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: К. С. Саженов
Преподаватель: И. Н. Симахин
Группа: М8О-308Б-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:

1. Сложение
2. Вычитание
3. Умножение
4. Возведение в степень
5. 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:

1. Больше
2. Меньше
3. 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

В данной лабораторной работе я использовал, так называемый, little-endian(от меньшего к большему) порядок записи числа в памяти.

Поскольку операторы сравнения реализовать довольно тривиально, я не буду здесь их описывать.

Сложение было реализовано методом «в столбик»:

1. Складываем младшие разряды,
2. Если произошло переполнение за основание, то добавляем к значению следующего разряда 1,
3. Продолжаем с 1 пункта для следующего разряда.

Вычитание также было сделано «в столбик».

Умножение было реализовано в двух вариантах(выбирать какой именно вариант использовать можно при помощи директивы препроцессора `USE_FFT`):

1. При помощи обычного умножения «в столбик» происходит перемножение каждого множителя с каждым(с учетом «переизбытка» по данному основанию),
2. При помощи, т.н., умножения при помощи Быстрого преобразования Фурье(FFT, англ.).

Деление было реализовано при помощи такого алгоритма:

1. При помощи бинарного поиска ищем число, произведение которого с основанием будет меньше первого разряда (в последующих итерациях количество разрядов может увеличиться),
2. Вычисляем остаток от деления,
3. Сохраняем нынешнее значение разряда в массив ответов,
4. Повторяем 1-3 пункты, пока делимое не обнулится,
5. Убираем лишние старшие нули.

2 Исходный код

Файл `longarithmetic.h`:

```
1
2
3  //
4  // Created by sakost on 06.03.2021.
5  //
6
7  #ifndef DA_LAB_6_LONGARITHMETIC_H
8  #define DA_LAB_6_LONGARITHMETIC_H
9
10 #include <cmath>
11 #include <iomanip>
12 #include <string>
13 #include <type_traits>
14 #include <vector>
15
16 #ifdef FAST_MUL
17 #include <complex>
18 #endif
19
20 static const long double PI = std::acos(-1.l);
21
22 class TInvalidOperands : public std::exception {
23 public:
24     const char *what() const noexcept override { return "Invalid operands"; }
25 };
26
27
28 namespace detail {
29     template<typename T, typename = std::enable_if<std::is_arithmetic_v<T>, T>>
30     constexpr T custom_constexpr_pow(const T &base, const T &power) {
31         T ans = 1;
32         for (size_t i(0); i < power; ++i) {
33             ans *= base;
34         }
35         return ans;
36     }
37 }// namespace detail
38
39 // template<typename T>
40 class TLongArithmetic {
41     using T = long long;
42     static_assert(std::is_integral_v<T>, "Integral required.");
43
44     static const T BASE_COUNT_DIGITS = 5;
45     static const T BASE = detail::custom_constexpr_pow<T>(10, BASE_COUNT_DIGITS);
46
```

```
47      friend std::ostream &operator<<(std::ostream &out, const TLongArithmetic &lhs);
48
49      friend std::istream &operator>>(std::istream &in, TLongArithmetic &rhs);
50
51      using ssizeType = typename std::string::size_type;
52      using vsizeType = typename std::vector<T>::size_type;
53
54  public:
55      TLongArithmetic() : TLongArithmetic(T()) {}
56
57      TLongArithmetic(T n) {// NOLINT(google-explicit-constructor)
58          mData.clear();
59          if (n < BASE)
60              mData.push_back(n);
61          else {
62              for (; n > 0; n /= BASE)
63                  mData.push_back(n % BASE);
64          }
65          Normalize();
66      }
67
68      explicit TLongArithmetic(const std::vector<T> &data) { mData = data; }
69
70      explicit TLongArithmetic(std::string &input) {
71          if (input.empty()) {
72              mData.push_back(0);
73              return;
74          }
75
76          input = TrimLeadingZeroes(input);
77
78          std::stringstream tempSS;
79          mData.clear();
80
81          for (long long i = (long long) input.size() - 1; i >= 0; i -= BASE_COUNT_DIGITS
                ) {
82              ssizeType start = std::max(i - (long long) BASE_COUNT_DIGITS + 1ll, 0ll);
83              ssizeType end = i - start + 1;
84
85              tempSS << input.substr(start, end);
86              T radix = 0;
87              tempSS >> radix;
88              mData.push_back(radix);
89              tempSS.clear();
90          }
91      }
92
93      explicit operator std::string() const {
94          if (mData.empty())
```

```
95              return "0";
96          std::stringstream res;
97          res << mData.back();
98
99          for (auto el = std::next(mData.crbegin()); el != mData.crend(); ++el) {
100             res << std::setfill('0') << std::setw(BASE_COUNT_DIGITS) << *el;
101         }
102
103         return res.str();
104     }
105
106     bool operator==(const TLongArithmetic &rhs) const {
107         return this == &rhs || mData == rhs.mData ||
108                ((mData.empty() || (mData.size() == 1 && mData.front() == 0)) &&
109                 (rhs.mData.empty() || (rhs.mData.size() == 1 && rhs.mData.front() == 0))
                    );
110     }
111
112     bool operator!=(const TLongArithmetic &rhs) const {
113         return mData != rhs.mData;
114     }
115
116     bool operator<(const TLongArithmetic &rhs) const {
117         if (mData.size() != rhs.mData.size()) {
118             return mData.size() < rhs.mData.size();
119         }
120         return std::lexicographical_compare(mData.rbegin(), mData.rend(),
121                                             rhs.mData.rbegin(), rhs.mData.rend());
122     }
123
124     bool operator>(const TLongArithmetic &rhs) const {
125         return !(*this < rhs) && (*this != rhs);
126     }
127
128     bool operator<=(const TLongArithmetic &rhs) const {
129         return (*this == rhs) || (*this < rhs);
130     }
131
132     bool operator>=(const TLongArithmetic &rhs) const {
133         return (*this == rhs) || (*this > rhs);
134     }
135
136     TLongArithmetic &operator+=(const TLongArithmetic &rhs) {
137         auto maxSize = std::max(mData.size(), rhs.mData.size()) + 1;
138         mData.resize(maxSize, (T) 0);
139
140         T carry = (T) 0;
141         for (vsizeType i = 0; i < maxSize || carry != 0; i++) {
142             if (i >= mData.size()) {
```

```
143                 mData.push_back((T) 0);
144             }
145             T cur = mData[i] + rhs.At(i) + carry;
146             mData[i] = cur % BASE;
147             carry = cur >= BASE;
148         }
149         this->Normalize();
150         return *this;
151     }
152 
153     TLongArithmetic operator+(const TLongArithmetic &rhs) const {
154         TLongArithmetic temp = *this;
155         temp += rhs;
156         return temp;
157     }
158 
159     TLongArithmetic &operator-=(const TLongArithmetic &rhs) {
160         if (*this < rhs) {
161             throw TInvalidOperands();
162         }
163         auto min_size = rhs.mData.size();
164 
165         T carry = (T) 0;
166         for (vsizeType i = 0; i < min_size || carry != 0; i++) {
167             T cur = mData[i] - rhs.At(i) - carry;
168             carry = cur < 0;
169             if (carry != 0) {
170                 cur += BASE;
171             }
172             mData[i] = cur;
173         }
174         this->Normalize();
175         return *this;
176     }
177 
178     TLongArithmetic operator-(const TLongArithmetic &rhs) const {
179         TLongArithmetic temp = *this;
180         temp -= rhs;
181         return temp;
182     }
183 
184     const TLongArithmetic &operator--() {
185         *this -= 1;
186         return *this;
187     }
188 
189     const TLongArithmetic operator--(int) { return --(*this); }
190 
191     const TLongArithmetic &operator++() {
```

```
192         *this += 1;
193         return *this;
194     }
195 
196     const TLongArithmetic operator++(int) { return ++(*this); }
197 
198     TLongArithmetic &operator*=(const T &rhs) {
199         T carry = (T) 0;
200         for (typename std::vector<T>::size_type i = 0;
201              i < mData.size() || carry != 0; i++) {
202             if (i == mData.size()) {
203                 mData.push_back(0);
204             }
205             T cur = carry + mData[i] * rhs;
206             mData[i] = cur % BASE;
207             carry = cur / BASE;
208         }
209         this->Normalize();
210         return *this;
211     }
212 
213     TLongArithmetic operator*(const T &rhs) const {
214         TLongArithmetic temp = *this;
215         temp *= rhs;
216         return temp;
217     }
218 
219     TLongArithmetic &operator*=(const TLongArithmetic &rhs) {
220         *this = (*this * rhs);
221         return *this;
222     }
223 
224     TLongArithmetic operator*(const TLongArithmetic &rhs) const {
225         TLongArithmetic res(0);
226 
227 #ifdef FAST_MUL
228         std::vector<base> fa(mData.begin(), mData.end()),
229                 fb(rhs.mData.begin(), rhs.mData.end());
230         std::size_t n = 1;
231         while (n < std::max(fa.size(), fb.size())) {
232             n <<= 1;
233         }
234         n <<= 1;
235         fa.resize(n), fb.resize(n);
236 
237         FFT(fa, false), FFT(fb, false);
238         for (std::size_t i = 0; i < n; ++i) {
239             fa[i] *= fb[i];
240         }
```

```
241
242         FFT(fa, true);
243
244         res.mData.resize(n);
245         T carry = 0;
246         for (std::size_t i = 0; i < n || carry != 0; ++i) {
247             T cur = std::llroundl(fa[i].real()) + carry;
248             res.mData[i] = cur % BASE;
249             carry = cur / BASE;
250         }
251
252 #else
253         vsizeType lhsSize = mData.size(), rhsSize = rhs.mData.size();
254
255         res.mData.resize(lhsSize + rhsSize, 0);
256
257         for (vsizeType i = 0; i < lhsSize; ++i) {
258             T carry = 0;
259             if (mData[i] == 0) {
260                 continue;
261             }
262             for (vsizeType j = 0; j < rhsSize || carry != 0; ++j) {
263                 T cur = mData[i] * rhs.At(j) + carry + res.mData[i + j];
264                 carry = cur / BASE;
265                 res.mData[i + j] = cur % BASE;
266             }
267         }
268 #endif
269
270         res.Normalize();
271         return res;
272     }
273
274     TLongArithmetic operator/(const TLongArithmetic &rhs) const {
275         if (rhs == 0) {
276             throw TInvalidOperands();
277         }
278         if (*this == rhs)
279             return 1;
280         if (*this < rhs)
281             return 0;
282
283         TLongArithmetic res, cv = 0;
284         res.mData.resize(mData.size());
285
286         for (long long i = (long long) mData.size() - 1; i >= 0; --i) {
287             cv.mData.insert(cv.mData.begin(), mData[i]);
288             if (cv.mData.back() == 0) {
289                 cv.mData.pop_back();
```

```
290             }
291             T x = 0, l = 0, r = BASE;
292             while (l <= r) {
293                 T m = (l + r) / 2;
294                 TLongArithmetic cur(rhs * m);
295                 if (cur <= cv) {
296                     x = m;
297                     l = m + 1;
298                 } else {
299                     r = m - 1;
300                 }
301             }
302             res.mData[i] = x;
303             cv = cv - rhs * x;
304         }
305         res.Normalize();
306         return res;
307     }
308 
309     TLongArithmetic &operator/=(const TLongArithmetic &rhs) {
310         *this = *this / rhs;
311         return *this;
312     }
313 
314     TLongArithmetic operator%(const TLongArithmetic &rhs) {
315         if (rhs == 2) {
316             return mData.front() % 2;
317         }
318         if (rhs == 0) {
319             throw TInvalidOperands();
320         }
321         if (mData.empty() || rhs == 1 || *this == rhs) {
322             return 0;
323         }
324 
325         if (*this < rhs) {
326             return *this;
327         }
328         auto temp = *this / rhs;
329         return *this - temp * (*this);
330     }
331 
332     TLongArithmetic Power(const TLongArithmetic &rhs) {
333         if (*this == 0) {
334             if (rhs == 0) {
335                 throw TInvalidOperands();
336             }
337             return 0;
338         }
```

```
339         if (rhs == 0 || *this == 1) {
340             return 1;
341         }
342         if (rhs == 1) {
343             return *this;
344         }
345         TLongArithmetic result = 1, power = rhs, number = *this;
346         while (power > 0) {
347             if (power % 2 > 0) {
348                 result *= number;
349             }
350             number *= number;
351             power /= 2;
352         }
353         return result;
354     }
355 
356 private:
357     void Normalize() {
358         if (mData.empty()) {
359             mData.push_back((T) 0);
360             return;
361         }
362         while (mData.back() == 0 && mData.size() > 1) {
363             mData.pop_back();
364         }
365     }
366 
367     [[nodiscard]] T At(const vsizeType &i) const {
368         if (i < 0 || i >= mData.size())
369             return (T) 0;
370         return mData[i];
371     }
372 
373     static std::string TrimLeadingZeroes(const std::string &input) {
374         ssizeType i = 0;
375         while (input.size() != i - 1 && input[i] == '0') {
376             i++;
377         }
378         return input.substr(i);
379     }
380 
381 #ifdef FAST_MUL
382     using double_base = long double;
383     using base = std::complex<double_base>;
384 
385     void FFT(std::vector<base> &a, bool inverse = false) const {
386         std::vector<base>::size_type n = a.size();
387         if (n == 1)
```

```
            return;

        std::vector<base> a0(n / 2), a1(n / 2);
        for (std::size_t i(0), j(0); i < n; i += 2, ++j) {
            a0[j] = a[i];
            a1[j] = a[i + 1];
        }

        FFT(a0, inverse);
        FFT(a1, inverse);

        double_base ang = 2 * PI / n * (inverse ? -1 : 1);
        base w(1), wn(std::cos(ang), std::sin(ang));
        for (std::size_t i = 0; i < n / 2; ++i) {
            a[i] = a0[i] + w * a1[i];
            a[i + n / 2] = a0[i] - w * a1[i];
            if (inverse) {
                a[i] /= 2;
                a[i + n / 2] /= 2;
            }
            w *= wn;
        }
    }

#endif

    std::vector<T> mData;
};

std::ostream &operator<<(std::ostream &out, const TLongArithmetic &lhs) {
    out << (std::string) lhs;
    return out;
}

std::istream &operator>>(std::istream &in, TLongArithmetic &rhs) {
    std::string inp;
    in >> inp;
    rhs = TLongArithmetic(inp);
    return in;
}

#endif// DA_LAB_6_LONGARITHMETIC_H
```

Файл `main.cpp`:

```
#include <iostream>
#include <vector>

#define FAST_MUL
#include "longarithmetic.h"
```

```
6
7  using namespace std;
8
9
10 int main() {
11     ios_base::sync_with_stdio(false);
12     cout.tie(nullptr);
13     cin.tie(nullptr);
14
15     TLongArithmetic a, b;
16     std::vector<std::string> result = {"false", "true"};
17
18     char op;
19     while (std::cin >> a >> b >> op) {
20         try {
21             switch (op) {
22                 case '+':
23                     cout << a + b << '\n';
24                     break;
25                 case '-':
26                     cout << a - b << '\n';
27                     break;
28                 case '*':
29                     cout << a * b << '\n';
30                     break;
31                 case '^':
32                     cout << a.Power(b) << '\n';
33                     break;
34                 case '/':
35                     cout << a / b << '\n';
36                     break;
37                 case '>':
38                     cout << result[a > b] << '\n';
39                     break;
40                 case '<':
41                     cout << result[a < b] << '\n';
42                     break;
43                 case '=':
44                     cout << result[a == b] << '\n';
45                     break;
46                 default:
47                     cout << "Error\n";
48             }
49         } catch (TInvalidOperands &e) {
50             cout << "Error\n";
51         }
52     }
53     cout.flush();
54     return 0;
```

```
55 || }
```

3 Консоль

```
~/university/2 course/diskran/lab6/cmake-build-debug on master
$ cmake ../
--The C compiler identification is AppleClang 13.0.0.13000029
--The CXX compiler identification is AppleClang 13.0.0.13000029
--Detecting C compiler ABI info
--Detecting C compiler ABI info -done
--Check for working C compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/
-skipped
--Detecting C compile features
--Detecting C compile features -done
--Detecting CXX compiler ABI info
--Detecting CXX compiler ABI info -done
--Check for working CXX compiler: /Applications/Xcode.app/Contents/Developer/Toolchain
-skipped
--Detecting CXX compile features
--Detecting CXX compile features -done
--Configuring done
--Generating done
--Build files have been written to: /Users/k.sazhenov/university/2 course/diskran/lab6

~/university/2 course/diskran/lab6/cmake-build-debug on master
$ cmake --build .
[ 50%] Building CXX object CMakeFiles/da_lab_6.dir/main.cpp.o
[100%] Linking CXX executable da_lab_6
[100%] Built target da_lab_6

~/university/2 course/diskran/lab6/cmake-build-debug on master
$ cat ../input.txt
38943432983521435346436
354353254328383
+
904094384738493247293847 3843
2343543
-
972323
2173937
>
2
```

```
3
-

~/university/2 course/diskran/lab6/cmake-build-debug on master
$ ./da_lab_6 <../input.txt
38943433337874689674819
90409438473849324729361303 00
false
Error
```

4 Тест производительности

Поскольку в C++ нет встроенной длинной арифметики, я буду сравнивать реализованную мной библиотеку с длинной арифметикой Python-a. А также я приведу сравнение реализации с FFT.

Тест будет проводиться на рандомно сгенерированных **валидных** данных:

```
~/university/2 course/diskran/lab6 on master
$ time cmake-build-debug/da_lab_6_slow < tests/03.t > /dev/null
cmake-build-debug/da_lab_6_slow < tests/03.t > /dev/null  2.19s user 0.02s
system 85% cpu 2.595 total

~/university/2 course/diskran/lab6 on master
$ time cmake-build-debug/da_lab_6_fast < tests/03.t > /dev/null
cmake-build-debug/da_lab_6_fast <tests/03.t >/dev/null  6.54s user 0.01s system
99% cpu 6.564 total

~/university/2 course/diskran/lab6 on master
$ time python solver.py <tests/03.t >/dev/null
python solver.py < tests/03.t > /dev/null  0.64s user 0.25s system 95% cpu
0.934 total
```

Как видно из тестов выше, мы **крайне** неэффективно используем память и совершаем много лишних действий. Поэтому python оказался и быстрее больше, чем в 10 раз, хоть он и является интерпретируемым и, по слухам, якобы медленным языком.

5 Выводы

Задача об эффективной реализации длинной арифметики является одной из самых актуальных на данный момент времени, поскольку при помощи длинной аримфетики происходит огромное количество вычислений, которые, чаще всего, являются жизненнонеобходимыми для людей, использующих их.

Одна из таких задач – поиск различных чисел на суперкомпьютере. В данной задаче помимо алгоритмов длинной арифметики, используется ещё куча других, что никак не умаляет их важности.

Одна из реализаций длинной аримфетики есть в языке программирования Python, который является интерпретируемым языком программирования.

Как ни странно, но моя реализация быстрого умножения(которая основана на FFT) получилась медленней наивной реализации. Такое произошло, поскольку, чтобы реализовать умножение с FFT(теоретическая сложность которого $O(nlog(n))$), необходимо совершать очень много операций копирования – примерно $O(n^2log(n))$ в моей реализации, что очень неэффективно. Исходя из количества копирований памяти, получается, что моя реализация быстрого умножения имеет сложность $O(n^2log(n))$. Очень интересно, как можно было бы соптимизировать данную асимптотику.

В общем и целом – длинная арифметика является очень актуальной задачей, которую, оказывается, можно решить не только наивным алгоритмом, но и «более математическим», что показлось мне очень интересным фактом.

Список литературы

[1] Т. Кормен, Ч Лейзерсон, Р Ривест, К Штайн. *Алгоритмы: построение и анализ, 3-е издание.* — Издательский дом «Вильямс», 2013. Перевод с английского: ООО «И.Д. Вильямс» — 1328 с. (ISBN 978-5-8459-1794-2 (рус.))