Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский химико-технологический университет имени Д.И. Менделеева»

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5

Выполнил студент группы КС-30 Джотян Давид Араикович

Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/DzhotyanDA\_30/tree/main/lab5\_algo

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: 23.03.2025

Оглавление

[Описание задачи. 2](#_Toc63548272)

[Описание метода/модели. 2](#_Toc63548273)

[Выполнение задачи. 2](#_Toc63548274)

[Заключение. 2](#_Toc63548275)

# Описание задачи.

1. Создайте взвешенный граф, состоящий из [10, 20, 50, 100] вершин.

* Каждая вершина графа связана со случайным количеством вершин, минимум с [3, 4, 10, 20].
* Веса ребер задаются случайным значением от 1 до 20.
* Каждая вершина графа должна быть доступна, т.е. до каждой вершины графа должен обязательно существовать путь до каждой вершины, не обязательно прямой.

1. Выведите, получившийся граф в виде матрицы смежности.
2. Для каждого графа требуется провести серию из 5 - 10 тестов, в зависимости от времени затраченного на выполнение одного теста., необходимо: Найти кратчайшие пути между всеми вершинами графа и их длину с помощью алгоритма Флойда — Уоршелла.
3. В рамках каждого теста, необходимо замерить потребовавшееся время на выполнение задания из пункта 3 для каждого набора вершин. По окончанию всех тестов необходимо построить график используя полученные замеры времени, где на ось абсцисс (Х) нанести N – количество вершин, а на ось ординат(Y) - значения затраченного времени.

# Описание метода/модели.

# Алгоритм Флойда – Уоршэлла

Поиск-всех-кратчайших-путей-Флойда (Матрица смежности)

Инициируем 3 счетчика, 2 счетчика для матрицы и 1 для номера промежуточной вершины.

Итерируем по счетчику промежуточных вершин до тех пор пока не кончатся вершины

Итерируем по первому счетчику вершин, до тех пор пока не кончатся вершины

Итерируем по второму счетчику вершин, до тех пор пока не кончатся вершины

Считаем длину пути используя промежуточную вершину как узловую точку, т.е. от вершины по первому счетчику до узловой плюс от узловой до вершины по второму счетчику.

Проверяем будет ли новый путь меньше предыдущего уже найденного через другую вершину, и записываем новое значение если оно подходит.

Асимптотическая сложность равно O(n^3), что равно n вызову алгоритмов Дейкстры.

Алгоритм можно использовать для определения существования путей между двумя точками на графе.

# Выполнение задачи.

Файл Vertex.cs

namespace lab5\_algo

{

internal class Vertex

{

public int Number { get; }

public Vertex(int number)

{

Number = number;

}

public override string ToString()

{

return Number.ToString();

}

}

}

Файл Edge.cs

namespace lab5\_algo

{

internal class Edge

{

public Vertex Start { get; }

public Vertex End { get; }

public int Weight { get; }

public Edge(Vertex start, Vertex end, int weight = 1)

{

Start = start;

End = end;

Weight = weight;

}

public override string ToString()

{

return $"{Start} -> {End}, Weight = {Weight}";

}

}

}

Файл Graph.cs

using System.Drawing;

namespace lab5\_algo

{

internal class Graph

{

public List<Vertex> Vertices = new List<Vertex>();

public List<Edge> Edges = new List<Edge>();

private bool Oriented;

public int VertexCount => Vertices.Count;

public int EdgeCount => Edges.Count;

List<List<int>> AdjacencyMatrix = new List<List<int>>();

public Graph(bool oriented = false)

{

Oriented = oriented;

}

public void AddVertex(Vertex vertex)

{

Vertices.Add(vertex);

}

public void AddEdge(Edge edge)

{

Edges.Add(edge);

UpdateAdjacencyMatrix(edge);

}

/// <summary>

/// Алгоритм Флойда-Уоршелла

/// </summary>

public int[,] FloydWarshall()

{

int size = AdjacencyMatrix.Count;

int[,] dist = new int[size, size];

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

{

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

{

if (i == j)

dist[i, j] = 0;

else if (AdjacencyMatrix[i][j] == 0)

dist[i, j] = int.MaxValue;

else

dist[i, j] = AdjacencyMatrix[i][j];

}

}

for (int k = 0; k < size; k++)

{

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

{

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

{

if (dist[i, k] != int.MaxValue && dist[k, j] != int.MaxValue)

{

dist[i, j] = Math.Min(dist[i, j], dist[i, k] + dist[k, j]);

}

}

}

}

return dist;

}

/// <summary>

/// Обновление матрицы смежности

/// </summary>

/// <param name="edge"></param>

private void UpdateAdjacencyMatrix(Edge edge)

{

var row = edge.Start.Number - 1;

var column = edge.End.Number - 1;

int size = Math.Max(row, column) + 1;

while (AdjacencyMatrix.Count < size)

{

AdjacencyMatrix.Add(new List<int>(new int[size]));

}

foreach (var rowList in AdjacencyMatrix)

{

while (rowList.Count < size)

{

rowList.Add(0);

}

}

AdjacencyMatrix[row][column] = edge.Weight;

if (!Oriented)

{

AdjacencyMatrix[column][row] = edge.Weight;

}

}

/// <summary>

/// Получение матрицы смежности

/// </summary>

/// <returns></returns>

public List<List<int>> GetAdjacencyMatrix() => AdjacencyMatrix;

/// <summary>

/// Вывод ребер графа

/// </summary>

public void PrintEdges()

{

Console.WriteLine($"Граф (Вершин: {Vertices.Count}, Ребер: {Edges.Count}, Ориентированный: {Oriented})");

foreach (var edge in Edges)

{

Console.WriteLine($"{edge.Start} -> {edge.End}");

}

Console.WriteLine();

}

}

}

Файл GraphGenerator.cs

using System.ComponentModel.DataAnnotations;

using System.Drawing;

namespace lab5\_algo

{

internal class GraphGenerator

{

const int MIN\_WEIGHT = 1;

const int MAX\_WEIGHT = 20;

const int EXTRA\_NEUGHBOURS = 2;

private Random random = new Random();

public Graph GenerateGraph(int verticesCount, int minEdgesCount, bool oriented)

{

Graph graph = new Graph(oriented);

Random random = new Random();

int minNeughboursCount = random.Next(minEdgesCount, minEdgesCount + EXTRA\_NEUGHBOURS);

Console.WriteLine($"Минимальное количество вершин, с которым связана вершина: {minNeughboursCount}");

List<Vertex> vertices = new List<Vertex>();

for (int i = 1; i <= verticesCount; i++)

{

var vertex = new Vertex(i);

vertices.Add(vertex);

graph.AddVertex(vertex);

}

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

{

Vertex start = vertices[i];

List<Vertex> notNeughbours = vertices.Where(v => v != start).ToList();

int edgesToAdd = Math.Min(minNeughboursCount, notNeughbours.Count);

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

{

if (notNeughbours.Count == 0)

break;

Vertex end = notNeughbours[random.Next(notNeughbours.Count)];

notNeughbours.Remove(end);

int weight = random.Next(MIN\_WEIGHT, MAX\_WEIGHT + 1);

Edge edge = new Edge(start, end, weight);

graph.AddEdge(edge);

if (!oriented)

{

Edge reverseEdge = new Edge(end, start, weight);

graph.AddEdge(reverseEdge);

}

}

}

return graph;

}

}

}

Файл Program.cs

using System.Diagnostics;

namespace lab5\_algo

{

class Program

{

static void Main()

{

Stopwatch stopwatch = new Stopwatch();

GraphGenerator graphGenerator = new GraphGenerator();

const int MAX\_ITER = 7;

int[] verticesCount = { 10, 20, 50, 100 };

int[] minEdgesCount = { 3, 4, 10, 20 };

long[] microsecFWAverage = new long[verticesCount.Length];

for (int i = 0; i < verticesCount.Length; i++)

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine($"Граф {i + 1}");

Console.ResetColor();

long[] microsecFW = new long[MAX\_ITER];

int iter = 0;

while (iter < MAX\_ITER)

{

for (int j = 0; j < minEdgesCount.Length; j++)

{

Graph graph = graphGenerator.GenerateGraph(verticesCount[i], minEdgesCount[j], false);

PrintAdjacencyMatrix(graph);

stopwatch.Start();

var matrix = graph.FloydWarshall();

stopwatch.Stop();

microsecFW[j] = (long)stopwatch.Elapsed.TotalMicroseconds;

stopwatch.Reset();

PrintFWMatrix(graph, matrix);

}

iter++;

}

Console.WriteLine($"Ср. время: {microsecFW.Average()} мкс");

microsecFWAverage[i] = (long)microsecFW.Average();

}

WriteToFile(microsecFWAverage);

}

static void PrintAdjacencyMatrix(Graph graph)

{

var matrix = graph.GetAdjacencyMatrix();

int size = matrix.Count;

Console.ForegroundColor = ConsoleColor.Yellow;

Console.WriteLine("Матрица смежности");

Console.ResetColor();

Console.Write(" │");

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

Console.Write($" {i + 1,2} ");

Console.WriteLine("\n───┼" + new string('─', 4 \* size + 1));

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

{

Console.Write($"{i + 1,2} │");

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

{

int value = matrix[i][j];

if (value == 0)

Console.Write($" INF");

else

Console.Write($" {value,2} ");

}

Console.WriteLine();

}

Console.WriteLine();

}

static void PrintFWMatrix(Graph graph, int[,] dist)

{

int size = graph.GetAdjacencyMatrix().Count;

Console.ForegroundColor = ConsoleColor.Yellow;

Console.WriteLine("Матрица кратчайших путей: Флойд-Уоршэлл");

Console.ResetColor();

Console.Write(" │");

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

Console.Write($" {i + 1,2} ");

Console.WriteLine("\n───┼" + new string('─', 4 \* size + 1));

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

{

Console.Write($"{i + 1,2} │");

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

{

if (dist[i, j] == int.MaxValue)

Console.Write(" -");

else

Console.Write($" {dist[i, j],2} ");

}

Console.WriteLine();

}

}

public static void WriteToFile(long[] microsecondsFW)

{

var path = Path.Combine(Directory.GetCurrentDirectory(), "..", "..", "..", "Measurements.txt");

using (StreamWriter sw = new StreamWriter(path, false, System.Text.Encoding.Default))

{

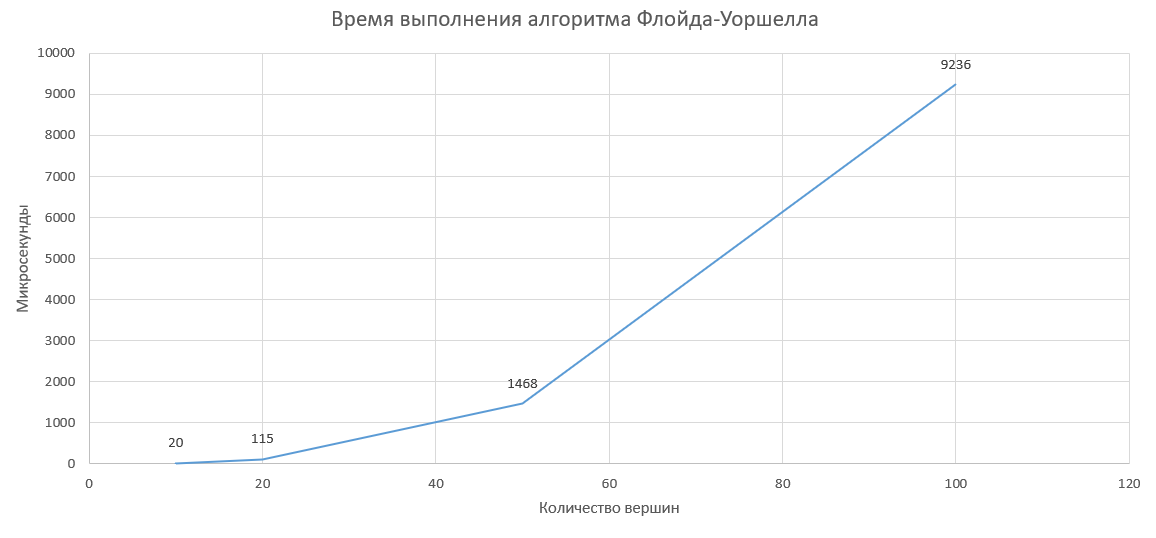
sw.WriteLine($"Время выполнения алгоритма Флойда-Уоршелла: {string.Join(" ", microsecondsFW)}");

}

}

}

}



# 

# Заключение.

Т.к. сложность по времени составляет О(N3), то при увеличении числа вершин время выполнения возрастает очень быстро.

Данный алгоритм подойдет для малых и средних графов (до 1000 вершин).

Для больших графов (>1000 вершин) становится слишком медленным. В таких случаях лучше использовать алгоритм Дейкстры, где сложность O(N2).