МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Информационные системы»

**Лабораторная работа №3**по дисциплине «Основы алгоритмизации и программирования»

Выполнил:  
студент(ка) гр. ПИбд-12 Дозорова Алена

Ульяновск  
2021

Задание на лабораторную работу:

1. Реализовать структуру данных по вариантам:

Вариант 6: бинарную кучу

1. Реализовать алгоритм на графе:

Вариант 6: Обход в ширину (списки смежности)

1. Реализовать алгоритм на взвешенном графе

Вариант 6: Алгоритм поиска минимального остового дерева (матрица смежности

Краткое описание используемых алгоритмов и структур

В программе реализовано три класса под каждое задание. Бинарная куча реализована через массив, прописаны функции добавления элементов и возвращения баланса дереву. Используя класс Graf реализован обход в ширину через списки смежности, то есть, прописан словарь с кортежами, где указаны связанные вершины. Для поиска минимального остового дерева используется матрица смежности и массив, реализующий граф

Код лабораторной работы:

class Heap

{

int SIZE = 100;

int HeapSize;

public int[] h;

public Heap(int size) // конструктор кучи

{

HeapSize = size;

Random r = new Random();

h = new int[SIZE];

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

{

h[i] = r.Next(SIZE) + 1;

}

for (int i = HeapSize / 2; i >= 0; i--)

{

heapify(i);

}

}

public void addelem(int n) // добавление элемента кучи

{

int i, parent;

i = HeapSize-1;

h[i] = n;

parent = (i - 1) / 2;

while (parent >= 0 && i > 0)

{

if (h[i] > h[parent])

{

int temp = h[i];

h[i] = h[parent];

h[parent] = temp;

}

i = parent;

parent = (i - 1) / 2;

}

HeapSize++;

}

public void outHeap()// вывод элементов кучи в форме кучи

{

int i = 0;

int k = 1;

while (i < HeapSize)

{

while ((i < k) && (i < HeapSize))

{

Console.Write(h[i] + " ");

i++;

}

Console.Write("\n");

k = k \* 2 + 1;

}

}

public int getmax()// удаление вершины (максимального элемента)

{

int x;

x = h[0];

h[0] = h[HeapSize - 1];

HeapSize-=1;

heapify(0);

return x;

}

public void heapify(int i)// упорядочение кучи

{

int left, right;

int temp;

left = 2 \* i + 1;

right = 2 \* i + 2;

if (left < HeapSize)

{

if (h[i] < h[left])

{

temp = h[i];

h[i] = h[left];

h[left] = temp;

heapify(left);

}

}

if (right < HeapSize)

{

if (h[i] < h[right])

{

temp = h[i];

h[i] = h[right];

h[right] = temp;

heapify(right);

}

}

}

}

class BFS

{

public HashSet<T> BFS\_alg<T>(Graph<T> graph, T start)

{

Stopwatch stop = new Stopwatch();

stop.Start();

var visited = new HashSet<T>();

if (!graph.AdjacencyList.ContainsKey(start))

return visited;

var queue = new Queue<T>();

queue.Enqueue(start);

while (queue.Count > 0)

{

var vertex = queue.Dequeue();

if (visited.Contains(vertex))

continue;

visited.Add(vertex);

foreach (var neighbor in graph.AdjacencyList[vertex])

if (!visited.Contains(neighbor))

queue.Enqueue(neighbor);

}

stop.Stop();

Ostov.time(stop);

return visited;

}

}

public class Graph<T>

{

public Graph() { }

public Graph(IEnumerable<T> vertices, IEnumerable<Tuple<T, T>> edges)

{

foreach (var vertex in vertices)

AddVertex(vertex);

foreach (var edge in edges)

AddEdge(edge);

}

public Dictionary<T, HashSet<T>> AdjacencyList { get; } = new Dictionary<T, HashSet<T>>();

public void AddVertex(T vertex)

{

AdjacencyList[vertex] = new HashSet<T>();

}

public void AddEdge(Tuple<T, T> edge)

{

if (AdjacencyList.ContainsKey(edge.Item1) && AdjacencyList.ContainsKey(edge.Item2))

{

AdjacencyList[edge.Item1].Add(edge.Item2);

AdjacencyList[edge.Item2].Add(edge.Item1);

}

}

}

class Ostov

{

public static void time(Stopwatch stopWatch)

{

TimeSpan ts = stopWatch.Elapsed;

string elapsedTime = Convert.ToString(ts.Ticks);

// ts.Hours)+":"+ Convert.ToString(ts.Minutes)+":"+ Convert.ToString(ts.Seconds)+":"+ Convert.ToString(ts.Milliseconds);

Console.WriteLine("\n Время выполнения алгоритма: " + elapsedTime);

}

static int[,] G = new int[5,5] { {0, 9, 75, 0, 0}, {9, 0, 95, 19, 42}, {75, 95, 0, 51, 66}, {0, 19, 51, 0, 31}, {0, 42, 66, 31, 0}};

public static void Prim()

{

Stopwatch stop = new Stopwatch();

stop.Start();

int V = 5;

int no\_edge;

int [] selected = new int [5];

// set number of edge to 0

no\_edge = 0;

// the number of egde in minimum spanning tree will be

// always less than (V -1), where V is number of vertices in

//graph

// choose 0th vertex and make it true

selected[0] = 1;

int x; // row number

int y; // col number

// print for edge and weight

Console.WriteLine("Edge : Weight");

while (no\_edge < V - 1)

{

int min = 10000000;

x = 0;

y = 0;

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

{

if (selected[i]==1)

{

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

{

if (selected[j]==0 && G[i,j]!=null)

{ // not in selected and there is an edge

if (min > G[i,j])

{

min = G[i,j];

x = i;

y = j;

}

}

}

}

}

Console.WriteLine(Convert.ToString(x) + " - "+ Convert.ToString(y) + " : " + Convert.ToString(G[x,y]));

selected[y] = 1;

no\_edge++;

}

stop.Stop();

time(stop);

}

}

static void Main(string[] args)

{

int s = 0;

Console.WriteLine("Введите размер кучи: ");

s = Convert.ToInt32(Console.ReadLine());

Heap h = new Heap(s);

Console.WriteLine("Куча: ");

h.outHeap();

Console.WriteLine("Введите значение добавляемого элемента: ");

s = Convert.ToInt32(Console.ReadLine());

h.addelem(s);

Console.WriteLine("Обновленная куча: ");

h.outHeap();

Console.WriteLine("Максимум в куче: " + h.getmax());

var vertices = new[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

var edges = new[]{Tuple.Create(1,2), Tuple.Create(1,3),

Tuple.Create(2,4), Tuple.Create(3,5), Tuple.Create(3,6),

Tuple.Create(4,7), Tuple.Create(5,7), Tuple.Create(5,8),

Tuple.Create(5,6), Tuple.Create(8,9), Tuple.Create(9,10)};

var graph = new Graph<int>(vertices, edges);

var algorithms = new BFS();

Console.WriteLine("Depth First Search\n Введите вершину (от 1 до 10), с которой хотите начать обход: ");

s = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Путь в графе: "+string.Join(", ", algorithms.BFS\_alg(graph, s)));

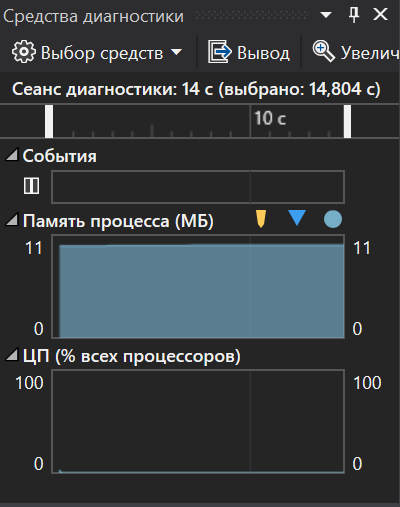
Console.WriteLine("\n5) Алгорит Прима для поиска остовного дерева");

Ostov.Prim();

Console.ReadKey();

}

Результаты тестирования:



Выводы:

Я повторила реализацию кучи, поиска в ширину и узнала об алгоритме Прима