

Национальный исследовательский университет ИТМО

Кафедра программных систем

**Лабораторная работа 3**

Выполнил: Орел

Даниил Максимович

Группа № K3221

Проверил: Иванов С.Е.

Санкт-Петербург

2020

**Цель работы:**

Программно реализовать основные алгоритмы на графах: задание графа с помощью матрицы смежности и инцидентности, поиск в ширину и глубину, алгоритм Дейкстры.

**Ход работы:**

1. Класс **Node** описывает узел графа – номер графа и список смежных с ним вершин:

class Node

{

public int Name { get; }

public List<Node> Children { get; }

public Node(int nodeName)

{

Name = nodeName;

Children = new List<Node>();

}

public Node AddChildren(Node node, bool isDirected = false)

{

Children.Add(node);

if (!isDirected)

{

node.Children.Add(this);

}

return this;

}

}

1. Класс **Graph** представляет собой коллекцию объектов Node и позволяет по этим спискам формировать матрицы смежности и инцидентности:

class Graph

{

private bool[] visited;

private int edges;

private int currEdge;

public List<Node> Nodes { get; set; }

public int[,] AdjMatrix { get; set; }

public int[,] IncMatrix { get; set; }

public Graph(List<Node> nodes)

{

Nodes = nodes;

}

public void CreateAdjMatrix()

{

AdjMatrix = new int[Nodes.Count + 1, Nodes.Count + 1];

foreach (var node in Nodes)

{

foreach (var childNode in node.Children)

{

AdjMatrix[node.Name, childNode.Name ] = 1;

}

}

PrintMatrix(AdjMatrix);

}

private int GetNumberOfEdges()

{

edges = 0;

visited = new bool[Nodes.Count + 1];

foreach (var node in Nodes)

{

visited[node.Name] = true;

foreach (var child in node.Children)

{

if (!visited[child.Name] || !child.Children.Contains(node))

edges++;

}

}

return edges;

}

public void CreateIncMatrix()

{

GetNumberOfEdges();

IncMatrix = new int[Nodes.Count + 1, edges + 1];

visited = new bool[Nodes.Count + 1];

currEdge = 1;

foreach (var node in Nodes)

{

visited[node.Name] = true;

foreach (var child in node.Children)

{

if (!visited[child.Name])

{

IncMatrix[node.Name, currEdge] = 1;

if (child.Children.Contains(node))

IncMatrix[child.Name, currEdge] = 1;

else IncMatrix[child.Name, currEdge] = -1;

}

else if (!child.Children.Contains(node)) IncMatrix[child.Name, currEdge] = -1;

else continue;

currEdge++;

}

}

PrintMatrix(IncMatrix);

}

private void PrintMatrix(int[,] matrix)

{

int rows = matrix.GetUpperBound(0) + 1;

int columns = matrix.Length / rows;

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

{

if (i != 0) Console.Write(i);

for (int j = 1; j < columns; j++)

{

if (i == 0) Console.Write($"\t{j}");

else Console.Write($"\t{matrix[i, j]}");

}

Console.WriteLine();

}

}

}

1. Класс **BFS** осуществляет поиск в ширину:

class BFS

{

private Node currNode;

private Queue<Node> queue;

private bool[] visited;

public void StartBFS(Node startNode, int nodes)

{

currNode = startNode;

queue = new Queue<Node>();

queue.Enqueue(startNode);

visited = new bool[nodes + 1];

visited[startNode.Name] = true;

while(queue.Count != 0)

{

currNode = queue.Dequeue();

Console.Write($"{currNode.Name} ");

foreach(var node in currNode.Children)

{

int to = node.Name;

if (!visited[to])

{

visited[to] = true;

queue.Enqueue(node);

}

}

}

Console.WriteLine();

}

}

1. Класс **DFS** осуществляет поиск в глубину графа:

class DFS

{

bool[] \_visited;

Queue<int> queue;

public void StartDFS(Node startNode, int nodes)

{

\_visited = new bool[nodes + 1];

\_visited[startNode.Name] = true;

queue = new Queue<int>();

queue.Enqueue(startNode.Name);

foreach(var node in startNode.Children.Where(node => !\_visited[node.Name]))

{

StartDFS(node);

}

PrintQueue();

}

private void StartDFS(Node startNode)

{

\_visited[startNode.Name] = true;

queue.Enqueue(startNode.Name);

foreach (var node in startNode.Children.Where(node => !\_visited[node.Name]))

{

StartDFS(node);

}

}

private void PrintQueue()

{

foreach (var nodeName in queue)

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

Console.WriteLine();

}

}

1. Класс **Dijkstra** реализует алгоритм Дейкстра. Вначале задается вес всех существующих связей, а потом находятся оптимальные пути от начального узла до все всех остальных узлов:

class Dijkstra

{

private List<Node> Nodes { get; set; }

private int[,] weightOfEdges;

private int weight;

private int[] path;

private bool[] visited;

private Node startNode, currNode;

private Queue<Node> queue;

public Dijkstra(List<Node> nodes)

{

Nodes = nodes;

}

private void SetWeightForEdges()

{

weightOfEdges = new int[Nodes.Count + 1, Nodes.Count + 1];

foreach (var currNode in Nodes)

{

foreach(var childNode in currNode.Children)

{

if(childNode.Name < currNode.Name

&& weightOfEdges[childNode.Name, currNode.Name] != 0)

{

weightOfEdges[currNode.Name, childNode.Name] = weightOfEdges[childNode.Name, currNode.Name];

}

else

{

Console.Write($"Вес связи от узла {currNode.Name} до узла {childNode.Name}: ");

weight = int.Parse(Console.ReadLine());

weightOfEdges[currNode.Name, childNode.Name] = weight;

}

}

}

}

private void PrintWeightOfEdges()

{

foreach (var currNode in Nodes)

{

foreach (var childNode in currNode.Children)

{

Console.WriteLine($"Вес связи от узла {currNode.Name} до узла {childNode.Name}: {weightOfEdges[currNode.Name, childNode.Name]}");

}

}

}

public void Start(Node node)

{

SetWeightForEdges();

startNode = node;

StartDijkstra();

PrintShortestPaths();

Console.WriteLine();

}

private void PrintShortestPaths()

{

for (int i = 1; i < path.Length; i++)

{

if (startNode.Name != i)

Console.WriteLine($"Наикратчаший путь из {startNode.Name} в {i}: {path[i]}");

}

}

private void StartDijkstra()

{

int to;

path = new int[Nodes.Count + 1];

visited = new bool[Nodes.Count + 1];

queue = new Queue<Node>();

currNode = startNode;

queue.Enqueue(currNode);

visited = new bool[Nodes.Count + 1];

visited[currNode.Name] = true;

while (queue.Count != 0)

{

currNode = queue.Dequeue();

foreach (var childNode in currNode.Children)

{

to = childNode.Name;

if (!visited[to])

{

if (path[to] == 0)

path[to] = path[currNode.Name] + weightOfEdges[currNode.Name, childNode.Name];

else if (path[to] >= path[currNode.Name] + weightOfEdges[currNode.Name, childNode.Name])

path[to] = path[currNode.Name] + weightOfEdges[currNode.Name, childNode.Name];

else path[to] += weightOfEdges[currNode.Name, childNode.Name];

visited[to] = true;

queue.Enqueue(childNode);

}

else

{

if (weightOfEdges[currNode.Name, childNode.Name] + path[currNode.Name] < path[to])

path[to] = path[currNode.Name] + weightOfEdges[currNode.Name, childNode.Name];

}

}

}

}

}

1. В классе Program сформированы все графы, данные в задании, и в нем демонстрируются все алгоритмы, реализованные в данной программе:

class Program

{

static void Main(string[] args)

{

Node n1 = new Node(1);

Node n2 = new Node(2);

Node n3 = new Node(3);

Node n4 = new Node(4);

Node n5 = new Node(5);

Node n6 = new Node(6);

Node n7 = new Node(7);

Node n8 = new Node(8);

Node n9 = new Node(9);

Node n10 = new Node(10);

List<Node> nodes1 = new List<Node>();

// Задание 1

n1.AddChildren(n2).AddChildren(n3);

n2.AddChildren(n4).AddChildren(n5);

n3.AddChildren(n4).AddChildren(n5);

nodes1.Add(n1);

nodes1.Add(n2);

nodes1.Add(n3);

nodes1.Add(n4);

nodes1.Add(n5);

Graph graph1 = new Graph(nodes1);

Console.WriteLine("Задание 1\n\nМатрица смежности");

graph1.CreateAdjMatrix();

Console.WriteLine("Матрица инцидентности");

graph1.CreateIncMatrix();

Console.WriteLine();

//Задание 2 - 3

foreach (var node in nodes1)

{

node.Children.Clear();

}

nodes1.Add(n6);

nodes1.Add(n7);

//Неориентированный граф

n3.AddChildren(n1).AddChildren(n2).AddChildren(n4).AddChildren(n5).AddChildren(n6);

n1.AddChildren(n2);

n7.AddChildren(n5).AddChildren(n6);

Console.WriteLine("Задание 2-3. Неориентированный граф\nМатрица смежности");

graph1.CreateAdjMatrix();

Console.WriteLine("Матрица инцидентности");

graph1.CreateIncMatrix();

Console.WriteLine();

//Ориентированный граф

foreach (var node in nodes1)

{

node.Children.Clear();

}

n1.AddChildren(n2, true).AddChildren(n3, true);

n2.AddChildren(n3, true);

n3.AddChildren(n4, true).AddChildren(n5, true).AddChildren(n6, true);

n5.AddChildren(n7, true);

n6.AddChildren(n7, true);

Console.WriteLine("Задание 2-3. Ориентированный граф\nМатрица смежности");

graph1.CreateAdjMatrix();

Console.WriteLine("Матрица инцидентности");

graph1.CreateIncMatrix();

Console.WriteLine();

//Тестирование поиска в глубину и ширину

foreach (var node in nodes1)

{

node.Children.Clear();

}

n1.AddChildren(n2, true).AddChildren(n3);

n2.AddChildren(n3, true).AddChildren(n4);

n3.AddChildren(n4, true);

graph1.Nodes = nodes1;

var dfs = new DFS();

var bfs = new BFS();

Console.WriteLine("Задание 4\n");

Console.WriteLine("Поиск в глубину: ");

dfs.StartDFS(n4, 4);

dfs.StartDFS(n3, 4);

dfs.StartDFS(n2, 4);

dfs.StartDFS(n1, 4);

Console.WriteLine();

Console.WriteLine("Поиск в ширину: ");

bfs.StartBFS(n4, 4);

bfs.StartBFS(n3, 4);

bfs.StartBFS(n2, 4);

bfs.StartBFS(n1, 4);

Console.WriteLine();

n1.Children.Clear();

n2.Children.Clear();

n3.Children.Clear();

n4.Children.Clear();

n1.AddChildren(n5).AddChildren(n2);

n3.AddChildren(n5).AddChildren(n2);

n4.AddChildren(n5).AddChildren(n2);

Console.WriteLine("Задание 5\n");

Console.Write("Поиск в глубину: ");

dfs.StartDFS(n5, 5);

Console.Write("Поиск в ширину: ");

bfs.StartBFS(n5, 5);

Console.WriteLine();

n1.Children.Clear();

n2.Children.Clear();

n3.Children.Clear();

n4.Children.Clear();

n5.Children.Clear();

n1.AddChildren(n2).AddChildren(n7);

n2.AddChildren(n3).AddChildren(n4);

n4.AddChildren(n5).AddChildren(n6);

n7.AddChildren(n8);

n8.AddChildren(n9).AddChildren(n10);

Console.WriteLine("Собственный пример графа: ");

Console.Write("Поиск в глубину: ");

dfs.StartDFS(n8, 10);

Console.Write("Поиск в ширину: ");

bfs.StartBFS(n8, 10);

Console.WriteLine();

n1.Children.Clear();

n2.Children.Clear();

n3.Children.Clear();

n4.Children.Clear();

n5.Children.Clear();

n6.Children.Clear();

n1.AddChildren(n2, true).AddChildren(n3).AddChildren(n5, true);

n2.AddChildren(n4);

n3.AddChildren(n4);

n4.AddChildren(n6, true);

n5.AddChildren(n6, true);

List<Node> graph = new List<Node>();

graph.Add(n1);

graph.Add(n2);

graph.Add(n3);

graph.Add(n4);

graph.Add(n5);

graph.Add(n6);

Dijkstra dijkstra = new Dijkstra(graph);

Console.WriteLine("Задание 6. Алгоритм Дейкстры:\n");

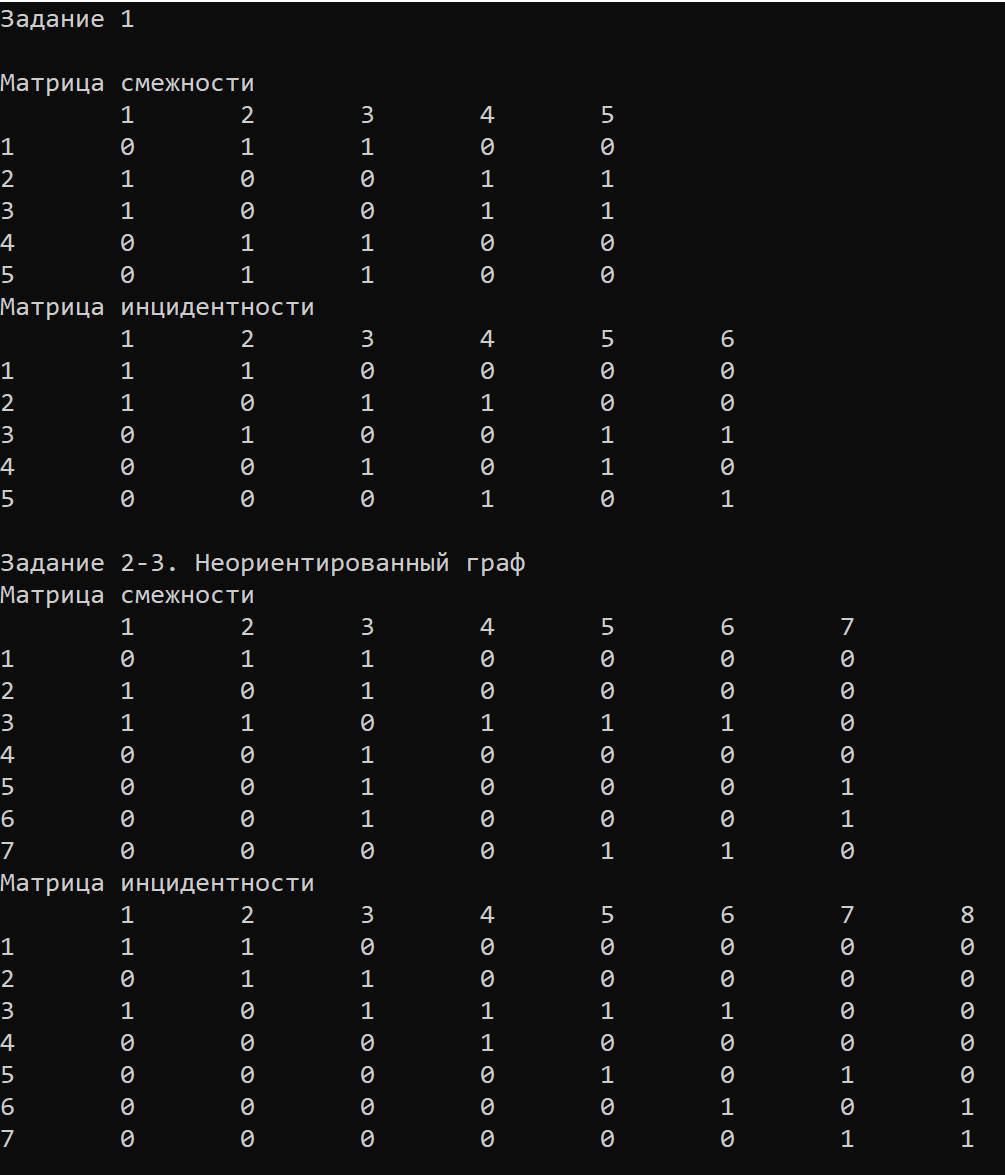
dijkstra.Start(n1);

Console.ReadKey();

}

}

1. Результат выполнения программы:

1.

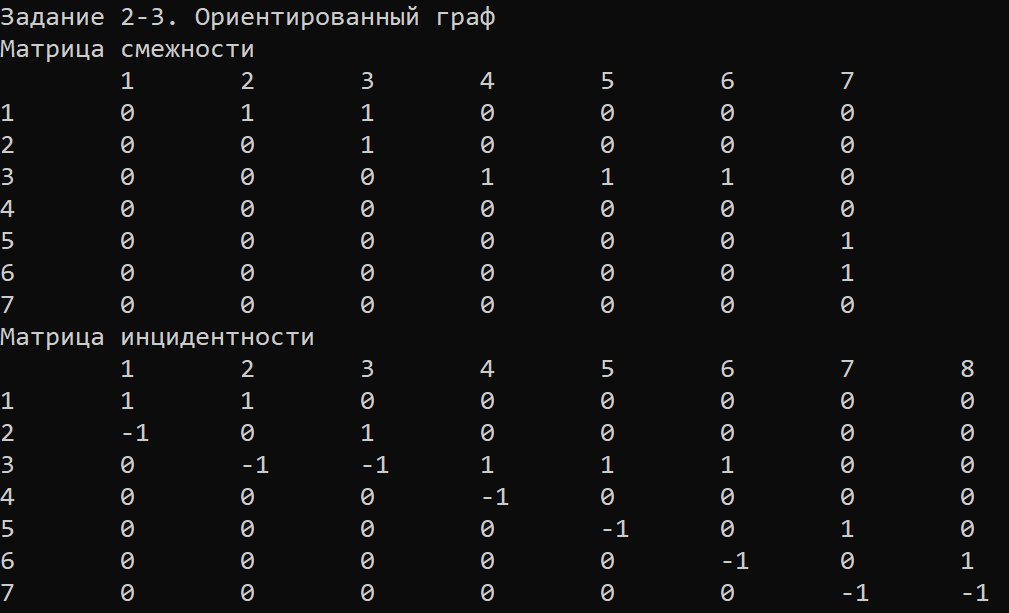
2.

Рисунок 1, 2 – Матрицы смежности и инцидентности графов, данных в заданиях 1-3

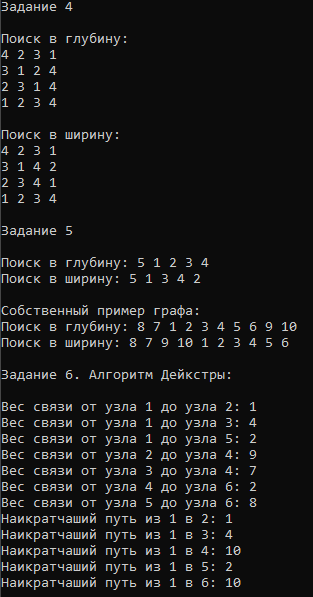


Рисунок 3 – Результат поиска в глубину и ширину в графах из заданий 4, 5, а также результат работы алгоритма Дейкстры

**Вывод:**

В ходе выполнения данной практической работы были реализованы основные алгоритмы для работы с графами: задание графа с помощью матрицы смежности и инцидентности, поиск в ширину и глубину, алгоритм Дейкстры.