Практическое занятие 4.

Тема: Эвристические алгоритмы.

Цель: Ознакомление с эвристическими алгоритмами.

Задание.

* + Найти кратчайший путь между двумя точками в графе.

namespace Lab4

{

class Program

{

static void Main(string[] args)

{

var g = new Graph();

//добавление вершин

g.AddVertex("A");

g.AddVertex("B");

g.AddVertex("C");

g.AddVertex("D");

g.AddVertex("E");

g.AddVertex("F");

g.AddVertex("G");

//добавление ребер

g.AddEdge("A", "B", 40);

g.AddEdge("A", "C", 33);

g.AddEdge("A", "D", 61);

g.AddEdge("B", "C", 47);

g.AddEdge("B", "E", 93);

g.AddEdge("C", "D", 11);

g.AddEdge("C", "E", 79);

g.AddEdge("C", "F", 63);

g.AddEdge("D", "F", 41);

g.AddEdge("E", "F", 17);

g.AddEdge("E", "G", 58);

g.AddEdge("F", "G", 84);

var dijkstra = new Dijkstra(g);

var path = dijkstra.FindShortestPath("A", "G");

Console.WriteLine(path);

Console.ReadKey();

}

}

}

class GraphVertex

{

/// <summary>

/// Название вершины

/// </summary>

public string Name { get; }

/// <summary>

/// Список ребер

/// </summary>

public List<GraphEdge> Edges { get; }

/// <summary>

/// Конструктор

/// </summary>

/// <param name="vertexName">Название вершины</param>

public GraphVertex(string vertexName)

{

Name = vertexName;

Edges = new List<GraphEdge>();

}

/// <summary>

/// Добавить ребро

/// </summary>

/// <param name="newEdge">Ребро</param>

public void AddEdge(GraphEdge newEdge)

{

Edges.Add(newEdge);

}

/// <summary>

/// Добавить ребро

/// </summary>

/// <param name="vertex">Вершина</param>

/// <param name="edgeWeight">Вес</param>

public void AddEdge(GraphVertex vertex, int edgeWeight)

{

AddEdge(new GraphEdge(vertex, edgeWeight));

}

/// <summary>

/// Преобразование в строку

/// </summary>

/// <returns>Имя вершины</returns>

public override string ToString() => Name;

}

class Graph

{

/// <summary>

/// Список вершин графа

/// </summary>

public List<GraphVertex> Vertices { get; }

/// <summary>

/// Конструктор

/// </summary>

public Graph()

{

Vertices = new List<GraphVertex>();

}

/// <summary>

/// Добавление вершины

/// </summary>

/// <param name="vertexName">Имя вершины</param>

public void AddVertex(string vertexName)

{

Vertices.Add(new GraphVertex(vertexName));

}

/// <summary>

/// Поиск вершины

/// </summary>

/// <param name="vertexName">Название вершины</param>

/// <returns>Найденная вершина</returns>

public GraphVertex FindVertex(string vertexName)

{

foreach (var v in Vertices)

{

if (v.Name.Equals(vertexName))

{

return v;

}

}

return null;

}

/// <summary>

/// Добавление ребра

/// </summary>

/// <param name="firstName">Имя первой вершины</param>

/// <param name="secondName">Имя второй вершины</param>

/// <param name="weight">Вес ребра соединяющего вершины</param>

public void AddEdge(string firstName, string secondName, int weight)

{

var v1 = FindVertex(firstName);

var v2 = FindVertex(secondName);

if (v2 != null && v1 != null)

{

v1.AddEdge(v2, weight);

v2.AddEdge(v1, weight);

}

}

}

class GraphEdge

{

/// <summary>

/// Связанная вершина

/// </summary>

public GraphVertex ConnectedVertex { get; }

/// <summary>

/// Вес ребра

/// </summary>

public int EdgeWeight { get; }

/// <summary>

/// Конструктор

/// </summary>

/// <param name="connectedVertex">Связанная вершина</param>

/// <param name="weight">Вес ребра</param>

public GraphEdge(GraphVertex connectedVertex, int weight)

{

ConnectedVertex = connectedVertex;

EdgeWeight = weight;

}

}

class GraphVertexInfo

{

/// <summary>

/// Вершина

/// </summary>

public GraphVertex Vertex { get; set; }

/// <summary>

/// Не посещенная вершина

/// </summary>

public bool IsUnvisited { get; set; }

/// <summary>

/// Сумма весов ребер

/// </summary>

public int EdgesWeightSum { get; set; }

/// <summary>

/// Предыдущая вершина

/// </summary>

public GraphVertex PreviousVertex { get; set; }

/// <summary>

/// Конструктор

/// </summary>

/// <param name="vertex">Вершина</param>

public GraphVertexInfo(GraphVertex vertex)

{

Vertex = vertex;

IsUnvisited = true;

EdgesWeightSum = int.MaxValue;

PreviousVertex = null;

}

}

class Dijkstra

{

Graph graph;

List<GraphVertexInfo> infos;

/// <summary>

/// Конструктор

/// </summary>

/// <param name="graph">Граф</param>

public Dijkstra(Graph graph)

{

this.graph = graph;

}

/// <summary>

/// Инициализация информации

/// </summary>

void InitInfo()

{

infos = new List<GraphVertexInfo>();

foreach (var v in graph.Vertices)

{

infos.Add(new GraphVertexInfo(v));

}

}

/// <summary>

/// Получение информации о вершине графа

/// </summary>

/// <param name="v">Вершина</param>

/// <returns>Информация о вершине</returns>

GraphVertexInfo GetVertexInfo(GraphVertex v)

{

foreach (var i in infos)

{

if (i.Vertex.Equals(v))

{

return i;

}

}

return null;

}

/// <summary>

/// Поиск непосещенной вершины с минимальным значением суммы

/// </summary>

/// <returns>Информация о вершине</returns>

public GraphVertexInfo FindUnvisitedVertexWithMinSum()

{

var minValue = int.MaxValue;

GraphVertexInfo minVertexInfo = null;

foreach (var i in infos)

{

if (i.IsUnvisited && i.EdgesWeightSum < minValue)

{

minVertexInfo = i;

minValue = i.EdgesWeightSum;

}

}

return minVertexInfo;

}

/// <summary>

/// Поиск кратчайшего пути по названиям вершин

/// </summary>

/// <param name="startName">Название стартовой вершины</param>

/// <param name="finishName">Название финишной вершины</param>

/// <returns>Кратчайший путь</returns>

public string FindShortestPath(string startName, string finishName)

{

return FindShortestPath(graph.FindVertex(startName), graph.FindVertex(finishName));

}

/// <summary>

/// Поиск кратчайшего пути по вершинам

/// </summary>

/// <param name="startVertex">Стартовая вершина</param>

/// <param name="finishVertex">Финишная вершина</param>

/// <returns>Кратчайший путь</returns>

public string FindShortestPath(GraphVertex startVertex, GraphVertex finishVertex)

{

InitInfo();

var first = GetVertexInfo(startVertex);

first.EdgesWeightSum = 0;

while (true)

{

var current = FindUnvisitedVertexWithMinSum();

if (current == null)

{

break;

}

SetSumToNextVertex(current);

}

return GetPath(startVertex, finishVertex);

}

/// <summary>

/// Вычисление суммы весов ребер для следующей вершины

/// </summary>

/// <param name="info">Информация о текущей вершине</param>

void SetSumToNextVertex(GraphVertexInfo info)

{

info.IsUnvisited = false;

foreach (var e in info.Vertex.Edges)

{

var nextInfo = GetVertexInfo(e.ConnectedVertex);

var sum = info.EdgesWeightSum + e.EdgeWeight;

if (sum < nextInfo.EdgesWeightSum)

{

nextInfo.EdgesWeightSum = sum;

nextInfo.PreviousVertex = info.Vertex;

}

}

}

/// <summary>

/// Формирование пути

/// </summary>

/// <param name="startVertex">Начальная вершина</param>

/// <param name="endVertex">Конечная вершина</param>

/// <returns>Путь</returns>

string GetPath(GraphVertex startVertex, GraphVertex endVertex)

{

var path = endVertex.ToString();

while (startVertex != endVertex)

{

endVertex = GetVertexInfo(endVertex).PreviousVertex;

path = endVertex.ToString() + path;

}

return path;

}

}

Вывод: ознакомился с эвристическими алгоритмами и их временной оценке.