Лабораторная работа № 6 по курсу дискретного анализа: Калькулятор

Выполнил студент группы М8О-308Б-21 МАИ *Белоносов Кирилл.*

Условие

1. Общая постановка задачи: Реализовать калькулятор использующий длинную арифметику

2. Задание:

   Алгоритм: 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 ($\hat{}$).
   - Равно (=).

   Замечание: при реализации деления можно ограничить делитель цифрой внутреннего представления «длинных» чисел, в этом случае максимальная оценка, которую можно получить за лабораторную работу, будет ограничена оценкой 3 («удовлетворительно»).

   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.
   Формат ввода: Входный файл состоит из последовательностей заданий, каждое задание состоит их трех строк:

- Первый операнд операции.
- Второй операнд операции.
- Символ арифметической операции или проверки условия (+, -, *, ; /, >, <, =).

Числа, поступающие на вход программе, могут иметь «ведущие нули».
Формат вывода: Для каждого задания из входного файла нужно распечатать результат на отдельной строке в выходном файле:

- Числовой результат для арифметических операций.
- Строку Error в случае возникновения ошибки при выполнении арифметичесокй операции.
- Строку true или false при выполнении проверки условия.

В выходных данных вывод чисел должен быть нормализован, то есть не содержать в себе «ведущих» нулей.

Метод решения

Для выполнения данной лабораторной работы требуется использовать длинную арифметику. Для ее представления используем вектор состоящий из разрядов. Для большей производительности пусть один разряд будет в диапозоне [0,1000). Сложение и вычитание в таком представлении не вызовут каких либо сложностей. Выполняются они "в столбик со сложностью O(n). Крайне важно не забывать после сложения и вычитания "нормализировать"числа и остатки перербросить в старшие(меньшие) разряды соответственно. Для умножения используем алгоритм быстрого преобразования фурье - FFT. Который помогает перемножать многочлены (В нашем случае вектора из разрядов) за сложность O(nlogn). После умножения также важно не забыть нормализовать числа и удаливать возможные ведущие нули (появляются в результате работы алгоритма FFT - округление до ближайшей степени двойки). Так как найти все делители для большого числа - задача объёмная по памяти, то будем сами искать делитель. Для этого нам поможет бинарный поиск. Работа алгоритма похожа на принцип деления уголком - мы ищем такое число, чтобы остаток от деления был 0 или минимально возможным. Сложность $O(n^2logb)$ Для возведения в степень воспользуемся известным алгоритмом бинарного возведения в степень и уже имеющегося у нас быстрого умножения

Описание программы

Код для данной программы достаточно простой и однотипный, поэтому я укажу только самые важные участки

Реализация алгоритма FFT

```
static void FFT(vector<base>& a, bool invert) {
    ll n = static_cast<ll>(a.size());

    for(ll i = 1, j = 0; i < n; ++i) {
        ll bit = n >> 1;
        for(; j >= bit; bit >>= 1)
            j -= bit;
        j += bit;
        if (i < j)
            swap (a[i], a[j]);
    }

    for (ll len = 2; len <= n; len <<= 1) {
        double angle = 2 * PI / static_cast<double>(len) * (invert ?
           -1 : 1);
        base wlen (cos(angle), sin(angle));
        for (ll i = 0; i < n; i += len) {
            base w (1);
            for (ll j = 0; j < len/2; ++j) {
                base u = a[i + j],  v = a[i + j + len / 2] * w;
                a[i + j] = u + v;
                a[i + j + len / 2] = u - v;
                w *= wlen;
            }
        }
    }
    if (invert)
        for (ll i = 0; i < n; ++i)
            a[i] /= static_cast<double>(n);
}

friend BigInt operator* (const BigInt& lhs, const BigInt& rhs) {
    ull n = 1;
    while(n < max(lhs.Size(), rhs.Size()))
        n <<= 1;
    n <<= 1;
    vector<ll> mult(n);
    vector<base> lhsComplex = transformToComplex(lhs, n);
    vector<base> rhsComplex = transformToComplex(rhs, n);

    lhsComplex.resize(n),
            rhsComplex.resize(n);

    FFT(lhsComplex, false),
            FFT(rhsComplex, false);
```

```
    for (ull i = 0; i < n; ++i)
        lhsComplex[i] *= rhsComplex[i];
    FFT(lhsComplex, true);

    mult.resize(n);
    for (size_t i = 0; i < n; ++i)
        mult[i] = round(lhsComplex[i].real());
    Normalize(mult);
    RemoveEndingZeros(mult);
    reverse(mult.begin(), mult.end());
    return BigInt(mult);
}
```

Реализация деления с использованием бинарного поиска для нахождения делителя

```
friend BigInt operator/ (const BigInt& lhs, const BigInt& rhs) {
    if(rhs > lhs) {
        return {0};
    }
    vector<ll> div;
    ull i = 0;
    BigInt remainder;
    do {
        remainder.AddDigit(lhs[i]);
        ++i;
    } while(remainder < rhs);
    for(;;) {
        ll l = -1, r = BASE;
        while(l + 1 < r) {
            ll m = (l + r) / 2;
            if(rhs * m > remainder)
                r = m;
            else
                l = m;
        }
        div.push_back(l);
        remainder = remainder - l * rhs;
        if(i == lhs.Size())
            break;
        if(remainder == 0)
            remainder[0] = lhs[i];
        else
            remainder.AddDigit(lhs[i]);
        ++i;
    }
```

```
    return BigInt(div);
}
```

Реализация бинарного возведения в степень

```
friend BigInt operator^ (BigInt& lhs, BigInt& rhs) {
    BigInt power(1);
    BigInt zero(0);
    while(rhs > zero) {
        auto [div, mod] = BitShift(rhs);
        if(mod)
            power *= lhs;
        lhs = lhs * lhs;
        rhs = div;
    }
    return power;
}
```

Дневник отладки

Попытка 1

Возникла ошибка с умножением. В результате умножения не переносил остатки в старшие разряды

Попытка 2

Нашел небольшой баг, связанный с вычитанием

Попытка 3

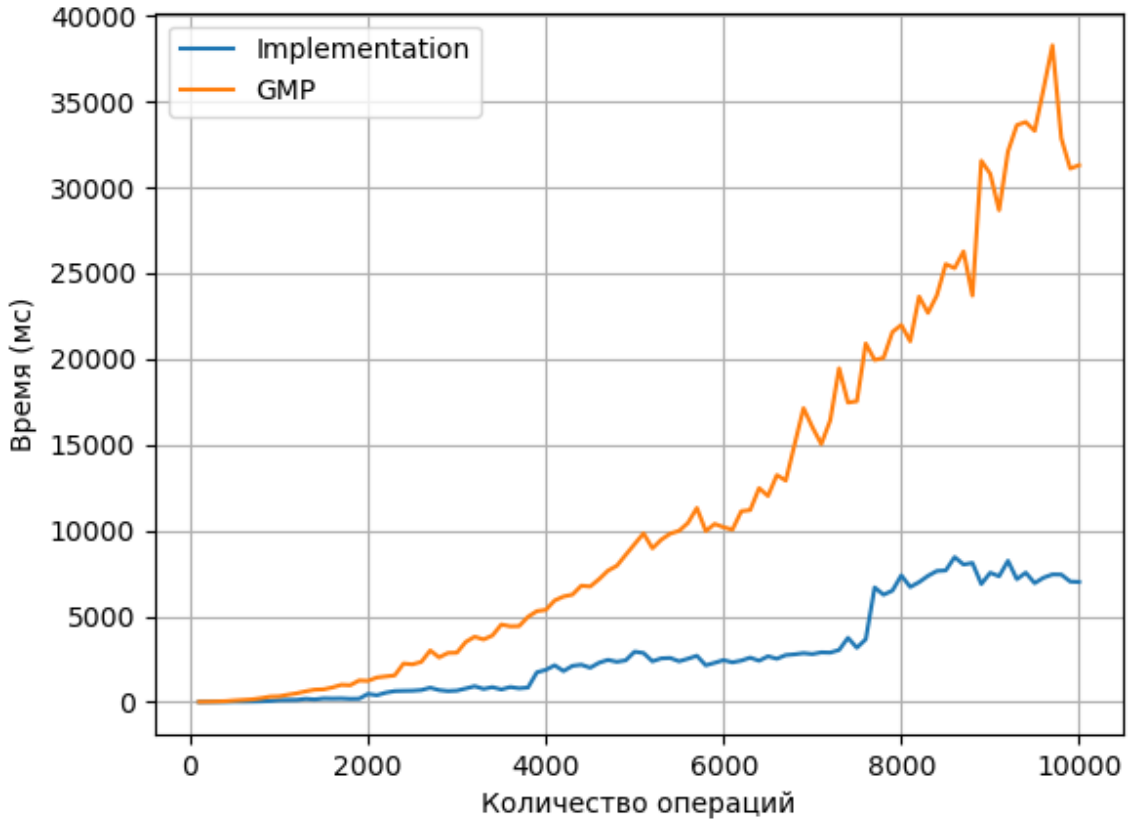
Программа прошла тестирование

0.1 Результаты тестирования

Тест производительности

Тестирование программы производилось на 100 сгенерированных тестах для всех операций, за исключением возведения в степень. Для сравнения было выбрана библиотека GMP:

Вставка:



Получившиеся результаты сильно удивляют. Получившаяся реализация длинной арифметики оказалась значительно быстрее той, что представлена в библотеке. Сложно сказать, с чем это связано. Возможно с внутренней реализацией библиотеки для работы с дробями.

Выводы

В данной лабораторной работе я реализовал написал программу-калькулятор использующуу длинную арифметику. В рамках данной работы я познакомился и изучил несколько интересных алгоритмов. Такие как: Быстрое преобразование Фурье, Бинарное возведение в степень, Деление с бинарным поиском делителя. Также я поработал с библиотекой GMP для сравнения со своей реализацией быстрой арифметики. Не сложно заметить, что сделанные оптимизации помогли значительно ускорить программу, относительно стандартного инструмента. Также данная программа. В целом работа показалась одной из самый объемных и в тоже время интересных. Интересной частью работы было также проектирование самого представления длинной арифметики