Московский авиационный институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №6 по курсу «Дискретный анализ»

Студент: Е. А. Кондратьев
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-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:

- Сложение ($+$).
- Вычитание ($-$).
- Умножение ($*$).
- Возведение в степень ($\wedge$).
- 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.

1 Описание

Метод решения

Необходимо написать реализацию простейших арифметических и логических операций с длинными числами. В качестве внутреннего представления числа логично выбрать вектор, в который будут добавляться “разряды” длинного числа. Числа в векторе располагаются от младшего разряда к старшему, максимальное значение числа в одном разряде ограничено выбранным основанием системы счисления, я выбрал 10^4.

```
\textbf{TBigInt::TBigInt(const std::string& inp)
    : num({})
{
    if (inp.empty())
        return;
    size_t pos = 0;
    while (inp[pos] == '0')
        ++pos;
    long long start, end = static_cast<long long>(inp.size());
    for (start = end - CELL_LENGTH; start >= pos && end > 0; start -= CELL_LENGTH) {
        if (start < 0)
            start = 0;

        num.push_back(atoi(inp.substr(static_cast<size_t>(start), static_cast<size_t>(
            end - start)).c_str()));
        end -= CELL_LENGTH;
    }
}}
```

Сложение Реализуется тривиально: начиная с младших разрядов числа (начало вектора), суммируем оба числа. Если результат больше чем основание системы счисления, то записывается остаток от деления результата на основание системы счисления. При этом целая часть прибавляется к старшему разряду.

```
TBigInt const operator+(TBigInt const& lhs, TBigInt const& rhs)
{
    TBigInt res;
    int32_t carry = 0;
    size_t n = std::max(lhs.data.size(), rhs.data.size());
    res.data.resize(n);
    for(size_t i = 0; i < n; ++i)
    {
        int32_t sum = carry;
        if(i < rhs.data.size()) { sum += rhs.data[i]; }
        if(i < lhs.data.size()) { sum += lhs.data[i]; }
        carry = sum / TBigInt::BASE;
        res.data[i] = sum % TBigInt::BASE;
    }
    if(carry != 0) { res.data.push_back(static_cast<int32_t>(1)); }
    res.DeleteLeadingZeros();
    return res;
}
```

Вычитание Реализуется аналогично сложению: из большего вычитается меньшее, если при вычитании разрядов получается отрицательное число, то к нему прибавляется основание системы счисления и занимается единица из старшего разряда.

```
TBigInt const operator-(TBigInt const& lhs, TBigInt const& rhs)
{
    TBigInt res;
    int32_t carry = 0;
    size_t n = std::max(lhs.data.size(), rhs.data.size());
    res.data.resize(n);
    for(size_t i = 0; i < n; ++i)
    {
        int32_t sub = lhs.data[i] - carry;
        if(i < rhs.data.size()) { sub -= rhs.data[i]; }
        carry = 0;
        if(sub < 0)
        {
            carry = 1;
            sub += TBigInt::BASE;
        }
        res.data[i] = sub % TBigInt::BASE;
    }
    res.DeleteLeadingZeros();
    return res;
}
```

Умножение Алгоритм вычисления такой же, как и для обычных чисел (по разрядам, столбиком), за исключением того случая, когда результат становится больше основания системы счисления. Тогда целую часть от деления результата надо прибавить к следующему результату, а остаток от деления прибавить к разряду с номером, равным сумме позиций умножаемых разрядов двух чисел. Сложность наивного алгоритма умножения $O(n*m)$, что не очень хорошо, когда количество разрядов числа слишком велико, поэтому для более сложных случаев применяются алгоритмы Карацубы или Шёнхаге-Штрассена.

```
TBigInt const operator*(TBigInt const& lhs, TBigInt const& rhs)
{
    if(rhs.data.size() == 1) { return lhs.MultShort(rhs); }
    if(lhs.data.size() == 1) { return rhs.MultShort(lhs); }
    TBigInt res;
    size_t n = lhs.data.size() * rhs.data.size();
    res.data.resize(n + 1);
    int32_t k = 0;
    int32_t r = 0;
    for(size_t i = 0; i < lhs.data.size(); ++i)
    {
        for(size_t j = 0; j < rhs.data.size(); ++j)
        {
            k = rhs.data[j] * lhs.data[i] + res.data[i+j];
            r = k / TBigInt::BASE;
            res.data[i+j+1] = res.data[i+j+1] + r;
            res.data[i+j] = k % TBigInt::BASE;
        }
    }
    res.DeleteLeadingZeros();
    return res;
}
```

Возведение в степень Быстрое возведение в степень - алгоритм, учитывающий четность степени, позволяет возводить число со сложностью $O(logn)$, где n - количество перемножений, которые надо совершить.

```
TBigInt const operator^(TBigInt const& lhs, TBigInt const& power)
{
    TBigInt res("1");
    TBigInt two("2");
    TBigInt one("1");
    TBigInt zero("0");
    if(power == zero) { return res; }
    if(power == one || lhs == one) { return lhs; }
    if(power.data[0] % 2 == 0)
    {
        TBigInt res = lhs ^ (power / two);
        return res * res;
    }
    else
    {
        TBigInt res = lhs ^ (power - one);
        return lhs * res;
    }
}
```

Деление Осуществляется уголком: выбираем количество старших разрядов делимого числа так, чтобы получившийся срез по длине был равен делителю. Затем находим с помощью бинарного поиска (на отрезке от 0 до основания системы счисления) максимально возможный множитель,такой, что разница между срезом и умноженным делителем минимальна и положительна (0 тоже допустим это означает, что срез меньше делителя). Разницу запоминаем для дальнейшего деления, множитель записываем в старший разряд ответа. В начало остатка от деления записываем следующий старший разряд делителя и продолжаем алгоритм, пока не дойдём до младшего разряда делителя. Сложность умножения в данном случае будет $O(m)$.

```
TBigInt const operator/(TBigInt const& lhs, TBigInt const& rhs)
{
    TBigInt curr, res;
    size_t lhs_size = lhs.data.size();
    res.data.resize(lhs_size);
    int l = 0;
    int r = TBigInt::BASE;
    int m = 0;
    int data_res = 0;
    for(int i = lhs_size - 1; i >= 0; --i)
    {
        m = 0;
        l = 0;
        r = TBigInt::BASE;
        curr.ShiftRight();
        curr.data[0] = lhs.data[i];
        curr.DeleteLeadingZeros();
        while(l <= r)
        {
            m = (l + r) / 2;
            if(rhs * TBigInt(std::to_string(m)) <= curr)
            {
                data_res = m;
                l = m + 1;
            }
            else { r = m - 1; }
        }
        res.data[i] = data_res;
        curr = curr - rhs * TBigInt(std::to_string(data_res));
```

```
30      }
31      res.DeleteLeadingZeros();
32      return res;
33  }
```

Операции сравнения Операции сравнения реализуются очень просто, сначала сравниваются длины векторов, и в случае если они равны, то поразрядно. Поэтому их код я приводить не буду.

2 Тест производительности

Для сравнения использовал библиотеку int_width для 128-битных чисел и chrono для замера времени.

```
a = 753275733897352885583252657455685685,b = 723587383839

Сложение:
753275733897352885583253381043069524
Моя реализация: 2.2498e-05
Библиотека <int_width>: 0.00179459

Вычитание:
753275733897352885583251933868301846
Моя реализация: 4.602e-06
Библиотека <int_width>: 0.000723548

Деление:
1041029391503263989488349
Моя реализация: 1.76e-05
Библиотека <int_width>: 0.000360422
```

Из приведенных тестов видно, что операции сложения, вычитания и деления проходят быстрее.

3 Выводы

В ходе шестой лабораторной работы я познакомился с длинной арифметикой. Реализовал класс BigInt и операции для работы с ним. Самым сложным алгоритмом оказалось деление. Сначала пришлось расписать весь код на листочке, попутно разбирая множество примеров и лишь затем программировать, зато таким образом я потратил очень мало времени на дебаг.

Список литературы

[1] *Поисковик - Google.*
URL: `https://www.google.com/`

[2] *Сайт с подробной документацией библиотек C++*
URL: `https://en.cppreference.com/`

[3] *Про длинную арифметику*
URL: `https://e-maxx.ru/algo/big_integer/`