//Author Zaynoun Hammoud, September 2025

using System;

public class TSP

{

char[] Cities = { 'A', 'B', 'C', 'D', 'E' ,'F','G','H','I'};

string path="A";

string finalPath;

int pathCost=0;

int origin=0;

int minCost=int.MaxValue;

public int factorial( int n)//to calculate permutations

{

if(n==1)

return 1;

else

return n\* factorial (n-1);

}

public void printArr ( int [,] n)

{

for ( int i=0;i<n.GetLength(0);i++)

{

Console.WriteLine("---");

for ( int j=0;j<n.GetLength(1);j++)

Console.Write(" "+n[i,j]);

}

}

public void printArr3by3 ( int [,,] n)

{

for ( int i=0;i<n.GetLength(0);i++)

for ( int j=0;j<n.GetLength(1);j++)

for ( int z=0;z<n.GetLength(2);z++)

;

}

public int BruteForce(int [,] \_ConnectedGraph)

{

int log=0;

bool flush=false;

int depth=\_ConnectedGraph.GetLength(0);//route path depth from origin

//This also represents the number of cities to visit from origin

bool [] visitedCities= new bool[depth];

int depthCounter=0;

int selectedCityCounter=0;

int route=0;

int iterationControl=depth-1;

int numberOfPossibleRoutes=factorial(depth-1);

Console.WriteLine(" max routes number>"+numberOfPossibleRoutes);

int iteration=0;

//Console.WriteLine("iterationControl="+iterationControl);

int [,] visitRegister= new int[iterationControl,iterationControl];

int [,] selectionControl = new int[1, depth-1];

for (int i=0;i<selectionControl.GetLength(1);i++)

{

selectionControl[0,i]=factorial(iterationControl)/iterationControl;

iterationControl--;

}

// printArr(visitRegister);

while(numberOfPossibleRoutes>0)

{

if(visitRegister[iteration,selectedCityCounter]<selectionControl[0,iteration])

if(visitedCities[selectedCityCounter+1]==false)

{

// Console.Write(">"+Cities[selectedCityCounter+1]);

path=path+Cities[selectedCityCounter+1];

pathCost=pathCost+\_ConnectedGraph[origin,selectedCityCounter+1];

origin=selectedCityCounter+1;

visitRegister[iteration,selectedCityCounter]++;

//printArr(visitRegister);

iteration++;

depthCounter++;

visitedCities[selectedCityCounter+1]=true;

//printArr(visitRegister);

}

selectedCityCounter++;

if(selectedCityCounter==depth-1)

{

selectedCityCounter=0;

}

if(depthCounter==depth-1)

{

// Console.WriteLine("");

// Console.WriteLine("resetting");

log++;

pathCost=pathCost+\_ConnectedGraph[0,origin];

path=path+"A";

if(minCost>pathCost)

{minCost=pathCost;

finalPath=path;

}

path="A";

iteration=0;

depthCounter=0;

selectedCityCounter=0;

origin=0;

pathCost=0;

route++;

numberOfPossibleRoutes--;

for(int x=0;x<depth;x++)

visitedCities[x]=false;

int control=selectionControl.GetLength(1)-3;

if(control<0)

control=0;

if(route==selectionControl[0,control])

{

flush=true;

//printArr(visitRegister);

for(int x=1;x<depth-1;x++)

for(int y=0;y<depth-1;y++)

visitRegister[x,y]=0;

route=0;

}

}

}//while loop

Console.WriteLine("Traveling Salesman using Brute force Shortest path is "+finalPath);

return minCost;

}

public int NearestNeighbour (int [,] \_ConnectedGraph, bool [,] \_visitedCity)

{

string visited;

int cityRow=0;

int cityColumn=0;

int newCityRow=0;

int citiesToVisit=\_ConnectedGraph.GetLength(0);

int totalDistance=0;

int distance=int.MaxValue;

for( int i=0;i< \_ConnectedGraph.GetLength(0);i++)

{

for( int j=0;j< \_ConnectedGraph.GetLength(0);j++)

{

if (\_ConnectedGraph[newCityRow,j]!=0 && (!\_visitedCity[newCityRow,j]) )

if (\_ConnectedGraph[newCityRow,j]<distance)

{

distance=\_ConnectedGraph[newCityRow,j];

cityRow=i;

cityColumn=j;

}

if (j==\_ConnectedGraph.GetLength(0)-1 && citiesToVisit>1)

{

Console.WriteLine("distance is "+distance+" city coordiates are : row "+cityRow+" columun"+cityColumn);

newCityRow=cityColumn;

totalDistance=totalDistance+distance;

citiesToVisit--;

distance=1000;//reset

Console.WriteLine("citiesToVisit"+citiesToVisit);

Console.WriteLine("total distance="+totalDistance);

for (int xx=0;xx<\_visitedCity.GetLength(0);xx++)

{ \_visitedCity[xx,0]=true;

\_visitedCity[xx,cityColumn]=true;

}

// printGraph(\_visitedCity);

}// if

}//for j

}//i

Console.WriteLine("return to origin from row-"+\_ConnectedGraph[newCityRow,0]+" Total Distance using TSP NN is");

totalDistance=totalDistance+\_ConnectedGraph[newCityRow,0];

return totalDistance;

} // for i

}

public class HelloWorld

{

public static void Main(string[] args)

{

// Assumptions is that the graph is connected graph

int[,] distanceMatrix = new int[, ]

{

// A B C D E F G

{ 0, 10, 15, 20, 25, 30, 35 }, // A

{ 10, 0, 35, 25, 17, 28, 23 }, // B

{ 15, 35, 0, 30, 20, 34, 31 }, // C

{ 20, 25, 30, 0, 22, 18, 27 }, // D

{ 25, 17, 20, 22, 0, 24, 19 }, // E

{ 30, 28, 34, 18, 24, 0, 26 }, // F

{ 35, 23, 31, 27, 19, 26, 0 } // G

//{ 0, 7, 10, 13 },

//{ 7, 0, 6, 9 },

//{ 10, 6, 0, 8 },

// { 13, 9, 8, 0 }

//{ 0, 5, 9, 14 },

//{ 5, 0, 8, 12 },

//{ 9, 8, 0, 6 },

//{ 14, 12, 6, 0 }

// A B C D E

// { 0, 10, 15, 20, 25 }, // A

//{ 10, 0, 35, 25, 17 }, // B

//{ 15, 35, 0, 30, 28 }, // C

//{ 20, 25, 30, 0, 22 }, // D

//{ 25, 17, 28, 22, 0 } // E

// A B C D E

// { 0, 12, 29, 22, 17 }, // A

//{ 12, 0, 19, 31, 23 }, // B

//{ 29, 19, 0, 28, 34 }, // C

//{ 22, 31, 28, 0, 26 }, // D

//{ 17, 23, 34, 26, 0 } // E

// A B C

//{ 0, 10, 15 }, // A

//{ 10, 0, 20 }, // B

//{ 15, 20, 0 } // C

//{ 0, 12, 18, 24 },

//{ 12, 0, 28, 22 },

//{ 18, 28, 0, 30 },

//{ 24, 22, 30, 0 }

// A B C D E

//{ 0, 10, 15, 20, 25 }, // A

//{ 10, 0, 35, 25, 30 }, // B

//{ 15, 35, 0, 30, 20 }, // C

//{ 20, 25, 30, 0, 15 }, // D

//{ 25, 30, 20, 15, 0 } // E

// A B C D E F

//{ 0, 10, 15, 20, 25, 30 }, // A

//{ 10, 0, 35, 25, 17, 28 }, // B

//{ 15, 35, 0, 30, 20, 26 }, // C

//{ 20, 25, 30, 0, 18, 24 }, // D

//{ 25, 17, 20, 18, 0, 22 }, // E

// { 30, 28, 26, 24, 22, 0 } // F

//{ 0, 10, 15, 20, 25 }, // A

//{ 10, 0, 35, 25, 17 }, // B

//{ 15, 35, 0, 30, 28 }, // C

//{ 20, 25, 30, 0, 22 }, // D

//{ 25, 17, 28, 22, 0 } // E

};

bool [,] visitedCity = new bool[distanceMatrix.GetLength(0), distanceMatrix.GetLength(0)];

TSP t= new TSP();

//TSP solution using Brute force and NearestNeighbour

Console.WriteLine (t.BruteForce(distanceMatrix));

Console.WriteLine(t.NearestNeighbour(distanceMatrix, visitedCity));

}

}