Министерство науки и высшего образования Российской Федерации

Федеральное государственное образовательное учреждение

высшего образования

**ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

Институт математики и информационных систем

Факультет автоматики и вычислительной техники

Кафедра систем автоматизации управления

**Дисциплина: Алгоритмы и структуры данных**

**Отчёт по лабораторной работе №9**

**Тема: «Способы задания графов. Разновидности графов. Алгоритмы поиска в глубину, в ширину, Дейкстры»**

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил студент  группы ИТб 2302-02-20 |  | Ердяков Роман Александрович |  |
|  |  |
| Проверила |  | Кашина Елена Вячеславовна |  |

СОДЕРЖАНИЕ

[1 Задание 3](#_Toc200883426)

[2 Тестирование (задача 1) 4](#_Toc200883427)

[3 Программа на C# (задача 1): 5](#_Toc200883428)

[4 Тестирование (задача 2) 7](#_Toc200883429)

[5 Программа на C# (задача 2): 8](#_Toc200883430)

[6 Тестирование (задача 3) 10](#_Toc200883431)

[7 Программа на C# (задача 3): 11](#_Toc200883432)

[8 Задача 4 13](#_Toc200883433)

[9 Вывод 16](#_Toc200883434)

# Задание

Задача 1.

Написать программу, задающую неориентированный граф перечнем рёбер. Реализовать поиск в ширину.

Задача 2.

Написать программу, задающую неориентированный граф матрицей смежности. По матрице смежности графа построит матрицу инцидентности. Реализовать поиск в ширину.

Задача 3.

Написать программу, реализующую  алгоритм Дейкстры.

Задача 4.

Выпишите (или оформите в виде текстового документа) разновидности графов с картинками и определениями (описаниями).

# Тестирование (задача 1)

Примеры выполнения программы представлены на рисунке 1 и 2.

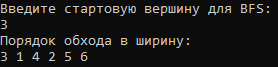


Рисунок 1 – Экранная форма программы

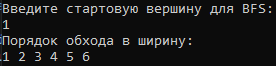


Рисунок 2 – Экранная форма программы

# Программа на C# (задача 1):

class Program

{

static void Main(string[] args)

{

List<(int, int)> edges = new List<(int, int)>

{

(1, 2),

(1, 3),

(2, 4),

(3, 4),

(4, 5),

(5, 6)

};

var graph = BuildGraph(edges);

Console.WriteLine("Введите стартовую вершину для BFS:");

int start = int.Parse(Console.ReadLine()!);

Console.WriteLine("Порядок обхода в ширину:");

BFS(graph, start);

}

static Dictionary<int, List<int>> BuildGraph(List<(int, int)> edges)

{

var graph = new Dictionary<int, List<int>>();

foreach (var (u, v) in edges)

{

if (!graph.ContainsKey(u))

graph[u] = new List<int>();

if (!graph.ContainsKey(v))

graph[v] = new List<int>();

graph[u].Add(v);

graph[v].Add(u);

}

return graph;

}

static void BFS(Dictionary<int, List<int>> graph, int start)

{

var visited = new HashSet<int>();

var queue = new Queue<int>();

visited.Add(start);

queue.Enqueue(start);

while (queue.Count > 0)

{

int current = queue.Dequeue();

Console.Write(current + " ");

foreach (var neighbor in graph[current])

{

if (!visited.Contains(neighbor))

{

visited.Add(neighbor);

queue.Enqueue(neighbor);

}

}

}

}

}

# Тестирование (задача 2)

Примеры выполнения программы представлены на рисунках 3, 4.

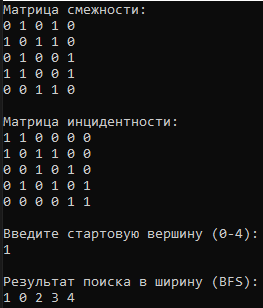


Рисунок 3 – Экранная форма программы

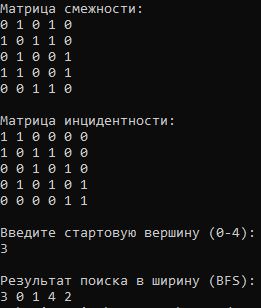


Рисунок 4 – Экранная форма программы

# Программа на C# (задача 2):

class Program

{

static void Main()

{

int[,] adjacencyMatrix = {

{0, 1, 0, 1, 0},

{1, 0, 1, 1, 0},

{0, 1, 0, 0, 1},

{1, 1, 0, 0, 1},

{0, 0, 1, 1, 0}

};

Console.WriteLine("Матрица смежности:");

PrintMatrix(adjacencyMatrix);

int[,] incidenceMatrix = BuildIncidenceMatrix(adjacencyMatrix);

Console.WriteLine("\nМатрица инцидентности:");

PrintMatrix(incidenceMatrix);

Console.WriteLine("\nВведите стартовую вершину (0-" + (adjacencyMatrix.GetLength(0) - 1) + "):");

int start = int.Parse(Console.ReadLine()!);

Console.WriteLine("\nРезультат поиска в ширину (BFS):");

BFS(adjacencyMatrix, start);

}

static int[,] BuildIncidenceMatrix(int[,] adjacency)

{

int n = adjacency.GetLength(0);

var edges = new List<(int, int)>();

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

for (int j = i + 1; j < n; j++)

if (adjacency[i, j] == 1)

edges.Add((i, j));

int m = edges.Count;

int[,] incidence = new int[n, m];

for (int e = 0; e < m; e++)

{

var (u, v) = edges[e];

incidence[u, e] = 1;

incidence[v, e] = 1;

}

return incidence;

}

static void BFS(int[,] adjacency, int start)

{

int n = adjacency.GetLength(0);

bool[] visited = new bool[n];

Queue<int> queue = new Queue<int>();

visited[start] = true;

queue.Enqueue(start);

while (queue.Count > 0)

{

int current = queue.Dequeue();

Console.Write(current + " ");

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

{

if (adjacency[current, i] == 1 && !visited[i])

{

visited[i] = true;

queue.Enqueue(i);

}

}

}

}

static void PrintMatrix(int[,] matrix)

{

int rows = matrix.GetLength(0);

int cols = matrix.GetLength(1);

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

{

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

Console.Write(matrix[i, j] + " ");

Console.WriteLine();

}

}

}

# Тестирование (задача 3)

Примеры выполнения программы представлены на рисунках 5, 6.

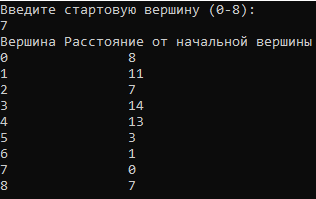


Рисунок 6 – Экранная форма программы

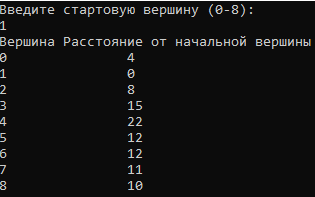


Рисунок 7 – Экранная форма программы

# Программа на C# (задача 3):

class Program

{

static void Main()

{

// Матрица смежности графа (0 - отсутствие ребра, положительное число - вес ребра)

int[,] graph = {

{ 0, 4, 0, 0, 0, 0, 0, 8, 0 },

{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },

{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },

{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },

{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },

{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },

{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },

{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },

{ 0, 0, 2, 0, 0, 0, 6, 7, 0 }

};

Console.WriteLine("Введите стартовую вершину (0-8):");

int start = int.Parse(Console.ReadLine()!);

Dijkstra(graph, start);

}

static void Dijkstra(int[,] graph, int start)

{

int n = graph.GetLength(0);

int[] dist = new int[n];

bool[] visited = new bool[n];

const int INF = int.MaxValue;

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

dist[i] = INF;

dist[start] = 0;

for (int count = 0; count < n - 1; count++)

{

int u = MinDistance(dist, visited);

if (u == -1) break;

visited[u] = true;

for (int v = 0; v < n; v++)

{

if (!visited[v] && graph[u, v] != 0 && dist[u] != INF &&

dist[u] + graph[u, v] < dist[v])

{

dist[v] = dist[u] + graph[u, v];

}

}

}

Console.WriteLine("Вершина\tРасстояние от начальной вершины");

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

{

if (dist[i] == INF)

Console.WriteLine($"{i}\t\tНет пути");

else

Console.WriteLine($"{i}\t\t{dist[i]}");

}

}

static int MinDistance(int[] dist, bool[] visited)

{

int min = int.MaxValue;

int minIndex = -1;

for (int v = 0; v < dist.Length; v++)

{

if (!visited[v] && dist[v] <= min)

{

min = dist[v];

minIndex = v;

}

}

return minIndex;

}

}

# Задача 4

Выпишите (или оформите в виде текстового документа) разновидности графов с картинками и определениями (описаниями).

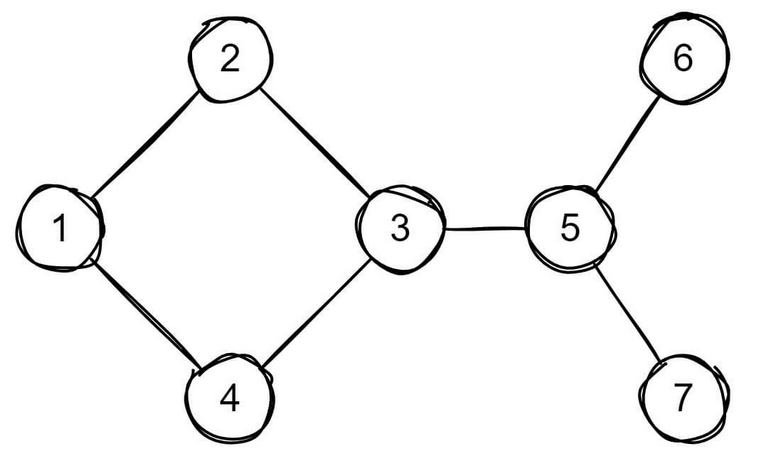
**Неориентированный граф** - граф, в котором между их узлами нет заданных направлений. Поэтому ребро из узла A в B будет идентично ребру из B в A: 

Рисунок 8. Неориентированный граф

**Направленные графы** – граф в котором есть ориентация или направление между узлами. Получается, если есть ребро из узла A в B, вы можете двигаться только из A в B, так как другое направление не указано.

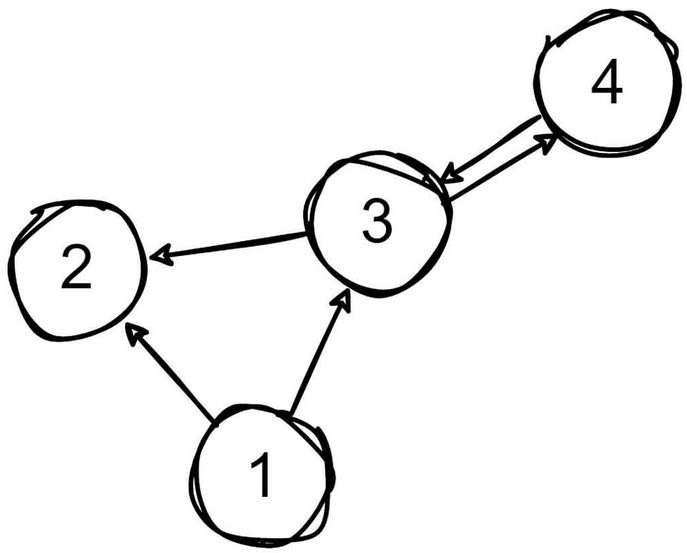


Рисунок 9. Направленный граф

**Взвешенный графов – граф у которого** есть ребра, которым присвоены **весы** — некоторые числа. Например, они могут отражать стоимость перевозки груза, расстояние, которое нужно преодолеть.

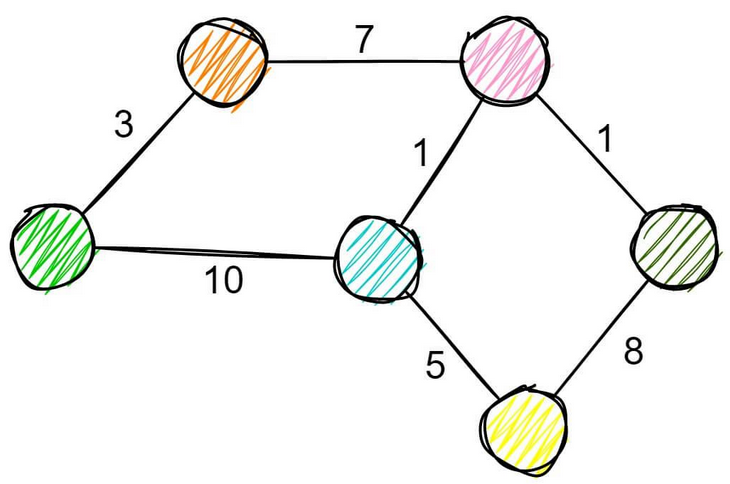


Рисунок 10. Взвешенный граф

**Мул**ь**тиграф** - граф, в котором разрешается присутствие кратных рёбер (их также называют «параллельными»), то есть рёбер, имеющих те же самые конечные вершины.

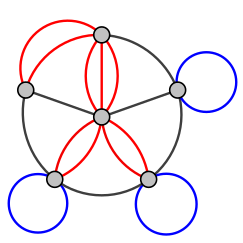
. 

Рисунок 11. Мультиграф граф

**Петля** – ребро, соединяющее вершину с ней же самой.

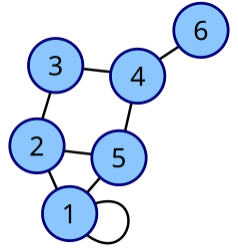


Рисунок 12. Петля

**Цикл** — граф, состоящий из единственного цикла, или, другими словами, некоторого числа вершин, соединённых замкнутой цепью

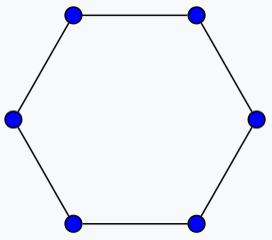
****

Рисунок 9. Цикл

**Дерево** — связный ациклический граф. Связность означает наличие маршрута между любой парой вершин, ацикличность — отсутствие циклов

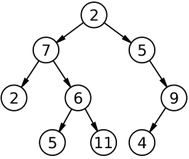


Рисунок 13. Дерево

# Вывод

В ходе выполнения лабораторных работ были успешно реализованы ключевые алгоритмы теории графов: поиск в глубину и ширину для анализа структуры графов, а также алгоритм Дейкстры для нахождения кратчайших путей во взвешенных графах