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: Д. С. Ляшун
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

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 написать реализацию длинной арифметики. Каждая из операций над длинными числами будет реализована с использованием следующих идей:

1. **Операция сложения** – посредством обычного сложения столбиком, т.е. последовательного поразрядного сложения начиная с младших разрядов чисел, в случае переполнения значения в разряде результирующего числа (т.е. когда значение в нём превосходит основание системы счисления представления) увеличить следующий старший разряд на значение целочисленного деления переполненного разряда, а сам его представить как остаток от деления на основание системы счисления представления.

   Сложение длинных чисел будет работать за $O(n + m)$ (где n, m – длины складываемых чисел), поскольку оно требует линейного прохода по каждому числу.

2. **Операция вычитания** – на основе вычитания столбиком, будет происходить поразрядное вычитание разрядов уменьшаемого и вычитаемого с записью результата в соответствующий разряд в результирующем числе-разности, при этом в случае превышения значения разряда вычитаемого по сравнению с уменьшаемым в старший разряд разности будет добавлено -1 и в младший разряд значения основания т.е. тем самым будет происходить занятие значения у старшего разряда уменьшаемого.

   Вычитание длинных чисел будет работать за $O(n+m)$ (где n, m – длины вычитаемого и уменьшаемого), поскольку оно требует линейного прохода по каждому числу.

3. **Операции сравнения** (больше, меньше и равно) – сперва оценить количество значимых разрядов в числах, в случае их совпадения произвести поразрядный проход по числам начиная со старших разрядов (для проверки на равенство порядок прохода не принципиален) и сравнивать каждый разряд, остановку произвести в случае несовпадения и вернуть результат.

   В худшем случае, когда требуется поразрядный проход по числам, операции сравнения будут работать за $O(n)$, где n – длина одного из входных чисел (совпадает с длиной другого числа); в противном случае операции сравнения работают за $O(1)$, поскольку дают ответ по длинам чисел.

4. **Операция умножения** – как обычное умножение столбиком, т.е. зафиксировать один из множителей и последовательно пробегать по разрядам другого множителя, при этом производя умножение разрядов из первого множителя на текущий рассматриваемый разряд во втором множителе, добавляя результаты произведений в разряды результирующего числа, номер разряда определяется как сумма номеров умножаемых разрядов из множителей.

Такое умножение длинных чисел будет работать за $O(nm)$ (где n, m – длины умножаемых чисел), поскольку умножение зафиксированного множителя на один разряд другого множителя происходит путем линейного прохода по всему числу длины n, т.е. выполняется m итераций за время $O(n)$ каждая.

5. **Операция деления** – также, как и обычное деление столбиком. Будет производиться последовательный проход по делимому от старшего разряда к младшему и добавление из него разрядов текущему малому делимому, который на предыдущей итерации прохода являлся остатком от деления, вначале оно является пустым числом. Путем бинарного поиска будут подбираться значения в разряде частного, при умножении на которое делителя получится наибольшее не превосходящее малое делимое число. Остаток от деления малого делимого как было описано выше будет использоваться дальше при делении.

   Такая реализация операции деления будет работать за время $O(log_2(b)nm)$ (где b – основание, в котором представляются числа, а m, n – длина делителя и частного соответственно), поскольку выполняется линейный проход по разрядам делимого длины n, и на каждой итерации происходит бинарный поиск для значения разряда частного – выполняется $O(log_2(b))$ итераций бинарного поиска, требующих произвести умножение длинного числа частного на длинное, но одноразрядное число за время работы умножения – $O(1 * m) = O(m)$, а также произвести разность двух длинных чисел за время $O(2m)$ (будем считать, что длина малого делимого не превосходит длины частного, поскольку иначе оно будет разделено), что в итоге даёт эту сложность.

6. **Операция возведения в степень** – по принципу бинарного возведения в степень, т.е. на каждой итерации будет проверяться текущий показатель степени на чётность, в случае нечётного значения происходит умножение числа-ответа на рабочее число, которое на каждой итерации умножается на само себя (в начале работы равное основанию степени), иначе же умножение не происходит. После этого показатель степени делится на 2 и происходит повторная итерация (пока показатель не станет равным 0). Такая идея возведения в степень основана на двоичных представлениях чисел - в случае 1 в текущем рассматриваемом разряде (что говорит и нечётности) умножение происходит, а при 0 нет.

   Операция возведения в степень работает за время $O(log_2(y)(log_2(b)n+m^2)$, (где y – значение показателя степени, b – основание системы счисления представления чисел, n, m – длина показателя и основания степени соответственно), поскольку всего происходит $log_2(y)$ итераций работы, на каждой из которых выполняется проверка текущего показателя на чётность за O(1), его деление на 2 за время $O(log_2(b)*1*n) = O(log_2(b)n)$ – сложность работы деления при делимом длины n и делителе, равном 2 длины 1, а также умножение текущего основания степени на само себя за $O(m^2)$.

2 Исходный код

Для представления длинных чисел был описан класс $TBigInteger$ со следующими полями и методами:

bigInteger.hpp	
TBigInteger() = default	Конструктор по умолчанию.
TBigInteger(const std::string&)	Конструктор, принимающий строковое представление числа.
void RemoveLeadZeros()	Процедура удаления ведущих нулей из числа.
friend std::istream& operator»(std::istream&, TBigInteger&)	Перегруженный оператор ввода.
friend std::ostream& operator«(std::ostream&, const TBigInteger&)	Перегруженный оператор вывода.
TBigInteger operator+(const TBigInteger&) const	Перегруженный оператор сложения.
TBigInteger operator-(const TBigInteger&) const	Перегруженный оператор вычитания.
TBigInteger operator*(const TBigInteger&) const	Перегруженный оператор умножения.
TBigInteger operator/(const TBigInteger&) const	Перегруженный оператор деления.
friend TBigInteger Pow(const TBigInteger&, const TBigInteger&)	Функция возведения в степень.
bool operator<(const TBigInteger&) const	Перегруженный оператор меньше.
bool operator>(const TBigInteger&) const	Перегруженный оператор больше.
bool operator==(const TBigInteger&) const	Перегруженный оператор равенства.
std::vector<int32_t> Number	Вектор, хранящий разряды числа.
static const int BASE = 10000	Константа основания системы счисления представления.
static const int RADIX = 4	Число цифр в одном разряде числа.

В основном коде программы будет производиться чтение входных чисел как строк *string* и перевод их в объекты $TBigInteger$, чтение требуемых операций, их выполнение и вывод результатов:

```cpp
#include "bigInteger.hpp"
int main() {
    NMyStd::TBigInteger first, second, result;
    char operation;
    while (std::cin >> first >> second >> operation) {
        if (operation == '+') {
            std::cout << first + second << "\n";
        }
        else if (operation == '-') {
            try {
                std::cout << first - second << "\n";
            }
            catch (const std::exception& e) {
                std::cout << e.what() << "\n";
            }
        }
        else if (operation == '*') {
            result = first * second;
            std::cout << result << "\n";
        }
        else if (operation == '/') {
            try {
                result = first / second;
                std::cout << result << "\n";
            }
            catch (const std::exception& e) {
                std::cout << e.what() << "\n";
            }
        }
        else if (operation == '^') {
            try {
                result = Pow(first, second);
                std::cout << result << "\n";
            }
            catch (const std::exception& e) {
                std::cout << e.what() << "\n";
            }
        }
        else if (operation == '>') {
            std::cout << (first > second ? "true" : "false") << "\n";
        }
        else if (operation == '<') {
            std::cout << (first < second ? "true" : "false") << "\n";
        }
        else if (operation == '=') {
            std::cout << (first == second ? "true" : "false") << "\n";
        }
    }
}
```

3 Консоль

```
dmitry@dmitry-VirtualBox:~/Work_place/DA_labs/lab6$ make
g++ -std=c++11 -O2 -Wextra -Wall -Werror -Wno-sign-compare -Wno-unused-result
-pedantic -c bigInteger.cpp
g++ -std=c++11 -O2 -Wextra -Wall -Werror -Wno-sign-compare -Wno-unused-result
-pedantic bigInteger.o main.cpp -o solution
dmitry@dmitry-VirtualBox:~/Work_place/DA_labs/lab6$ ls
bigInteger.cpp  bigInteger.o  Makefile  test.py  test.txt
bigInteger.hpp  main.cpp      solution  tests
dmitry@dmitry-VirtualBox:~/Work_place/DA_labs/lab6$ cat test.txt
38943432983521435346436
354353254328383
+
904094384738493247293847 3843
2343543
-
972323
2173937
>
2
3
-
dmitry@dmitry-VirtualBox:~/Work_place/DA_labs/lab6$ ./solution <test.txt
38943433337874689674819
904094384738493247293613 0300
false
Error
dmitry@dmitry-VirtualBox:~/Work_place/DA_labs/lab6$
```

4 Тест производительности

Тест производительности представляет из себя сравнение суммарного времени работы написанной библиотеки длинной арифметики со сторонней библиотекой GMP. Количество тестов для каждой операции составляет 10^3.

Длина входных чисел	Время работы написанной библиотеки	Время работы встроенной библиотеки GMP
до 10^2	Сложение - 6,288 мс Вычитание - 4,789 мс Умножение - 14,472 мс Деление - 663,651 мс	Сложение - 6,881 мс Вычитание - 6,127 мс Умножение - 5,571 мс Деление - 6,869 мс
до 10^2	Сложение - 37,22 мс Вычитание - 37,06 мс Умножение - 592,46 мс Деление - 23578,46 мс	Сложение - 43,35 мс Вычитание - 50,34 мс Умножение - 58,98 мс Деление - 58,26 мс
до 10^4	Сложение - 348,3 мс Вычитание - 374,1 мс Умножение - 5663,3 мс Деление - 128338,3 мс	Сложение - 890,0 мс Вычитание - 899,6 мс Умножение - 1163,3 мс Деление - 1400,5 мс

Как видно, с учётом различных длин входных чисел написанная библиотека по работе с длинными числами немного быстрее GMP при выполнении операций сложения и вычитания, однако заметно проигрывает ей при умножении и делении. Это объясняется тем, что в моей библиотеке были использованы обычные наивные реализации этих операций, которые никак не учитывают размер входных операндов, в то время как GMP учитывает их размер и в соответствии с этим подбирает соответствующие алгоритмы работы – так было указано в документации к библиотеке [2].

5 Выводы

Выполнив шестую лабораторную работу по курсу «Дискретный анализ», я научился реализовывать простейшие алгоритмы длинной арифметики, основанной на выполнении вычислений «столбиком», т.е. исключительно с разрядами чисел. В ходе выполнения работы основную трудность для меня составила реализация операции деления, поскольку она требует правильного использования алгоритма бинарного поиска при нахождении разряда в частном и использует другие операции с длинными числами - вычитание и умножение.

Как мне кажется, применять сторонние библиотеки по типу GMP для работы с длинной арифметикой предпочтительнее несмотря на то, что мы не знаем, как они в действительности работают – для реальных задач это не является необходимостью, важно лишь то, что они выполняются гораздо эффективнее по времени.

Список литературы

[1] *С.М. Окулов. «Длинная» арифметика*
URL: `http://comp-science.narod.ru/DL-AR/okulov.htm` (дата обращения: 8.03.2021).

[2] *GMP Manual*
URL: `https://gmplib.org/gmp-man-6.2.1.pdf` (дата обращения: 10.03.2021).