**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ВТ**

Курсовая работа

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: измерение временной сложности алгоритма**

**Вариант 24**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7308 |  | Замыслов Н.Ю. |
| Студентка гр. 7308 |  | Цебульский С.А. |
| Преподаватель |  | Колинько П.Г. |

Санкт-Петербург

2019

**Цель работы**

Провести эксперимент по определению оценки сложности цепочки операций и проанализировать результат.

**Постановка задачи**

Выполнить статистический эксперимент по измерению временной сложности алгоритма обработки данных, использующего стандартную библиотеку шаблонов.

Доработать программу лабораторной работы 6 с красно-чёрным деревом таким образом, чтобы она генерировала множества мощностью, меняющейся от 10 до 200, измеряла время выполнения цепочки операций над множествами и последовательностями и выводила результат в текстовый файл. Первая строка этого файла должна содержать количество опытов, а последующие — по паре значений «размер входа — время» для каждого опыта. Затем эти данные обрабатываются, и по результатам обработки делается заключение о временной сложности алгоритма.

Для повышения надёжности эксперимента предусмотреть в программе перехват исключительных ситуаций таким образом, чтобы сбой сводился просто к пропуску очередного шага эксперимента. В частности, рекомендуется перехватывать ситуацию *bad\_alloc*, возбуждаемую конструктором при нехватке памяти.

**Реализованная программа**

**Основа курсовой работы**

Для работы с множествами и последовательностями в лабораторной 6 был разработан класс MySeq. Для хранения множеств без повторений использовалась структура set<int>. Данная структура реализует красно-черное дерево с хранением только ключей, поэтому был выбран именно этот контейнер для хранения элементов множества.

Также в классе есть поле vector<set<int>::iterator> для хранения итераторов на элементы дерева, задающих нужный порядок обхода дерева для вывода последовательности в исходном порядке.

Для операций над множествами в классе перегружены операторы &, |, / и ^, которые используют функции set\_union, set\_intersection, set\_difference, set\_symmetric\_difference из стандартной библиотеки.

Благодаря тому, что вектор итераторов при одинаковых элементах в последовательности хранит одинаковые итераторы на элементы дерева, отпала необходимость использовать шаблон multiset<int> для хранения множества с повторениями, что позволило сэкономить память.

Программа реализует три операции над последовательностями:

* Erase – Из последовательности исключается часть, ограниченная порядковыми номерами от *p*1 до *p*2
* Mul – Последовательность сцепляется сама с собой заданное количество раз.
* Excl – Вторая последовательность исключается из первой, если она является её частью.

**Генерация множеств**

Исходная программа была доработана так, чтобы последовательно генерировать множества размерностей от 10 до 200 и производить над ними цепочку операций. Для этого была написана функция regenerate(), которая вызывается каждую новую итерацию и генерирует множество новой мощности. Также добавлена проверка bad\_alloc исключения.

**Оценка временной сложности используемых операций**

**Таблица 1**

**Временная сложность операций**

|  |  |
| --- | --- |
| **Операция** | **Временная сложность** |
| & | O(N) |
| **|** | O(N) |
| **^** | O(N) |
| **/** | O(N) |
| excl | O(N) |
| mul | O(N) |
| erase | O(N) |

Таким образом, временная сложность цепочки из этих операций будет O(N).

**Результаты эксперимента**

**Таблица 2**

**Входные данные**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| N | Time | N | Time | N | Time | N | Time |
| 10 | 0,002599 | 58 | 0,016258 | 106 | 0,018239 | 154 | 0,028399 |
| 11 | 0,002617 | 59 | 0,015645 | 107 | 0,018152 | 155 | 0,028986 |
| 12 | 0,002599 | 60 | 0,018699 | 108 | 0,019502 | 156 | 0,030801 |
| 13 | 0,002821 | 61 | 0,015923 | 109 | 0,020023 | 157 | 0,029837 |
| 14 | 0,003254 | 62 | 0,013193 | 110 | 0,021559 | 158 | 0,03513 |
| 15 | 0,003521 | 63 | 0,014879 | 111 | 0,028048 | 159 | 0,031405 |
| 16 | 0,00375 | 64 | 0,015179 | 112 | 0,021011 | 160 | 0,036774 |
| 17 | 0,003829 | 65 | 0,015513 | 113 | 0,022016 | 161 | 0,033309 |
| 18 | 0,008207 | 66 | 0,014605 | 114 | 0,019663 | 162 | 0,029757 |
| 19 | 0,00434 | 67 | 0,015548 | 115 | 0,02765 | 163 | 0,031653 |
| 20 | 0,006944 | 68 | 0,017871 | 116 | 0,022322 | 164 | 0,041707 |
| 21 | 0,005602 | 69 | 0,016207 | 117 | 0,022243 | 165 | 0,038244 |
| 22 | 0,010792 | 70 | 0,015645 | 118 | 0,024531 | 166 | 0,03483 |
| 23 | 0,005529 | 71 | 0,018724 | 119 | 0,029258 | 167 | 0,035599 |
| 24 | 0,006639 | 72 | 0,028739 | 120 | 0,035379 | 168 | 0,032825 |
| 25 | 0,008871 | 73 | 0,015812 | 121 | 0,034591 | 169 | 0,037798 |
| 26 | 0,009199 | 74 | 1,63E-02 | 122 | 0,035848 | 170 | 0,033452 |
| 27 | 0,00785 | 75 | 0,016581 | 123 | 0,039444 | 171 | 0,035858 |
| 28 | 0,014315 | 76 | 0,014713 | 124 | 0,031234 | 172 | 0,03442 |
| 29 | 0,007279 | 77 | 0,019718 | 125 | 0,022761 | 173 | 0,035135 |
| 30 | 0,007558 | 78 | 0,020948 | 126 | 0,023748 | 174 | 0,038916 |
| 31 | 0,007607 | 79 | 0,021882 | 127 | 0,022891 | 175 | 0,040234 |
| 32 | 0,008081 | 80 | 0,019901 | 128 | 0,02455 | 176 | 0,057155 |
| 33 | 0,008271 | 81 | 0,020968 | 129 | 0,027002 | 177 | 0,048676 |
| 34 | 0,008814 | 82 | 0,01918 | 130 | 0,022876 | 178 | 0,038636 |
| 35 | 0,01234 | 83 | 0,018981 | 131 | 0,023128 | 179 | 0,036022 |
| 36 | 0,009173 | 84 | 0,013605 | 132 | 0,023669 | 180 | 0,037363 |
| 37 | 0,009658 | 85 | 0,015832 | 133 | 0,024332 | 181 | 0,045893 |
| 38 | 0,011091 | 86 | 0,019701 | 134 | 0,024155 | 182 | 0,03896 |
| 39 | 0,012 | 87 | 0,020227 | 135 | 0,027431 | 183 | 0,045505 |
| 40 | 0,012289 | 88 | 0,015191 | 136 | 0,026316 | 184 | 0,036753 |
| 41 | 0,011626 | 89 | 0,022658 | 137 | 0,023537 | 185 | 0,038319 |
| 42 | 0,010951 | 90 | 0,019999 | 138 | 0,025662 | 186 | 0,040629 |
| 43 | 0,009401 | 91 | 0,015468 | 139 | 0,025062 | 187 | 0,037612 |
| 44 | 0,011321 | 92 | 0,016286 | 140 | 0,026249 | 188 | 0,038401 |
| 45 | 0,010691 | 93 | 0,016825 | 141 | 0,028802 | 189 | 0,04363 |
| 46 | 0,012009 | 94 | 0,01847 | 142 | 0,024993 | 190 | 0,040837 |
| 47 | 0,012075 | 95 | 0,016824 | 143 | 0,028253 | 191 | 0,040685 |
| 48 | 0,010806 | 96 | 0,020193 | 144 | 0,024473 | 192 | 0,041791 |
| 49 | 0,012565 | 97 | 0,017648 | 145 | 0,02629 | 193 | 0,040269 |
| 50 | 0,013109 | 98 | 0,017051 | 146 | 0,034872 | 194 | 0,042003 |
| 51 | 0,011751 | 99 | 0,016477 | 147 | 0,033296 | 195 | 0,062584 |
| 52 | 0,013125 | 100 | 0,017781 | 148 | 0,027505 | 196 | 0,050527 |
| 53 | 0,013053 | 101 | 0,017462 | 149 | 0,030068 | 197 | 0,040794 |
| 54 | 0,014299 | 102 | 0,018579 | 150 | 0,030098 | 198 | 0,045483 |
| 55 | 0,014611 | 103 | 0,024821 | 151 | 0,032104 | 199 | 0,047508 |
| 56 | 0,015157 | 104 | 0,017344 | 152 | 0,040322 | 200 | 0,045028 |
| 57 | 0,012185 | 105 | 0,020392 | 153 | 0,036617 |  |  |

**Таблица 3**

**Результаты работы программы RG32**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | (1) | (ln N) | (N) | (N ln N) | (N^2) | (N^3) | (N^4) |  |  |
| **Вариант** | **D** | **S** | **K** | **c0** | **c1** | **c2** | **c3** | **c4** | **c5** | **c6** |  |  |
| **1** | 1,47E-04 | 1,21E-02 | 1 | 2,31E-02 | 0,00E+00 | 0,00E+00 | 0,00E+00 | 0,00E+00 | 0,00E+00 | 0,00E+00 | 0 | 191 |
| **2** | 3,57E-05 | 5,97E-03 | 2 | -4,24E-02 | 1,47E-02 | 0,00E+00 | 0,00E+00 | 0,00E+00 | 0,00E+00 | 0,00E+00 | 91 | 0 |
| **3** | 1,73E-05 | 4,16E-03 | 2 | 1,46E-03 | 0,00E+00 | 2,06E-04 | 0,00E+00 | 0,00E+00 | 0,00E+00 | 0,00E+00 | 115 | 0 |
| **4** | 1,71E-05 | 4,13E-03 | 3 | 8,73E-03 | -2,31E-03 | 2,35E-04 | 0,00E+00 | 0,00E+00 | 0,00E+00 | 0,00E+00 | 267 | 0 |
| **5** | 1,66E-05 | 4,07E-03 | 3 | 5,89E-03 | 0,00E+00 | -1,12E-04 | 5,73E-05 | 0,00E+00 | 0,00E+00 | 0,00E+00 | 286 | 0 |
| **6** | 1,51E-05 | 3,89E-03 | 4 | -3,42E-02 | 2,00E-02 | -2,00E-03 | 3,54E-04 | 0,00E+00 | 0,00E+00 | 0,00E+00 | 495 | 0 |
| **7** | 1,61E-05 | 4,01E-03 | 3 | 4,76E-03 | 0,00E+00 | 1,19E-04 | 0,00E+00 | 4,17E-07 | 0,00E+00 | 0,00E+00 | 327 | 0 |
| **8** | 1,49E-05 | 3,86E-03 | 4 | -7,15E-03 | 0,00E+00 | 1,69E-03 | -3,60E-04 | 2,43E-06 | 0,00E+00 | 0,00E+00 | 573 | 0 |
| **9** | 1,50E-05 | 3,87E-03 | 5 | -4,30E-03 | -1,97E-03 | 2,03E-03 | -4,24E-04 | 2,64E-06 | 0,00E+00 | 0,00E+00 | 1090 | 0 |
| **10** | 1,54E-04 | 1,24E-02 | 4 | -6,67E-02 | 0,00E+00 | 2,74E-03 | 0,00E+00 | -2,56E-05 | 7,54E-08 | 0,00E+00 | 292 | 0 |
| **11** | 2,38E-05 | 4,88E-03 | 5 | -1,37E-01 | 6,20E-02 | -2,47E-03 | 0,00E+00 | 1,47E-05 | -3,02E-08 | 0,00E+00 | 527 | 0 |
| **12** | 2,52E-05 | 5,02E-03 | 5 | -7,61E-02 | 0,00E+00 | 1,38E-02 | -3,53E-03 | 4,06E-05 | -6,55E-08 | 0,00E+00 | 538 | 0 |
| **13** | 2,14E-05 | 4,63E-03 | 6 | -2,43E-01 | 1,82E-01 | -3,58E-02 | 7,34E-03 | -4,22E-05 | 5,01E-08 | 0,00E+00 | 978 | 0 |
| **14** | 7,46E-05 | 8,64E-03 | 5 | -6,96E-02 | 0,00E+00 | 4,24E-03 | 0,00E+00 | -6,81E-05 | 4,39E-07 | -9,55E-10 | 458 | 0 |
| **15** | 1,87E-05 | 4,32E-03 | 6 | -7,94E-02 | 0,00E+00 | 1,79E-02 | -5,15E-03 | 9,75E-05 | -3,59E-07 | 6,04E-10 | 858 | 0 |
| **16** | 1,88E-05 | 4,34E-03 | 7 | 3,50E-02 | -1,37E-01 | 5,99E-02 | -1,49E-02 | 2,05E-04 | -6,59E-07 | 1,02E-09 | 1844 | 0 |

**Обработка результатов эксперимента**

По таблице значений отношений дисперсий было выбрано третье уравнение регрессии, исходя из значения квантиля распределения Фишера: для данного количества измерений (191) он равен 1,26.

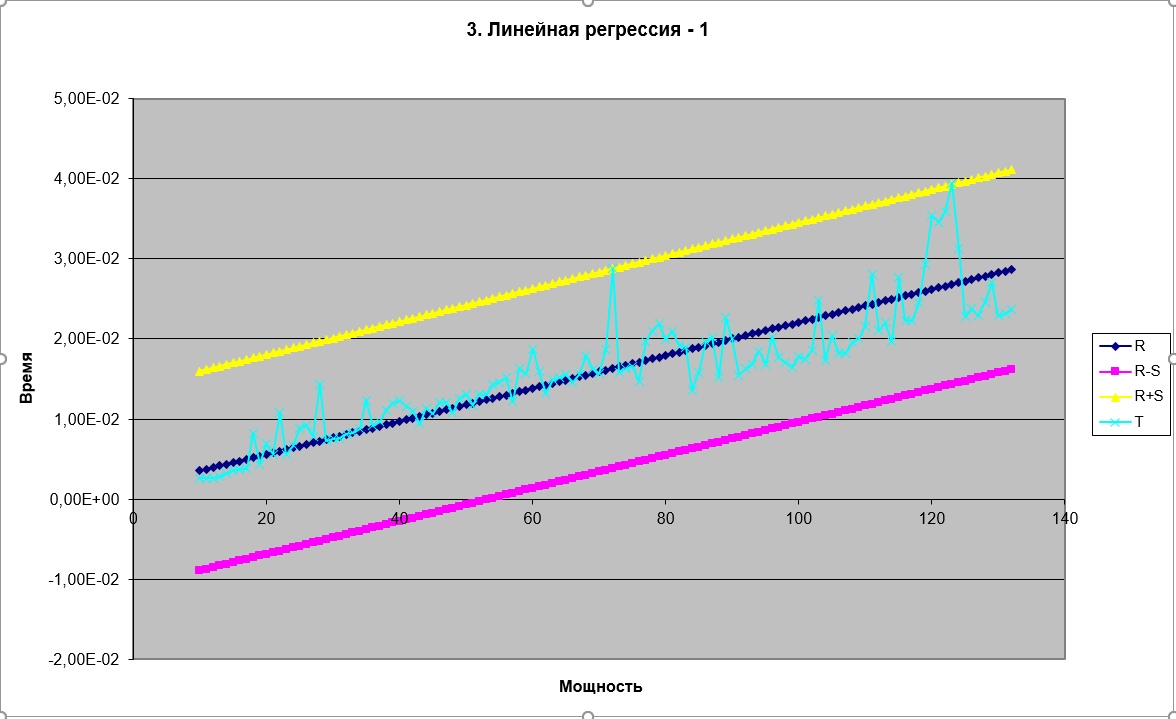
Также по таблице 4 можем определить уравнение на основе разниц между значениями по строкам в каждом столбце.

**Таблица 4**

**Отношение дисперсий**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** |
| **1** | 1,00 | 0,24 | 0,12 | 0,12 |
| **2** | 4,13 | 1,00 | 0,48 | 0,48 |
| **3** | 8,53 | 2,07 | 1,00 | 0,99 |
| **4** | 8,63 | 2,09 | 1,01 | 1,00 |
| **5** | 8,89 | 2,15 | 1,04 | 1,03 |
| **6** | 9,73 | 2,36 | 1,14 | 1,13 |
| **7** | 9,17 | 2,22 | 1,08 | 1,06 |
| **8** | 9,90 | 2,40 | 1,16 | 1,15 |
| **9** | 9,85 | 2,38 | 1,15 | 1,14 |
| **10** | 0,96 | 0,23 | 0,11 | 0,11 |
| **11** | 6,19 | 1,50 | 0,73 | 0,72 |
| **12** | 5,85 | 1,42 | 0,69 | 0,68 |
| **13** | 6,87 | 1,66 | 0,81 | 0,80 |
| **14** | 1,97 | 0,48 | 0,23 | 0,23 |
| **15** | 7,88 | 1,91 | 0,92 | 0,91 |
| **16** | 7,84 | 1,90 | 0,92 | 0,91 |

На основе этого получаем следующие графики линейных регрессий (рис. 1, 2).

Рис. 1. Результаты эксперимента 1

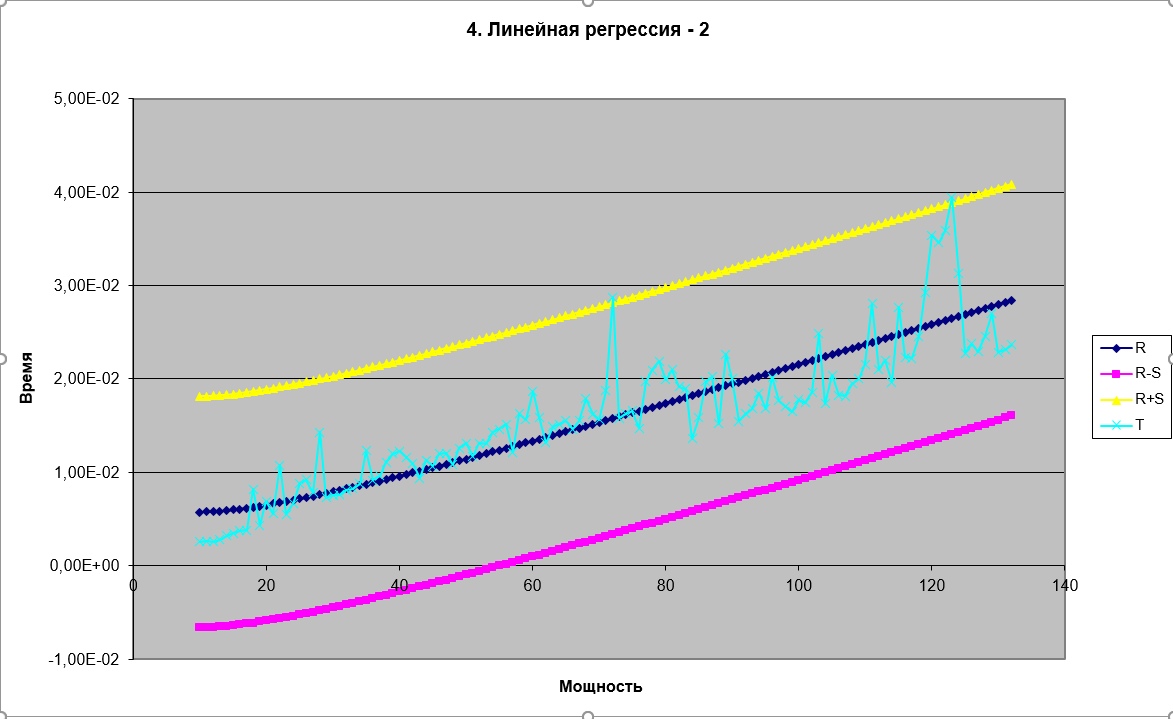


Рис. 2. Результаты эксперимента 2

На графиках следующие обозначения: R – среднее ожидаемое значение, S – модуль доверительной погрешности, T – данные результата эксперимента.

**Выводы**

В результате выполнения курсовой работы была доработана программа лабораторной 6 и проведен эксперимент по оценки временной сложности цепочки операций.

Анализ полученных результатов эксперимента привёл к выводу, что сложность цепочки операций O(N), что совпадает с теоретической оценкой. Таким образом, можно утверждать, что эксперимент прошел удачно.

**Список использованных источников**

Колинько П. Г. Алгоритмы и структуры данных. Часть 2: Методические указания к практическим занятиям на ПЭВМ и курсовому проектированию. Вып. 1902. — СПб.: СПбГЭТУ «ЛЭТИ», 2019. — 56 с.: ил.

**Листинг**

**Source.cpp**

#include <set>

#include <algorithm>

#include <iterator>

#include <conio.h>

#include <vector>

#include <ctime>

#include <iostream>

#include <iomanip>

#include <fstream>

#include <chrono>

using namespace std;

const size\_t SIZE = 300; // Мощность размещаемого в структурах множества

size\_t N = 10; // Мощность генерируемых множеств

// Класс для работы с последовательностями

class MySeq {

vector<set<int>::iterator> seq;

set<int> tree;

void seqRestart() {

seq.clear();

for (set<int>::iterator it = tree.begin(); it != tree.end(); ++it)

seq.push\_back(it);

}

// Перезапись дерева по элементам вектора

void treeRestart() {

set<int> tempTree;

vector<set<int>::iterator> tempSeq;

for (size\_t i = 0; i < seq.size(); ++i) {

tempTree.insert(\*seq.at(i));

tempSeq.push\_back(tempTree.find(\*seq.at(i)));

}

swap(tempTree, tree);

swap(tempSeq, seq);

}

public:

// Конструктор с генерацией последовательности

MySeq() {

std::pair<set<int>::iterator, bool> tempPair;

for (size\_t i = 0; i < N;) {

tempPair = tree.insert(rand() % SIZE + 1);

if (tempPair.second)

++i;

seq.push\_back(tempPair.first);

}

}

// Конструктор копирования

MySeq(const MySeq & Seq) {

for (size\_t i = 0; i < Seq.seq.size(); ++i)

tree.insert(\*Seq.seq.at(i));

seqRestart();

}

// Перегенерация множества

void regenerate() {

seq.clear();

tree.clear();

std::pair<set<int>::iterator, bool> tempPair;

for (size\_t i = 0; i < N;) {

tempPair = tree.insert(rand() % SIZE + 1);

if (tempPair.second)

++i;

seq.push\_back(tempPair.first);

}

}

// Вывод последовательности

void print() {

cout << "Tree [" << tree.size() << "] :\t";

for (set<int>::iterator it = tree.begin(); it != tree.end(); ++it)

cout << setw(5) << \*it;

cout << "\nSeq [" << seq.size() << "] :\t";

for (size\_t i = 0; i < seq.size(); ++i)

cout << setw(5) << \*seq[i];

cout << endl;

}

// Исключение из последовательности подпоследовательности с индекса left до right

void erase(size\_t left, size\_t right) {

if (right >= seq.size())

right = seq.size() - 1;

if (left <= right) {

set<int> tempSet;

vector<set<int>::iterator> tempSeq;

for (size\_t i = 0; i < seq.size(); ++i) {

if (!(i >= left && i <= right)) {

tempSet.insert(\*seq[i]);

tempSeq.push\_back(tempSet.find(\*seq[i]));

}

}

swap(tempSet, tree);

swap(tempSeq, seq);

treeRestart();

}

}

// Добавление к последовательности в конец этой же последовательности count раз

void mul(size\_t count) {

size\_t tempSize = seq.size();

for (size\_t i = 0; i < count; ++i) {

for (size\_t j = 0; j < tempSize; ++j) {

seq.push\_back(seq[j]);

}

}

}

// Исключение из последовательности всех вхождений подпоследовательности exclSeq

void excl(MySeq exclSeq) {

if (seq.size() >= exclSeq.seq.size() && exclSeq.seq.size() != 0) {

int j = 0;

for (int i = 0; i < seq.size(); ++i) {

if (\*seq.at(i) == \*exclSeq.seq.at(j)) {

++j;

}

else {

i -= j;

j = 0;

}

if (j == exclSeq.seq.size()) {

for (int g = i; g > i - j; --g)

seq.erase(seq.begin() + g);

i -= j;

j = 0;

}

}

}

treeRestart();

}

// Пересечение множеств

MySeq operator & (const MySeq & Seq) {

MySeq result = MySeq();

result.seq.clear();

result.tree.clear();

set\_intersection(tree.begin(), tree.end(), Seq.tree.begin(), Seq.tree.end(), inserter(result.tree, result.tree.begin()));

result.seqRestart();

return MySeq(result);

}

// Объединение множеств

MySeq operator | (const MySeq & Seq) {

MySeq result = MySeq();

result.seq.clear();

result.tree.clear();

set\_union(tree.begin(), tree.end(), Seq.tree.begin(), Seq.tree.end(), inserter(result.tree, result.tree.begin()));

result.seqRestart();

return MySeq(result);

}

// Разность множеств

MySeq operator / (const MySeq & Seq) {

MySeq result = MySeq();

result.seq.clear();

result.tree.clear();

set\_difference(tree.begin(), tree.end(), Seq.tree.begin(), Seq.tree.end(), inserter(result.tree, result.tree.begin()));

result.seqRestart();

return MySeq(result);

}

// Симметрическая разность множеств

MySeq operator ^ (const MySeq & Seq) {

MySeq result = MySeq();

result.seq.clear();

result.tree.clear();

set\_symmetric\_difference(tree.begin(), tree.end(), Seq.tree.begin(), Seq.tree.end(), inserter(result.tree, result.tree.begin()));

result.seqRestart();

return MySeq(result);

}

// Оператор присваивания

MySeq & operator = (const MySeq & Seq) {

tree.clear();

seq.clear();

for (size\_t i = 0; i < Seq.seq.size(); ++i) {

tree.insert(\*Seq.seq.at(i));

seq.push\_back(tree.find(\*Seq.seq.at(i)));

}

return \*this;

}

};

int main()

{

srand(time(0));

ofstream output("output.txt");

//............................................................РЕШЕНИЕ ЦЕПОЧКИ ОПЕРАЦИЙ НАД МНОЖЕСТВАМИ

try {

MySeq A, B, C, D, E, Result;

MySeq S1, S2;

size\_t left, right; // Границы для операции erase

size\_t count; // Количество вставок mul

size\_t G = 201; // Количество генераций

left = 0;

right = 5;

count = 1;

output << G - 10 << endl;

for (N = 10; N < G; ++N) {

left = 0;

right = N / 2;

cout << N << " ";

try {

A.regenerate();

B.regenerate();

C.regenerate();

D.regenerate();

E.regenerate();

S1.regenerate();

S2.regenerate();

std::chrono::high\_resolution\_clock::time\_point t1 =

std::chrono::high\_resolution\_clock::now();

// Цепочка операций над множествами

Result = (A | B) ^ ((C & D) / E);

// Цепочка операций над последовательностями

S2.erase(left, right);

S1.mul(count);

S1.excl(S2);

chrono::high\_resolution\_clock::time\_point t2 = chrono::high\_resolution\_clock::now();

auto dt = chrono::duration\_cast<chrono::duration<double>>(t2 - t1);

output << N << ' ' << (dt.count()) << endl;

}

catch (const std::bad\_alloc& er) {

cout << "Error! Step " << N << " crashed. Skipped.\n" << er.what() << "\n";

}

}

}

catch (const std::bad\_alloc& er) {

cout << "Error! First step crashed.\n" << er.what() << "\n";

output.close();

}

output.close();

\_getch();

}