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: Н. П. Ежов
Преподаватель: Н. С. Капралов
Группа: М8О-204Б
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().
Деление(/).
В случае возникновения переполнения в результате вычислений, попытки вычесть из меньшего числа большее, деления на ноль
или возведение 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

Требуется написать реализацию алгоритмов поразрядного сложения, вычитания, умножения, возведения в степень, а также деления. Можно работать с числами, как со строками, но разделение на разряды будет позволять быстрее работать (за счёт операций внутри над сравнительно маленькими числами).
В качестве основания для наших чисел будем использовать 10^7. Большие числа организуем при помощи класса TBigInt. В классе будет вектор "цифр"числа. В векторе цифр значения хранятся от младшего разряда к старшему, т.е. $bigInt[0]$ это самый младший разряд числа. Также определим для данного класса базовые арифметические операции при помощи перегрузки операторов.

TBigInt	
$size_t Size() const$	возвращает размер вектора большого числа
TBigInt() = default	Конструктор по умолчанию.
TBigInt(int num)	Конструктор для маленького числа.
explicit TBigInt(const size_t &size) : bigInt(size, 0)	Конструктор для создания числа, заданного размером size.
explicit TBigInt(const std::string &str)	Конструктор для строки.
TBigInt operator = (const TBigInt &other)	Перегрузка оператора присваивания
friend TBigInt operator + (TBigInt &first, TBigInt &second)	Перегрузка оператора сложения.
friend TBigInt operator − (TBigInt &first, TBigInt &second)	Перегрузка оператора вычитания.
friend TBigInt operator ∗ (TBigInt &first, TBigInt &second)	Пегегрузка оператора умножения для больших чисел.
friend TBigInt operator ∗ (TBigInt &first, const int &second)	Пегегрузка оператора для умножения большого числа на малое.
friend TBigInt operator / (TBigInt &first, TBigInt &second)	Пегегрузка оператора деления для больших чисел.
friend TBigInt operator / (TBigInt &first, const int &second)	Пегегрузка оператора для деления большого числа на малое.
friend TBigInt operator ^ (TBigInt &first, TBigInt &second)	Возведение большого числа в указанную степень.
friend std::ostream& operator <<	Оператор вывода большого числа.
friend std::istream& operator >>	Оператор ввода большого числа.
friend TBigInt operator < (TBigInt &first, TBigInt &second)	Оператор сравнения <.
friend TBigInt operator > (TBigInt &first, TBigInt &second)	Оператор сравнения >.
friend TBigInt operator == (TBigInt &first, TBigInt &second)	Оператор сравнения =.

Замечание: во всех операторах после выполнения основной функции производится удаление лидирующих нулей, если такие есть.

1 Сложение

Сложение реализовано просто - путём сложения "в столбик". Единственная разница в том, что основание условной "системы исчисления"у нас не 10, а 10^7, что позволяет складывать большие числа быстрее. Если первое число меньше второго, то мы рекурсивно вызываем оператор $second + first$, т.к. удобно реализовать алгоритм для случая, когда $first \geq second$, а в случае обратного просто менять их местами.

2 Вычитание

Вычитание также реализовано школьным методом, "в столбик" но при этом мы проверяем, что $first \geq second$, иначе выводим $Error$.

3 Умножение

Умножение реализовывается "в столбик". Реализуем две функции - одна для перемножения двух больших чисел, а вторая - для перемножения большого числа на относительно малое (вмещается в один разряд).

4 Деление

Деления большого числа на малое производится при помощи деления "в столбик" а для деления двух больших чисел алгоритм другой. Согласно [1] алгоритм деления можно описать следующим образом: у нас есть два числа, $u = (u_{m+n+1}...u_1 * u_0)$ и $(v_{n-1}...v_1 * v_0)$., представленным по основанию b, в предположении что $v_{n-1} \neq 0$ и $n > 1$, находим в системе счисления по основанию b частное $[u/v] = (q_m * q_{m-1}...q_0)$. Найти $d = [b/v_{n-1} + 1]$, после чего умножить оба числа на полученное d. Присвоить $j \leftarrow m$. Пока $j \geq 0$ выполняем следующие шаги:
$d \leftarrow [u_{j+n} * b + \frac{u_{j+n-1}}{v_{n-1}}]$
Тут начинается разница между алгоритмом из [1] и итоговой реализацией. На ряде тестов, по неизвестным причинам (неправильно был переписан алгоритм в код или он просто не подходит), два раза проверить указанные неравенства недостаточно, чтобы получить правильный d. Зная, что $d \geq$искомого значения, алгоритм был модифицирован следующим образом: от d отнимают по 1, если $v * d < \hat{r}$, где $\hat{r} = u[i] + r * b$. Затем считаем новый $r \leftarrow r - v * d$, добавляем d последним разрядом в результат и начинаем действия выше с начала.

5 Возведение в степень

Используется т.н. алгоритм бинарного возведения в степень. Это приём, позволяющий возводить любое число в n-ую степень за $O(logn)$ умножений (вместо n).
Алгоритм:
Заметим, что для любого числа a и **чётного** числа n выполнимо тождество $a^n = (a^{n/2})^2 = a^{n/2} * a^{n/2}$.
Оно и является основным в методе бинарного возведения в степень. А если же n нечётный, то для него верно следующее тождество $a^n = a^{n-1} * a$, где $n-1$ - гарантированно чётное число. Очевидно, что для a^{n-1} можно будет применить тождество, верное для чисел с чётным показателем степени.

6 Операции сравнения

Числа сравниваются "лексикографически" т.е. поразрядно (можно провести аналогию со строками).

2 Исходный код

```
//
// main.cpp
//

#include <vector>
#include <string>
#include <iostream>
#include <cmath>
#include <iomanip>
#include <algorithm>

class TBigInt {
private:
    static const long int BASE = 1e7;
    static const int DIGIT_LENGTH = 7;
    std::vector<int> bigInt;
public:
    size_t Size() const {
        return bigInt.size();
    }

    TBigInt() = default;

    TBigInt(int num) {
        if (!num) {
            bigInt.push_back(0);
        }
        else {
            while (num) {
                bigInt.push_back(num % BASE);
                num /= BASE;
            }
        }
    }

    explicit TBigInt(const size_t &size) : bigInt(size, 0) {};

    explicit TBigInt(const std::string &str) {
        bigInt.clear();
        for (int i = (int)str.length(); i > 0; i -= DIGIT_LENGTH)
            if (i < DIGIT_LENGTH)
                bigInt.push_back (atoi(str.substr (0, i).c_str()));
            else
                bigInt.push_back (atoi(str.substr (i-DIGIT_LENGTH, DIGIT_LENGTH).c_str()
                    ));
        while (bigInt.size() > 1 && bigInt.back() == 0)
            bigInt.pop_back();
```

```
47      }
48      TBigInt operator = (const TBigInt &other) {
49          this->bigInt = other.bigInt;
50          return *this;
51      }
52
53      friend TBigInt operator + (TBigInt &first, TBigInt &second) {
54          if (first.Size() < second.Size()) {
55              return second + first;
56          }
57          TBigInt result = first;
58          int carry = 0;
59          for (int i = 0; i < std::max(result.Size(), second.Size()) || carry; ++i) {
60              if (i == result.Size()) {
61                  result.bigInt.emplace_back(0);
62              }
63              result.bigInt[i] += carry + (i < second.Size() ? second.bigInt[i] : 0);
64              carry = result.bigInt[i] / BASE;
65              result.bigInt[i] %= BASE;
66          }
67          while (result.bigInt.size() > 1 and result.bigInt.back() == 0) {
68              result.bigInt.pop_back();
69          }
70          return result;
71      }
72
73      friend TBigInt operator - (TBigInt &first, TBigInt &second) {
74          if (first < second) {
75              throw std::runtime_error("error");
76          }
77          int carry = 0;
78          TBigInt result(first);
79          for (int i = 0; i < static_cast<int>(second.Size()) || carry; ++i) {
80              result.bigInt.at(i) -= carry + (i < static_cast<int>(second.Size()) ?
                    second.bigInt[i] : 0);
81              carry = result.bigInt[i] < 0;
82              if (carry != 0) {
83                  result.bigInt.at(i) += BASE;
84              }
85          }
86          while (result.bigInt.size() > 1 and result.bigInt.back() == 0) {
87              result.bigInt.pop_back();
88          }
89          return result;
90      }
91
92
93
94      friend TBigInt operator * (TBigInt &first, TBigInt &second) {
```

```
95          TBigInt result(first.Size() + second.Size());
96          int carry = 0;
97          for (int i = 0; i < first.Size(); ++i) {
98              carry = 0;
99              for (int j = 0; j < int(second.Size()) || carry; ++j) {
100                 long long curr = result.bigInt[i + j] + carry + first.bigInt[i] * 1ll *
                        (j < (int)second.Size() ? second.bigInt[j] : 0);
101                 result.bigInt[i + j] = int(curr % BASE);
102                 carry = int(curr / BASE);
103             }
104         }
105         while (result.bigInt.size() > 1 and result.bigInt.back() == 0) {
106             result.bigInt.pop_back();
107         }
108         return result;
109     }
110
111     friend TBigInt operator * (TBigInt &first, const int &second) {
112         TBigInt result(first);
113         int carry = 0;
114         for (int i = 0; i < result.Size() || carry; ++i) {
115             if (i == result.Size()) {
116                 result.bigInt.push_back(0);
117             }
118             long long curr = carry + result.bigInt[i] * 1ll * second;
119             result.bigInt[i] = int (curr % BASE);
120             carry = (int) (curr / BASE);
121         }
122         while (result.bigInt.size() > 1 and result.bigInt.back() == 0) {
123             result.bigInt.pop_back();
124         }
125         return result;
126     }
127
128     friend TBigInt operator / (TBigInt &first, TBigInt &second) {
129         if(first < second) {
130             return TBigInt("0");
131         }
132         if((second.Size() == 1) && second.bigInt[0] == 0) {
133             throw std::runtime_error("error");
134         }
135         int norm = BASE / (second.bigInt.back() + 1);
136         TBigInt lCopy = first * norm;
137         TBigInt rCopy = second * norm;
138         TBigInt q(lCopy.Size());
139         TBigInt r;
140         for (int i = lCopy.Size() - 1; i >= 0; --i) {
141             r = r * BASE;
142             TBigInt increment(std::to_string(lCopy.bigInt[i]));
```

```
143             r = r + increment;
144             int s1 = r.Size() <= rCopy.Size() ? 0 : r.bigInt[rCopy.Size()];
145             int s2 = r.Size() <= rCopy.Size() - 1 ? 0 : r.bigInt[rCopy.Size() - 1];
146             int d = static_cast<int>((static_cast<int>(s1) * BASE + s2) / rCopy.bigInt.
                    back());
147             TBigInt tmp = rCopy * d;
148             while (tmp > r){
149                 tmp = tmp - rCopy;
150                 d--;
151             }
152             r = r - tmp;
153             q.bigInt[i] = d;
154         }
155         while (q.bigInt.size() > 1 and q.bigInt.back() == 0) {
156             q.bigInt.pop_back();
157         }
158         return q;
159     }
160 
161     friend TBigInt operator / (TBigInt &first, const int &second) {
162         if (second == 0) {
163             throw std::runtime_error("error");
164         }
165         int carry = 0;
166         TBigInt result(first);
167         for (int i = result.Size() - 1; i >= 0; --i) {
168             long long curr = result.bigInt[i] + carry *1ll * BASE;
169             result.bigInt[i] = int (curr/second);
170             carry = int (curr % second);
171         }
172         while (result.bigInt.size() > 1 and result.bigInt.back() == 0) {
173             result.bigInt.pop_back();
174         }
175         return result;
176     }
177 
178     friend TBigInt operator ^ (TBigInt &first, TBigInt &second) {
179         if ((first.Size() == 1 && first.bigInt[0] == 0) && (second.Size() == 1 &&
                second.bigInt[0] == 0)) {
180             throw std::runtime_error("error");
181         }
182         TBigInt decrement("1");
183         TBigInt num(first);
184         TBigInt result("1");
185         while (!(second.Size() == 1 && second.bigInt[0] == 0)) {
186             if ((second.bigInt[0] % 10) % 2 == 0) {
187                 num = num * num;
188                 second = second / 2;
189             } else {
```

```
            result = result * num;
            second = second - decrement;
        }
    }
    while (result.bigInt.size() > 1 and result.bigInt.back() == 0) {
        result.bigInt.pop_back();
    }
    return result;
}

friend std::ostream& operator << (std::ostream &os, const TBigInt &num) {
    int num_len = num.Size();
    os << num.bigInt.back();
    for (int i = num_len - 2; i >= 0; --i) {
        os << std::setfill('0') << std::setw(DIGIT_LENGTH) << num.bigInt[i];
    }
    return os;
}

friend std::istream& operator >> (std::istream &is, TBigInt &num) {
    std::string str;
    is >> str;
    num = TBigInt(str);
    return is;
}

friend bool operator < (const TBigInt &first, const TBigInt &second) {
    if (first.Size() != second.Size()) {
        return first.Size() < second.Size();
    }
    for (int i = first.Size() - 1; i >= 0 ; --i) {
        if (first.bigInt[i] != second.bigInt[i]) {
            return first.bigInt[i] < second.bigInt[i];
        }
    }
    return false;
}

friend bool operator > (const TBigInt &first, const TBigInt &second) {
    if (first.Size() != second.Size()) {
        return first.Size() > second.Size();
    }
    for (int i = first.Size() - 1; i >=0 ; --i) {
        if (first.bigInt[i] != second.bigInt[i]) {
            return first.bigInt[i] > second.bigInt[i];
        }
    }
    return false;
}
```

```
239
240     friend bool operator == (const TBigInt &first, const TBigInt &second) {
241         if (first.Size() != second.Size()) {
242             return false;
243         }
244         for (int i = first.Size() - 1; i >=0 ; --i) {
245             if (first.bigInt[i] != second.bigInt[i]) {
246                 return false;
247             }
248         }
249         return true;
250     }
251
252 };
253
254 int main() {
255     TBigInt a, b;
256     char action;
257     while(!std::cin.eof()) {
258         std::cin >> a >> b >> action;
259         if (std::cin.eof()) {
260             break;
261         }
262         if (action == '+') {
263             std::cout << a + b << "\n";
264         } else if (action == '-') {
265             try {
266                 std::cout << a - b << '\n';
267             } catch (...) {
268                 std::cout << "Error\n";
269             }
270         } else if (action == '*') {
271             std::cout << a * b << '\n';
272         } else if (action == '/') {
273             try {
274                 std::cout << a / b << '\n';
275             } catch (...) {
276                 std::cout << "Error\n";
277             }
278         } else if (action == '^') {
279             try {
280                 std::cout << (a ^ b) << '\n';
281             } catch (...) {
282                 std::cout << "Error\n";
283             }
284         } else if (action == '<') {
285             if (a < b) {
286                 std::cout << "true\n";
287             } else {
```

```
288                 std::cout << "false\n";
289             }
290         } else if (action == '>') {
291             if (a > b) {
292                 std::cout << "true\n";
293             } else {
294                 std::cout << "false\n";
295             }
296         } else if (action == '=') {
297             if (a == b) {
298                 std::cout << "true\n";
299             } else {
300                 std::cout << "false\n";
301             }
302         }
303     }
304 }
```

3 Консоль

```
nezhov@killswitch:~/CLionProjects/Diskran/lab6$ make
g++ -std=c++14 -O3 -Wextra -Wall -Wno-sign-compare -Wno-unused-result -pedantic
-o solution main.cpp
nezhov@killswitch:~/CLionProjects/Diskran/lab6$ cat tests/01.t
7674067 3029597 /
3970773 5168295 <
6265336 3259557 <
2166558 7442197 >
1161742 8220457 <
6059909 6215020 +
5263670 8140694 -
7468521 9612409 <
4035847 2054982 /
5535188 7 ^
nezhov@killswitch:~/CLionProjects/Diskran/lab6$ ./solution <tests/01.t
2
true
false
false
true
12274929
Error
true
1
15919397700560759456870674420179969687894774579 2
```

4 Тест производительности

Замеряется всё время работы программы, но, т.к. ввод одинаковый, то результаты репрезентативны. Для сравнения производительности была выбрана библиотека GMP. Количество строк (число число операция) для каждого файла равно десять в степени номер теста.
Например, $02.t$ содержит сто строк, а $07.t$ десять миллионов.

```
nezhov@killswitch:~/CLionProjects/Diskran/lab6$ ./benchmark <tests/04.t >04.a
custom long arithmetics 46 ms
nezhov@killswitch:~/CLionProjects/Diskran/lab6$ ./benchmark1 <tests/04.t >04.a
custom long arithmetics 55 ms
nezhov@killswitch:~/CLionProjects/Diskran/lab6$ ./benchmark2 <tests/04.t >04.a
GMP arithmetics 33 ms
nezhov@killswitch:~/CLionProjects/Diskran/lab6$ ./benchmark1 <tests/06.t >06.a
custom long arithmetics 3666 ms
nezhov@killswitch:~/CLionProjects/Diskran/lab6$ ./benchmark1 <tests/05.t >05.a
custom long arithmetics 366 ms
nezhov@killswitch:~/CLionProjects/Diskran/lab6$ ./benchmark2 <tests/05.t >05.a
GMP arithmetics 280 ms
nezhov@killswitch:~/CLionProjects/Diskran/lab6$ ./benchmark1 <tests/06.t >06.a
custom long arithmetics 3751 ms
nezhov@killswitch:~/CLionProjects/Diskran/lab6$ ./benchmark2 <tests/06.t >06.a
GMP arithmetics 3060 ms
nezhov@killswitch:~/CLionProjects/Diskran/lab6$ ./benchmark1 <tests/07.t >07.a
custom long arithmetics 40647 ms
nezhov@killswitch:~/CLionProjects/Diskran/lab6$ ./benchmark2 <tests/07.t >07.a
GMP arithmetics 30655 ms
```

Глядя на результаты, видно, что моя реализация однозначно уступает библиотеке GMP, но не стоит забывать, что для большинства операций мы использовали наивный алгоритм.

5 Выводы

В ходе выполнения данной работы по дискретному анализу я познакомился с длинной арифметикой и реализовал её на практике. Также я познакомился с библиотекой GMP. Самым трудным было реализовать операцию деления длинных чисел. Также повлияло то, что прямое переписывание алгоритма деления [1] не помогло и пришлось самому придумывать способ выхода из данной ситуации.

Список литературы

[1] Дональд Е. Кнут. *Искусство программирования 2-й том, 3-е издание.* — Издатели Виктор Шпонда, Геннадий Петриковец, Алексей Орлович. Перевод с английского: Неизвестно. — 788 с. стр. 302 - 303 (дата обращения: 12.03.2020).

[2] *MAXimal - Бинарное возведение в степень.*
URL: `https://e-maxx.ru/algo/binary_pow` (дата обращения: 12.03.2020).