МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: «КОМБИНИРОВАННЫЕ СТРУКТУРЫ ДАННЫХ И СТАНДАРТНАЯ БИБЛИОТЕКА ШАБЛОНОВ»

Студенты гр. 3312 _____ Нукусси Ф.М

Преподаватель _____ Колинько П.Г.

Санкт-Петербург

2025

Цель работы

Изучить особенности реализации итераторов чтения и вставки для различных структур данных, включая дерево двоичного поиска и его модификации, на примере реализации множества с использованием красно-черного дерева. Исследовать поведение итераторов при выполнении операций вставки, в том числе двуместных (слияния), оценить временные характеристики операций и определить эффективные подходы к работе с упорядоченными и произвольными последовательностями ключей.

Задание

Реализовать индивидуальное задание темы «Множества + последовательности» в виде программы, используя свой контейнер для заданной структуры данных (вариант 6: К-ч-д 1), и доработать его для поддержки операций с последовательностями. Для операций с контейнером рекомендуется использовать возможности библиотеки алгоритмов. Программа должна реализовывать цепочку операций над множествами, имеющимися в выражении, взятом по номеру варианта задания (таблица 1) с базовым контейнером и операциями с последовательностью из таблицы (таблица 2). Результат каждого шага цепочки операций выводится на экран. Реализация каждой операции должна обеспечивать расширенные гарантии устойчивости к исключениям.

Вариант: 14

Мощность множества: 32

Операции с множествами и пример их использования: $A \setminus B \oplus (C \setminus D \setminus E)$

Описание структуры данных

Хеш-таблица (HT) — это структура данных, реализующая интерфейс ассоциативного массива с использованием хеш-функции для быстрого доступа к элементам. В данной работе представлена реализация хеш-таблицы с методом цепочек (separate chaining) для разрешения коллизий.

Основные компоненты

[+]Массив бакетов (bucket array)

Фиксированный массив указателей на узлы (`Node** bucket`), где каждый бакет содержит цепочку элементов.
Размер массива: `Buckets = 32` (по умолчанию)

[+]Узел (Node)

```
struct Node {
int key;
Node *down;
Node() : down(nullptr) {}
Node(int k, Node* d = nullptr) : key(k), down(d)
{}
~Node( ) { delete down; }
};

[+]Хеш-функция

int hash(int k) const {
        return (k * (Buckets - 1) + 7) %
Buckets;
}
Равномерно распределяет ключи по бакетам.
```

[+]Последовательность (Sequence)

```
vector<Node*> seq;
```

Операции

- **Вставка (`insert`)**

Добавляет элемент в начало цепочки соответствующего бакета.
Сложность: O(1) в среднем случае.

- **Поиск (`find`)**

Проверяет наличие ключа в цепочке бакета.
Сложность: O(1) в среднем, O(n) в худшем (все элементы в одном бакете).

- **Удаление (`Remove`)**

Удаляет ключ из цепочки. Сложность: аналогична поиску.

- **Операции над множествами**

Реализованы через комбинацию базовых операций:

- Объединение (`A | B`)
- Пересечение (`A & B`)
- Разность (`A - B`)
- XOR (`A ^ B`)

4. Пример работы

```
    -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   50  17  16  -   -   13  -   43  -   -   -
    -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
  Sequence
23-17-10-39-50-35-16-38-26-18-48-36-37-41-32-43-22-13-20-30-11-29-24-14-25-31-12-28-45-49-27-15-
== TABLE B ==

B:  -   38  37  36  35  -   -   32  31  30  -   28  59  58  57  24  23  54  21  52  51  50  49  48  47  46  45  -   43  42  41  -
    -   -   -   -   -   -   -   -   -   -   -   60  -   -   25  56  55  -   53  20  -   -   -   -   -   -   -   -   -   -   -
    -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
  Sequence
45-55-49-42-60-48-25-51-46-20-32-38-37-54-31-57-23-59-47-30-56-28-50-53-52-58-35-43-41-24-36-21-
== TABLE C ==

C:  39  70  69  36  35  66  65  32  31  62  61  60  -   58  57  -   55  54  -   52  -   -   49  48  47  46  45  -   43  42  41  40
    -   38  37  -   -   34  33  64  63  -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
    -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
  Sequence
33-55-64-40-57-60-34-54-63-66-42-47-41-36-58-37-48-69-38-62-61-43-49-70-35-39-65-46-45-32-52-31-
== TABLE D ==

D:  -   70  69  -   -   66  65  64  -   -   61  60  59  58  57  56  55  54  53  -   51  50  49  48  47  78  45  44  43  42  73  40
    -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   80  79  -   77  76  75  74  -   -
    -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
  Sequence
78-61-79-51-65-70-57-50-74-76-47-69-49-44-77-59-64-42-54-55-73-66-56-53-58-75-43-40-45-80-60-48-
== TABLE E ==

E:  -   70  69  68  67  66  65  64  63  62  61  60  -   58  89  56  55  86  53  84  51  82  81  80  79  -   -   76  -   74  73  72
    -   -   -   -   -   -   -   -   -   -   -   -   -   90  57  -   87  54  -   -   83  -   -   -   -   -   -   -   -   -   -   -
    -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
  Sequence
84-73-68-83-82-66-64-90-67-69-58-57-56-70-81-51-60-53-79-89-87-62-80-61-55-63-54-86-74-65-72-76-
== (A \ B)==

H:  39  -   -   -   -   -   -   -   -   -   29  -   27  26  -   -   -   22  -   -   -   18  17  16  15  14  13  12  11  10  -   -
    -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
  Sequence
39-29-27-26-22-18-17-16-15-14-13-12-11-10-
== C \ D \ E ==

N:  39  38  37  36  35  34  33  32  31  -   -   -   -   -   -   -   -   -   -   52  -   -   -   -   -   46  -   -   -   -   41  -
    -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
  Sequence
39-38-37-36-35-34-33-32-31-52-46-41-

RESULTAT: R = (A \ B) ⊖ (C \ D \ E)

Z:  -   38  37  36  35  34  33  32  31  -   29  -   27  26  -   -   -   22  -   52  -   18  17  16  15  14  13  12  11  10  41  -
    -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   46  -   -   -   -   -   -
    -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
  Sequence
38-37-36-35-34-33-32-31-29-27-26-22-52-18-17-16-15-46-14-13-12-11-10-41-
limitless@limitless-Vivobook-Go-E1504FA-E1504FA:~/Lab works/Labs of Algorithme and Structure of Data/Second semester/lab 3/MyCont00_HT$
```

Оценка временной сложности

1. Вставка (`insert`)

В среднем: O(1)

В худшем случае: O(n) (если все элементы попадают в один бакет)

2. Удаление (`Remove`)

В среднем: O(1)

В худшем случае: O(n) (коллизии)

3. Поиск (`find`)

В среднем: O(1)

В худшем случае: O(n)

4. Отображение (`Display`)

Всегда: O(n+m), где:

n — количество элементов,

m — количество бакетов (размер таблицы).

5. Объединение (`operator|`)

O(n+m), где:

n — размер текущей таблицы,

m — размер таблицы `other`.

6. Пересечение (`operator&`)

O(min(n,m)) в среднем

O(n×m) в худшем случае (если все элементы в одном бакете).

7. Разность (`operator-`)

O(n), где n — размер текущей таблицы.

8. Симметрическая разность/XOR (`operator^`)

O(n+m) (комбинация разности и объединения).

Код программы

+HT0.h

```
#pragma once
#include <bits/stdc++.h>
using namespace std;


class HT;

struct Node {
	int key;
	Node *down;
	Node() : down(nullptr) {}
	Node(int k, Node* d = nullptr) : key(k), down(d) {}
	~Node( ) { delete down; }
};

//template<class Container = HT>
//struct myiter : public std::iterator<std::forward_iterator_tag, int> //Obsolete depuis c++17
```

```
struct myiter
{
        using iterator_category = std::forward_iterator_tag;
        using value_type = int;
        using difference_type = std::ptrdiff_t;
        using pointer = int*;
        using reference = int&;
        myiter(Node *p) : bct(nullptr), pos(0), Ptr(p) {}
        bool operator == (const myiter & Other) const { return Ptr ==
Other.Ptr; }
        bool operator != (const myiter & Other) const { return Ptr !=
Other.Ptr; }
        //Pre-increment
        myiter operator++();
        //Post-increment
        myiter operator++(int) { myiter temp(*this); ++*this; return temp; }
        pointer operator->() { return & Ptr->key; }
        reference operator*() { return Ptr->key; }
        //protected:
        //Container& c;
        Node **bct;
        size_t pos;
        Node * Ptr;
};

template <typename Container, typename Iter = myiter>
//class outiter : public std::iterator<std::output_iterator_tag, typename
Container::value_type>
class outiter
{
```

```
protected:
        Container& container;
        Iter iter;
public:

        //Types requis
        using itrerator_category = std::output_iterator_tag;
        using value_type = void;
        using difference_type = void;
        using pointer = void;
        using reference = void;

        explicit outiter(Container& c, Iter it) : container(c), iter(it) { }

        const outiter<Container>&
          operator = (const typename Container::value_type& value) {
          iter = container.insert(value, iter).first;
          return *this;
        }
        const outiter<Container>&
          operator = (const outiter<Container>&) { return *this; }

        outiter<Container>& operator* () { return *this; }

        outiter<Container>& operator++ () { return *this; }
        outiter<Container>& operator++ (int) { return *this; }
};

// Helper fonction pour cree un iterateur
template <typename Container, typename Iter>
```

```
inline outiter<Container, Iter> outinserter(Container& c, Iter it)
{
        return outiter<Container, Iter>(c, it);
}
//myiter myiter0(nullptr);

class HT {
static  size_t tags;
   char tag;
        Node **bucket;
        size_t count = 0;
        static myiter myiter0;

        //Adding of sequence:
        vector<Node*> seq;
public:
static const size_t Buckets = 32;
        using key_type = int;
        using value_type = int;
        using key_compare = equal_to<int> ;
        void swap(HT & rgt)
        {
          std::swap(tag, rgt.tag);
          std::swap(bucket, rgt.bucket);
          std::swap(count, rgt.count);
          std::swap(seq, rgt.seq);
        }
        int hash(int k)const { return (k*(Buckets - 1) + 7) % Buckets; }
        size_t bucket_count() { return Buckets; }
```

```
        myiter Insert(const int& k,myiter where) { return insert(k,
where).first; }
        void Display();
        myiter begin()const;
        myiter end()const { return myiter(nullptr); }
//      const myiter cbegin() const { return begin(); }
//      const myiter cend()const { return cend(); }
        pair<myiter, bool> insert(int, myiter = myiter(nullptr));
        pair<myiter, bool> Remove(int);
        HT() :  tag('A' + tags++), bucket(new Node*[Buckets]){
          for (int i = 0; i < Buckets; ++i) bucket[i] = nullptr;
        }
        size_t power() const {
          size_t count = 0;
          for (size_t i = 0; i < Buckets; ++i)
          {
                Node* element = bucket[i];
                while (element)
                {
                      count++;
                      element = element->down;
                }
          }
          return count;
        }
//      HT(size_t Buckets);
//      void resize(size_t Buckets);
        int size() { return count; }
        template<typename MyIt>
         HT(MyIt, MyIt);
```

```
~HT();
myiter find(int)const;
HT(const HT&rgt) : HT() {
   for (auto x = rgt.begin(); x != rgt.end(); ++x) insert(*x);
}
HT(HT&& rgt) : HT() {
   swap(rgt);
}
HT& operator=(const HT & rgt)
{
   HT temp;
   for (auto x = rgt.begin(); x != rgt.end(); ++x) insert(*x);
   swap(temp);
   return *this;
}
HT& operator=(HT && rgt)
{
   swap(rgt);
   return *this;
}
HT& operator |= (const HT &);
HT operator | (const HT & rgt) const
{
   HT result(*this); return (result |= rgt);
}
HT& operator &= (const HT &);
HT operator & (const HT & rgt) const
{
   HT result(*this); return (result &= rgt);
}
```

```
        HT& operator -= (const HT &);
        HT operator - (const HT & rgt) const
        {
          HT result(*this); return (result -= rgt);
        }
        HT& operator ^= (const HT& rgt);
        HT operator ^ (const HT & rgt) const
        {
          HT result(*this); return (result ^= rgt);
        }
};
void generator(std::vector<int> &arr, int min, int max, int SIZE);
```

HT0.cpp

```
//#include "pch.h"
#include "HT0.h"
#include <iostream>
#include <iomanip>
using std::cout;
#include <cstdlib>
#include <ctime>
#include <random>
const int SIZE = 32;

// Retourne un itérateur pointant sur le premier élément de la table de hachage
myiter HT::begin()const {
   myiter p(nullptr);
   p.bct = bucket;
   for (; p.pos < Buckets; ++p.pos) {
      p.Ptr = bucket[p.pos];
```

```
        if (p.Ptr) break;  // Trouvé un bucket non vide
    }
    return p;
}

// Opérateur de pré-incrémentation pour l'itérateur
myiter myiter::operator++()
{
    if (!Ptr) { // Itérateur invalide ?
        return *this; // Ne rien faire si déjà invalide
    }
    else
    { // Chercher le prochain élément
        if(Ptr->down) { // Si élément suivant dans le même bucket
            Ptr = Ptr->down;
            return (*this);
        }
        while(++pos < HT::Buckets) {    // Chercher dans les buckets suivants
            if(Ptr = bct[pos])
                return *this;
        }
        Ptr = nullptr;  // Fin de la table
        return (*this);
    }
}

// Destructeur de la table de hachage
HT :: ~HT()
{
   for(auto t = 0; t < Buckets; ++t)
```

```
      delete bucket[t];    // Libérer chaque chaîne
   delete []bucket;        // Libérer le tableau de buckets
}

// Affichage de la table de hachage
void HT::Display()
{
   Node** P = new Node*[Buckets];
   for(auto t = 0; t < Buckets; ++t) P[t] = bucket[t];
   bool prod = true;
   cout << "\n" << tag << ":";
   while(prod) {
      prod = false;
      for(auto t = 0; t < Buckets; ++t) {
         if(P[t]) {
            cout << setw(4) << P[t]->key;
            P[t] = P[t]->down;
            prod = true;
         }
         else cout << "  - ";
      }
      cout << "\n  ";
   }

   //Display the sequence;
   cout << "Sequence" << endl;
   for (int i = 0; i < seq.size(); i++)
   {
      cout << seq[i]->key << "-" ;
   }
```

```
    cout << endl;

}

// Recherche d'un élément dans la table
myiter HT::find(int k)const
{
    auto t(hash(k));  // Calcul du hash
    Node*p(bucket[t]);
    while (p) {
        if (p->key == k)
            return myiter(p);
        else p = p->down;
    }
    return end();  // Non trouvé
}

// Insertion d'un élément dans la table
pair<myiter, bool> HT::insert(int k, myiter)
{
    auto t(hash(k));
    Node * p(bucket[t]);
    while(p) {
        if(p->key == k) {
            seq.push_back(p);
            return make_pair(myiter(p), false);  // Déjà présent
        }
        else p = p->down;
    }
    bucket[t] = new Node(k, bucket[t]);  // Insertion en tête
```

```
    ++count;
    //Insertion in the sequence:
    seq.push_back(bucket[t]);
    return make_pair(myiter(bucket[t]), true);  // Insertion réussie
}

// Suppression d'un élément de la table
pair<myiter, bool> HT::Remove(int k)
{

    auto t(hash(k));
    auto p(bucket[t]), q(p);
    if (p) {
        if (p->key == k) { // Suppression en tête de liste
            bucket[t] = p->down;
            p->down = nullptr;
            delete p;
            return make_pair(myiter(q), true);
        }
        p = p->down;
        while (p) {
            if (p->key == k) {
                q->down = p->down;
                p->down = nullptr;
                delete p;
                --count;
                return make_pair(myiter(q), true);
            }
            else q = p, p = p->down;
        }
```

```
    }
    //For the sequence i decided to delete all the instance
    /*seq.erase(__remove_if(seq.begin(), seq.end(),
          [k](const Node* p){return p->key == k; }), seq.end());*/
    for (int i = 0; i < seq.size(); i++)
    {
       if (seq[i]->key == k)
       {
          seq.erase(seq.begin() + i);
          --i;
       }

    }

    return make_pair(myiter(q), false);  // Non trouvé
}

// Constructeur à partir d'une séquence d'éléments
template<class MyIt>
HT::HT(MyIt first, MyIt last) : HT() {
    for (; first != last; ++first) insert(*first);
}

// Opérateur d'union avec affectation
HT & HT::operator |= (const HT & rgt) {
    for (myiter x = rgt.begin(); x != rgt.end(); ++x)
       insert(*x, nullptr);
    return *this;
}
```

```
// Opérateur d'intersection avec affectation
HT & HT::operator &= (const HT & rgt) {
   HT temp;
   for (auto x = begin(); x != end(); ++x)
      if (rgt.find(*x) != rgt.end())
         temp.insert(*x);
   swap(temp);
   return *this;
}

// Opérateur de différence avec affectation
HT& HT::operator -= (const HT & rgt) {
   HT temp;
   for (auto x = begin(); x != end(); ++x)
      if (rgt.find(*x) == rgt.end()) temp.insert(*x);
   swap(temp);
   return *this;
}

//Operateur de XOR avec affectation (element soit dans A, soit dans B mais pas dans les deux)
HT& HT::operator ^= (const HT& rgt){
            HT temp;
            for (auto x = begin(); x != end(); ++x)
               if (rgt.find(*x) == rgt.end()) temp.insert(*x);

            for (auto x = rgt.begin(); x != rgt.end(); ++x)
               if(find(*x) == end()) temp.insert(*x);

            swap(temp);
```

```
            return *this;
}

size_t HT::tags = 0;  // Compteur pour générer des tags uniques

void generator(std::vector<int> &arr, int min, int max, int SIZE) {
   std::random_device rd;
   std::mt19937 rng(rd());
   std::unordered_set<int> usedValues;

   for (int i = 0; i < SIZE; ++i) {
      int val;
      do {
         val = min + (rng() % (max - min + 1));
      } while (usedValues.find(val) != usedValues.end());

      arr[i] = val;
      usedValues.insert(val);
   }
}
// myiter& myiter::operator++(){
//            if (Ptr && Ptr->down){
//               Ptr = Ptr->down;
//            } else {
//               while (++pos < HT::Buckets)
//               {
//                        if (Ptr = bct[pos]) break;
//               }
//            }
//            return *this;
```

```
// };
```

+MyCont00.cpp

```
// MyCont00.cpp : Хеш-таблица + итераторы
// Вставка, удаление, копирование и двуместные операции
// Демонстрационная программа: (C)lgn, 17-23.03.19


//include "pch.h"
#include <iostream>
#include <clocale>
#include <vector>
#include <list>
#include <stack>
#include <iterator>
#include <algorithm>
#include "HT0.h"
using namespace std;
const int SIZE = 32;

int main()
{


        // Initialisation des tableaux avec des plages différentes
   int AT[SIZE], BT[SIZE], CT[SIZE], DT[SIZE], ET[SIZE];
   cout << "Creation of sets..." << endl;
   generator(AT, 10, 50);
   generator(BT, 20, 60);
```

```
    generator(CT, 30, 70);
    generator(DT, 40, 80);
    generator(ET, 50, 90);

    HT A, B, C, D, E;
    for (int i = 0; i < SIZE; ++i) {
        A.insert(AT[i]);
        B.insert(BT[i]);
        C.insert(CT[i]);
        D.insert(DT[i]);
        E.insert(ET[i]);
    }
    cout << "== TABLE A ==" << endl; A.Display();
    cout << "== TABLE B ==" << endl; B.Display();
    cout << "== TABLE C ==" << endl; C.Display();
    cout << "== TABLE D ==" << endl; D.Display();
    cout << "== TABLE E ==" << endl; E.Display();

    // Calcul de R = (A \ B) XOR (C \ D \ E)
    cout << "== (A \\ B)==" << endl;
    HT one = A - B;
    one.Display();
    cout << "== C \\ D \\ E ==" << endl;
    HT two = C - D - E;
    two.Display();
    HT R = (A - B) ^ (C - D - E);
    cout << "\nRESULTAT: R = (A \ B) ⊖ (C \ D \ E)" << endl;
    R.Display();

}
```

Выводы

Исследование подтвердило, что корректная реализация итераторов критична для производительности операций над множествами, особенно при работе с динамическими данными. Полученные результаты могут быть применены для оптимизации STL-совместимых контейнеров и специализированных хранилищ ключей.й.

Список использованных источников

1. П.Г. Колинько – «Пользовательские контейнеры» учебно-метод. пособие, 2025 г.