Лабораторная работа № 6 по курсу дискретного анализа: Калькулятор

Выполнил студент группы 08-207 МАИ *Днепров Иван.*

Условие

1. Необходимо разработать программную библиотеку на языке C или C++, реализующую простеишие арифметические деиствия и проверку условии над целыми неотрицательными числами. На основании этои библиотеки, нужно составить программу, выполняющую вычисления над парами десятичных чисел и выводящую результат на стандартныи фаил вывода.

Метод решения

Для работы с длинной арифметикой я реализовал библиоткеу с классом BigInt и типом BigIntData на основе вектора целых чисел, причём разряды числа хранятся в векторе в обратном прядке для того, чтобы при увеличении числа разрядов, достаточно было добавить элемент в конец вектора, а не добавлять его в начало и сдвигать все последующие элементы.

В этой библиотеке я сделал два конструктора: первый получает на вход целочисленную переменную и, последовательно вычисляя остаток от деления на основание, заполняет вектор и второй, он получае на вход строку, состоящую из числовых символов, удаляет ведущие нули, вычленяет куски заданной длинны, перобразуем их к типу int и дописываем в результирующий вектор.

Также переопределил 9 операторов: 5 функциональных операторов: сложение, вычитание, умножение, деление и возведение в степень, 3 логических оператора: равно, меньше и больше и один оператор перенапрвления потоков.

Сложение и вычитание устроены практически идентично, по этому я рассмотрю только сложение. Оно реализовано как сложение в столбик (наивный алгоритм): у нас есть интовая переменная transferOverflow, в которую будет записываься значение для переноса в следующую ячейку при переполнении текущей, всё действо происходит в цикле, который последовательно суммирует ячейки, не забывая про переполнения с прошлой итерации, пока мы не пройдем длинну наибольшего числа и не выполнили все обязательства по переносу переполнений.

Умножение тоже реализовано наивно: я задаю размер результирующего числа равный сумме длин двух перемножаемых чисел (в случае6 если результат окажется меньше, я изменю размер вектора). Далее запускаю два вложегнных цикла (да, сложность плучается квадратичная): первый бежит по разрядам первого числа, второй – по разрядам второго. Внутри второго цикла я в элемент вектора с номером равным сумме номеров текущих разрядов первого и второго числа записываю произведение этих разрядов + переполнение с прошлой итерации, после чего проверяю переполнение для

текущей.

Деление – это самая сложная функция в этой лабораторной работе. Мы пробегаем по всем разрядам делимого числа и добавляем разряд в currentDischarge – промежуточное число, для которого методом половинного деления подбираем коэффицент resultDischarge, при перемножении которого с делителем получится число, максимально приближенное к currentDischarge, которвый записываем в соответствующий разряд результирующего вектора и уменьшаем currentDischarge на произведение делителя и resultDischarge. Посути это аналог деления в столбик с подбором коэффицента методом бинарного поиска.

Возведение в степень работает несколько хитрее, чем в наивном алгоритме: чтобы минимизировать количество умножений, число возводится в квадрат каждую итерацию и если степень не чётная, то домножается на само себя.

Оператор сравнения построен на стандартном сравнении двух векторов.

Операторы больше и меньше очень похожи, поэтому я разберу только один из них. В операторе меньше в начале мы сравниваем длинны двух чисел и если они различаются, на основании этого делаем вывод, какое число больше (на вход подаются числа без ведущих нулей), если же числа равной длинны, последовательно сравниваем разряды, начиная со старших.

Оператор перенаправления потока был переопределен таким образом, что он последовательно добавляет все разряды числа в поток, начиная со старших.

Также в данную библиотеку я добавил функцию удаления ведущих нулей DeleteLeadingZeros, которая в цикле удаляет пустые ведущие разряды.

Для демонстрации работы библиотеки в main есть цикл, который работает, пока есть входные параметры, внутри которого switch case, с проверками на корректость данных и вызовом необходимых операций, в зависимости от входных парметров.

Описание программы

Библиотека состоит из двух файлов: основного файла BigInt.cpp и заголовочного – BigInt.h. Она подключена к main.cpp, в котором обрабатываются запросы из потока ввода и вызываются необходимые библиотечные функции.

Выводы

Из-за того, что большинство функций я реализовал наивным способом, эту библиотеку сложно назвать оптимальной с точки зрения скорости изполнения и количества используемых ресурсов. Простейшие опирации вроде сложения, вычитания и сравнений работают за линейное время, умножение и деление работают за квадрат, а возведение в степень за логорифм относительно умножения, то есть сложность возведеия в степень равна $O(n^2 \cdot \log n)$. Это не куб, конечно, но, думаю, можно было бы реализовать это эффективнее. Моя библиотека проиграет по скорости исполнения любой стандартной библиотеке для работы с большими числами, что было чоень хорошо заметно при тестировании (я генерировал входные и выходные данные на python, и это работало в

разы быстрее, чем обработка теством моей прогарммой). В свое оправдание могу заметить, что, благодаря наивным методам, код получился легко читаемым и достаточно простым в понимании. Я бы ни в коем случае не рекомендовал написанное мной для коммерческого исполозования, но в образовательных целях эта библиотека применима. Благодаря ней я, не углубляясь в тонкости сложных алгоритмов понял принципы работы библиотек, опирирующих с большими числами.