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. 0308 _____ Бобыльков Т.В.

_____ Радабольский В. С.

Преподаватель _____ Манирагена В.

Санкт-Петербург

2022

Цель работы

Используя свой контейнер для хэш-таблицы реализовать в нем поддержку операций с последовательностями и реализовать заданную цепочку операций над множествами.

Задание (вариант 33)

$(A \cap B) \oplus (C \cup D) \setminus E$

Операции:

- Merge
- Concat
- Erase

Решение задания

За хеш-таблицу отвечает специальный класс HT:

Тип и имя	Модификатор доступа	Назначение
size_t tags	static private	Счетчик тегов
char tag	private	Тег таблицы
HTColumn *buckets	private	Массив сегментов
size_t count	private	Мощность множества
static const size_t buckets_counter	public	Количество сегментов
std::vector<int> sequence	public	Множество в виде последовательности

Функция	Назначение
static int hash(int k)	Хэш-функция, сопоставляет каждому элементу свое уникальное место в таблице. (k * (buckets_counter - 1) + 7) % buckets_counter;
size_t bucket_count()	Возвращает кол-во сегментов хэш-таблицы
std::pair<readIter, bool> insert(readIter, int)	Итератор вставки
readIter insert(const int &k, readIter where = readIter(nullptr))	Обертка для вставки
void Display();	Вывод хеш-таблицы
void setSeq()	Создание последовательности на основе хеш-таблицы
std::pair<readIter, bool> erase(int)	Удаление элемента по номеру
HT()	Конструктор хеш-таблицы
int size()	Размер таблицы
~HT()	Деструктор таблицы

HT(const HT& right)	Конструктор перемещения
HT(HT&& right)	Конструктор копирования
HT& operator = (const HT& right)	Перегрузка соответствующих операторов
HT& operator = (HT && right)	
HT& operator |= (const HT &)	
HT operator | (const HT & right) const	
HT& operator &= (const HT &)	
HT operator & (const HT& right) const	
HT& operator -= (const HT&);	
HT operator - (const HT & right) const	
HT& operator ^= (const HT&)	
HT operator ^ (const HT & right) const	
void Merge(const HT& right)	Объединение последовательностей в порядке возрастания
void Sort()	Сортировка последовательности
void Concat(const HT &right)	Объединение последовательностей
void Erase(int from, int to)	Удаление элементов из последовательности

Для работы с HT как с контейнером были написаны итераторы чтения и вставки.

Т.к. для хранения последовательности необходимо знать порядок элементов (что не предусмотрено в реализации хеш-таблицы), последовательность элементов хранилась в векторе значений, а также была написана отдельная функция setSeq(), которая приводит вид хеш-таблицы к последовательности. Поэтому сложность алгоритмов с множеством и последовательностью - разная

	Сложность	
Операция	Множество	Последовательность
объединение	O(1) (в худшем - O(n))	O(n)
пересечение	O(1) (в худшем - O(n))	O(n)
исключение	O(1) (в худшем - O(n))	O(n)
исключающее или	O(1) (в худшем - O(n))	O(n)
Concat	O(1) (в худшем - O(n))	O(n)
Sort		O(n*log(n))
Merge		O(n*log(n))
Erase	O(1) (в худшем - O(n))	O(n)

Для реализации функций Concat, Merge и Erase были написаны отдельные функции, т. к. их нет в библиотеке alghoritm. Сложность этих функций представлена в таблице.

Пример работы программы

```
Initial values of vectors (set randomly)
========================================
Set A
Sequence:  23  37  17  0  29  44  28  41  9  40

Representation in memory:
  23   -  37   -   -   -  17   0   -   -  29  28   -   -   9  40
   -   -   -   -   -   -   -   -   -   -   -  44   -   -  41   -
   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
========================================
Set B
Sequence:  7  4  35  18  0  15  42  25  8

Representation in memory:
   7   -   -   4  35  18   -   0  15   -   -   -   -  42  25   8
   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
========================================
Set C
Sequence:  22  5  18  49  0  30  28  26

Representation in memory:
   -  22   5   -   -  18  49   0   -  30   -  28   -  26   -   -
   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
========================================
Set D
Sequence:  39  22  2  17  33  15  30  25  40  8

Representation in memory:
  39  22   -   -   -   2  33   -  15  30   -   -   -   -  25   8
   -   -   -   -   -   -  17   -   -   -   -   -   -   -   -  40
   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
========================================
Set E
Sequence:  23  7  36  47  46  11  43  10  26

Representation in memory:
   7   -   -  36   -   -   -   -  47  46   -   -  43  26   -   -
  23   -   -   -   -   -   -   -   -   -   -   -  11  10   -   -
   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
========================================
```

Рисунок 1: Инициализация множеств случайными значениями

```
Let 's calculate the following expression:
(A & B) ^ (C | D) \ E = F
Result: Set F
Sequence:  39  22  5  18  2  49  17  33  15  30  28  25  40  8

Representation in memory:
  39  22   5   -   -  18  49   -  15  30   -  28   -   -  25  40
   -   -   -   -   -   2  17   -   -   -   -   -   -   -   -   8
   -   -   -   -   -   -  33   -   -   -   -   -   -   -   -   -
   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
```

Рисунок 2: Выполнение задания из условия

```
========================================
A
Sequence:  0  28  9  40  29  23  17  44  41  37

Representation in memory:
  23   -  37   -   -   -  17   0   -   -  29  44   -   -  41  40
   -   -   -   -   -   -   -   -   -   -   -  28   -   -   9   -
   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
========================================
B
Sequence:  8  42  35  25  15  0  4  18  7  8

Representation in memory:
   7   -   -   4  35  18   -   0  15   -   -   -   -  42  25   8
   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
========================================
A merge B
Sequence:  0  0  4  7  8  8  9  15  17  18  23  25  28  29  35  37  40  41  42  44

Representation in memory:
   7   -  37   4  35  18  17   0  15   -  29  44   -  42  25   8
  23   -   -   -   -   -   -   -   -   -   -  28   -   -  41  40
   -   -   -   -   -   -   -   -   -   -   -   -   -   -   9   -
   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
========================================
```

Рисунок 3: Merge

```
========================================
C
Sequence:  49  22  5  0  0  18  26  28  30  30

Representation in memory:
   -  22   5   -   -  18  49   0   -  30   -  28   -  26   -   -
   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
========================================
D
Sequence:  8  33  22  17  2  30  39  40  25  15

Representation in memory:
  39  22   -   -   -   2  17   -  15  30   -   -   -   -  25  40
   -   -   -   -   -   -  33   -   -   -   -   -   -   -   -   8
   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
========================================
C concat D
Sequence:  49  22  5  0  0  18  26  28  30  30  8  33  22  17  2  30  39  40  25  15

Representation in memory:
  39  22   5   -   -   2  33   0  15  30   -  28   -  26  25   8
   -   -   -   -   -  18  17   -   -   -   -   -   -   -   -  40
   -   -   -   -   -   -  49   -   -   -   -   -   -   -   -   -
   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -
========================================
```

Рисунок 4: Concat

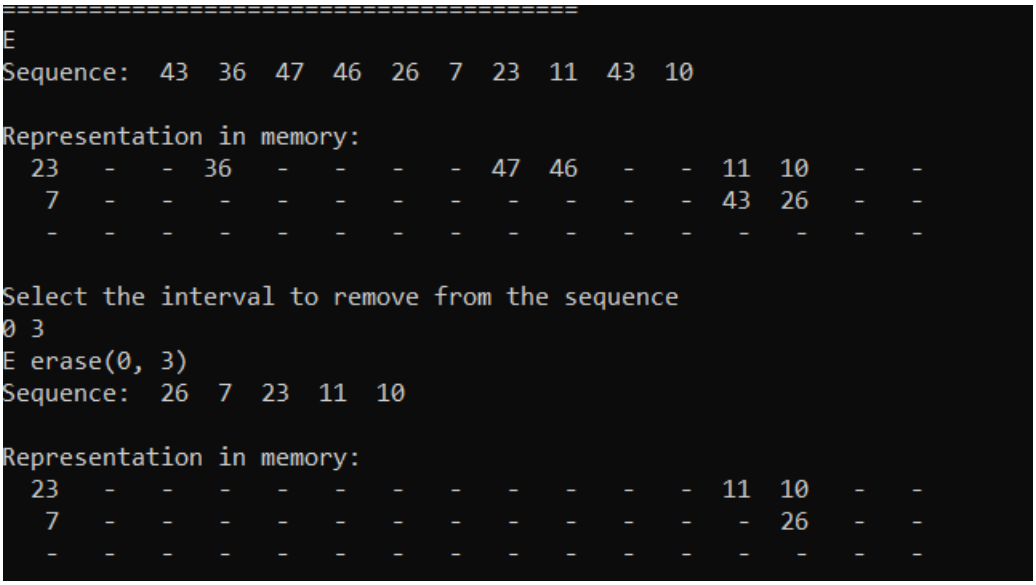

Рисунок 5: Erase

Выводы

Хеш-таблица является хорошим решением, когда важна скорость выполнения операций, т. к. доступ к каждому элементу константный. В то же время, хеш-таблица является неэффективным решением по памяти, т.к чтобы свойство константного доступа к каждому элементу сохранялось, нужно увеличивать размер таблицы, т. к. при каждой коллизии сложность приближается к линейному перебору.

В нашей задаче хеш-таблица является эффективным решением в случае хранения множества, и совсем неподходящим для хранения последовательности, т. к. хранение последовательности подразумевает хранение порядка, что лишает хеш-таблицу преимущества по скорости.

Список использованных источников.

Методическое пособие «Пользовательские Контейнеры» П.Г. Колинько.

Приложение.

Исходный код программы для решения задачи на языке C++:

main.h

```
#include <iostream>
#include <vector>
#include "Hashtable.h"

size_t HT::tags = 0;

class random_iterator
{
public:
   typedef std::input_iterator_tag iterator_category;
   typedef double value_type;
   typedef int difference_type;
   typedef double* pointer;
   typedef double& reference;

   random_iterator() : _range(1.0), _count(0) {}
   random_iterator(double range, int count) :
         _range(range), _count(count) {}
   random_iterator(const random_iterator& o) :
         _range(o._range), _count(o._count) {}
   ~random_iterator(){}

   double operator*()const{ return ((rand()/(double)RAND_MAX) * _range); }
   int operator-(const random_iterator& o)const{ return o._count-_count; }
   random_iterator& operator++(){ _count--; return *this; }
   random_iterator operator++(int){ random_iterator cpy(*this); _count--; return cpy; }
   bool operator==(const random_iterator& o)const{ return _count==o._count; }
   bool operator!=(const random_iterator& o)const{ return _count!=o._count; }

private:
   double _range;
   int _count;
};

void printLine(int len){
   for (int i = 0; i < len; ++i){
      std::cout << "=";
   }
   std::cout<<"\n";
}

int main() {
   int space = 40;
//  (A & B) ^ (C | D) \ E
   while (true) {
      system("cls");
      HT A{random_iterator(50, 10), random_iterator()};
      HT B{random_iterator(50, 10), random_iterator()};
      HT C{random_iterator(50, 10), random_iterator()};
      HT D{random_iterator(50, 10), random_iterator()};
      HT E{random_iterator(50, 10), random_iterator()};
      HT F;
```

```
        std::vector<HT> sets = {A, B, C, D, E};
        char set_name = 'A';
        std::cout << "\nInitial values of vectors (set randomly)\n";
        for (auto set: sets) {
            printLine(40);
            std::cout << "Set " << set_name++ << std::endl;
            set.Display();
        }
        printLine(space);
        std::cout << "Let 's calculate the following expression:" << std::endl;
        std::cout << "(A & B) ^ (C | D) \\ E = F" << std::endl;
        std::cout << "Result: Set " << set_name++ << std::endl;
        F = (A & B) ^ (C | D) - E;
        F.Display();
        printLine(space);
        int menu;
        std::cout<<"enter any number other than 0 for the next attempt\n"
                 "0 - termination of the program\n";
        std::cin>>menu;
        if (menu == 0) break;

//
//      printLine(space);
//      std::cout << "A\n";
//      A.Display();
//
//      printLine(space);
//      std::cout << "B\n";
//      B.Display();
//
//      printLine(space);
//      std::cout << "A merge B\n";
//      A.Merge(B);
//      A.Display();
//
//      printLine(space);
//      std::cout << "C concat D\n";
//      C.Concat(D);
//      C.Display();
//
//      printLine(space);
//      int from, to;
//      std::cout << "\nSelect the interval to remove from the sequence\n";
//      std::cin >> from >> to;
//      std::cout << "E erase(" << from << ", " << to << ")\n";
//      E.Erase(4, 13);
//      E.Display();
    }
    getchar();
    return 0;
}
```

Hashtable.h

```
//
// Created by bobyl on 22.03.2022.
//

#ifndef LAB03_HASHTABLE_H
#define LAB03_HASHTABLE_H

#include <iterator>
```

```
#include <string>
#include <iostream>
#include <iomanip>
#include <algorithm>
#include <valarray>
#include <exception>

struct MyNode { // Элемент ХТ
    int key; // вес
    MyNode* down; // ссылка вниз
    MyNode() : down(nullptr), key() {} // конструктор по умолчанию
    explicit MyNode(int k, MyNode *down = nullptr) : key(k), down(down) {} //
конструктор
    ~MyNode() { delete down; } // Деструктор узла

};


// ИТЕРАТОР ЧТЕНИЯ
typedef MyNode* HTColumn;

struct readIter: public std::iterator<
        std::forward_iterator_tag,
        int
        > {

    HTColumn *columns; //массив сегментов
    size_t columnIndex; //позиция в массиве
    HTColumn elemInColumn; // Указатель на данные


    explicit readIter(MyNode* p = nullptr, int colIndex = 0) : columns(nullptr),
columnIndex(colIndex), elemInColumn(p) {} // конструктор по умолчанию


    bool operator == (const readIter & Other) const {
        return elemInColumn == Other.elemInColumn; // сравнение итераторов
    }

    bool operator != (const readIter & Other) const {
        return elemInColumn != Other.elemInColumn;
    }

    readIter operator++(); //объявление инкремента
    readIter operator++(int) {
        readIter temp(*this); // сохраняем значение
        ++(*this); // Увеличиваем значение
        return temp;
    }

    pointer operator -> () { // указатель на значение
        return & (elemInColumn->key); // возвращаем указатель на значение
    }

    reference operator * () {
        return elemInColumn->key; // возвращаем значение по ссылке
    }

};


// Итератор вставки (универсальный)
template <typename Container, typename ReadIter = readIter>
```

```
class insertIter:
public std::iterator<std::output_iterator_tag, typename Container::value_type> {
protected:
    Container& container; // контейнер для вставки элементов
    ReadIter readiter; // Текущее значение итератора
public:
    insertIter(Container& container, ReadIter riter) : container(container),
readiter(riter) {} // Конструктор итератора вставки

    const insertIter<Container>& // Равенство возвращает адрес итератора вставки
контейнера, а получает value_type
        operator = (const typename Container::value_type& value) {
            readiter = container.insert(readiter, value).first;
            return *this;
    }
    const insertIter<Container>& // Присваивание копии - фиктивное
        operator = (const insertIter<Container>&) { return *this; }

    insertIter<Container>& operator* () {return *this;} // Разыменование -
пустая операция
    insertIter<Container>& operator++ () {return *this;} // Инкремент - пустая
операция
    insertIter<Container>& operator++ (int) {return *this;} // Инкремент -
пустая операция
};




template <typename Container, typename Iter>
//Функция для создания итератора вставки
insertIter<Container, Iter> myInserter(Container& c, Iter it) {
    return insertIter<Container, Iter>(c, it);
}



class HT { //контейнер - хеш-таблица
    static size_t tags; // Счетчик тегов
    char tag; //тег таблицы
    HTColumn *buckets; // массив сегментов
    size_t count = 0; // мощность множества
public:
    static const size_t buckets_counter = 16; // К-во сегментов в демо-варианте
    std::vector<int> sequence; // последовательность Указателей на ключи
    using key_type = int;
    using value_type = int;
    using key_compare = std::equal_to<int>;

    void swap(HT& right) {
        std::swap(tag, right.tag);
        std::swap(buckets, right.buckets);
        std::swap(count, right.count);
    }

    static int hash(int k) { // Хэш-функция
        return (k * (buckets_counter - 1) + 7) % buckets_counter;
    }
    size_t bucket_count() {return buckets_counter;}

    std::pair<readIter, bool> insert(readIter, int); // вставка
```

```
    readIter insert(const int &k, readIter where = readIter(nullptr)) { //
Обертка для вставки
        return insert(where, k).first;
    }

    void Display(); // Вывод ХТ

    void setSeq(){sequence = std::vector<int>{begin(), end()};}

    readIter begin() const;
    readIter end() const {
        return readIter(nullptr); // итератор в никуда
    };

    readIter cbegin() const{
        return begin();
    }

    readIter cend() const{
        return end();
    }




    std::pair<readIter, bool> erase(int);
    readIter eraseElem(int num) {
        if (erase(num).second)
            return erase(num).first;
        else //Todo можно бросить exception
            return readIter(nullptr);
    }

    HT(): tag('A' + tags++), buckets(new HTColumn[buckets_counter]) {
        for (int i = 0; i < buckets_counter; ++i) buckets[i] = nullptr;
    }


    int size() { return count; }

    template<typename MyIt> HT(MyIt, MyIt); //Конструктор из последовательности
    ~HT() {
        --tags;
        for (int bucket = 0; bucket < buckets_counter; ++bucket) delete
buckets[bucket];
        delete[] buckets;
    }


    readIter find(int toFind) const; // Поиск по ключу

    HT(const HT& right) : HT() {
        for (readIter i = right.begin(); i != right.end(); ++i) this-
>insert(*i);
    }

    HT(HT&& right) : HT() { //Конструктор копирования
        swap(right);
    }

    HT& operator = (const HT& right) {
        HT temp;
        for (int & x : right) this->insert(x);
```

```
        swap(temp);
        return *this;
    }

    HT& operator = (HT && right) {
        swap(right);
        return *this;
    }

    HT& operator |= (const HT &);

    HT operator | (const HT & right) const {
        HT result(*this); return(result |= right);
    };

    HT& operator &= (const HT &);

    HT operator & (const HT& right) const {
        HT result(*this);
        return (result &= right);
    }

    HT& operator -= (const HT&);

    HT operator - (const HT & right) const {
        HT result(*this);
        return (result -= right);
    }

    HT& operator ^= (const HT&);

    HT operator ^ (const HT & right) const {
        HT result(*this);
        return (result ^= right);
    }


    void Merge(const HT& right);

    void Sort();

    void Concat(const HT &right);

    void Erase(int from, int to);
};

readIter HT::begin() const {//Итератор на начало
    readIter begin(nullptr); // Итератор чтения
    begin.columns = this->buckets;
    for (; begin.columnIndex < this->buckets_counter; ++begin.columnIndex) { //
проходимся по всем колонкам таблицы
        begin.elemInColumn = buckets[begin.columnIndex];
        if (begin.elemInColumn) break; //Выход, если сегмент не пуст, результат
- его начало
    }
    return begin;
}

readIter readIter::operator++() // Инкремент итератора = шаг по ХТ
{
    if(!elemInColumn) { //Первое обращение?
        return *this; // Текущая колонка еще не выставлена
    }
```

```
    else { //Текущий итератор указывает на элемент из колонки
        if(elemInColumn->down) { // Есть следующий в колонке - вниз
            elemInColumn = elemInColumn->down;
            return (*this);
        }
        while (++columnIndex < HT::buckets_counter) {//Поиск очередной не пустой
колонки с элементом
            if (columns[columnIndex]) { //Найден непустая колонка
                elemInColumn = columns[columnIndex]; // Устанавливаем итератор
на голову колонки
                return *this;
            }

        }
        elemInColumn = nullptr; //Таблица закончилась
        return *this;
    }
}


void HT::Display() {

    HTColumn* toPrint = new HTColumn[buckets_counter]; // Массив колонок
    for (auto i = 0 ; i < buckets_counter; ++i) toPrint[i] = buckets[i];
    bool notAllPrinted = true;
//    std::cout << tag << ':' << std::endl;
    std::cout << "Sequence: ";
    if ((sequence.empty()) && (size() != 0)) sequence =
std::vector<int>{begin(), end()};
    for (auto x: sequence) std::cout << " " << x << " ";
    std::cout << "\n\nRepresentation in memory:\n";
    while (notAllPrinted) {
        notAllPrinted = false;

        for (auto t = 0; t < buckets_counter; ++t) {

            if(toPrint[t]) {
                std::cout << std::setw(4) << toPrint[t]->key; // выводим ключ
                toPrint[t] = toPrint[t]->down; // спускаемся ниже
                notAllPrinted = true;
            } else {
                 std::cout << std::setw(4) << "-";
            }
        }
        std::cout << std::endl;
    }
}

// Поиск ключа toFind с выдачей итератора на него или end()
readIter HT::find(int toFind) const {
    auto colIndex = hash(toFind);
    HTColumn col = buckets[colIndex];
    while (col) {
        if (col->key == toFind) return readIter(col, colIndex);
        else col = col->down;
    }
    return end();
}

std::pair<readIter, bool> HT::insert(readIter, int k) //Вставка нового значения
k
{
    auto colIndex(hash(k));
```

```
    sequence.push_back(k); // Добавляем элемент в последовательность
    HTColumn elem = buckets[colIndex];
    while (elem) {
        if (elem->key == k) return make_pair(readIter(elem), true); // уже есть
        else elem = elem->down;
    }
    // Новый элемент
    buckets[colIndex] = new MyNode(k, buckets[colIndex]);
    ++count;
    return make_pair(readIter(buckets[colIndex]), true);
}

template<typename MyIt>
HT::HT(MyIt begin, MyIt end) : HT() {
    for (MyIt iter = begin; iter != end; ++iter) insert(*iter);
}

HT &HT::operator|=(const HT &right) {
    //Copy из библиотеки std копирует в таблицу все значения и игнорирует
дубликаты (из-за реализации Колинько)
    // Если скопировать что-то в пустую таблицу, получим просто копирование
    // Но можно и в не пустую, тогда все повторяющиеся элементы в резульатате не
повторятся
    // Ее предлагается использовать для реализации объединения множеств
    std::copy(right.begin(), right.end(), // source - Откуда
              myInserter(*this, readIter(nullptr))); // destination - куда
    this->sequence.insert(this->sequence.cend(),
                          right.sequence.cbegin(),
                          right.sequence.cend());
    Sort();
    return *this;
}

HT &HT::operator^=(const HT & right) { // xor = (left + right) - (left * right)
    HT leftTemp(*this);
    *this |= right;
    *this -= (leftTemp & right);
    return *this;
}

HT &HT::operator&=(const HT & right) {

    for (auto x : *this) {
        if (right.find(x) == end()) { // Элемент не нашелся
            bool deleted = this->erase(x).second;
            if (deleted) std::remove(sequence.begin(), sequence.end(), x);
        }
    }
    return *this;
}

HT &HT::operator-=(const HT & right) {
    // Удаляем те элементы, которые есть в right
    for (auto x : right)
        this->erase(x);

    return *this;
}

std::pair<readIter, bool> HT::erase(int toErase) {
    readIter founded = find(toErase);
    if (founded == end()) return std::make_pair(readIter(nullptr), false); //
элемент в XT не найден
```

```
    //Элемент найден
    sequence.erase(std::remove(sequence.begin(), sequence.end(), toErase), se-
quence.cend()); //Удаляет все элементы из последовательности с этим значением
//    sequence.erase(sequence.begin() + toErase);
    MyNode* head = this->buckets[founded.columnIndex]; // Голова списка
    if (head == founded.elemInColumn) { //удаляется голова списка
        this->buckets[founded.columnIndex] = founded.elemInColumn->down; //
Следующий элемент - голова
        founded.elemInColumn = nullptr; // Отвязываем для удаления
        delete founded.elemInColumn;
        return std::make_pair(founded, true); // возвращаем итератор на
освобожденную ячейку
    }
    MyNode* up = head;
    while (up->down != founded.elemInColumn) up = up->down; // доходим до
элемента
    up->down = founded.elemInColumn->down; // Перевязываем верх и низ
    founded.elemInColumn->down = nullptr; // Отвязываем низ
    delete founded.elemInColumn; // Чистим память

    return std::make_pair(founded, true); //возвращаем итератор на освобожденную
ячейку
}

void HT::Sort() { //Сортировка последовательности через algorithm
    std::sort(sequence.begin(), sequence.end());
}

void HT::Merge(const HT &right) {
//    std::vector<int> res(sequence.size() + right.sequence.size());
//    std::merge(right.sequence.begin(), right.sequence.end(), sequence.begin(),
sequence.end(), res.begin());
//    sequence = res;
    Concat(right);
    std::sort(sequence.begin(), sequence.end());
//    Sort();
}

void HT::Concat(const HT & right) {
    std::vector<int> res = sequence;
    res.insert(res.end(),
                          right.sequence.begin(),
                          right.sequence.end());
    *this |= right;
    sequence = res;
}

void HT::Erase(int from, int to) {
    try {
        // Проверка на exception
        sequence.at(from);
        sequence.at(to);
        for (int i = from; i <= to; ++i) {
            int toErase = sequence.at(from);
//            std::cout<<"\nTest: "<<toErase <<std::endl;
            erase(toErase); // Удаляем, что удаляется
//            sequence.erase(sequence.begin() + i);
        }
    } catch (std::out_of_range ex) {
        std::cout << "Erase out of range" << std::endl;
    }
```

```
}

#endif //LAB03_HASHTABLE_H
```