Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
Санкт-Петербургский государственный
электротехнический университет «ЛЭТИ» имени В.И. Ульянова (Ленина)

Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

Зачётная работа № 6
по дисциплине «Алгоритмы и структуры данных»
«Использование стандартной библиотеки шаблонов»

Вариант№21

Выполнил: студент группы 5005
Зинатуллин Азат Салаватович

Проверил: старший преподаватель
Колинько Павел Георгиевич

Цель работы

Научиться использовать контейнеры и функции из стандартной библиотеки для работы с множествами и последовательностями.

Набор подходящих контейнеров и функции STL, использованные для работы с ними.

Для моего варианта способом хранения множества является хеш-таблица, и для этого используется ассоциативный контейнер unordered_multiset из стандартной библиотеки. Последовательность хранится в контейнере vector.

Функции и алгоритмы, использованные для работы с контейнерами:
Алгоритм copy() копирует последовательность элементов, ограниченную парой итераторов [first, last), в другой контейнер, начиная с позиции на которую указывает first2. Алгоритм возвращает итератор, указывающий на элемент второго контейнера, следующий за последним вставленным.
Алгоритм find() сравнивает элементы из заданных множеств, как только соответствие найдено, find возвращает итератор типа InputIterator, указывающий на найденный элемент, в противном случае возвращается last.
Алгоритм min() возвращает меньшее из двух значений.
Алгоритм merge() объединяет две отсортированные последовательности, ограниченные диапазонами [first1,last1) и [first2,last2), в единую отсортированную последовательность. Результирующий итератор записи указывает на элемент за концом новой последовательности.
Алгоритм sort() переупорядочивает элементы в диапазоне [first,last) по возрастанию, используя оператор “меньше”, определенный для типа элементов контейнера.
Алгоритм swap() обменивает значения объектов ob1 и ob2.

Функция push_back() вставляет элемент в конец вектора.
Функция insert() в общем случае принимает два параметра: указатель на один из элементов списка и новое значение, которое вставляется после указанного элемента.
Адаптер итератора back_inserter() позволяет вставлять элементы в конец вектора, переданного ему в качестве аргумента.
Begin() и end() устанавливают итератор соответственно на первый элемент вектора и на элемент, который следует за последним. Вместе эти две функции задают диапазон элементов вектора.

Оценка временной сложности

Для двуместной операции над множествами хранящимися в хеш-таблице справедлива оценка временной сложности O(n)^2.
Для двуместной операции над последовательностями так же предполагается временная сложность O(n)^2.

Результаты работы программы

1. На данном изображение выполняются логические операции (в порядке приоритета) между случайно сгенерированными множествами. Первая операция - это конъюнкция двух множеств, второе - дизъюнкция множества получившегося в результате первой операции и третьего множества, далее еще одна дизъюнкция результата предидущих операций и четвертого множества, и последнее - вычитание пятого множества из результата предидущих операций.

```
C: 143 717 343 884 181 251 812 622 479 817
 < 622 181 251 884 817 717 143 812 479 343 >
R: 777 285 721 479 507 67 669 519 343 840
 < 840 669 507 519 721 285 67 777 479 343 >
=== R&=C ===(2)
R: 479 343
 < 479 343 >
B: 710 780 453 729 531 399 529 529
 < 531 399 529 729 453 780 710 529 >
=== R|=B ===
R: 529 529 729 531 399 479 710 780 343 453
 < 479 343 710 780 453 729 531 399 529 529 >
E: 892 695 24 667 984 729 950 730
 < 984 950 667 729 730 695 892 24 >
=== R|=E ===
R: 730 950 24 695 529 529 667 892 984 453 729 729 531 399 479 710 780 343
 < 479 343 710 780 453 729 531 399 529 529 892 695 24 667 984 729 950 730 >
A: 120 526 742 930 235 928 873 990
 < 742 930 235 990 526 928 120 873 >
=== R-=A ===
R: 780 343 710 479 399 531 453 729 729 892 984 529 529 667 695 24 730 950
 < 730 950 24 695 529 529 667 892 984 453 729 729 531 399 479 710 780 343 >
```

2. На этом изображении демонстрируется операция исключения случайно сгенерированной последовательности "F" из последовательности "R" (результат предидущих операций), совпадающие числа исключаются. Затем демонстрируется операция включения случайно сгенерированного множества "G" и его последовательности в множество "R" с указанной позиции (в данном тесте с 5-ой). Здесь берется первая последовательность до 5 позиции, затем включается вторая, а за ней остаток первой. Третья операция заменяет последовательность первого множества с 15 позиции (отчет начинает с "0").

```
R: 780 343 710 479 399 531 453 729 729 892 984 529 529 667 695 24 730 950
 < 730 950 24 695 529 529 667 892 984 453 729 729 531 399 479 710 780 343 >
=== R.Excl(F) ===
F: 780 479 710 399 729 729 531 453 343
 < 453 729 729 531 399 479 710 780 343 >
R: 984 667 695 24 529 529 892 730 950
 < 730 950 24 695 529 529 667 892 984 >
=== R.Subst (G, 5) ===
G: 733 687 935 703 615 307 279 488
 < 703 488 279 935 615 307 687 733 >
R: 892 984 687 733 935 703 730 615 307 488 695 24 529 529 667 950 279
 < 730 950 24 695 529 703 488 279 935 615 307 687 733 529 667 892 984 >
=== R.Change (H, 15) ===
H: 528 211 505 811 985 353 678 808
 < 985 353 678 808 811 505 211 528 >
R: 211 505 528 811 678 985 687 733 935 703 730 615 307 353 488 695 24 529 529 667 950 279 808
 < 730 950 24 695 529 703 488 279 935 615 307 687 733 529 667 985 353 678 808 811 505 211 528 >
=== Time === (8 : 15 * 11 DT=0.0214329)
```

Список использованных источников

1. Алгоритмы и структуры данных: методические указания к лабораторным работам, практическим занятиям и курсовому проектированию. Ч.2 Вып. 1707 (для заочников) / сост.: П. Г. Колинько. — СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2017. — 52 с.

2. Язык программирования С++. Вводный курс. Стенли Б, Липпман, Жози Лажойе. 3-е издание. 1185 стр.

Приложение

main.cpp - файл с программой.

```
#include "stdafx.h"
#include <iostream>
#include <algorithm>
#include <unordered_set>
#include <ctime>
#include <iterator>
#include <chrono>


typedef std::unordered_multiset<int> MySet;
typedef std::unordered_multiset<int>::iterator MyIt;
typedef std::vector<MyIt> MySeq;

const int lim = 1000;
class MyCont {
    int power;
    char tag;
    MySet A;
    MySeq sA;
    MyCont& operator = (const MyCont &);
    MyCont& operator = (MyCont &&);
public:
    MyCont ( int, char );
    MyCont (const MyCont &);
    MyCont (MyCont &&);
    MyCont& operator |= (const MyCont &);
    MyCont& operator &= (const MyCont &);
    MyCont& operator -= (const MyCont &);
    void Merge (const MyCont &);
    void Concat (const MyCont &);
    void Mul (int);
    void Erase (size_t, size_t);
    void Excl (const MyCont &);
    void Subst (const MyCont &, size_t);
    void Change (const MyCont &, size_t);
    void Show ( ) const;
    size_t Power( ) const { return sA.size( ); }
    void PrepareExcl(const MyCont& );
    friend  void PrepareAnd(MyCont &, MyCont&, const int);
};

MyCont::MyCont( int p = 0, char t = 'R')
: power(p), tag(t) {
    for(int i = 0; i < power; ++i) { sA.push_back(A.insert(std::rand( )%lim)); }
}

MyCont::MyCont (MyCont && source)
: power(source.power), tag(source.tag),
A(source.A), sA(source.sA) { }

MyCont::MyCont (const MyCont & source)
: power(source.power), tag(source.tag) {
    for (auto x : source.A) sA.push_back(A.insert(x));
}

void MyCont::Show( ) const {
    using std::cout;
    cout << "\n" << tag << ": ";
    /*  unsigned n = A.bucket_count( );
     for (unsigned i = 0; i < n; ++i)
     if (A.bucket_size(i))
     {
     cout << "\n" <<  i << "("<< A.bucket_size(i) << "):" ;
     //     auto it0 = A.begin(i), it1 = A.end(i);
     for (auto it = A.begin(i); it != A.end(i); ++it) cout << " " << *it;
     } */
    for(auto x : A) cout << x << " ";
    cout << "\n < ";
    for(auto x : sA) cout << *x << " ";
    cout << ">";
}

void PrepareAnd(MyCont & first, MyCont& second, const int quantity) {
```

```
    for (int i = 0; i < quantity; ++i) {
        int x = rand( )%lim;
        first.sA.push_back(first.A.insert(x));
        second.sA.push_back(second.A.insert(x));
    }
}

MyCont& MyCont::operator -= (const MyCont & rgt){
    MySet temp;
    MySeq stemp;
    //  for (auto x : A) if(find(rgt.A.begin( ), rgt.A.end( ), x) == A.end( )) stemp.insert(x);
    /// for (auto x : A) if(rgt.A.find(x) == rgt.A.end( )) temp.insert(x);
    for (auto x : A)
        if(rgt.A.find(x) == rgt.A.end( ))
            stemp.push_back(temp.insert(x));
    /// for (auto x : temp) stemp.push_back(&*temp.find(x));
    temp.swap(A);
    stemp.swap(sA);
    return *this;
}

MyCont& MyCont::operator &= (const MyCont & rgt){
    MySet temp;
    MySeq stemp;
    //  for (auto x : A) if(find(rgt.A.begin( ), rgt.A.end( ), x) != A.end( )) stemp.insert(x);
    /// for (auto x : A) if(rgt.A.find(x) != rgt.A.end( )) temp.insert(x);
    /// for (auto x : temp) stemp.push_back(&*temp.find(x));
    for (auto x : A) if(rgt.A.find(x) != rgt.A.end( )) stemp.push_back(temp.insert(x));
    temp.swap(A);
    stemp.swap(sA);
    return *this;
}

MyCont& MyCont::operator |= (const MyCont & rgt) {
    for (auto x : rgt.A) sA.push_back(A.insert(x));
    return *this;
}


void MyCont::PrepareExcl( const MyCont& rgt ){
    int a = rand( )%rgt.Power( ), b = rand( )%rgt.Power( );
    if (b>a) {
        for (int x = a; x <= b; ++x)  {
            int y =*(rgt.sA[x]); sA.push_back(A.insert(y));
        }
    }
}

void MyCont::Excl (const MyCont & rgt) {
    size_t n(Power( )), m(rgt.Power( ));
    if(m) for (size_t p = 0; p < n; ++p) {
        bool f(true);

        if(*sA[p] == *rgt.sA[0]) {
            size_t q(p), r(0);
            if (m > 1) do {
                ++q, ++r;
                size_t c(*sA[q]), d(*rgt.sA[r]);
                f &= c == d;
            } while ((r<m-1) && f);
            if(f) {
                MySet temp;
                MySeq stemp;
                for(size_t i = 0; i < p; ++i) stemp.push_back(temp.insert(*sA[i]));
                for(size_t i = p+m; i < Power( ); ++i) stemp.push_back(temp.insert(*sA[i]));
                A.swap(temp);
                sA.swap(stemp);
                break;
            }
        }
    }
}


void MyCont::Concat(const MyCont & rgt) {
    //  std::copy(rgt.sA.begin( ), rgt.sA.end( ), back_inserter(sA));
    for(auto x : rgt.sA) sA.push_back(A.insert(*x));
}

void MyCont::Mul(int k) {
    auto p = sA.begin( ), q = sA.end( );
    if(p != q && (k = k%5) > 1) {
        std::vector<int> temp(A.begin( ), A.end( ));
        MySeq res(sA);
        for(int i = 0; i < k-1; ++i) {
            std::copy(p, q, back_inserter(res));
            A.insert(temp.begin( ), temp.end( ));
        }
        sA.swap(res);
    }
}
void MyCont::Erase (size_t p, size_t q) {
    using std::min;
    size_t r(Power( ));
```

```
    p = min(p, r); q = min(q+1, r);
    if(p <= q) {
        MySet temp;
        MySeq stemp;
        for(size_t i = 0; i < p; ++i)
            stemp.push_back(temp.insert(*sA[i]));
        for(size_t i = q; i < r; ++i)
            stemp.push_back(temp.insert(*sA[i]));
        A.swap(temp);
        sA.swap(stemp);
    }
}
void MyCont::Merge(const MyCont & rgt) {
    using std::sort;
    MySeq temp(rgt.sA), res;
    auto le = [ ] (MyIt a, MyIt b)->bool { return *a < *b; };
    sort(sA.begin( ), sA.end( ), le);
    sort(temp.begin( ), temp.end( ), le);
    std::merge(sA.begin( ), sA.end( ), temp.begin( ), temp.end( ), std::back_inserter(res), le);
    A.insert(rgt.A.begin( ), rgt.A.end( ));
    sA.swap(res);
}



void MyCont::Subst (const MyCont & rgt, size_t p) {
    if(p >= Power( )) Concat(rgt);
    else {
        MySeq stemp(sA.begin( ), sA.begin( ) + p);
        std::copy(rgt.sA.begin( ), rgt.sA.end( ), back_inserter(stemp));
        std::copy(sA.begin( )+p, sA.end( ), back_inserter(stemp));
        MySet temp;
        sA.clear( );
        for (auto x : stemp) sA.push_back(temp.insert(*x));
        A.swap(temp);
    }
}

void MyCont::Change (const MyCont & rgt, size_t p) {
    if(p >= Power( )) Concat(rgt);
    else {
        MySeq stemp(sA.begin( ), sA.begin( ) + p);
        std::copy(rgt.sA.begin( ), rgt.sA.end( ), back_inserter(stemp));
        size_t q = p + rgt.Power( );
        if (q < Power( ))
            std::copy(sA.begin( )+q, sA.end( ), back_inserter(stemp));
        MySet temp;
        sA.clear( );
        for (auto x : stemp) sA.push_back(temp.insert(*x));
        A.swap(temp);
    }
}

int main()
{  using std::cout;
    using namespace std::chrono;
    setlocale(LC_ALL, "Russian");
    srand((unsigned int)4);
    srand((unsigned int)time(nullptr));
    bool debug = true;
    auto MaxMul = 5;
    int middle_power = 0, set_count = 0;
    auto Used = [&] (MyCont & t){ middle_power += t.Power( ); ++set_count; };
    auto DebOut = [debug] (MyCont & t) { if(debug) { t.Show( ); system("Pause"); }};
    auto rand = [ ] (int d) { return std::rand( )%d; };
    int p = rand(10) + 1;
    //=== ===
    MyCont A(p, 'A');
    MyCont B(p, 'B');
    MyCont C(p, 'C');
    MyCont D(p, 'D');
    MyCont E(p, 'E');
    MyCont F(0, 'F');
    MyCont G(p, 'G');
    MyCont H(p, 'H');
    MyCont R(p);
    int q_and(rand(MaxMul) + 1);
    PrepareAnd(C, R, q_and);
    if (debug) C.Show( ); Used(C);
    if (debug) R.Show( ); Used(R);
    //=== Operation ===
    std::chrono::high_resolution_clock::time_point t1 = std::chrono::high_resolution_clock::now( );
    if (debug) cout << "\n=== R&=C ===(" << q_and << ") ";
    R&=C;   DebOut(R); Used(R);

    if (debug) B.Show( ); Used(B);
    if (debug) cout << "\n=== R|=B ===";
    R|=B;  DebOut(R); Used(R);

    if (debug) E.Show( ); Used(E);
    if (debug) cout << "\n=== R|=E ===";
    R|=E;  DebOut(R); Used(R);

    if (debug) A.Show(), Used(A);
```

```
    if (debug) cout << "\n=== R-=A ===";
    R-=A;  DebOut(R); Used(R);

    /*
     int a = rand(R.Power( )), b = rand(R.Power( ));
     if (debug) cout << "\n=== R.Erase (" << a << "," << b << ")===";
     if (a>b) cout << "(skipped!)";
     R.Erase(a, b);   DebOut(R);    Used(R);

     if (debug) cout << "\n=== R.Concat(D) ===";
     D.Show( ); Used(D);
     R.Concat(D);   DebOut(R);  Used(R);

     if (debug) cout << "\n=== R.Merge(E) ===";
     E.Show( ); Used(E);
     R.Merge(E);   DebOut(R);   Used(R);
     */

    if (debug) cout << "\n=== R.Excl(F) ===";
    F.PrepareExcl(R);
    if(debug && !F.Power()) cout << "(skipped)!";
    F.Show();  Used(F);
    R.Excl(F);   DebOut(R); Used(R);

    int d = rand(R.Power( ));
    if (debug) cout << "\n=== R.Subst (G, " << d << ") ===";
    G.Show(); Used(G);
    R.Subst(G, d);   DebOut(R); Used(R);

    int e = rand(R.Power( ));
    if (debug) cout << "\n=== R.Change (H, " << e << ") ===";
    H.Show( ); Used(H);
    R.Change(H, e);   DebOut(R);    Used(R);

    /*
     int c = rand(MaxMul);
     if (debug) cout << "\n=== R.Mul(" << c << ")===";
     if (c < 2) cout << "(skipped!)";
     R.Mul(c);   DebOut(R); Used(R);
     */

    std::chrono::high_resolution_clock::time_point t2 = std::chrono::high_resolution_clock::now();
    auto dt = duration_cast<duration<double>>(t2-t1);
    middle_power /= set_count;
    cout << "\n=== Time === (" << p << " : " << set_count << " * " << middle_power << " DT=" << (dt.count())
<<")\n";
    system("Pause");
    //  return 0;
}
```