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 №11**

**Тема: «Нахождение максимального потока в графе. Нахождение Эйлерова цикла в графе. Нахождение гамильтонова цикла в графе»**

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

СОДЕРЖАНИЕ

[1 Задание 3](#_Toc200883465)

[2 Тестирование (задача 1) 4](#_Toc200883466)

[3 Программа на C# (задача 1): 5](#_Toc200883467)

[4 Тестирование (задача 2) 8](#_Toc200883468)

[5 Программа на C# (задача 2): 9](#_Toc200883469)

[6 Тестирование (задача 3) 12](#_Toc200883470)

[7 Программа на C# (задача 3): 13](#_Toc200883471)

[8 Вывод 15](#_Toc200883472)

# Задание

Задача 1.

Написать программу, которая реализует нахождение максимального потока. алгоритмом Эдмондса-Карпа.

Задача 2.

Нахождение Эйлерова цикла в графе.

Задача 3.

Нахождение гамильтонова цикла в графе.

# Тестирование (задача 1)

Примеры выполнения программы представлены на рисунке 1.

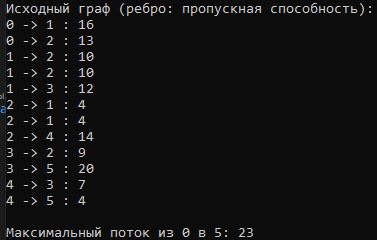


Рисунок 1 – Экранная форма программы

# Программа на C# (задача 1):

class EdmondsKarp

{

private int vertices;

private int[,] capacity;

private int[,] originalCapacity;

private List<int>[] adj;

public EdmondsKarp(int n)

{

vertices = n;

capacity = new int[n, n];

originalCapacity = new int[n, n];

adj = new List<int>[n];

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

adj[i] = new List<int>();

}

public void AddEdge(int from, int to, int cap)

{

capacity[from, to] = cap;

originalCapacity[from, to] = cap;

adj[from].Add(to);

adj[to].Add(from);

}

public void PrintGraph()

{

Console.WriteLine("Исходный граф (ребро: пропускная способность):");

for (int u = 0; u < vertices; u++)

{

foreach (int v in adj[u])

{

if (originalCapacity[u, v] > 0)

Console.WriteLine($"{u} -> {v} : {originalCapacity[u, v]}");

}

}

}

private int BFS(int s, int t, int[] parent)

{

Array.Fill(parent, -1);

parent[s] = -2;

Queue<(int vertex, int flow)> queue = new Queue<(int, int)>();

queue.Enqueue((s, int.MaxValue));

while (queue.Count > 0)

{

var (current, flow) = queue.Dequeue();

foreach (int next in adj[current])

{

if (parent[next] == -1 && capacity[current, next] > 0)

{

parent[next] = current;

int new\_flow = Math.Min(flow, capacity[current, next]);

if (next == t)

return new\_flow;

queue.Enqueue((next, new\_flow));

}

}

}

return 0;

}

public int MaxFlow(int s, int t)

{

int flow = 0;

int[] parent = new int[vertices];

int new\_flow;

while ((new\_flow = BFS(s, t, parent)) != 0)

{

flow += new\_flow;

int current = t;

while (current != s)

{

int prev = parent[current];

capacity[prev, current] -= new\_flow;

capacity[current, prev] += new\_flow;

current = prev;

}

}

return flow;

}

}

class Program

{

static void Main()

{

// Граф из 6 вершин

int n = 6;

EdmondsKarp graph = new EdmondsKarp(n);

graph.AddEdge(0, 1, 16);

graph.AddEdge(0, 2, 13);

graph.AddEdge(1, 2, 10);

graph.AddEdge(2, 1, 4);

graph.AddEdge(1, 3, 12);

graph.AddEdge(3, 2, 9);

graph.AddEdge(2, 4, 14);

graph.AddEdge(4, 3, 7);

graph.AddEdge(3, 5, 20);

graph.AddEdge(4, 5, 4);

int source = 0;

int sink = 5;

graph.PrintGraph();

int maxFlow = graph.MaxFlow(source, sink);

Console.WriteLine($"\nМаксимальный поток из {source} в {sink}: {maxFlow}");

}

}

# Тестирование (задача 2)

Примеры выполнения программы представлены на рисунке 2.

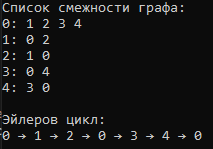


Рисунок 2 – Экранная форма программы

# Программа на C# (задача 2):

class Graph

{

private Dictionary<int, List<int>> adjacencyList = new Dictionary<int, List<int>>();

public void AddEdge(int u, int v)

{

if (!adjacencyList.ContainsKey(u))

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

if (!adjacencyList.ContainsKey(v))

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

adjacencyList[u].Add(v);

adjacencyList[v].Add(u);

}

public void PrintGraph()

{

Console.WriteLine("Список смежности графа:");

foreach (var pair in adjacencyList)

{

Console.Write($"{pair.Key}: ");

foreach (var neighbor in pair.Value)

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

Console.WriteLine();

}

}

public bool HasEulerianCycle()

{

foreach (var kvp in adjacencyList)

{

if (kvp.Value.Count % 2 != 0)

return false;

}

return true;

}

public List<int> FindEulerianCycle()

{

var stack = new Stack<int>();

var cycle = new List<int>();

// Глубокая копия графа

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

foreach (var kvp in adjacencyList)

localGraph[kvp.Key] = new List<int>(kvp.Value);

int current = GetStartVertex();

stack.Push(current);

while (stack.Count > 0)

{

if (localGraph[current].Count > 0)

{

stack.Push(current);

int next = localGraph[current][0];

localGraph[current].Remove(next);

localGraph[next].Remove(current);

current = next;

}

else

{

cycle.Add(current);

current = stack.Pop();

}

}

cycle.Reverse();

return cycle;

}

private int GetStartVertex()

{

foreach (var kvp in adjacencyList)

return kvp.Key;

return 0;

}

}

class Program

{

static void Main()

{

var graph = new Graph();

graph.AddEdge(0, 1);

graph.AddEdge(1, 2);

graph.AddEdge(2, 0);

graph.AddEdge(0, 3);

graph.AddEdge(3, 4);

graph.AddEdge(4, 0);

graph.PrintGraph();

if (!graph.HasEulerianCycle())

{

Console.WriteLine("\nЭйлеров цикл невозможен: не все вершины имеют чётную степень.");

}

else

{

var cycle = graph.FindEulerianCycle();

Console.WriteLine("\nЭйлеров цикл:");

Console.WriteLine(string.Join(" → ", cycle));

}

}

}

# Тестирование (задача 3)

Примеры выполнения программы представлены на рисунках 3.

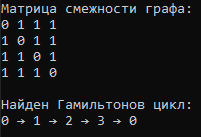


Рисунок 3 – Экранная форма программы

# Программа на C# (задача 3):

class Graph

{

private int V;

private bool[,] adj;

public Graph(int vertices)

{

V = vertices;

adj = new bool[V, V];

}

public void AddEdge(int u, int v)

{

adj[u, v] = true;

adj[v, u] = true;

}

public void PrintGraph()

{

Console.WriteLine("Матрица смежности графа:");

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

{

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

Console.Write((adj[i, j] ? "1 " : "0 "));

Console.WriteLine();

}

}

public bool FindHamiltonianCycle()

{

var path = new List<int> { 0 };

var visited = new bool[V];

visited[0] = true;

if (HamiltonianDFS(0, path, visited))

{

Console.WriteLine("\nНайден Гамильтонов цикл:");

Console.WriteLine(string.Join(" → ", path) + " → " + path[0]);

return true;

}

else

{

Console.WriteLine("\nГамильтонов цикл не найден.");

return false;

}

}

private bool HamiltonianDFS(int current, List<int> path, bool[] visited)

{

if (path.Count == V)

return adj[current, path[0]];

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

{

if (adj[current, next] && !visited[next])

{

visited[next] = true;

path.Add(next);

if (HamiltonianDFS(next, path, visited))

return true;

visited[next] = false;

path.RemoveAt(path.Count - 1);

}

}

return false;

}

}

class Program

{

static void Main()

{

Graph graph = new Graph(4);

graph.AddEdge(0, 1);

graph.AddEdge(1, 2);

graph.AddEdge(2, 3);

graph.AddEdge(3, 0);

graph.AddEdge(1, 3);

graph.AddEdge(0, 2);

graph.PrintGraph();

graph.FindHamiltonianCycle();

}

}

# Вывод

В ходе лабораторной работы реализованы алгоритмы для работы с графами: поиск максимального потока, нахождение Эйлерова и Гамильтонова циклов. Эти задачи помогли разобраться с различными типами обходов, условиями существования циклов и применением классических алгоритмов в графовой теории.