Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по лабораторной работе №4

Алгоритм Форда-Фалкерсона

Выполнил:

студент гр. 953504

Кондрашов И.Д.

Проверил:

кандидат физико-математических наук

Дугинов О.И.

Минск 2022

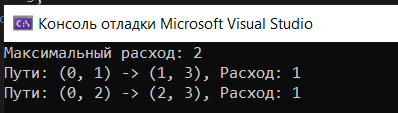
1. **Пример работы программы**

Входные данные:

Text

Description automatically generated

Ответ:



1. **Исходный код программы (C#)**

Реализованные методы:

1. BFS(Matrix<double> graph, int source, int sink, int[] path) – метод, реализующий поиск в ширину
2. FordFalkerson(Matrix<double> graph, int source, int sink)– метод, выполняющий задание лабораторной(алгоритм Форда-Фалкерсона)

using MathNet.Numerics.LinearAlgebra;

int source = 0;

int sink = 3;

Matrix<double> graph = Matrix<double>.Build.DenseOfArray(new double[,] { { 0, 1, 1, 0 },

{ 0, 0, 2, 2 },

{ 0, 0, 0, 3 },

{ 0, 0, 0, 0 } });

var (maxFlow, paths) = FordFalkerson(graph, source, sink);

Console.WriteLine($"Максимальный расход: {maxFlow}");

foreach (var path in paths)

{

Console.WriteLine($"Пути: {path.Item1}, Расход: {path.Item2}");

}

bool BFS(Matrix<double> graph, int source, int sink, int[] path)

{

bool[] visited = new bool[graph.RowCount];

Array.Fill(visited, false);

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

queue.Enqueue(source);

visited[source] = true;

path[source] = -1;

while (queue.Count != 0)

{

int i = queue.Dequeue();

for (int j = 0; j < graph.RowCount; j++)

{

if (!visited[j] && graph[i, j] != 0)

{

path[j] = i;

if (j == sink)

{

return true;

}

else

{

queue.Enqueue(j);

visited[j] = true;

}

}

}

}

return false;

}

Tuple<double, List<Tuple<string, double>>> FordFalkerson(Matrix<double> graph, int source, int sink)

{

Matrix<double> residualGraph = Matrix<double>.Build.DenseOfMatrix(graph);

int[] path = new int[graph.RowCount];

List<Tuple<string, double>> paths = new List<Tuple<string, double>>();

double maxFlow = 0.0;

while (BFS(residualGraph, source, sink, path))

{

double tetta = Double.PositiveInfinity;

List<string> reversedPath = new List<string>();

for (int j = sink; j != source; j = path[j])

{

var i = path[j];

reversedPath.Add($"({i}, {j})");

tetta = Math.Min(tetta, residualGraph[i, j]);

residualGraph[i, j] -= tetta;

residualGraph[j, i] += tetta;

}

reversedPath.Reverse();

paths.Add(Tuple.Create(String.Join(" -> ", reversedPath), tetta));

maxFlow += tetta;

}

return Tuple.Create(maxFlow, paths);

}