Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота № 1

з дисципліни

«Вступ до штучного інтелекту»

Виконав: Перевірив:

студент групи ІП-03 Трочун Євгеній

Янишин Віталій Володимирович

номер залікової книжки: 0327

Київ 2022

## Опис Роботи

Моя лабораторна робота написана на мові C# і виконана в Jupyter Notebook від anaconda3.

Дана робота має наступну структуру є класи:

**Vertex** який реалізує вершину графа яка має порядковий номер;

**Edge** який реалізує ребро графа і містить інформацію про вершини які він зв’язує та його вагу.

**Graph** який реалізовує ненаправлений граф, даний клас містить: поля Vertices і Edges які являють собою списки які зберігають вершини та ребра графа відповідно та VertexCount EdgeCount які зберігають кількість вершин і ребер графа відповідно; методи AddVertex приймає вершину і додає її в список вершин, AddEdge приймає дві вершини і вагу ребра і створює ребро між заданими вершинами з заданою вагою, IsWay який приймає дві вершини і повертає true якщо вони мають спільне ребро, GetVertexList приймає вершину і повертає список суміжних вершин.

Головним класом є клас **SquareGraph** який і виконує поставлену задачу через конструктор створюється квадратний граф величина сторони якого передається аргументом в конструктор і за зазамовчеванням дорівнює 5 метод RemoveEdges видаляє задану кількість ребер і метод ShowInGraphConsole який виводить наш граф на консоль і де вершини задані символом ‘o’ а ребра символами ‘-’ та ‘|’

## Програмний код генерації дороги

Клас **Vertex**

public class Vertex

{

public int Number { get; set; }

public Vertex(int number)

{

Number = number;

}

public override string ToString()

{

return Number.ToString();

}

public override bool Equals(object? obj)

{

if(obj is Vertex v)

{

return v.Number.Equals(Number);

}

return false;

}

}

Клас **Edge**

public class Edge

{

public Vertex Vertex1 { get; set; }

public Vertex Vertex2 { get; set; }

public int Weight { get; set; }

public Edge(Vertex vertex1 , Vertex vertex2, int weight=1)

{

Vertex1 = vertex1;

Vertex2 = vertex2;

Weight = weight;

}

public override string ToString()

{

return $"({Vertex1}; {Vertex2})";

}

public override bool Equals(object? obj)

{

if(obj is Edge edge)

{

return edge.Vertex1.Equals(Vertex1) &&edge.Vertex2.Equals(Vertex2);

}

return false;

}

}

Клас **Graph**

public class Graph

{

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

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

public int VertexCount => Vertices.Count;

public int EdgeCount =0;

public void AddVertex(Vertex vertex)

{

if (Vertices.Contains(vertex))

{

return;

}

Vertices.Add(vertex);

}

public void AddEdge(Vertex Vertex1, Vertex Vertex2, int weight=1)

{

if (Vertex1.Equals(Vertex2))

{

return;

}

var edge = new Edge(Vertex1, Vertex2, weight);

var edge2 = new Edge(Vertex2, Vertex1, weight);

if (Edges.Contains(edge))

{

Console.WriteLine($"Way ({Vertex1}:{Vertex2}) already exists");

return;

}

Edges.Add(edge);

Edges.Add(edge2);

EdgeCount++;

}

public bool IsWay(Vertex vertex1, Vertex vertex2)

{

var list = GetVertexLists(vertex1);

return list.Contains(vertex2);

}

public List<Vertex> GetVertexLists(Vertex vertex)

{

var result=new List<Vertex>();

foreach(var edge in Edges)

{

if (edge.Vertex1.Equals(vertex))

{

result.Add(edge.Vertex2);

}

}

return result;

}

public bool Wave(Vertex start, Vertex finish)

{

var list = new List<Vertex>();

list.Add(start);

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

{

var vertex = list[i];

foreach (var v in GetVertexLists(vertex))

{

if (!list.Contains(v))

{

list.Add(v);

}

}

}

return list.Contains(finish);

}

public bool RemoveEdge(Vertex Vertex1, Vertex Vertex2)

{

var edge = new Edge(Vertex1, Vertex2);

var edge2 = new Edge(Vertex2, Vertex1);

bool result = Edges.Remove(edge) && Edges.Remove(edge2);

if (result)

{

EdgeCount--;

}

return result;

}

}

Клас **SquareGraph**

public class SquareGraph

{

private Graph Graph{get;set;}

private Vertex[,] vertexes;

private int Size;

public SquareGraph (int size = 5)

{

Size = size;

Graph=GetSquareGraph(size);

}

private Graph GetSquareGraph(int size = 5)

{

var rnd = new Random();

vertexes =new Vertex[size, size];

if (size < 2)

{

Console.WriteLine("Side must be not less then 2");

size = 2;

}

var Graph=new Graph();

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

{

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

{

var vertex = new Vertex(i\* size + j);

Graph.AddVertex(vertex);

vertexes[i,j] = vertex;

}

}

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

{

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

{

if(i== size - 1 && j == size - 1)

{

continue;

}

else if(j== size - 1 && i < size - 1)

{

Graph.AddEdge(vertexes[i, j], vertexes[i + 1, j], rnd.Next(1, 9));

}

else if(i== size - 1 && j < size - 1)

{

Graph.AddEdge(vertexes[i, j], vertexes[i, j + 1], rnd.Next(1, 9));

}

else

{

Graph.AddEdge(vertexes[i, j], vertexes[i, j + 1], rnd.Next(1, 9));

Graph.AddEdge(vertexes[i, j], vertexes[i + 1, j], rnd.Next(1, 9));

}

}

}

return Graph;

}

public void RemoveEdges(int Count)

{

if (Count > Graph.EdgeCount - Graph.VertexCount + 1)

{

throw new ArgumentException("The graph must be connected", nameof(Count));

}

var rnd = new Random();

var removed = false;

while(Count!=0)

foreach(var vertex in vertexes)

{

var list = Graph.GetVertexLists(vertex);

var vertex2 = list[rnd.Next(list.Count)];

Graph.RemoveEdge(vertex, vertex2);

removed = true;

foreach(var ver in vertexes)

{

if (Graph.Wave(vertex, ver))

{

continue;

}

else

{

Graph.AddEdge(vertex, vertex2);

removed = false;

break;

}

}

if (removed)

{

Count--;

removed = false;

}

if(Count == 0)

{

break;

}

}

}

public void ShowInGraphConsole()

{

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

{

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

{

Console.Write("o");

if (j == Size - 1)

{

continue;

}

else

{

if(Graph.IsWay(vertexes[i,j], vertexes[i, j + 1]))

{

Console.Write(" - ");

}

else

{

Console.Write(" ");

}

}

}

Console.WriteLine();

if(i==Size - 1)

{

continue;

}

else

{

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

{

if (Graph.IsWay(vertexes[i, j], vertexes[i+1, j]))

{

Console.Write("|");

}

else

{

Console.Write(" ");

}

Console.Write(" ");

}

}

Console.WriteLine();

}

}

}

**Main**

internal class Program

{

static void Main(string[] args)

{

var graph=new SquareGraph();

graph.ShowInGraphConsole();

graph.RemoveEdges(10);

Console.WriteLine();

graph.ShowInGraphConsole();

}

}

# Висновок

Підчас лабораторної роботи я отримав навички роботи з JupyterNotebook. Та підготував оточення для виконання подальних лабораторних робіт