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: А. Н. Марков
Преподаватель: Н. А. Зацепин
Группа: М8О-308Б
Дата:
Оценка:
Подпись:

Москва, 2020

Условие

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.

Метод решения

Число хранится в векторе. База длинного числа 1000000000. Каждый элемент вектора - это одна цифра в 1000000000-чной системе счисления. Для того, чтобы операции над длинными числами было удобнее реализовывать, разряды числа хранятся в векторе в обратном порядке.

Сложение реализовано столбиком. Пробегаемся по каждому разряду чисел и складываем их. Начинаем с младших разрядов, складываем цифры, стоящие в этих разрядах, и прибавляем к сумме остаток от сложения предыдущих разрядов (изначально остаток равен 0). Записываем остаток от деления этой суммы на базу в результирующий вектор. А новым остатком будет являться частное от суммы и базы. Сложность

операции сложения $O(max(n, m))$, где n - количество разрядов в первом числе, m - количество разрядов во втором числе.

Вычитание реализовано столбиком. Так же, как и при сложении, пробегаемся по каждому разряду чисел, но уже вычитаем их. Начинаем с младших разрядов, вычитаем цифры, стоящие в этих разрядах, а также вычитаем цифру, которую занимали в предыдущем вычитании. Если разность получилась отрицательная, то прибавляем к полученной разности базу, тем самым как бы занимаем единицу из следующего разряда. Операция вычитания предполагает, что уменьшаемое число должно быть больше или равно вычитаемому. Если вычитаемое больше, чем уменьшаемое, то выбрасывается исключение. Сложность операции вычитания $O(n)$, где n - количество разрядов в уменьшаемом числе.

Операция умножения в конечном виде лабораторной работы реализовано столбиком. Для i-го разряда второго числа ($i = 0..m-1$) происходит умножение на каждый разряд j первого числа ($j = 0..n-1$). Полученное произведение i-го разряда второго числа и j-го разряда второго числа складывается с остатком предыдущего умножения. $i + j$-й разряд результирующего вектора увеличивается на частное от деления этой суммы на базу. А новым остатком будет являться результат деления суммы на базу. Сложность операции умножения столбиком равна $O(n * m)$, где n - количество разрядов в первом числе, m - количество разрядов во втором числе.

Изначально в лабораторной работе было реализовано умножение Карацубы. Но из-за неэффективной реализации (происходило множество копирований длинных чисел), я убрал данную реализацию. Суть алгоритма Карацубы в том, что каждое число можно разбить на две части длиной $n/2$. Тогда если даны два числа $A = a_{n-1}a_{n-2}...a_0$ и $B = b_{n-1}b_{n-2}...b_0$. То они разбиваются на:

$$A_0 = [a_0, ..., a_{n/2-1}], A_1 = [a_{n/2}, ..., a_{n-1}];$$
$$B_0 = [b_0, ..., b_{n/2-1}], B_1 = [b_{n/2}, ..., b_{n-1}].$$
$$A = A_0 + A_1 \cdot BASE^{n/2}, B = B_0 + B_1 \cdot BASE^{n/2}.$$
$$A \cdot B = (A_0 + A_1 \cdot BASE^{n/2}) \cdot (B_0 + B_1 \cdot BASE^{n/2}) =$$
$$A_0 \cdot B_0 + A_0 \cdot B_1 \cdot BASE^{n/2} + A_1 \cdot B_0 \cdot BASE^{n/2} + A_1 \cdot B_1 \cdot BASE^n =$$
$$A_0 \cdot B_0 + (A_0 \cdot B_1 + A_1 \cdot B_0) \cdot BASE^{n/2} + A_1 \cdot B_1 \cdot BASE^n.$$

$$(A_0 + A_1) \cdot (B_0 + B_1) = A_0 \cdot B_0 + A_0 \cdot B_1 + A_1 \cdot B_0 + A_1 \cdot B_1.$$

А значит,

$$A \cdot B = A_0 \cdot B_0 + BASE^{n/2} \cdot ((A_0 + A_1) \cdot (B_0 + B_1) - A_0 \cdot B_0 - A_1 \cdot B_1) +$$
$$+ A_1 \cdot B_1 \cdot BASE^n.$$

Сложность алгоритма Карацубы $T(n) = 3 \cdot T(n/2) + \theta(n) = O(n^{log_2 3})$. Но в моей реализации константа, на которую умножается $n^{log_2 3}$, слишком велика, поэтому данная реализация неэффективна.

Операция деления реализована уголком. Количество разрядов у частного от деления не превосходит количества разрядов делимого. Начинаем формировать результат со старшего разряда. На каждой итерации имеем текущее значение, которое пытаемся уменьшить на максимально большое количество раз делимым. Это количество раз вычисляется бинарным поиском. Сложность операции деления: $O(n * m * log(BASE)$.

Возведение в степень имеет сложность $O(n * m * log(m)$,, где n - число, возводимое в степень, m - степень.

Операции сравнения осуществляются поразрядно и имеют сложность $O(n)$.

Описание программы

Проект состоит из 3 файлов:

- main.cpp - главный файл, в котором реализована функция main
- long_number.hpp - заголовочный файл, в котором находятся объявления класса длинного числа и его методов.
- long_number.cpp - файл, в котором содержатся реализации методов класса длинного числа.

Дневник отладки

При создании этой таблицы была использована история посылок.

1-6	2020/11/17	TL	Проблема была в реализации алгоритма Карацубы. Данная реализация медленно работала из-за большого количества копирований длинных чисел.

Выводы

Лабораторная работа была относительно простой, но все же нужно было проявлять аккуратность при написании таких простых вещей, как умножение, деление, сложение и вычитание.

Длинная арифметика используется для вычислений, требующих работы с большими числами и высокой точностью.

Недостатком моей работы является то, что операция умножения требует значительных улучшений, если нужно достигнуть заявленной сложности алгоритма Карацубы.