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: Д. С. Пивницкий
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-19

Дата: 01.01.2021
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. Разработать программу на языке C или C++, реализующую построенный алгоритм.

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.

Формат входных данных: Входной файл состоит из последовательности заданий, каждое задание состоит из трех строк:

Первый операнд операции. Второй операнд операции.

Символ арифметической операции или проверки условия.

Числа, поступающие на вход программе, могут иметь «ведущие» нули.

Формат результата: Для каждого задания из выходного файла нужно распечатать результат на отдельной строке в выходном файле:

Числовой результат для арифметических операций.

Строку Error в случае возникновения ошибки при выполнении арифметической 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.

1 Описание

Требуется реализовать класс для хранения «длинных» чисел, реализовать операции сложения, вычитания, умножения, деления и возведения в степень. Сложение выполняется очень просто, складываем разряды, в текущий разряд записываем остаток, а целую часть от деления запоминаем для следующего разряда. Вычитание выполняется аналогичным образом, с запоминанием, требуется ли нам «занять» число. Умножение выполняется аналогичным образом, только для каждой пары разрядов и последующих сдвигов для больших разрядов. Деление в степень выполняется начиная с больших разрядов, берется минимальное число, которое делится нацело, при помощи бинарного поиска и вычитается как в обычном делении стобком, и так до конца разрядов. Возведение в степень выполняется, используя уже написанное умножение, рекурсивно, если степень четная, то умножаем полученный результат из рекурсии сам на себя и делим степень пополам, если степень нечетная, то умножаем на исходное число и вычитаем из степени 1.Сравнения происходят очень просто, сначала сравниваем длины, если равны, то сравниваем по разрядам.

В программе bigiht.cpp содержатся основные конструкторы и функции для реализации нашей программы. А так же операторы вычислительных действий (сложения, вычитания, умножения, деления, сравнения)

2 Исходный код

Код: main.cpp

```
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <cmath>
#include "bigint.hpp"

int main()
{
NBigInt::TBigInt num1, num2;
NBigInt::TBigInt zero("0");
char action = '?';
while(std::cin >> num1 >> num2 >> action)
{
if(action == '+')
{
NBigInt::TBigInt res = num1 + num2;
std::cout << res << "\n";
}
else if(action == '-')
{
if(num1 < num2)
{
std::cout << "Error\n";
continue;
}
NBigInt::TBigInt res = num1 - num2;
std::cout << res << "\n";
}
else if(action == '*')
{
NBigInt::TBigInt res = num1 * num2;
std::cout << res << "\n";
}
else if(action == '^')
{
if(num1 == zero && num2 == zero)
{
std::cout << "Error\n";
continue;
}
NBigInt::TBigInt res = num1 ^ num2;
std::cout << res << "\n";
}
else if(action == '/')
{
```

```
47 if(num2 == zero)
48 {
49 std::cout << "Error\n";
50 continue;
51 }
52 NBigInt::TBigInt res = num1 / num2;
53 std::cout << res << "\n";
54 }
55 else if(action == '<')
56 {
57 if(num1 < num2) { std::cout << "true\n"; }
58 else { std::cout << "false\n"; }
59 }
60 else if(action == '>')
61 {
62 if(num1 > num2) { std::cout << "true\n"; }
63 else { std::cout << "false\n"; }
64 }
65 else if(action == '=')
66 {
67 if(num1 == num2) { std::cout << "true\n"; }
68 else { std::cout << "false\n"; }
69 }
70 else { std::cout << "Error\n"; }
71 }
72 return 0;
73 }
```

Код: digiht.cpp

```cpp
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
#include <algorithm>
#include "bigint.hpp"

namespace NBigInt
{

TBigInt::TBigInt(std::string const& str) { this->Init(str); }

void TBigInt::Init(std::string const& str)
{
data.clear();
int size = static_cast<int>(str.size());
for (int i = size - 1; i >= 0; i = i - TBigInt::RADIX)
{
if (i < TBigInt::RADIX) { data.push_back(static_cast<int32_t>(atoll(str.substr(0, i + 1).c_str()))); }
else { data.push_back(static_cast<int32_t>(atoll(str.substr(i - TBigInt::RADIX + 1, TBigInt::RADIX).c_str())));}
}
DeleteLeadingZeros();
}

void TBigInt::DeleteLeadingZeros()
{
while(!data.empty() && data.back() == 0) { data.pop_back(); }
}

std::istream& operator>>(std::istream& in, TBigInt& rhs)
{
std::string str;
in >> str;
rhs.Init(str);
return in;
}

std::ostream& operator<<(std::ostream& out, TBigInt const& rhs)
{
if(rhs.data.empty())
{
out << "0";
return out;
}
out << rhs.data.back();
for(int i = rhs.data.size() - 2; i >= 0; --i)
```

```
47 {
48 out << std::setfill('0') << std::setw(TBigInt::RADIX) << rhs.data[i];
49 }
50 return out;
51 }
52
53 TBigInt const operator+(TBigInt const& lhs, TBigInt const& rhs)
54 {
55 TBigInt res;
56 int32_t carry = 0;
57 size_t n = std::max(lhs.data.size(), rhs.data.size());
58 res.data.resize(n);
59 for(size_t i = 0; i < n; ++i)
60 {
61 int32_t sum = carry;
62 if(i < rhs.data.size()) { sum += rhs.data[i]; }
63 if(i < lhs.data.size()) { sum += lhs.data[i]; }
64 carry = sum / TBigInt::BASE;
65 res.data[i] = sum % TBigInt::BASE;
66 }
67 if(carry != 0) { res.data.push_back(static_cast<int32_t>(1)); }
68 res.DeleteLeadingZeros();
69 return res;
70 }
71
72 TBigInt const operator-(TBigInt const& lhs, TBigInt const& rhs)
73 {
74 TBigInt res;
75 int32_t carry = 0;
76 size_t n = std::max(lhs.data.size(), rhs.data.size());
77 res.data.resize(n);
78 for(size_t i = 0; i < n; ++i)
79 {
80 int32_t sub = lhs.data[i] - carry;
81 if(i < rhs.data.size()) { sub -= rhs.data[i]; }
82 carry = 0;
83 if(sub < 0)
84 {
85 carry = 1;
86 sub += TBigInt::BASE;
87 }
88 res.data[i] = sub % TBigInt::BASE;
89 }
90 res.DeleteLeadingZeros();
91 return res;
92 }
93
94 TBigInt const operator*(TBigInt const& lhs, TBigInt const& rhs)
95 {
```

```
96  if(rhs.data.size() == 1) { return lhs.MultShort(rhs); }
97  if(lhs.data.size() == 1) { return rhs.MultShort(lhs); }
98  TBigInt res;
99  size_t n = lhs.data.size() * rhs.data.size();
100 res.data.resize(n + 1);
101 int32_t k = 0;
102 int32_t r = 0;
103 for(size_t i = 0; i < lhs.data.size(); ++i)
104 {
105 for(size_t j = 0; j < rhs.data.size(); ++j)
106 {
107 k = rhs.data[j] * lhs.data[i] + res.data[i+j];
108 r = k / TBigInt::BASE;
109 res.data[i+j+1] = res.data[i+j+1] + r;
110 res.data[i+j] = k % TBigInt::BASE;
111 }
112 }
113 res.DeleteLeadingZeros();
114 return res;
115 }
116
117 TBigInt const TBigInt::MultShort(TBigInt const& rhs) const
118 {
119 int32_t carry = 0;
120 int32_t mult = 0;
121 size_t size = data.size();
122 TBigInt res;
123
124 for (size_t i = 0; i < size; ++i)
125 {
126 mult = data[i] * rhs.data[0] + carry;
127 res.data.push_back(mult % BASE);
128 carry = mult / BASE;
129 }
130 if (carry != 0) { res.data.push_back(carry); }
131 res.DeleteLeadingZeros();
132 return res;
133 }
134
135 TBigInt const operator^(TBigInt const& lhs, TBigInt const& power)
136 {
137 TBigInt res("1");
138 TBigInt two("2");
139 TBigInt one("1");
140 TBigInt zero("0");
141 if(power == zero) { return res; }
142 if(power == one || lhs == one) { return lhs; }
143 if(power.data[0] % 2 == 0)
144 {
```

```
TBigInt res = lhs ^ (power / two);
return res * res;
}
else
{
TBigInt res = lhs ^ (power - one);
return lhs * res;
}
}

void TBigInt::ShiftRight()
{
if (data.size() == 0)
{
data.push_back(0);
return;
}
data.push_back(data[data.size() - 1]);
for (size_t i = data.size() - 2; i > 0; --i) { data[i] = data[i - 1]; }
data[0] = 0;
}

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; }
}
```

```
194 res.data[i] = data_res;
195 curr = curr - rhs * TBigInt(std::to_string(data_res));
196 }
197 res.DeleteLeadingZeros();
198 return res;
199 }
200
201 bool operator<(TBigInt const& lhs, TBigInt const& rhs)
202 {
203 if(lhs.data.size() != rhs.data.size()) { return lhs.data.size() < rhs.data.size(); }
204 for(int i = lhs.data.size() - 1; i >= 0; --i)
205 {
206 if(lhs.data[i] != rhs.data[i]) { return lhs.data[i] < rhs.data[i]; }
207 }
208 return false;
209 }
210
211 bool operator==(TBigInt const& lhs, TBigInt const& rhs) { return !(lhs < rhs) && !(rhs
        < lhs); }
212 bool operator!=(TBigInt const& lhs, TBigInt const& rhs) { return !(lhs == rhs); }
213 bool operator<=(TBigInt const& lhs, TBigInt const& rhs) { return !(rhs < lhs); }
214 bool operator>(TBigInt const& lhs, TBigInt const& rhs) { return !(lhs <= rhs); }
215 bool operator>=(TBigInt const& lhs, TBigInt const& rhs) { return !(lhs < rhs); }
216
217 }
```

3 Консоль

```
(py37) ~ /DA_labs/lab6$ make
g++ -g -O2 -pedantic -std=c++17 -Wall -Wextra -Werror main.cpp -o solution
(py37) ~ /DA_labs/lab6$ cat test_sum.txt
050604130001
021410140107
+
(py37) ~ /DA_labs/lab6$ ./solution <test_sum.txt
72014270108
(py37) ~ /DA_labs/lab6$ cat test_sub.txt
040500130012
021410140107
-
(py37) ~ /DA_labs/lab6$ ./solution <test_sub.txt
19089989905
(py37) ~ /DA_labs/lab6$ cat test__mult.txt
050604130001
```

```
021410140107
*
(py37) ~ /DA_labs/lab6$ ./solution <test_mult.txt
1083441513314252050107
(py37) ~ /DA_labs/lab6$ cat test_power.txt
14
02
^
(py37) ~ /DA_labs/lab6$ ./solution <test_power.txt
196
(py37) ~ /DA_labs/lab6$ cat test_div.txt
050604130001
031014
/
(py37) ~ /DA_labs/lab6$ ./solution <test_div.txt
1631654
(py37) ~ /DA_labs/lab6$ cat test_less.txt
001203
100103
<
(py37) ~ /DA_labs/lab6$./solution <test_less.txt
true
(py37) ~ /DA_labs/lab6$ cat test_more.txt
9927
0234
>
(py37) ~ /DA_labs/lab6$ ./solution <test_more.txt
true
(py37) ~ /DA_labs/lab6$ cat test_eq.txt
9927
00927
=
(py37) ~ /DA_labs/lab6$ ./solution <test_eq.txt
true
```

4 Тест производительности

Тест состоит из 3-х подчастей для сложения, умножения и вычитания. Каждая часть теста проходит с константами, заданными перед началом замера времени, количество выполнений константых операций для всех операций равно 100 млн.

Моя реализация:

```
(py37) ~ /DA_labs/lab6$ make
g++ -g -O2 -pedantic -std=c++17 -Wall -Wextra -Werror main.cpp -o solution
(py37) ~ /DA_labs/lab6$ make bench
g++ -g -O2 -pedantic -std=c++17 -Wall -Wextra -Werror benchmark.cpp -o benchmark
(py37) ~ /DA_labs/lab6$ ./benchmark
Bigint sum time: 5.65296 seconds
Gmp sum time: 0.907357 seconds
Bigint sub time: 5.56938 seconds
Gmp sub time: 0.993383 seconds
Bigint mult time: 9.86926 seconds
Gmp mult time: 0.723498 seconds
```

Как можно увидеть, время работы примерно gmp достаточно сильно меньше на всех операциях, чем сложнее операция, тем больше разница, так мы реализовывали достаточно примитивные алгоритмы, не дающие такой скорости, как реализация в gmp.

5 Выводы

Выполнив шестую лабораторную работу по курсу «Дискретный анализ», я познакомился с различными алгоритмами работы с «длинными» числами. Посмотрел на способы их внутреннего представления, научился писать основные арифметические операции для них, за приемлимое время выполнения данных операций.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] *Поразрядная сортировка — Вики университета ITMO.* URL: `https://neerc.ifmo.ru/wiki/index.php?title=Цифровая_сортировка` (дата обращения: 01.10.2020).