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: М. А. Инютин
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б-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:

1. Сложение (+).
2. Вычитание (-).
3. Умножение (*).
4. Возведение в степень ($\wedge$).
5. 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:

1. Больше (>).
2. Меньше (<).
3. 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 реализовать алгоритм для работы с длинными числами. Можно работать с числами, как со строками, то есть в системе счисления 10, но если использовать систему счисления больше, можно ускорить программу в несколько раз.

Проще всего хранить массив цифр числа, как и сказано в [1]. Вводить число нужно будет строкой, поэтому нужно уметь преобразовывать строку в массив цифр. При печати числа на экран нужно помнить о недостающих нулях в цифрах числа и выводить их.

Складывать числа будем школьным в столбик — это наиболее простой алгоритм, который имеет асимптотику $O(n)$. Сравнение чисел ничуть не отличается от лексикографического сравнения строк. Вычитать тоже будем в столбик, предварительно проверив, что первое число не меньше второго.

Для умножения чисел можно использовать алгоритм Карацубы, который имеет сложность $O(n^{\log_2 3}) \approx O(n * \sqrt{n})$, но в алгоритме не считаются сдвиги чисел, которые могут сильно замедлить алгоритм до $O(n^2 * \sqrt{n})$. Умножение в столбик не требует сдвигов, так как мы сразу знаем, в какой разряд поместить произведение разрядов. Простое умножение всегда выполняется за $O(n * m) \approx O(n^2)$.

Деление — самая сложная операция над длинными числами. При делении в столбик нужно уметь быстро определять число, на которое будет делится текущий остаток на делитель. Можно попробовать предпосчитать все множители делителя, но это требует $O(n^2)$ памяти, что слишком велико даже для моего компьютера. Будем использовать бинарный поиск и получим сложность $O(n*(n+m)*\log_2 b) \approx O(n^2*\log_2 b)$, где b — основание выбранной системы счисления.

Самый простой способ возведения в степень требует очень большая количества умножений, уже на небольших числах возникают трудности с вычислением. Используем алгортим бинарного возведения в степень, который использует идею, что показатель степени можно представить как сумму степеней двойки. Подробно этот алгоритм описан в [2]. Всего нужно будет поделить показатель $\log_2 m$ раз и столько же выполнить умножение основания само на себя. При умножении число знаков в основании увеличивается в два раза, поэтому сложность операции $O(\log_2 y*(y^2+(n*2^{\log_2 y})^2)) \approx O(\log_2 y * n^2 * y^2)$, где y — показатель степени.

2 Исходный код

Выполнение работы разобьём на следующие шаги:

1. Реализация класса для хранения длинных чисел. Перегрузка основных операторов.
2. Перегрузка операторов сложения, вычитания и сравнения.
3. Перегрузка операторов умножения, деления и возведения в степень.
4. Реализация ввода данных и вычислений.
5. Тесты производительности, сравнение с библиотекой.

Опишем класс $TBigNum$ для работы с длинными числами. Перегрузим необходимые операторы, чтобы работать с числами как с обычным целочисленным типом. Если требуется изменить длинну разряда, то достаточно изменить значения переменных $BASE$, $RADIX$, MAX_DIGIT, характеризующих систему счисления для хранения длинных чисел.

big_num_lib.hpp	
void Normalize()	Удаление «ведущих» нулей и реверс числа
static void DivMod(const TBigNum & lhs, const TBigNum & rhs, TBigNum & div, TBigNum & mod)	Функция для деления с остатком. Возвращает результат в div и mod
TBigNum()	Конструктор по умолчанию
TBigNum(const size_t & size)	Конструктор числа заданной длинны
TBigNum(const std::string & s)	Конструктор числа из строки
TBigNum(const TBigNum & num)	Конструктор копирования
TBigNum & operator = (const TBigNum & num)	Перегрузка оператора присваивания
size_t Size() const	Получение длинны числа
friend TBigNum operator + (const TBigNum & lhs, const TBigNum & rhs)	Перегрузка оператора сложения
friend TBigNum operator - (const TBigNum & lhs, const TBigNum & rhs)	Перегрузка оператора вычитания
friend TBigNum operator * (const TBigNum & lhs, const TBigNum & rhs)	Перегрузка оператора умножения
friend TBigNum operator ∧ (const TBigNum & lhs, const TBigNum & rhs)	Перегрузка оператора возведения в степень

friend TBigNum operator / (const TBigNum & lhs, const TBigNum & rhs)	Перегрузка оператора деления
friend TBigNum operator < (const TBigNum & lhs, const TBigNum & rhs)	Перегрузка оператора меньше
friend TBigNum operator > (const TBigNum & lhs, const TBigNum & rhs)	Перегрузка оператора больше
friend TBigNum operator == (const TBigNum & lhs, const TBigNum & rhs)	Перегрузка оператора равенства
friend std::ostream & operator « (std::ostream & out, const TBigNum & num)	Перегрузка оператора вывода
friend std::istream & operator » (std::istream & in, TBigNum & num)	Перегрузка оператора ввода

Ввод чисел и вывод результата получается достаточно громоздким из-за большого количества условий. Для проверки деления на ноль или возведений нуля в нулевую степень дополнительно нужно задать глобальную переменную BIG_ZERO.

```
#include "big_num_lib.hpp"

const TBigNum BIG_ZERO = TBigNum("0");

int main() {
    TBigNum num1, num2;
    char action;
    while (std::cin >> num1 >> num2 >> action) {
        if (action == '+') {
            std::cout << num1 + num2 << '\n';
        }
        if (action == '-') {
            if (num1 < num2) {
                std::cout << "Error\n";
            } else {
                std::cout << num1 - num2 << '\n';
            }
        }
        if (action == '*') {
            std::cout << num1 * num2 << '\n';
        }
        if (action == '^') {
            if (num1 == BIG_ZERO and num2 == BIG_ZERO) {
                std::cout << "Error\n";
            } else {
                std::cout << (num1 ^ num2) << '\n';
            }
        }
```

```
29          if (action == '/') {
30              if (num2 == BIG_ZERO) {
31                  std::cout << "Error\n";
32              } else {
33                  std::cout << num1 / num2 << '\n';
34              }
35          }
36          if (action == '<') {
37              if (num1 < num2) {
38                  std::cout << "true\n";
39              } else {
40                  std::cout << "false\n";
41              }
42          }
43          if (action == '>') {
44              if (num1 > num2) {
45                  std::cout << "true\n";
46              } else {
47                  std::cout << "false\n";
48              }
49          }
50          if (action == '=') {
51              if (num1 == num2) {
52                  std::cout << "true\n";
53              } else {
54                  std::cout << "false\n";
55              }
56          }
57      }
58  }
```

3 Консоль

```
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab6$ cat sample.txt
38943432983521435346436
354353254328383
+
90409438473849324729384738 43
2343543
-
972323
2173937
>
2
3
-
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab6$ make
g++ -g -O2 -pedantic -std=c++17 -Wall -Wextra -Werror main.cpp -o solution
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab6$ ./solution <sample.txt
3894343333787468967 4819
90409438473849324729361303 00
false
Error
```

4 Тест производительности

Реализованная библиотека для работы с длинными числами сравнивается с библиоткеой GNU MP. Отдельно сравним линейные и нелинейные операции на числах разных длинн.

Количество строк в тесте	Длинна чисел в тесте	Моя библиотека, мс	GNU MP, мс
Тесты сложения и вычитания			
10^5	10^2	297.845 мс	550.636 мс
10^4	10^3	265.223 мс	510.745 мс
10^3	10^4	247.380 мс	516.828 мс
10^2	10^5	248.957 мс	721.084 мс
Тесты умножения			
10^4	10^2	34.261 мс	51.043 мс
10^3	10^3	61.965 мс	50.694 мс
10^2	10^4	352.730 мс	54.447 мс
Тесты деления			
10^5	10	220.151 мс	85.443 мс
10^4	10^2	438.113 мс	51.759 мс
10^3	10^3	3840.333 мс	45.642 мс

В тестах сложения и вычитания моя программа оказалась быстрее, потому что длинные числа представлены в системе счисления 10000. В тестах умножения моя программа замедляется при увеличении длинны чисел из-за квадратичной сложности, а GNU MP использует алгоритм Карацубы по стандарту, который медленнее на маленьких числах. Для деления GNU MP быстро находит $\frac{1}{B}$ и поэтому оказывается быстрее, чем моя реализация.

5 Выводы

В ходе выполнения лабораторной работы я реализовал длинную арифметику в системе счисления 10^4 и 10^8, познакомился с длинной арифметикой в факторизованном виде и библиотекой GNU MP.

Сложнее всего было реализовать операции умножения, возведения в степень и деления длинных чисел. Дополнительно я изучил алгоритм Карацубы, но не стал его реализовывать его из-за необходимости выполнения сдвигов, которые замедлили бы программу.

Список литературы

[1] *MAXimal :: algo :: Длинная арифметика*
URL: `https://e-maxx.ru/algo/big_integer` (дата обращения: 04.03.2021).

[2] *MAXimal :: algo :: Бинарное возведение в степень*
URL: `https://e-maxx.ru/algo/binary_pow` (дата обращения: 04.03.2021).