**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра ВТ**

**ОТЧЕТ**

**по лабораторной работе №6 (24 вариант)**

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

**Тема: использование стандартной библиотеки шаблонов**

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

**Цель работы**

Получить практические навыки по работе с стандартной библиотекой шаблонов в С++.

**Постановка задачи**

Написать программу с использованием контейнеров из стандартной библиотеки для работы с множествами и реализовать цепочку операций:

Доработать выбранный контейнер для работы с последовательностями и реализовать операции над последовательностями: mul, erase, excl.

Результат каждого шага цепочки операций выводить на экран.

**Реализованная программа**

Для работы с множествами и последовательностями 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.

На экран выводятся 5 множеств (рисунок 1) из дерева и в виде последовательности. Далее последовательно результаты промежуточных операций над множествами и ответ (рисунок 2).

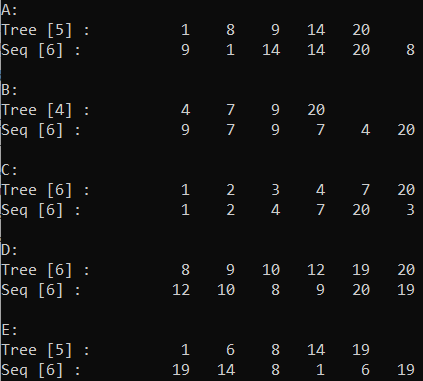
****

Рисунок 1 – Исходные множества

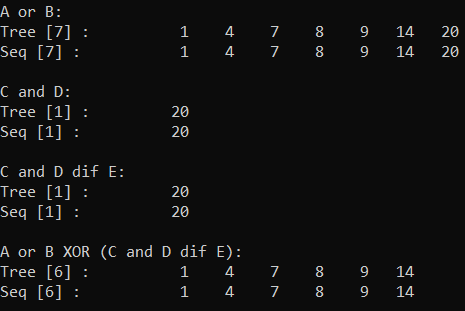


Рисунок 2 – Цепочка операций над множествами

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.

Для демонстрации работы с последовательностями на экран выводятся исходные сгенерированные множества из дерева и в виде последовательности. Затем результаты операций erase, mul и excl над последовательностями. Перед выводом результата операции также выводится последовательность до применения операции для сравнения результата (рисунок 3).

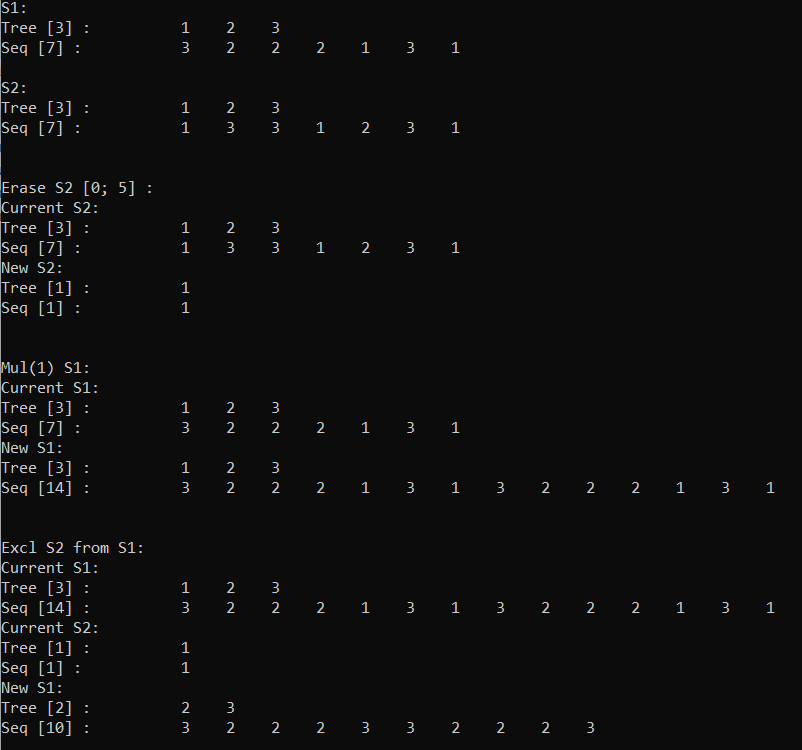


Рисунок 3 – Цепочка операций над последовательностями

Как видно, при изменении последовательности, также меняется множества и элементы в нём.

Таблица 1

**Оценки сложности алгоритмов над последовательностями**

|  |  |
| --- | --- |
| Алгоритм | Сложность в худшем случае |
| mul | O(n2) |
| excl | O(n2) |
| erase | O(n) |

**Вывод**

В результате выполнения лабораторной работы были изучены некоторые контейнеры и встроенные операции над ними из стандартной библиотеки С++. Также выбранные контейнеры были доработаны для работы с последовательностями.

Благодаря удачному выбору контейнеров удалось сэкономить память при хранении последовательности с повторяющимися элементами.

**Использованные источники**

1. 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.

**Приложения**

**Source.cpp**

#include <set>

#include <algorithm>

#include <iterator>

#include <conio.h>

#include <vector>

#include <ctime>

#include <iostream>

#include <iomanip>

using namespace std;

const size\_t SIZE = 3; // Мощность размещаемого в структурах множества

const size\_t COUNT = 7; // Размер последовательностей

// Класс для работы с последовательностями

class MySeq {

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:

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

set<int> tree;

// Конструктор с генерацией последовательности

MySeq() {

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

int temp = rand() % SIZE + 1;

tree.insert(temp);

seq.push\_back(tree.find(temp));

}

}

// 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 = new 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 \*result;

}

// Объединение множеств

MySeq & operator | (const MySeq & Seq) {

MySeq \* result = new 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 \*result;

}

// Разность множеств

MySeq & operator / (const MySeq & Seq) {

MySeq \* result = new 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 \*result;

}

// Симметрическая разность множеств

MySeq & operator ^ (const MySeq & Seq) {

MySeq \* result = new 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 \*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));

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

MySeq A, B, C, D, E, Temp1, Temp2, Temp3, Result;

// Вывод множеств

cout << "A: \n";

A.print();

cout << "\nB: \n";

B.print();

cout << "\nC: \n";

C.print();

cout << "\nD: \n";

D.print();

cout << "\nE: \n";

E.print();

// Вывод промежуточных значений и результата цепочки операций

cout << "\nA or B: \n";

Temp1 = A | B;

Temp1.print();

cout << "\nC and D: \n";

Temp2 = C & D;

Temp2.print();

cout << "\nC and D dif E: \n";

Temp3 = Temp2 / E;

Temp3.print();

cout << "\nA or B XOR (C and D dif E): \n";

Result = Temp1 ^ Temp3;

Result.print();

//............................................................ДЕМОНСТРАЦИЯ РАБОТЫ С ПОСЛЕДОВАТЕЛЬНОСТЯМИ

MySeq S1, S2;

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

size\_t count; // Количество вставок mul

left = 0;

right = 5;

count = 1;

// Вывод последовательностей

cout << "\n---------------------------------------------------------\n\n\n";

cout << "S1: \n";

S1.print();

cout << "\nS2: \n";

S2.print();

// Операции над последовательностями

// ERASE

cout << "\n\nErase S2 [" << left << "; " << right << "] : \n";

cout << "Current S2: \n";

S2.print();

S2.erase(left, right);

cout << "New S2: \n";

S2.print();

// MUL

cout << "\n\nMul(" << count << ") S1: \n";

cout << "Current S1: \n";

S1.print();

S1.mul(count);

cout << "New S1: \n";

S1.print();

// EXCL

cout << "\n\nExcl S2 from S1: \n";

cout << "Current S1: \n";

S1.print();

cout << "Current S2: \n";

S2.print();

S1.excl(S2);

cout << "New S1: \n";

S1.print();

\_getch();

}