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: А. К. Киреев
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 (^).

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 написать программную библиотеку для работы с большими числами.
Для операций сложения и вычитания давайте воспользуемся наивными алгоритмами сложения и вычитания в столбик. Эти алгоритмы работают за время $O(n)$, где n - длина числа, так как требуют лишь одного прохода по большому числу.
Для операций сравнения чисел также воспользуемся наивными алгоритмами: снчала сравним длины чисел, а затем, если длины равны. будем сравнивать числа со старших разрядов. Сравнение двух чисел также будет работать за $O(n)$, где n - длина числа.
Для операции умножения можно использовать также наивный алгоритм умножения столбиком или алгоритм Карацубы. Наивный алгоритм будет работать за $O(n * m)$, так как мы второе число будем умножать на каждую цифру из первого числа. Алгоритм Карацубы будет работать за $O(T^{log_2 3})$, однако при неаккуратной реализации у него будет очень большой множитель из-за множественного копирования больших чисел, и он будет даже проигрывать наивному.
Операцию деления реализуем следующим образом: алгоритм похож на наивный алгоритм деления столбиком, только на каждой итерации будем подбирать частное, которое пойдет в ответ, бинпоиком. Сложность будет $O(n*(log_2 Base*m+m)) = O(n*m)$
Операцию возведения в степень реализуем при помощи быстрого возведения в степень. В таком случае нам понадобится не m умножений, а $log_2 m$ умножений.
Также для оптимизации реализуем операции умножения и деления на короткое число, которые будем вызывать вместо обычных операций, если операнд имеет длину в одну цифру.

2 Исходный код

Сначала опишем интерфейс работы с классом больших чисел, чтобы понять, что он должен уметь делать. Нам бы хотелось пользоваться обычными арифметическими операторами языка, не вызывая дополнительных функций. Также по ТЗ мы должны обрабатывать случаи, когда мы вычитаем из меньшего большее, делим на ноль и т.п., для этого будем использовать механизм исключений.

Листинг 1: main.cpp

```
int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(0);

    std::string firstOperand;
    std::string secondOperand;
    char op;
    while (std::cin >> firstOperand >> secondOperand >> op) {
        NBigInt::TBigInt firstNum(firstOperand);
        NBigInt::TBigInt secondNum(secondOperand);

        if (op == '+') {
            std::cout << firstNum + secondNum << '\n';
        } else if (op == '-') {
            try {
                std::cout << firstNum - secondNum << '\n';
            } catch (std::logic_error) {
                std::cout << "Error\n";
            }
        } else if (op == '*') {
            std::cout << firstNum * secondNum << "\n";
        } else if (op == '^') {
            try {
                std::cout << NBigInt::TBigInt::FastPow(firstNum, secondNum) << '\n';
            } catch (std::logic_error) {
                std::cout << "Error\n";
            }
        } else if (op == '/') {
            try {
                std::cout << firstNum / secondNum << '\n';
            } catch (std::logic_error) {
                std::cout << "Error\n";
            }
        } else if (op == '>') {
            (firstNum > secondNum) ? std::cout << "true\n" : std::cout << "false\n";
        } else if (op == '<') {
            (firstNum < secondNum) ? std::cout << "true\n" : std::cout << "false\n";;
```

```
38          } else if (op == '=') {
39              (firstNum == secondNum) ? std::cout << "true\n" : std::cout << "false\n";;
40          } else {
41              std::cout << "Invalid operator!\n";
42          }
43      }
44      return 0;
45  }
```

Теперь объявим класс больших чисел. Зададим методам сигнатуры. Воспользуемся перегрузкой операторов, чтобы уметь использовать стандартные операторы языка для работы с большими числами. Конструктор зададим от строки. (именно в строковом представлении будем считывать числа)

Листинг 2: $\mathrm{big}_i nt.hpp$

```
1   namespace NBigInt {
2       const long long DefaultBase = 1e6;
3
4       class TBigInt {
5       private:
6           std::vector<long long> Data;
7           static const long long Base = DefaultBase;
8           static const long long Digits = 6;
9
10          void LeftShift(long long degree);
11          static void Split(const TBigInt& num, TBigInt& lhs, TBigInt& rhs);
12          static long long FindBin(const TBigInt& num, const TBigInt& div);
13          void RemoveNulls();
14
15      public:
16          TBigInt() = default;
17          TBigInt(const std::string& str);
18          TBigInt(const TBigInt& num);
19          TBigInt& operator=(const TBigInt& num);
20          ~TBigInt() = default;
21
22          friend bool operator<(const TBigInt& lhs, const TBigInt& rhs);
23          friend bool operator>(const TBigInt& lhs, const TBigInt& rhs);
24          friend bool operator==(const TBigInt& lhs, const TBigInt& rhs);
25
26          friend std::istream& operator>>(std::istream& is, TBigInt& num);
27          friend std::ostream& operator<<(std::ostream& os, const TBigInt& num);
28
29          friend TBigInt operator+(const TBigInt& lhs, const TBigInt& rhs);
30          friend TBigInt operator-(const TBigInt& lhs, const TBigInt& rhs);
31          friend TBigInt operator*(const TBigInt& lhs, const TBigInt& rhs);
32          friend TBigInt operator/(const TBigInt& lhs, const TBigInt& rhs);
```

```
33
34          static TBigInt FastPow(TBigInt base, TBigInt degree);
35          static TBigInt KaratsubaMult(TBigInt&& lhs, TBigInt&& rhs);
36          static TBigInt WeakDivision(const TBigInt& num, long long div);
37          static TBigInt WeakMultiply(const TBigInt& num, long long mul);
38      };
39  };
```

Теперь опишем конструктор от строки, конструктор копирования и оператор присваивания. В конструкторе от строки будем убирать ведущие нули, чтобы не хранить и не обрабатывать их потом.

Листинг 3: $\text{big}_i nt.pp$

```
1   TBigInt::TBigInt(const std::string& str) {
2       long long startPosition = 0; //
3       while (startPosition < str.size() && str[startPosition] == '0') {
4           ++startPosition;
5       }
6
7       if (startPosition == str.size()) { //
8           Data.push_back(0);
9           return;
10      }
11
12      for (long long i = str.size() - 1; i >= startPosition; i -= Digits) {
13          long long nextDigit = 0;
14          long long j = Digits - 1;
15          while (j >= 0) {
16              if (i - j >= startPosition) {
17                  nextDigit = nextDigit * 10 + str[i - j] - '0';
18              }
19              --j;
20          }
21          Data.push_back(nextDigit);
22      }
23  }
24
25  TBigInt::TBigInt(const TBigInt& num) {
26      for (long long i = 0; i < num.Data.size(); ++i) {
27          Data.push_back(num.Data[i]);
28      }
29  }
```

Теперь реализуем операторы ввода и вывода.

Листинг 4: big_int.cpp

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

std::ostream& operator<<(std::ostream& os, const TBigInt& num) {
    os << num.Data[num.Data.size() - 1];
    for (long long i = num.Data.size() - 2; i >= 0; --i) {
        os << std::setfill('0') << std::setw(TBigInt::Digits) << num.Data[i];
    }
    return os;
}
```

Реализуем операции сравнения больших чисел.

Листинг 5: big_int.cpp

```
bool operator<(const TBigInt& lhs, const TBigInt& rhs) {
    if (lhs.Data.size() != rhs.Data.size()) {
        return lhs.Data.size() < rhs.Data.size();
    }
    for (long long i = lhs.Data.size() - 1; i >= 0; --i) {
        if (lhs.Data[i] != rhs.Data[i]) {
            return lhs.Data[i] < rhs.Data[i];
        }
    }
    return false; // lhs == rhs
}

bool operator>(const TBigInt& lhs, const TBigInt& rhs) {
    if (lhs.Data.size() != rhs.Data.size()) {
        return lhs.Data.size() > rhs.Data.size();
    }
    for (long long i = lhs.Data.size() - 1; i >= 0; --i) {
        if (lhs.Data[i] != rhs.Data[i]) {
            return lhs.Data[i] > rhs.Data[i];
        }
    }
    return false; // lhs == rhs
}

bool operator==(const TBigInt& lhs, const TBigInt& rhs) {
    if (lhs.Data.size() != rhs.Data.size()) {
        return false; // lhs != rhs
```

```
28          }
29          for (long long i = lhs.Data.size() - 1; i >= 0; --i) {
30              if (lhs.Data[i] != rhs.Data[i]) {
31                  return false; // lhs != rhs
32              }
33          }
34          return true; // lhs == rhs
35      }
```

Релизуем операции сложения и вычитания. Для сложения и вычитания будем хранить временную переменную для добавления/заема к следующему разряду. Также после всех операций будем убирать ведущие нули, чтобы поддерживать вид чисел без ведущих нулей.

Листинг 6: $\text{big}_i nt.cpp$

```
1       TBigInt operator+(const TBigInt& lhs, const TBigInt& rhs) {
2           //      ,
3           const TBigInt& smaller = (lhs.Data.size() < rhs.Data.size()) ? lhs : rhs;
4           const TBigInt& bigger = (lhs.Data.size() >= rhs.Data.size()) ? lhs : rhs;
5           TBigInt res(bigger);
6           long long carry = 0;
7
8           for (long long i = 0; i < res.Data.size(); ++i) {
9               long long added = (i < smaller.Data.size()) ? smaller.Data[i] + carry :
                    carry;
10              res.Data[i] += added;
11              carry = res.Data[i] / TBigInt::Base;
12              res.Data[i] %= TBigInt::Base;
13          }
14
15          if (carry != 0) {
16              res.Data.push_back(carry);
17          }
18
19          return res;
20      }
21
22      TBigInt operator-(const TBigInt& lhs, const TBigInt& rhs) {
23          if (lhs < rhs) {
24              throw std::logic_error("Subtract the larger from the smaller.");
25          }
26
27          TBigInt res(lhs);
28          long long carry = 0;
29          for (long long i = 0; i < res.Data.size(); ++i) {
30              long long subtrahend = (i < rhs.Data.size()) ? carry + rhs.Data[i] : carry;
31              res.Data[i] -= subtrahend;
```

```
32              if (res.Data[i] < 0) {
33                  carry = 1;
34                  res.Data[i] += TBigInt::Base;
35              } else {
36                  carry = 0;
37              }
38          }
39
40          res.RemoveNulls();
41
42          return res;
43      }
```

Реализуем умножение двух длинных чисел и умножение длинного на короткое. Если один из операндов имеет длину 1, то будет вызвано умножение длинного на короткое.

Листинг 7: $\mathrm{big}_i nt.cpp$

```
1       TBigInt TBigInt::WeakMultiply(const TBigInt& num, long long mul) {
2           TBigInt res(num);
3           long long carry = 0;
4           for (long long i = 0; i < res.Data.size() || carry > 0; ++i) {
5               if (i == res.Data.size()) {
6                   res.Data.push_back(0);
7               }
8               long long curr = carry + res.Data[i] * mul;
9               res.Data[i] = curr % TBigInt::Base;
10              carry = curr / TBigInt::Base;
11          }
12
13          res.RemoveNulls();
14
15          return res;
16      }
17
18      TBigInt operator*(const TBigInt& lhs, const TBigInt& rhs) {
19          if (rhs.Data.size() == 1) {
20              return TBigInt::WeakMultiply(lhs, rhs.Data[0]);
21          } else if (lhs.Data.size() == 1) {
22              return TBigInt::WeakMultiply(rhs, lhs.Data[0]);
23          }
24
25          TBigInt res;
26          res.Data.resize(lhs.Data.size() + rhs.Data.size());
27          for (long long i = 0; i < lhs.Data.size() + rhs.Data.size(); ++i) {
28              res.Data.push_back(0);
29          }
30
```

```
31          for (long long i = 0; i < lhs.Data.size(); ++i) {
32              for (long long j = 0, carry = 0; j < rhs.Data.size() || carry; ++j) {
33                  long long curr = res.Data[i+j] + lhs.Data[i] * (j < rhs.Data.size() ?
                        rhs.Data[j] : 0) + carry;
34                  res.Data[i+j] = curr % TBigInt::Base;
35                  carry = curr / TBigInt::Base;
36              }
37          }
38
39          res.RemoveNulls();
40
41          return res;
42      }
```

Теперь реализуем деление длинного на длинное и длинного на короткое. При делении на число, имеющее длину 1, будет вызвано деление длинного на короткое для ускорения.

Листинг 8: $\text{big}_i nt.cpp$

```
1       TBigInt TBigInt::WeakDivision(const TBigInt& num, long long div) {
2           TBigInt res(num);
3           long long carry = 0;
4           for (long long i = res.Data.size() - 1; i >= 0; --i) {
5               long long curr = res.Data[i] + carry * TBigInt::Base;
6               res.Data[i] = curr / div;
7               carry = curr % div;
8           }
9
10          res.RemoveNulls();
11
12          return res;
13      }
14
15      long long TBigInt::FindBin(const TBigInt& num, const TBigInt& div) {
16          long long down = 0;
17          long long up = Base;
18          TBigInt tmpRes;
19          long long res = 0;
20          long long mid = 0;
21          while (down <= up) {
22              mid = (up + down) / 2;
23              tmpRes = WeakMultiply(div, mid);
24              if (num < tmpRes) {
25                  up = mid - 1;
26              } else {
27                  res = mid;
28                  down = mid + 1;
```

```
            }
        }
        return res;
    }

    TBigInt operator/(const TBigInt& lhs, const TBigInt& rhs) {
        if (rhs.Data.size() == 1 && rhs.Data[0] == 0) {
            throw std::logic_error("Division by zero.");
        }

        if (rhs.Data.size() == 1) {
            return TBigInt::WeakDivision(lhs, rhs.Data[0]);
        }

        TBigInt res;
        TBigInt tmpQuotient;
        for (long long i = lhs.Data.size() - 1; i >= 0; --i) {
            tmpQuotient.Data.insert(tmpQuotient.Data.begin(), lhs.Data[i]);
            long long tmpDivider = TBigInt::FindBin(tmpQuotient, rhs);
            res.Data.insert(res.Data.begin(), tmpDivider);
            tmpQuotient = tmpQuotient - TBigInt::WeakMultiply(rhs, tmpDivider);
        }

        res.RemoveNulls();

        return res;
    }
```

Осталось реализовать быстрое возведение в степень.

Листинг 9: $\text{big}_i nt.cpp$

```
    TBigInt TBigInt::FastPow(TBigInt base, TBigInt degree) {
        TBigInt null("0");
        if (base == null && degree == null) {
            throw std::logic_error("Uncertainty, 0^0.");
        }
        TBigInt res("1");
        while (degree > null) {
            if (degree.Data.back() % 2 == 1) {
                res = res * base;
            }
            base = base * base;
            degree = WeakDivision(degree, 2);;
        }
        return res;
    }
```

3 Консоль

```
MacBook-Air-K:solution AK$ cat test.t
124556
12434234235
+
1
2
<
123
123
*
MacBook-Air-K:solution AK$ ./solution <test.t
12434358791
true
15129
```

4 Тест производительности

Тест производительности представляет из себя следующее: моя реализация длинной арифметики сравнивается с реализацией из библиотеки GMP, время на считывание данных не учитывается. Тест производительности состоит из трёх тестов, каждый из которых состоит из текста длины 100, 1000 и 10000 символов. Каждый тест состоит из 7 подтестов: сложение, вычитание, умножение, сравнение на меньше, сравнение на равенство, деление и возведение в степень.

```
MacBook-Air-K:da_lab_06 AK$ ./wrapper.sh
[2021-03-16 16:03:03] [INFO] Compiling...
g++ -std=c++17 -O3 -Wextra -Wall -Wno-sign-compare -pedantic big_int.o bench.cpp
-o bench -lgmp -lgmpxx
[2021-03-16 16:03:06] [INFO] Generating tests (examples for each test=[10000])...
[2021-03-16 16:03:06] [INFO] Executing tests/01.t...
[2021-03-16 16:03:06] [INFO] Executing tests/02.t...
[2021-03-16 16:03:06] [INFO] Executing tests/03.t...
[2021-03-16 16:03:06] [INFO] Executing tests/04.t...
[2021-03-16 16:03:06] [INFO] Executing tests/05.t...
[2021-03-16 16:03:06] [INFO] Executing tests/06.t...
[2021-03-16 16:03:07] [INFO] Executing tests/07.t...
MacBook-Air-K:da_lab_06 AK$ cat res.txt
TBigInt     10283us
GMP         695us

TBigInt     13311us
GMP         488us

TBigInt     28069us
GMP         756us

TBigInt     843us
GMP         957us

TBigInt     786us
GMP         808us

TBigInt     1582502us
GMP         660us

TBigInt     258197us
```

```
GMP         760us

MacBook-Air-K:da_lab_06 AK$ ./wrapper.sh
[2021-03-16 16:03:03] [INFO] Compiling...
g++ -std=c++17 -O3 -Wextra -Wall -Wno-sign-compare -pedantic big_int.o bench.cpp
-o bench -lgmp -lgmpxx
[2021-03-16 16:03:06] [INFO] Generating tests (examples for each test=[1000])...
[2021-03-16 16:03:06] [INFO] Executing tests/01.t...
[2021-03-16 16:03:06] [INFO] Executing tests/02.t...
[2021-03-16 16:03:06] [INFO] Executing tests/03.t...
[2021-03-16 16:03:06] [INFO] Executing tests/04.t...
[2021-03-16 16:03:06] [INFO] Executing tests/05.t...
[2021-03-16 16:03:06] [INFO] Executing tests/06.t...
[2021-03-16 16:03:07] [INFO] Executing tests/07.t...
MacBook-Air-K:da_lab_06 AK$ cat res.txt
TBigInt 1088us
GMP     66us

TBigInt 773us
GMP     40us

TBigInt 2364us
GMP     67us

TBigInt 83us
GMP     92us

TBigInt 120us
GMP     171us

TBigInt 268659us
GMP     99us

TBigInt 30511us
GMP     87us

MacBook-Air-K:da_lab_06 AK$ ./wrapper.sh
[2021-03-16 16:03:03] [INFO] Compiling...
g++ -std=c++17 -O3 -Wextra -Wall -Wno-sign-compare -pedantic big_int.o bench.cpp
-o bench -lgmp -lgmpxx
[2021-03-16 16:03:06] [INFO] Generating tests (examples for each test=[100])...
```

```
[2021-03-16 16:03:06] [INFO] Executing tests/01.t...
[2021-03-16 16:03:06] [INFO] Executing tests/02.t...
[2021-03-16 16:03:06] [INFO] Executing tests/03.t...
[2021-03-16 16:03:06] [INFO] Executing tests/04.t...
[2021-03-16 16:03:06] [INFO] Executing tests/05.t...
[2021-03-16 16:03:06] [INFO] Executing tests/06.t...
[2021-03-16 16:03:07] [INFO] Executing tests/07.t...
MacBook-Air-K:da_lab_06 AK$ cat res.txt
TBigInt 101us
GMP     7us

TBigInt 109us
GMP     8us

TBigInt 756us
GMP     12us

TBigInt 12us
GMP     14us

TBigInt 8us
GMP     10us

TBigInt 16160us
GMP     5us

TBigInt 2728us
GMP     9us
```

Как видно из тестов, все арифметические операции, представленные в библиотеке GMP, кроме операций сравнения, значительно быстрее наивных алгоритмов.

5 Выводы

Выполнив шестую лабораторную работу по курсу «Дискретный анализ», я познакомился с алгоритмами длинной арифметики. Эти алгоритмы могут позволить удобно работать с большими числами, поддержки которых нет в стандартном C++. Вычисления над такими числами могут быть полезны в научной сфере, либо же при работе с большими данными,.

Список литературы

[1] *e-maxx.*
URL: `https://e-maxx.ru/algo/` (дата обращения: 09.03.2021).

[2] *C++ Reference.*
URL: `https://en.cppreference.com` (дата обращения: 08.03.2021).