МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное

учреждение высшего образования

«Южно-Уральский государственный университет

(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук

Кафедра системного программирования

**ОТЧЕТ**

о выполнении практического задания № 6

по дисциплине

«Структуры и алгоритмы обработки данных»

|  |  |
| --- | --- |
|  | Выполнил:  студент группы КЭ-203  Старостенок Д.В.  Проверил:  ст. преподаватель кафедры СП  Петрова Л.Н. |

Челябинск – 2022

ОГЛАВЛЕНИЕ

[1. Задача № 1 “ОЧЕРЕДИ” 2](#_Toc99446992)

[1.1. Физические величины: 2](#_Toc99446993)

[1.2. Область допустимых значений физических величин: 2](#_Toc99446994)

[1.3. Единицы измерения физических величин: 2](#_Toc99446995)

[1.4. Ограничения: 2](#_Toc99446996)

[1.5. Анализ ожидаемых результатов: 2](#_Toc99446997)

[1.6. Листинг программы: 3](#_Toc99446998)

[1.7. Контрольный тест: 4](#_Toc99446999)

[2. Задача № 2 “МНОЖЕСТВА”. 6](#_Toc99447000)

[2.1. Задача: 6](#_Toc99447001)

[2.2. Физические величины: 6](#_Toc99447002)

[2.3. Область допустимых значений физических величин: 6](#_Toc99447003)

[2.4. Единицы измерения физических величин: 6](#_Toc99447004)

[2.5. Ограничения: 6](#_Toc99447005)

[2.6. Анализ ожидаемых результатов: 6](#_Toc99447006)

[2.7. Листинг программы: 6](#_Toc99447007)

[2.8. Контрольный тест: 7](#_Toc99447008)

[Вывод 8](#_Toc99447009)

1. Задача № 1 “Поиск в лиНейных структурах”

Задача:

Используя бинарный поиск написать программу, которая:

* запрашивает у пользователя размер массива и ключ поиска;
* формирует массив случайных чисел;
* возвращает количество совпадений и их номера.
  1. Физические величины:

int, var

* 1. Область допустимых значений физических величин:

int – от -2147483648 до 2147483647

* 1. Единицы измерения физических величин:

int – целочисленный тип, 4 байта

* 1. Ограничения:

Время поиска, длина массива

* 1. Анализ ожидаемых результатов:

Бинарный поиск выбранного элемента в массива

* 1. Листинг программы:

class Program

{

private static int BinarySearch(int key, int[] arr) // Бинарный поиск

{

int low = 0; // Нижняя граница массива

int high = arr.Length - 1; // Верхняя граница массива

while (low <= high)

{

int midpoint = low + (high - low) / 2; // Середина массива

if (key < arr[midpoint]) // Сдвиг верхней границы

{

high = midpoint;

}

else if (key > arr[midpoint]) // Сдвиг нижней границы

{

low = midpoint + 1;

}

else

{

return midpoint;

}

}

return -1;

}

private static Dictionary<int, List<int>> OccurrenceCount(int key, int[] arr) // Поиск совпадений

{

int elements = BinarySearch(key, arr);

if (elements == -1)

Console.WriteLine("Элемент не был найден");

List<int> indexes = new List<int> {elements};

int count = 1;

int left = elements - 1;

while (left >= 0 && arr[left] == key) // Поиск в нижней границе

{

indexes.Add(left);

count++;

left--;

}

int right = elements + 1;

while (right < arr.Length && arr[right] == key) // Поиск в верхней границе

{

indexes.Add(right);

count++;

right++;

}

Dictionary<int, List<int>> result = new Dictionary<int, List<int>> // Добавление совпадений в словарь

{

[count] = indexes

};

return result;

}

public static void Main(string[] args)

{

Console.WriteLine("Введите размер массива ");

int arrSize = Convert.ToInt32(Console.ReadLine());

int[] array = new int[arrSize];

Random rnd = new Random();

for (int i = 0; i < arrSize; i++) // Заполнение массива

{

array[i] = rnd.Next(-10, 10);

}

Array.Sort(array); // Сортировка массива

Console.WriteLine("Отсортированный массив:");

foreach (var elements in array)

{

Console.Write(elements + " ");

}

Console.WriteLine("\nВведите ключ поиска ");

int searchKey = Convert.ToInt32(Console.ReadLine());

var occurrences = OccurrenceCount(searchKey, array);

foreach (var keyValuePair in occurrences)

{

Console.WriteLine($"Количество совпадений элементов = {keyValuePair.Key} \nНомера элементов:");

foreach (var elements in keyValuePair.Value)

{

Console.Write($"{elements} ");

}

}

}

}

* 1. Контрольный тест:

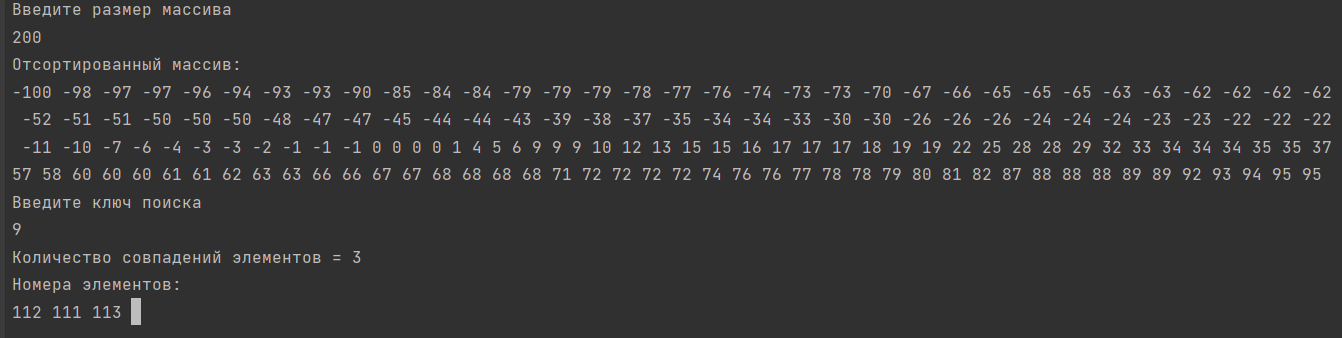


Рис. 1. Тест поиска 1

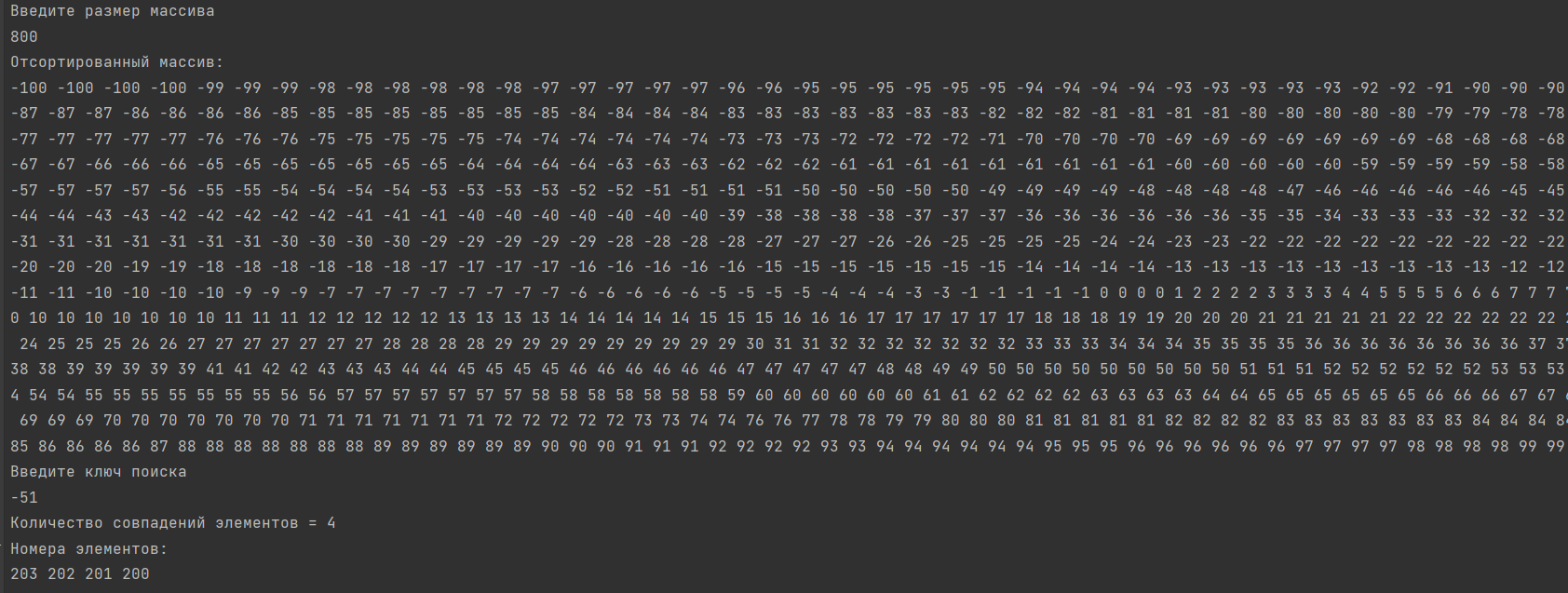


Рис. 2. Тест поиска 2

1. Задача № 2 “Использование деревьев в задачах поиска”.
   1. Задача:

Осуществить алгоритм упорядоченного дерева поиска. Найти ключ поиска и удалить его из дерева.

* 1. Физические величины:

int, var

* 1. Область допустимых значений физических величин:

int – от -2147483648 до 2147483647

* 1. Единицы измерения физических величин:

int – целочисленный тип, 4 байта

* 1. Ограничения:

Время на поиск, размер дерева

* 1. Анализ ожидаемых результатов:

Вывод созданного дерева, вывод дерева с удалением выбранного элемента

* 1. Листинг программы:

class Program

{

public class BinaryTree<T> where T : IComparable<T>

{

private BinaryTree<T> \_parent, \_left, \_right;

private T \_value;

private List<T> \_listForPrint = new List<T>();

public BinaryTree(T value, BinaryTree<T> parent)

{

\_value = value;

\_parent = parent;

}

public void Add(T val) // Добавление

{

if(val.CompareTo(\_value) < 0){

if(\_left==null){

\_left = new BinaryTree<T>(val, this);

}

else if(\_left != null)

\_left.Add(val);

}

else{

if(\_right==null){

\_right = new BinaryTree<T>(val, this);

}

else if(\_right != null)

\_right.Add(val);

}

}

private static BinaryTree<T> \_Search(BinaryTree<T> tree, T val) // Поиск

{

while (true)

{

if (tree == null) return null;

switch (val.CompareTo(tree.\_value))

{

case 1:

tree = tree.\_right;

continue;

case -1:

tree = tree.\_left;

continue;

case 0:

return tree;

default:

return null;

}

}

}

private BinaryTree<T> Search(T val)

{

return \_Search(this, val);

}

public void Remove(T val)

{

BinaryTree<T> tree = Search(val); //Проверка, существует ли данный узел

if(tree == null)

{

return;

}

BinaryTree<T> curTree;

//Если удаляем корень

if(tree == this){

if(tree.\_right!=null) {

curTree = tree.\_right;

}

else curTree = tree.\_left;

while (curTree.\_left != null) {

curTree = curTree.\_left;

}

T temp = curTree.\_value;

Remove(temp);

tree.\_value = temp;

return;

}

//Удаление листьев

if(tree.\_left==null && tree.\_right==null && tree.\_parent != null)

{

if(tree == tree.\_parent.\_left)

tree.\_parent.\_left = null;

else{

tree.\_parent.\_right = null;

}

return;

}

//Удаление узла, имеющего левое поддерево, но не имеющее правого поддерева

if(tree.\_left != null && tree.\_right == null){

//Меняем родителя

tree.\_left.\_parent = tree.\_parent;

if(tree == tree.\_parent?.\_left){

tree.\_parent.\_left = tree.\_left;

}

else if(tree == tree.\_parent?.\_right){

tree.\_parent.\_right = tree.\_left;

}

return;

}

//Удаление узла, имеющего правое поддерево, но не имеющее левого поддерева

if(tree.\_left == null && tree.\_right != null){

//Меняем родителя

tree.\_right.\_parent = tree.\_parent;

if(tree == tree.\_parent?.\_left){

tree.\_parent.\_left = tree.\_right;

}

else if(tree == tree.\_parent?.\_right){

tree.\_parent.\_right = tree.\_right;

}

return;

}

//Удаляем узел, имеющий поддеревья с обеих сторон

if(tree.\_right!=null && tree.\_left!=null) {

curTree = tree.\_right;

while (curTree.\_left != null) {

curTree = curTree.\_left;

}

//Если самый левый элемент является первым потомком

if(curTree.\_parent == tree) {

curTree.\_left = tree.\_left;

tree.\_left.\_parent = curTree;

curTree.\_parent = tree.\_parent;

if (tree == tree.\_parent?.\_left) {

tree.\_parent.\_left = curTree;

} else if (tree == tree.\_parent?.\_right) {

tree.\_parent.\_right = curTree;

}

return;

}

//Если самый левый элемент НЕ является первым потомком

if (curTree.\_right != null) {

curTree.\_right.\_parent = curTree.\_parent;

}

curTree.\_parent.\_left = curTree.\_right;

curTree.\_right = tree.\_right;

curTree.\_left = tree.\_left;

tree.\_left.\_parent = curTree;

tree.\_right.\_parent = curTree;

curTree.\_parent = tree.\_parent;

if (tree == tree.\_parent?.\_left) {

tree.\_parent.\_left = curTree;

} else if (tree == tree.\_parent?.\_right) {

tree.\_parent.\_right = curTree;

}

return;

}

}

private void \_print(BinaryTree<T> node)

{

while (true)

{

if (node == null) return;

\_print(node.\_left);

\_listForPrint.Add(node.\_value);

Console.Write(node + " ");

if (node.\_right != null)

{

node = node.\_right;

continue;

}

break;

}

}

public void Print()

{

\_listForPrint.Clear();

\_print(this);

Console.WriteLine();

}

public override string ToString()

{

return \_value.ToString();

}

}

public static void Main(string[] args)

{

BinaryTree<int> tree = new BinaryTree<int>(10, null);

Console.WriteLine("Обход дерева до удаления ключа");

tree.Add(2);

tree.Add(3);

tree.Add(8);

tree.Add(4);

tree.Add(6);

tree.Add(9);

tree.Print();

Console.WriteLine("Введите ключ для удаления");

int digitForRemove = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Обход дерева после удаления ключа");

tree.Remove(digitForRemove);

tree.Print();

}

}

* 1. Контрольный тест:

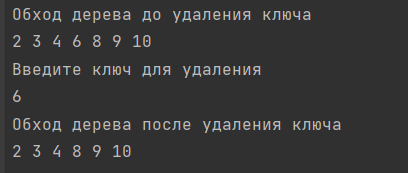


Рис. 3. Тест удаление элемента из дерева 1

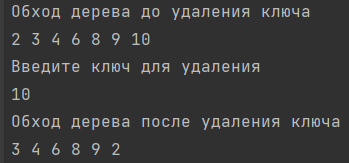


Рис. 4. Тест удаление элемента из дерева 2

1. Задача № 3 “Поиск в тексте”.
   1. Задача:

В текстовом файле хранится текст. Осуществить прямой поиск введенного пользователем слова с использованием алгоритма Боуера и Мура.

* 1. Физические величины:

int, var, string

* 1. Область допустимых значений физических величин:

int – от -2147483648 до 2147483647

string – ~1 миллиард символов

* 1. Единицы измерения физических величин:

int – целочисленный тип, 4 байта

string – строки, 2 ГБ

* 1. Ограничения:

Время на поиск, размер текста

* 1. Анализ ожидаемых результатов:

Вывод номера искомого слова

* 1. Листинг программы:

class Program

{

private static string GetTextFromFile(string name)

{

string text;

using (StreamReader reader = new StreamReader(name))

{

text = reader.ReadLine();

}

return text;

}

private static int BmSearch(string pattern, string txt)

{

byte[] shiftArray = new byte[0x10000]; // сдвиг массива

for (int i = 0; i < shiftArray.Length; i++)

{

shiftArray[i] = (byte)pattern.Length;

}

for (int i = 0; i < pattern.Length - 1; i++)

{

shiftArray[pattern[i]] = (byte)(pattern.Length - i - 1);

}

int startIndex = 0;

int k = startIndex;

while (k <= txt.Length - pattern.Length) // цикл, пока есть место для элемента поиска

{

int j = pattern.Length - 1; // проверка, есть ли совпадение в текущей позиции

while (j >= 0 && pattern[j] == txt[k + j])

{

j--;

}

if (j < 0)

{

return k; // совпадение найдено

}

k += Math.Max(shiftArray[txt[k + j]] - pattern.Length + 1 + j, 1); // переход к следующему сравнению

}

return -1; // элемент поиска не найден

}

public static void Main(string[] args)

{

string inputText = GetTextFromFile("text.txt");

Console.WriteLine("Текст в файле: " + inputText);

Console.Write("Введите строку для поиска: ");

string key = Console.ReadLine();

Console.WriteLine("Индекс найденного слова" + BmSearch(key, inputText));

}

}

* 1. Контрольный тест:

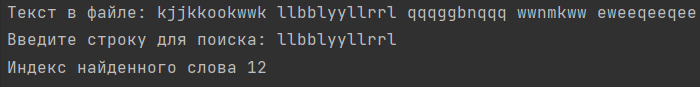


Рис. 5. Поиск слова 1

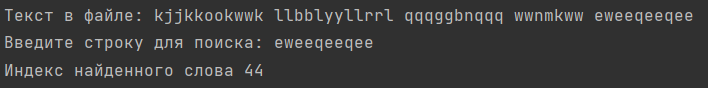


Рис. 6. Поиск слова 2

Вывод

В результате проделанной работы были получены навыки реализации и работы с поиском в линейных структурах, использованием деревьев в задачах поиска и задачах поиска в текста. По заданию созданы 3 программы, которые помогли закрепить полученные навыки.