**Министерство цифрового развития, связи**

**и массовых коммуникаций Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра «Математическая кибернетика и информационные технологии»

**Отчет по лабораторной работе №2**

по дисциплине «Структуры и алгоритмы обработки данных»

на тему «Методы поиска»

Выполнил: студент группы БВТ1901

Перевозчиков С. В.

Руководитель:

Мелехин А. А.

Москва 2021

Цель работы: изучить основные виды поиска элементов в массиве, структуру хэш-таблицы и методы рехэширования для устранения коллизий в них и написать их реализацию на одном из языков программирования.

Техническое задание:

Реализовать методы поиска в соответствии с заданием. Организовать генерацию начального набора случайных данных. Для всех вариантов добавить реализацию добавления, поиска и удаления элементов. Оценить время работы каждого алгоритма поиска и сравнить его со временем работы стандартной функции поиска, используемой в выбранном языке программирования.

Задание 1.

Написать функции поиска:

- Бинарный поиск

- Фибоначчиев поиск

- Интерполяционный поиск

Написать реализацию структуры бинарного дерева.

Задание 2.

Написать реализацию структуры хэш-таблицы и реализовать методы устранения коллизий:

- Простое рехэширование

- Рехэширование с помощью псевдослучайных чисел

- Метод цепочек

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. Написать программу, которая находит хотя бы один способ решения задач.

Выполнение задания:

Задание 1.

Бинарный поиск:

Код:

public static int binary(int[] array, int number)

{

int left = 0, right = array.length-1, index = -1, iter = 0;

while(iter < array.length)

{

if(array[left + (right - left) / 2] == number)

{

index = left + (right - left) / 2;

break;

}

if(array[left + (right - left) / 2] > number)

{

right = left + (right - left) / 2;

}

if(array[(right - left) / 2] < number)

{

left = left + (right - left) / 2;

}

iter++;

}

System.out.println("Index of " + Integer.toString(number) + " = " + Integer.toString(index));

return index;

}

Фибоначчиев поиск:

Код:

public static int f(int num)

{

if(num == 0)

{

return 0;

}

if(num == 1)

{

return 1;

}

return f(num - 1) + f(num - 2);

}

public static int fibonachi(int[] array, int number)

{

int index = -1, iter = 0, left = 0, right = array.length-1, i = 0;

while(iter < array.length)

{

if(left + f(i) <= right)

{

if(array[left + f(i)] == number)

{

index = left + f(i);

break;

}

if(array[left + f(i)] < number)

{

i++;

}

else

{

right = left + f(i);

left = left + f(i-1);

i = 0;

}

if(i != 0)

{

iter++;

}

}

else

{

if(array[right] == number)

{

index = right;

break;

}

if(array[right] < number)

{

break;

}

else

{

left = left + f(i-1);

i = 0;

}

if(i != 0)

{

iter++;

}

}

}

System.out.println("Index of " + Integer.toString(number) + " = " + Integer.toString(index));

return index;

}

Интерполяционный поиск:

Код:

public static int interpolar(int[] array, int number)

{

double left = 0, right = array.length-1;

int index = -1, iter = 0;

while(iter < array.length)

{

int d = (int) (left + (right - left) \* (number - array[(int) left]) / (array[(int) right] - array[(int) left]));

if(array[d] == number)

{

index = d;

break;

}

if(array[d] > number)

{

right = d;

}

if(array[d] < number)

{

left = d;

}

iter++;

}

System.out.println("Index of " + Integer.toString(number) + " = " + Integer.toString(index));

return index;

}

Бинарное дерево:

Код:

public class BinaryTree

{

private Integer root;

private BinaryTree leftTree;

private BinaryTree rightTree;

public void search(int num)

{

if(this.root == null)

{

System.out.println("Tree doesn't contain " + Integer.toString(num));

}

else

{

if(this.root == num)

{

System.out.println();

}

if(num < this.root)

{

if(this.leftTree == null)

{

System.out.println("Tree doesn't contain " + Integer.toString(num));

}

else

{

System.out.print("L");

this.leftTree.search(num);

}

}

if(num > this.root)

{

if(this.rightTree == null)

{

System.out.println("Tree doesn't contain " + Integer.toString(num));

}

else

{

System.out.print("R");

this.rightTree.search(num);

}

}

}

}

public void add(int num)

{

if(this.root == null)

{

this.root = num;

}

if(num < this.root)

{

if(this.leftTree == null)

{

this.leftTree = new BinaryTree();

}

this.leftTree.add(num);

}

if(num > this.root)

{

if(this.rightTree == null)

{

this.rightTree = new BinaryTree();

}

this.rightTree.add(num);

}

}

public int getRoot()

{

return this.root;

}

public BinaryTree getLeft()

{

return this.leftTree;

}

public BinaryTree getRight()

{

return this.rightTree;

}

public void setLeft(BinaryTree left)

{

this.leftTree = left;

}

public void setRight(BinaryTree right)

{

this.rightTree = right;

}

public void delete(int num)

{

if(this.root == num)

{

if(this.leftTree == null && this.rightTree == null)

{

this.root = null;

}

if(this.leftTree != null && this.rightTree == null)

{

this.root = leftTree.getRoot();

this.leftTree = leftTree.getLeft();

this.rightTree = leftTree.getRight();

}

if(this.leftTree == null && this.rightTree != null)

{

this.root = rightTree.getRoot();

this.leftTree = rightTree.getLeft();

this.rightTree = rightTree.getRight();

}

if(this.leftTree != null && this.rightTree != null)

{

this.root = leftTree.getRoot();

this.leftTree = leftTree.getLeft();

BinaryTree right = this.getRight();

this.rightTree = leftTree.getRight();

this.rightTree.setRight(right);

}

}

if(this.root != null)

{

if(num < this.root)

{

if(this.leftTree == null)

{

System.out.println("Tree doesn't contain " + Integer.toString(num));

}

else

{

this.leftTree.delete(num);

}

}

if(this.root != null && num > this.root)

{

if(this.rightTree == null)

{

System.out.println("Tree doesn't contain " + Integer.toString(num));

}

else

{

this.rightTree.delete(num);

}

}

}

}

}

Задание 2.

Хэш-таблица:

Код:

public class HashTable

{

private static Integer[] table;

private static ArrayList<Integer> randoms;

private static int tableSize;

private static ArrayList<Integer> stackTable;

private static ArrayList<Integer> links;

public static void main(String[] args)

{

try

{

if(args.length >= 2)

{

tableSize = Integer.parseInt(args[1]);

}

else

{

tableSize = 257;

}

if(args[0].equals("rehashSimple") || args[0].equals("rehashRandom"))

{

table = new Integer[tableSize];

}

if(args[0].equals("rehashRandom"))

{

randoms = new ArrayList<>();

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

{

while(true)

{

int num = (int)(Math.random()\*tableSize);

if(!randoms.contains(num))

{

randoms.add(num);

break;

}

}

}

}

if(args[0].equals("chainMethod"))

{

table = new Integer[tableSize];

stackTable = new ArrayList<>();

links = new ArrayList<>();

}

Scanner input = new Scanner(System.in);

System.out.println("Enter 'exit' to close the program\n'input' to add number into table\n'find' to search number in table\n'print' to print a table");

while(true)

{

String line = input.nextLine();

if(line.equals("exit"))

{

break;

}

if(line.equals("input"))

{

System.out.print("Enter number ");

if(input.hasNextInt())

{

int number = input.nextInt();

if(args[0].equals("rehashSimple"))

{

rehashSimple(number, number);

}

if(args[0].equals("rehashRandom"))

{

rehashRandom(number, 0);

}

if(args[0].equals("chainMethod"))

{

chainMethod(number);

}

}

else

{

System.out.println("Entered string isn't a number");

}

}

if(line.equals("find"))

{

System.out.print("Enter number ");

if(input.hasNextInt())

{

int number = input.nextInt();

if(args[0].equals("rehashSimple"))

{

rehashSimpleFind(number, number);

}

if(args[0].equals("rehashRandom"))

{

rehashRandomFind(number, 0);

}

if(args[0].equals("chainMethod"))

{

chainMethodFind(number);

}

}

else

{

System.out.println("Entered string isn't a number");

}

}

if(line.equals("print"))

{

if(args[0].equals("rehashSimple") || args[0].equals("rehashRandom"))

{

System.out.println(Arrays.toString(table));

}

if(args[0].equals("chainMethod"))

{

System.out.println(stackTable.toString());

System.out.println(links.toString());

}

}

}

input.close();

}

catch(Exception ex)

{

System.out.println("Arguments error, input form: java HashTable type\_of\_task <size\_of\_table>");

ex.printStackTrace();

}

}

//...Методы для поиска или добавления чисел в таблицу

}

Методы поиска и добавления элементов, реализующие простое рехэширование:

Код:

public static void rehashSimple(int number, int current)

{

if(table[current % tableSize] == null)

{

table[current % tableSize] = number;

}

else

{

if(current != number + tableSize)

{

rehashSimple(number, current + 1);

}

else

{

System.out.println("Table is filled");

}

}

}

public static Integer rehashSimpleFind(int number, int current)

{

if(table[current % tableSize] != null)

{

if(table[current % tableSize] == number)

{

System.out.println("Index of number = " + Integer.toString(current % tableSize));

return current % tableSize;

}

else

{

if(current != number + tableSize)

{

rehashSimpleFind(number, current + 1);

}

else

{

System.out.println("Table doesn't contain number");

}

}

}

else

{

System.out.println("Table doesn't contain number");

}

return null;

}

Методы поиска и добавления элементов, реализующие рехэширование с помощью псевдослучайных чисел:

Код:

public static void rehashRandom(int number, int rehashIndex)

{

if(table[number % tableSize] == null)

{

table[number % tableSize] = number;

}

else

{

if(table[randoms.get(rehashIndex)] == null)

{

table[randoms.get(rehashIndex)] = number;

}

else

{

if(rehashIndex + 1 < randoms.size()-1)

{

rehashRandom(number, rehashIndex + 1);

}

else

{

System.out.println("Table is filled");

}

}

}

}

public static Integer rehashRandomFind(int number, int rehashIndex)

{

if(table[number % tableSize] != null)

{

if(table[number % tableSize] == number)

{

System.out.println("Index of number = " + Integer.toString(number % tableSize));

return number % tableSize;

}

else

{

if(table[randoms.get(rehashIndex)] != null)

{

if(table[randoms.get(rehashIndex)] == number)

{

System.out.println("Index of number = " + Integer.toString(randoms.get(rehashIndex)));

return randoms.get(rehashIndex);

}

else

{

if(rehashIndex + 1 < randoms.size()-1)

{

rehashRandomFind(number, rehashIndex + 1);

}

else

{

System.out.println("Table doesn't contain number");

}

}

}

else

{

System.out.println("Table doesn't contain number");

}

}

}

else

{

System.out.println("Table doesn't contain number");

}

return null;

}

Методы поиска и добавления элементов, реализующие метод цепочек:

Код:

public static void chainMethod(int number)

{

if(table[number % tableSize] == null)

{

stackTable.add(number);

links.add(null);

table[number % tableSize] = stackTable.size() - 1;

}

else

{

int i = table[number % tableSize];

while(stackTable.get(i) != number || links.get(i) != null)

{

if(stackTable.get(i) == number)

{

System.out.println("Table contains number already");

}

else

{

if(links.get(i) != null)

{

i = links.get(i);

}

else

{

stackTable.add(number);

links.add(null);

links.set(i, links.size()-1);

}

}

}

}

}

public static Integer chainMethodFind(int number)

{

if(table[number % tableSize] != null)

{

int i = table[number % tableSize];

while(true)

{

if(stackTable.get(i) == number)

{

System.out.println("Index of number = " + Integer.toString(i));

return i;

}

else

{

if(links.get(i) != null)

{

i = links.get(i);

}

else

{

System.out.println("Table doesn't contain number");

return null;

}

}

}

}

else

{

System.out.println("Table doesn't contain number");

}

return null;

}

Задание 3.

Программа заполняет таблицу 8х8 числами, где позиции, на которых находится 1000, обозначают местоположение ферзей.

Код:

public class Lab2Q

{

public static int[][] board = new int[8][8];

public static int countOfQueens = 0;

public static void set()

{

set(new int[8][8], 0, 0);

}

public static void set(int[][] newBoard, int x0, int y0)

{

int[][] old = new int[8][8];

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

{

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

{

old[k][l] = newBoard[k][l];

}

}

if(countOfQueens < 8)

{

for(int j = y0; j < 8; j++)

{

if(board[x0][j] == 0)

{

countOfQueens++;

board[x0][j] += 1000;

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

{

if(board[x0][k] != 1000)

{

board[x0][k]++;

}

}

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

{

if(board[k][j] != 1000)

{

board[k][j]++;

}

}

for(int k = x0, l = j; k < 8 && l < 8;)

{

if(board[k][l] != 1000)

{

board[k][l]++;

}

k++;

l++;

}

for(int k = x0, l = j; k >= 0 && l < 8;)

{

if(board[k][l] != 1000)

{

board[k][l]++;

}

k--;

l++;

}

for(int k = x0, l = j; k < 8 && l >= 0;)

{

if(board[k][l] != 1000)

{

board[k][l]++;

}

k++;

l--;

}

for(int k = x0, l = j; k >= 0 && l >= 0;)

{

if(board[k][l] != 1000)

{

board[k][l]++;

}

k--;

l--;

}

set(board, x0, j);

if(countOfQueens < 8)

{

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

{

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

{

board[k][l] = old[k][l];

}

}

countOfQueens--;

}

}

}

for(int i = x0+1; i < 8; i++)

{

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

{

if(board[i][j] == 0)

{

countOfQueens++;

board[i][j] += 1000;

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

{

if(board[i][k] != 1000)

{

board[i][k]++;

}

}

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

{

if(board[k][j] != 1000)

{

board[k][j]++;

}

}

for(int k = i, l = j; k < 8 && l < 8;)

{

if(board[k][l] != 1000)

{

board[k][l]++;

}

k++;

l++;

}

for(int k = i, l = j; k >= 0 && l < 8;)

{

if(board[k][l] != 1000)

{

board[k][l]++;

}

k--;

l++;

}

for(int k = i, l = j; k < 8 && l >= 0;)

{

if(board[k][l] != 1000)

{

board[k][l]++;

}

k++;

l--;

}

for(int k = i, l = j; k >= 0 && l >= 0;)

{

if(board[k][l] != 1000)

{

board[k][l]++;

}

k--;

l--;

}

set(board.clone(), i, j);

if(countOfQueens < 8)

{

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

{

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

{

board[k][l] = old[k][l];

}

}

countOfQueens--;

}

}

}

}

}

}

}