|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
|  | | |
| Домашнее задание № 2 | | |
| по дисциплине «Методы построения и анализа алгоритмов» | | |
|  | | |
| **Сортировка и поиск** | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМИ-81 |
| Варианты: | 7 |
| Студент: | Кондратьев Игорь |
|  |  |
|  |  |
| Преподаватель: | Георгий Анатольевич Щукин |
|  |  |
|  | | |
| Новосибирск | | |
| 2019 | | |

1. **Постановка задачи**  
   Реализовать класс “Граф” и его базовые операции, такие как вставка вершины или ребра/дуги, получение списка вершин смежных с заданной вершиной и т.д. Если в задании указан взвешенный и/или ориентированный граф, учесть это при проектировании (добавить возможность указания веса ребра и т.д.)  
     
   Дано взвешенный ориентированный граф. Необходимо найти кратчайший путь между всеми парами вершин с помощью алгоритма Флойда-Уоршелла
2. **Описание использованных алгоритмов**  
     
   **Алгоритм Флойда — Уоршелла** — [динамический](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5" \o "Динамическое программирование) алгоритм для нахождения кратчайших расстояний между всеми вершинами [взвешенного](https://ru.wikipedia.org/wiki/%D0%92%D0%B7%D0%B2%D0%B5%D1%88%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84" \o "Взвешенный граф) [ориентированного графа](https://ru.wikipedia.org/wiki/%D0%9E%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84" \o "Ориентированный граф)

Псевдокод:

for k = 1 to n

for i = 1 to n

for j = 1 to n

W[i][j] = min(W[i][j], W[i][k] + W[k][j])

1. **Текст реализации алгоритмов на языке программирования**

Класс Граф:  
 public class Graph

{

List<int> nodeArray;

List<int> edgesFrom;

List<int> edgesTo;

List<int> edgesWeight;

public Graph()

{

nodeArray = new List<int>();

edgesFrom = new List<int>();

edgesTo = new List<int>();

edgesWeight = new List<int>();

}

public void AddEdge(int nodeFrom, int nodeTo, int weight)

{

if(weight <= 0)

{

throw new Exception("NonPositiveWeight = " + weight);

}

if(!IsExistNode(nodeFrom) || !IsExistNode(nodeTo))

{

throw new Exception("NonExistingNode");

}

else

{

if (!IsExistEdge(nodeFrom, nodeTo))

{

edgesFrom.Add(nodeFrom);

edgesTo.Add(nodeTo);

edgesWeight.Add(weight);

}

else

{

throw new Exception("Add already existing edge");

}

}

}

public void DeleteEdge(int nodeFrom, int nodeTo)

{

if (IsExistEdge(nodeFrom, nodeTo))

{

for (int i = 0; i < edgesFrom.Count; i++)

{

if (edgesFrom[i] == nodeFrom && edgesTo[i] == nodeTo)

{

edgesFrom.RemoveAt(i);

edgesTo.RemoveAt(i);

edgesWeight.RemoveAt(i);

}

}

}

else

{

throw new Exception("Trying to delete non-existing edge");

}

}

public void AddNode(int node)

{

if(IsExistNode(node))

{

throw new Exception("Trying to add already existing node");

}

else

{

nodeArray.Add(node);

}

}

public List<int> GetAdjacentFromNodes(int fromNode)

{

if(IsExistNode(fromNode))

{

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

for (int i = 0; i < edgesFrom.Capacity; i++)

{

if (edgesFrom[i] == fromNode)

{

adjacentNodes.Add(edgesTo[i]);

}

}

return adjacentNodes;

}

else

{

throw new Exception("NonExistingFromNode");

}

}

public List<int> GetAdjacentToNodes(int toNode)

{

if (IsExistNode(toNode))

{

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

for (int i = 0; i < edgesFrom.Capacity; i++)

{

if (edgesTo[i] == toNode)

{

adjacentNodes.Add(edgesFrom[i]);

}

}

return adjacentNodes;

}

else

{

throw new Exception("NonExistingToNode");

}

}

public bool IsExistNode(int node)

{

for (int i = 0; i < nodeArray.Count; i++)

{

if (nodeArray[i] == node)

{

return true;

}

}

return false;

}

public bool IsExistEdge(int nodeFrom, int nodeTo)

{

for (int i = 0; i < edgesFrom.Count; i++)

{

if (edgesFrom[i] == nodeFrom && edgesTo[i] == nodeTo)

{

return true;

}

}

return false;

}

public void DeleteNode(int node)

{

if (IsExistNode(node))

{

for (int i = 0; i < edgesFrom.Count; i++)

{

if (edgesFrom[i] == node || edgesTo[i] == node)

{

edgesFrom.RemoveAt(i);

edgesTo.RemoveAt(i);

edgesWeight.RemoveAt(i);

}

}

}

else

{

throw new Exception("Trying to delete non-existing node");

}

}

public void PrintNodeList()

{

if(nodeArray.Count == 0)

{

Console.WriteLine("Node list is empty");

}

else

{

foreach (int node in nodeArray)

{

Console.WriteLine("Node :" + node + "\n");

}

}

}

public List<List<int>> ConvertToWeightMatrix()

{

if(NodeCount > 0)

{

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

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

{

weightMatrix.Add(new List<int>());

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

{

weightMatrix[i].Add(-1);

}

}

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

{

weightMatrix[nodeArray.IndexOf(edgesFrom[i])][nodeArray.IndexOf(edgesTo[i])] = edgesWeight[i];

}

return weightMatrix;

}

else

{

throw new Exception("Trying to convert empty graph");

}

}

public int NodeCount

{

get { return nodeArray.Count; }

}

public int EdgeCount

{

get { return edgesFrom.Count; }

}

}

}

Класс работы с графом:

public class GraphAlgorithm

{

public static void FillRandomGraph(Graph graph, int count)

{

if(count < 0)

{

throw new Exception("NegativeCount");

}

if (graph == null)

{

graph = new Graph();

}

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

{

graph.AddNode(i);

}

Random random = new Random();

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

{

Console.WriteLine(i + " of " + count);

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

{

int randomWeight = random.Next(-10000, 1000000);

if (randomWeight > 0)

{

graph.AddEdge(i, j, randomWeight);

}

}

}

}

public static int FindMinWay(Graph graph)

{

List<List<int>> matrix = FloydWarshallFind(graph);

if(graph.EdgeCount > 0)

{

int minWay = int.MaxValue;

foreach (List<int> line in matrix)

{

foreach (int way in line)

{

if (way < minWay && way != -1)

{

minWay = way;

}

}

}

return minWay;

}

else

{

throw new Exception("NonEdgedGraph");

}

}

public static List<List<int>> FloydWarshallFind(Graph graph)

{

if(graph == null || graph.NodeCount <= 0)

{

throw new Exception("EmptyGraph");

}

List<List<int>> matrix = graph.ConvertToWeightMatrix();

int nodeCount = graph.NodeCount;

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

{

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

{

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

{

if (matrix[i][k] != -1 && matrix[k][j] != -1 && matrix[i][k] + matrix[k][j] < matrix[i][j])

{

matrix[i][j] = matrix[i][k] + matrix[k][j];

}

}

}

}

return matrix;

}

}

Демонстрирующий работу программы класс (с точкой выхода):

class Program

{

static void Main(string[] args)

{

Graph graph = new Graph();

GraphAlgorithm.FillRandomGraph(graph, 1400);

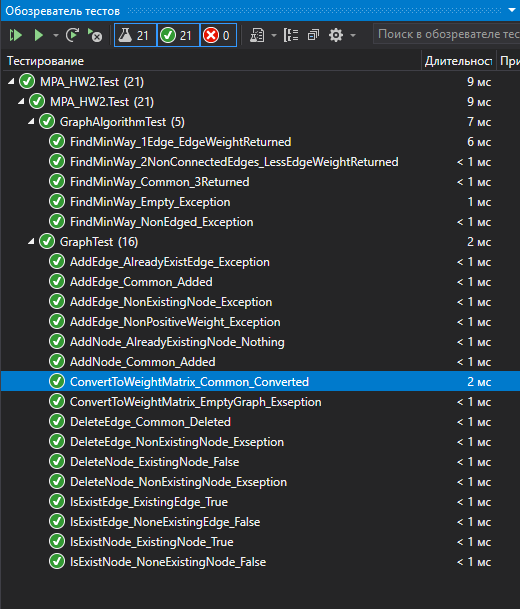
int minWay = GraphAlgorithm.FindMinWay(graph);

Console.WriteLine(minWay);

Console.ReadKey();

}

}

1. **Результаты тестирования**
2. **Результаты исследования времени работы алгортма**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Кол-во  узлов | 500 | 600 | 700 | 800 | 900 | 1000 | 1100 | 1200 | 1300 | 1400 |
| Время (с) | 2,8 | 3,8 | 5,3 | 7,2 | 9,7 | 12,8 | 16,5 | 21,1 | 26,8 | 33,0 |

1. **Сравнение полученных результатов с ожидаемыми:**

Ожидаемые результаты для пирамидальной сортировки в среднем вычисляются, как **O(n3)**.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Кол-во  узлов | 500 | 600 | 700 | 800 | 900 | 1000 | 1100 | 1200 | 1300 | 1400 |
| Время (с) | 1,25 | 2,16 | 3,4 | 5,1 | 7,2 | 10,0 | 13,3 | 17,2 | 21,9 | 27,4 |

Полученные результаты близки с ожидаемыми.

1. **Вывод**В процессе выполнения задачи я реализовал класс Граф с полным функционалом, протестировал его методы с помощью модульного тестирования. На основе данного класса я сделал реализацию нахождения самого короткого пути между всеми парами вершин заданного графа при помощи алгоритма Флойда-Уоршелла. Реализация также была удачно протестирована. Также я получил зависимость времени работы алгоритма от количества узлов в заданном графе и, сравнив с ожидаемыми теоритическими результатыми, получил приблизиетльное совпадение.