Министерство цифрового развития, связи и массовых коммуникаций

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»



Отчет по лабораторной работе №2

по дисциплине «Структура и алгоритмы обработки данных»

по теме «Методы поиска»

Выполнил: студент группы

БВТ1902

Бубенцов Н.В.

Проверил:

Москва

2021 г.

Оглавление

[Цель работы 3](#__RefHeading___Toc10834_3810339557)

[Задание 1 3](#__RefHeading___Toc10836_3810339557)

[Код программы 3](#__RefHeading___Toc10838_3810339557)

[Задание 2](#__RefHeading___Toc10840_3810339557) 8

[Код программы](#__RefHeading___Toc10842_3810339557) 8

[Задание 3](#__RefHeading___Toc10844_3810339557) 14

[Код программы](#__RefHeading___Toc10846_3810339557) 14

[Снимки экрана работы программ 1](#__RefHeading___Toc10848_3810339557)8

[Вывод 1](#__RefHeading___Toc10850_3810339557)9

## Цель работы

Реализовать методы поиска, простое рехеширование и метод цепочек в соответствии с заданием. Организовать генерацию начального набора случайных данных и оценить время работы каждого алгоритма поиска.

## Задание 1

|  |  |  |  |
| --- | --- | --- | --- |
| Бинарный поиск | Бинарное дерево | Фибоначчиев | Интерполяционный |

## Код программы

#include <iostream>

#include <clocale>

#include <algorithm>

#include<ctime>

//#include <iostream.h>

// #include "stdafx.h"

using namespace std;

const int A = 10;

int fibbonachi(const int arr[], int key, int A)

{

int fibM2 = 0;

int fibM1 = 1;

int fibM0 = fibM1 + fibM2;

while (fibM0 < A)

{

fibM2 = fibM1;

fibM1 = fibM0;

fibM0 = fibM1 + fibM2;

}

int offset = -1;

while (fibM0 > 1)

{

int i = std::min(fibM2 + offset, key - 1);

if (arr[i] < key)

{

fibM0 = fibM1;

fibM1 = fibM2;

fibM2 = fibM0 - fibM1;

}

else if (arr[i] > key)

{

fibM0 = fibM2;

fibM1 -= fibM2;

fibM2 = fibM0 - fibM1;

}

else

return i;

}

if (offset + 1 < A && arr[offset + 1] == key)

return offset + 1;

return -1;

}

int main()

{

setlocale(LC\_ALL, "Russian");

srand(static\_cast<unsigned int>(time(0)));

bool flag = false;

int l = 0; // левая граница

int r = 9; // правая граница

int min = 0;

int max = 100;

int mid;

int arr[A]; // создали массив на 10 элементов

int key; // создали переменную в которой будет находиться ключ

for (int i = 0; i < A; i++)

{

arr[i] = rand() % (max - min) + min;

}

cout << "Оригтнальный массив: " << endl;

for (int j = 0; j < A; j++)

{

cout << arr[j] << " ";

}

cout << endl;

sort(arr, arr + 10);

cout << "Сортированный массив: " << endl;

for (int j = 0; j < A; j++)

{

cout << arr[j] << " ";

}

int caseA = 0;

cout << "Выберите задачу: " << endl << "1. Бинарный поиск" << endl << "2. Фибоначчиев" << endl << "3. Интерполяционный" << endl;

cin >> caseA;

cout << endl;

switch (caseA)

{

case 1:

{

cout << endl << "Введите ключ: ";

cin >> key; // считываем ключ

while ((l <= r) && (flag != true))

{

mid = (l + r) / 2; // считываем срединный индекс отрезка [l,r]

if (arr[mid] == key) flag = true; //проверяем ключ со серединным элементом

if (arr[mid] > key) r = mid - 1; // проверяем, какую часть нужно отбросить

else l = mid + 1;

}

if (flag)

{

cout << "Индекс элемента " << key << " в массиве равен: " << mid << endl;

}

else

{

cout << "Извините, но такого элемента в массиве нет" << endl;

}

system("pause");

return 0;

}

case 2:

{

cout << endl << "Введите ключ: ";

cin >> key;

cout << fibbonachi(arr, key, A) << endl;

}

case 3:

{

int x = 0;

bool found;

cout << endl << "Введите ключ: ";

cin >> key;

for (found = false; (arr[l] < key) && (arr[r] > key) && !found; )

{

x = l + ((key - arr[l]) \* (r - l)) / (arr[r] - arr[l]);

if (arr[x] < key)

l = x + 1;

else if (arr[x] > key)

r = x - 1;

else

found = true;

}

if (arr[l] == key)

cout << key << " founded in element " << l << endl;

else if (arr[r] == key)

cout << key << " founded in element " << r << endl;

else

cout << "Sorry. Not found" << endl;

return 0;

}

case 4:

{

}

}

}

## Задание 2

|  |  |
| --- | --- |
| Простое рехеширование | Метод цепочек |

## Код программы

import java.util.ArrayList;  
  
class HashNode<K, V> {  
 K key;  
 V value;  
  
 HashNode<K, V> next;  
  
 public HashNode(K key, V value)  
 {  
 this.key = key;  
 this.value = value;  
 }  
}  
  
class Map<K, V>  
{  
 private ArrayList<HashNode<K, V> > bucketArray;  
  
 private int numBuckets;  
  
 private int size;  
  
 public Map()  
 {  
 bucketArray = new ArrayList<>();  
 numBuckets = 10;  
 size = 0;  
  
 for (int i = 0; i < numBuckets; i++)  
 bucketArray.add(null);  
 }  
  
 public int size() { return size; }  
  
 private int getBucketIndex(K key)  
 {  
 int hashCode = key.hashCode();  
 int index = hashCode % numBuckets;  
  
 index = index < 0 ? index \* -1 : index;  
 return index;  
 }  
  
 public V remove(K key)  
 {  
 int bucketIndex = getBucketIndex(key);  
  
 HashNode<K, V> head = bucketArray.get(bucketIndex);  
  
 HashNode<K, V> prev = null;  
  
 while (head != null)  
 {  
  
 if (head.key.equals(key))  
 break;  
  
 prev = head;  
 head = head.next;  
 }  
  
 if (head == null)  
 return null;  
  
 size--;  
  
 if (prev != null)  
 prev.next = head.next;  
 else  
 bucketArray.set(bucketIndex, head.next);  
  
 return head.value;  
 }  
  
 // Adds a key value pair to hash  
 public void add(K key, V value)  
 {  
 // Find head of chain for given key  
 int bucketIndex = getBucketIndex(key);  
 HashNode<K, V> head = bucketArray.get(bucketIndex);  
  
 // Check if key is already present  
 while (head != null) {  
 if (head.key.equals(key)) {  
 head.value = value;  
 return;  
 }  
 head = head.next;  
 }  
  
 // Insert key in chain  
 size++;  
 head = bucketArray.get(bucketIndex);  
 HashNode<K, V> newNode = new HashNode<K, V>(key, value);  
 newNode.next = head;  
 bucketArray.set(bucketIndex, newNode);  
  
 // If load factor goes beyond threshold, then  
 // double hash table size  
 if ((1.0 \* size) / numBuckets >= 0.7)  
 {  
 ArrayList<HashNode<K, V> > temp = bucketArray;  
 bucketArray = new ArrayList<>();  
 numBuckets = 2 \* numBuckets;  
 size = 0;  
 for (int i = 0; i < numBuckets; i++)  
 bucketArray.add(null);  
  
 for (HashNode<K, V> headNode : temp) {  
 while (headNode != null) {  
 add(headNode.key, headNode.value);  
 headNode = headNode.next;  
 }  
 }  
 }  
 }  
  
 // Driver method to test Map class  
 public static void main(String[] args)  
 {  
 Map<String, Integer> map = new Map<>();  
 map.add("this", 1);  
 map.add("coder", 2);  
 map.add("this", 4);  
 map.add("hi", 5);  
 System.*out*.println(map.size());  
 System.*out*.println(map.remove("this"));  
 System.*out*.println(map.remove("this"));  
 System.*out*.println(map.size());  
 }  
}

import java.util.ArrayList;  
import java.util.Scanner;  
  
class ReHashMap<K, V>  
{  
  
 class MapNode<K, V>  
 {  
  
 K key;  
 V value;  
 MapNode<K, V> next;  
  
  
 public MapNode(K key, V value)  
 {  
 this.key = key;  
 this.value = value;  
 next = null;  
 }  
 }  
  
 ArrayList<MapNode<K, V>> buckets;  
  
 // No. of pairs stored - n  
 int size;  
  
 // Size of the bucketArray - b  
 int numBuckets;  
  
 // Default loadFactor  
 final double DEFAULT\_LOAD\_FACTOR = 0.75;  
  
 public ReHashMap()  
 {  
 numBuckets = 5;  
  
 buckets = new ArrayList<>(numBuckets);  
  
 for (int i = 0; i < numBuckets; i++)  
 {  
 // Initialising to null  
 buckets.add(null);  
 }  
 System.*out*.println("Size of Map: " + numBuckets);  
 System.*out*.println("Default Load Factor : " + DEFAULT\_LOAD\_FACTOR + "\n");  
 }  
  
 private int getBucketId(K key)  
 {  
  
 int hashCode = key.hashCode();  
  
 return (hashCode % numBuckets);  
 }  
  
 public void insert(K key, V value)  
 {  
 int bucketId = getBucketId(key);  
  
 MapNode<K, V> head;  
 MapNode<K, V> newElementNode = new MapNode<K, V>(key, value);  
  
 head = buckets.get(bucketId);  
 newElementNode.next = head;  
  
 buckets.set(bucketId, newElementNode);  
  
 System.*out*.println("Pair(" + key + ", " + value + ") inserted");  
  
 size++;  
  
 double loadFactor = size\*1.0 / numBuckets;  
 System.*out*.println("Load factor = " + loadFactor);  
  
 //check for rehash  
 if (loadFactor > DEFAULT\_LOAD\_FACTOR)  
 {  
 System.*out*.println("Load factor rather then 0.75, rehashing");  
 rehash();  
  
 System.*out*.println("New Size of Map: " + numBuckets + "\n");  
 }  
 else  
 System.*out*.println("Size of Map: " + numBuckets + "\n");  
 }  
  
 private void rehash()  
 {  
 ArrayList<MapNode<K, V>> temp = buckets;  
  
 buckets = new ArrayList<MapNode<K, V>>(2 \* numBuckets);  
  
 for (int i = 0; i < 2 \* numBuckets; i++)  
 {  
 // Initialised to null  
 buckets.add(null);  
 }  
 size = 0;  
 numBuckets \*= 2;  
  
 for (int i = 0; i < temp.size(); i++)  
 {  
  
 MapNode<K, V> head = temp.get(i);  
  
 while (head != null)  
 {  
 K key = head.key;  
 V val = head.value;  
  
 insert(key, val);  
 head = head.next;  
 }  
 }  
 System.*out*.println("Rehashing Done \n");  
 }  
  
 public void printMap()  
 {  
  
 // The present bucket list is made temp  
 ArrayList<MapNode<K, V>> temp = buckets;  
  
 System.*out*.println("Current HashMap:");  
 // loop through all the nodes and print them  
 for (int i = 0; i < temp.size(); i++)  
 {  
  
 MapNode<K, V> head = temp.get(i);  
  
 while (head != null)  
 {  
 System.*out*.println("key = " + head.key + ", val = " + head.value);  
  
 head = head.next;  
 }  
 }  
 System.*out*.println();  
 }  
  
 public static void main(String[] args)  
 {  
 Scanner scanner= new Scanner(System.*in*);  
 int count=1;  
 ReHashMap<Integer, String> reHashMap = new ReHashMap<Integer, String>();  
  
 String word;  
  
 do  
 {  
 System.*out*.println("Insert object for HashMap, '0' for exit");  
 word=scanner.nextLine();  
 if (!word.equals("0"))  
 {  
 reHashMap.insert(count, word);  
 reHashMap.printMap();  
 count++;  
 }  
 } while (!word.equals("0"));  
  
 }  
}

## Задание 3

Расставить на стандартной 64-клеточной шахматной доске 8 ферзей так, чтобы ни один из них не находился под боем другого». Подразумевается, что ферзь бьёт все клетки, расположенные по вертикалям, горизонталям и обеим диагоналям

## Код программы

#include <iostream>

#include "Chess.cpp"

using namespace std;

int main()

{

chess table;

vector<figure> board;

board = table.solve();

table.printBoard(board);

for (auto i : board)

cout << i.getx() << " " << i.gety() << "; ";

}

#include <vector>

#include <iostream>

using namespace std;

class figure

{

private:

int x;

int y;

public:

figure() {x=0; y=0;}

figure(int xNew, int yNew) {x=xNew; y=yNew; }

int getx() { return x;}

int gety() { return y;}

void setx(int xNew) { x=xNew;}

void sety(int yNew) { y=yNew;}

};

class chess

{

private:

bool board[8][8]={true, };

public:

chess(){reset();}

void reset()

{

for (int i=0; i<8; i++)

{

for (int j=0; j<8; j++)

{

board[i][j]=true;

}

}

}

void removePositions(int i, int j)

{

//cols and rows

for (int k=0; k<8; k++)

{

board[i][k]=false;

board[k][j]=false;

}

int tempi=i;

int tempj=j;

//main diag

while (tempi!=0 && tempj!=0)

{

tempi--;

tempj--;

}

while (tempi!=7 && tempj!=7)

{

board[tempi][tempj]=false;

tempi++;

tempj++;

}

while(i!=7 && j!=7)

{

i++;

j--;

}

while (i!=0 && j!=7)

{

board[i][j]=false;

i--;

j++;

}

}

vector<figure> solve()

{

figure place;

vector<figure> places;

int i=0;

int j=0;

while(places.size()!=8)

{

if (board[i][j])

{

place.setx(j);

place.sety(i);

places.push\_back(place);

removePositions(i, j);

i+=2;

j++;

}

if (i>7)

{

i = 0;

}

while(!board[i][j])

{

i++;

}

}

return places;

}

bool check(figure temp, vector<figure> table)

{

for (auto i: table)

if (i.getx()==temp.getx() && i.gety()==temp.gety())

return true;

return false;

}

void printBoard(vector<figure> table)

{

figure quieen;

for (int i=0; i<8; i++)

{

for (int j=0; j<8; j++)

{

quieen.setx(j);

quieen.sety(i);

if (check(quieen, table))

{

cout <<"Q ";

}

else if (board[i][j])

cout << "+ ";

else

cout <<"- ";

}

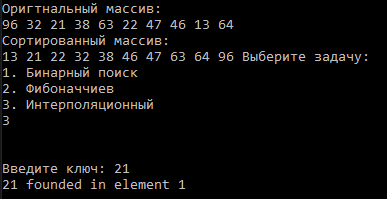
cout<<endl;

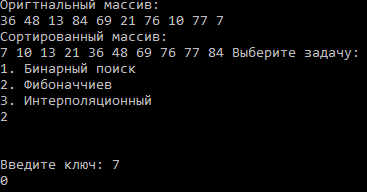
}

}

};

## Снимки экрана работы программ





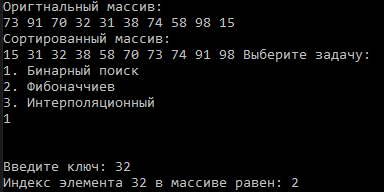
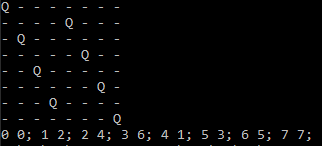


Рисунок 1 – Проверка корректности поисков

Рисунок 2 – Результат поиска позиций для 8 ферзей

## Вывод

Я реализовал методы поиска: бинарный, фиббоначиев, интерполяционный. Создал структуры данных: бинарноре дерево поиска, хеш карта на основе рехеширования и хеш карту на методе цепочек. Организовал генерации начальных значений. А так же написал алгоритм решения задачи про 8 ферзей.