GL:~/da_lab6$ make
g++ -pedantic -Wall -std=c++17 -Werror -Wno-sign-compare -O2 -lm -g -Wextra
main.cpp TBigInt.cpp -o solution
mosik@LAPTOP-69S778GL:~/da_lab6$ ./solution
281639
23412
+
305051
812689317
143283
-
812546034
189495
2351412
*
445580816940
19613847
243
/
80715
1384627
12
^
4965849489673689654421131339804206467141712883839243539176083019273418872l
948573845
4389279827
<
true
49357873
2897548743
>
false
243234
243234
=
true
```

4 Тест производительности

Так как в языке C++ нет встроенной поддержки длинной арифметики, то сравним мою библиотеку с длинной арифметикой языка Python.

Код программы на языке Python приведен ниже.

```
1  import time
2
3  start = time.time()
4
5  while True:
6      try:
7          num1 = int(raw_input())
8      except:
9          break
10     num2 = int(raw_input())
11     op = raw_input()
12
13     if op == '+':
14         print(num1 + num2)
15     elif op == '-':
16         if num1 < num2:
17             print("Error")
18         else:
19             print(num1 - num2)
20     elif op == '*':
21         print(num1 * num2)
22     elif op == '/':
23         if num2 == 0:
24             print("Error")
25         else:
26             print(num1 // num2)
27     elif op == '^':
28         if num1 == num2 == 0:
29             print("Error")
30         else:
31             print(num1 ** num2)
32     else:
33         print("Error")
34
35 end = time.time()
36 print("Time: " + str(end - start) + "s")
```

Тест 1. Сложение и вычитание - 100000 запросов, числа до 10^{1000}.

```
mosik@LAPTOP-69S778GL:~/da_lab6$ ./cpp_calc <input
Time: 1.53125s
```

```
mosik@LAPTOP-69S778GL:~/da_lab6$ python calculator.py <input
Time: 3.77142s
```

Тест 2. Умножение - 10000 запросов, числа до 10^{1000}.

```
mosik@LAPTOP-69S778GL:~/da_lab6$ ./cpp_calc <input
Time: 4.04688s

mosik@LAPTOP-69S778GL:~/da_lab6$ python calculator.py <input
Time: 0.95746s
```

Тест 3. Деление - 10000 запросов, числа до 10^{1000}.

```
mosik@LAPTOP-69S778GL:~/da_lab6$ ./cpp_calc <input
Time: 39.8906s

mosik@LAPTOP-69S778GL:~/da_lab6$ python calculator.py <input
Time: 0.2508s
```

Тест 4. Возведение в степень - 500 запросов, числа до 10^{10}, показатель степени не превышает 1000.

```
mosik@LAPTOP-69S778GL:~/da_lab6$ ./cpp_calc <input >result
Time: 2.26562s

mosik@LAPTOP-69S778GL:~/da_lab6$ python calculator.py <input >result
Time: 0.26286s
```

Хотя на сложении и вычитании моя библиотека неплохо выиграла, на остальных тестах Питон оказался в разы быстрее. Это происходит из-за того, что в Python используются более оптимизированные алгоритмы.

5 Выводы

Несмотря на то, что часто возникает необходимость работать со сколь угодно большими числами, язык C++ поддерживает только ограниченный диапазон целых чисел. Выполнив данную лабораторную работу, я изучил, как можно обойти это ограничение.

Для этого необходимо использовать длинную арифметику, основная суть которой состоит в том, что мы работаем не с числом, а с массивом, в котором расположены разряды этого числа. Для удобства и ускорения следует перейти в систему счисления с основанием как минимум 10000.

Длинные числа могут поддерживать все те операции, что и встроенные численные типы. Но выполняться они будут дольше.

Хотя в рамках данной работы мне нужно было реализовать калькулятор, поддерживающий только целые неотрицательные числа, его можно модифицировать и для отрицательных, и для дробных чисел.

Но на самом деле язык C++ - не лучший инструмент для работы с большими числами. Хотя пользователями было написано несколько библиотек для длинной арифметики, быстрее и проще будет обратиться к такому языку, который уже умеет работать с большими числами - например, Python. К тому же, он справляется с вычислениями быстрее, чем моя библиотека на C++.

Список литературы

[1] *Длинная арифметика - e-maxx.*
URL: `http://e-maxx.ru/algo/big_integer` (дата обращения: 05.03.2020).

[2] *Бинарное возведение в степень - e-maxx.*
URL: `https://e-maxx.ru/algo/binary_pow` (дата обращения: 05.03.2020).