public class RomeniaMapProblemBreadthFirstSearch: IMiniProblem

{

private decimal \_costOfTheWay;

private int \_depth;

private string \_initialState;

private string \_objectiveState;

private Node \_solutionNode;

private IList<string> \_statesSpace;

private IList<Node> \_nodes;

public **RomeniaMapProblemBreadthFirstSearch**(string initialStateName, string objectiveStateName, IList<string> statesSpace)

{

\_costOfTheWay = 0; \_depth = 0;

\_nodes = new List<Node>();

\_statesSpace = statesSpace;

\_initialState = InstanceStateByTheName(initialStateName);

\_objectiveState = InstanceStateByTheName(objectiveStateName);

}

private string InstanceStateByTheName(string stateName)

{

if (string.IsNullOrEmpty(stateName)){

throw new ArgumentNullException("O nome do estado não pode ser nulo ou vazio.");

}

var state = \_statesSpace.Where(field => field.Equals(stateName));

if (!state.Any()){

throw new ArgumentException($"O nome do estado '{stateName}', ....");

}

\_nodes.Add(new Node(stateName, null, "", 0));

return state.FirstOrDefault();

}

public IList<string> statesSpace { get => \_statesSpace; }

public string InitialState { get => \_initialState; }

public IList<Node> Nodes { get => \_nodes; }

public Node SolutionNode { get => \_solutionNode; }

public int Depth { get => \_depth; }

public decimal costOfTheWay { get => \_costOfTheWay; }

public void **AddChildToParent**(string childNodeName, string parentNodeName, decimal costOfTheWay, string action)

{

if (string.IsNullOrEmpty(childNodeName)){

throw new ArgumentNullException("O nome ... um valor nulo ou vazio.");

}

if (string.IsNullOrEmpty(parentNodeName)){

throw new ArgumentNullException("O nome ... um valor nulo ou vazio.");

}

**GetParentNode**(childNodeName, parentNodeName, costOfTheWay, action);

}

private void **GetParentNode**(string childNodeName, string parentNodeName, decimal costOfTheWay, string action)

{

var parentNode = \_nodes.FirstOrDefault(field => field.State == parentNodeName);

if (parentNode == null) {

throw new ArgumentException($"O nome do estado pai '{parentNodeName}', ...");

}

AddChildNodeTo(parentNode, childNodeName, costOfTheWay, action);

}

private void **AddChildNodeTo**(Node parentNode, string childNodeName, decimal costOfTheWay, string action)

{

var childState = \_statesSpace.Where(field => field.Equals(childNodeName)).FirstOrDefault();

if (childState == null){

throw new ArgumentException($"O nome do estado filho '{childNodeName}', ...");

}

\_nodes.Add(new Node(childState, parentNode, action, costOfTheWay));

}

public void **SearchTree**()

{

var edge = \_nodes

.Where(field => field.ParentNode == null && field.State == \_initialState).ToList();

if (!**IsSolutionContainedIn**(edge)){

ExpandLevel(edge);

}

}

private bool **IsSolutionContainedIn**(IList<Node> edge)

{

\_solutionNode = (from Node selectedNode in edge

where selectedNode.State == \_objectiveState

select selectedNode).FirstOrDefault();

if (\_solutionNode == null){

return false;

}

return true;

}

private void **ExpandLevel**(List<Node> edge)

{

while (\_solutionNode == null)

{

var expandedNodes = new List<Node>();

edge.ForEach(node => {

if (\_solutionNode == null){

expandedNodes.AddRange(GetExpandedNode(node));

if (IsSolutionContainedIn(expandedNodes)){

return;

}

}

});

\_depth += 1;

**ExpandLevel**(expandedNodes);

}

}

private List<Node> **GetExpandedNode**(Node node)

{

var childrenNodes = **Sucessor**(node);

foreach (var childNode in childrenNodes){

\_costOfTheWay += childNode.CostOfTheWay;

}

return childrenNodes;

}

private List<Node> **Sucessor**(Node selectedNode)

{

if (selectedNode.ParentNode == null)

{

return \_nodes.Where(node =>

node.ParentNode != null &&

node.ParentNode.State == selectedNode.State)

.OrderBy(field => field.CostOfTheWay).ToList();

}

return \_nodes.Where(node =>

node.ParentNode != null &&

node.ParentNode.State == selectedNode.State &&

node.State != selectedNode.ParentNode.State)

.OrderBy(field => field.CostOfTheWay.ToList();

}

}

**[TestClass]**

public class RomeniaMapProblemTest

{

private IList<string> \_stateSpaces;

[TestInitialize]

public void Initialize()

{

**AddSatesSpace**();

}

private void **AddSatesSpace**()

{

\_stateSpaces = new List<string>();

\_stateSpaces.Add("Arad");

\_stateSpaces.Add("Zerind");

\_stateSpaces.Add("Oradea");

\_stateSpaces.Add("Timisoara");

\_stateSpaces.Add("Sibiu");

\_stateSpaces.Add("Lugoj");

\_stateSpaces.Add("Mehadia");

\_stateSpaces.Add("Dobreta");

\_stateSpaces.Add("Craiova");

\_stateSpaces.Add("Rimnieu Vilcea");

\_stateSpaces.Add("Pitesti");

\_stateSpaces.Add("Fagaras");

\_stateSpaces.Add("Bucareste");

\_stateSpaces.Add("Giurgiu");

\_stateSpaces.Add("Urziceni");

\_stateSpaces.Add("Neamt");

\_stateSpaces.Add("Nome Apagado");

\_stateSpaces.Add("Vaslui");

\_stateSpaces.Add("Hirsova");

\_stateSpaces.Add("Eforie");

}

private void **AddNodesToProblem**(ref RomeniaMapProblem romeniaMapProblem)

{

romeniaMapProblem.AddChildToParent("Zerind", "Arad", 75, "GO");

romeniaMapProblem.AddChildToParent("Arad", "Zerind", 75, "BACK");

romeniaMapProblem.AddChildToParent("Sibiu", "Arad", 140, "GO");

romeniaMapProblem.AddChildToParent("Arad", "Sibiu", 140, "BACK");

romeniaMapProblem.AddChildToParent("Timisoara", "Arad", 118, "GO");

romeniaMapProblem.AddChildToParent("Arad", "Timisoara", 118, "BACK");

romeniaMapProblem.AddChildToParent("Oradea", "Zerind", 71, "GO");

romeniaMapProblem.AddChildToParent("Zerind", "Oradea", 71, "BACK");

romeniaMapProblem.AddChildToParent("Sibiu", "Oradea", 151, "GO");

romeniaMapProblem.AddChildToParent("Oradea", "Sibiu", 151, "BACK");

romeniaMapProblem.AddChildToParent("Lugoj", "Timisoara", 111, "GO");

romeniaMapProblem.AddChildToParent("Timisoara", "Lugoj", 111, "BACK");

romeniaMapProblem.AddChildToParent("Mehadia", "Lugoj", 70, "GO");

romeniaMapProblem.AddChildToParent("Lugoj", "Mehadia", 70, "BACK");

romeniaMapProblem.AddChildToParent("Dobreta", "Mehadia", 75, "GO");

romeniaMapProblem.AddChildToParent("Mehadia", "Dobreta", 75, "BACK");

romeniaMapProblem.AddChildToParent("Craiova", "Dobreta", 120, "GO");

romeniaMapProblem.AddChildToParent("Dobreta", "Craiova", 120, "BACK");

romeniaMapProblem.AddChildToParent("Pitesti", "Craiova", 138, "GO");

romeniaMapProblem.AddChildToParent("Craiova", "Pitesti", 138, "BACK");

romeniaMapProblem.AddChildToParent("Rimnieu Vilcea", "Craiova", 146, "GO");

romeniaMapProblem.AddChildToParent("Craiova", "Rimnieu Vilcea", 146, "BACK");

romeniaMapProblem.AddChildToParent("Rimnieu Vilcea", "Sibiu", 80, "GO");

romeniaMapProblem.AddChildToParent("Sibiu", "Rimnieu Vilcea", 80, "BACK");

romeniaMapProblem.AddChildToParent("Fagaras", "Sibiu", 99, "GO");

romeniaMapProblem.AddChildToParent("Sibiu", "Fagaras", 99, "BACK");

romeniaMapProblem.AddChildToParent("Bucareste", "Fagaras", 211, "GO");

romeniaMapProblem.AddChildToParent("Fagaras", "Bucareste", 211, "BACK");

romeniaMapProblem.AddChildToParent("Pitesti", "Rimnieu Vilcea", 97, "GO");

romeniaMapProblem.AddChildToParent("Rimnieu Vilcea", "Pitesti", 97, "BACK");

romeniaMapProblem.AddChildToParent("Bucareste", "Pitesti", 101, "GO");

romeniaMapProblem.AddChildToParent("Pitesti", "Bucareste", 101, "BACK");

romeniaMapProblem.AddChildToParent("Giurgiu", "Bucareste", 90, "GO");

romeniaMapProblem.AddChildToParent("Bucareste", "Giurgiu", 90, "BACK");

romeniaMapProblem.AddChildToParent("Urziceni", "Bucareste", 85, "GO");

romeniaMapProblem.AddChildToParent("Bucareste", "Urziceni", 85, "BACK");

romeniaMapProblem.AddChildToParent("Hirsova", "Urziceni", 98, "GO");

romeniaMapProblem.AddChildToParent("Urziceni", "Hirsova", 98, "BACK");

romeniaMapProblem.AddChildToParent("Eforie", "Hirsova", 86, "GO");

romeniaMapProblem.AddChildToParent("Hirsova", "Eforie", 86, "BACK");

romeniaMapProblem.AddChildToParent("Vaslui", "Urziceni", 142, "GO");

romeniaMapProblem.AddChildToParent("Urziceni", "Vaslui", 142, "BACK");

romeniaMapProblem.AddChildToParent("Nome Apagado", "Vaslui", 92, "GO");

romeniaMapProblem.AddChildToParent("Vaslui", "Nome Apagado", 92, "BACK");

romeniaMapProblem.AddChildToParent("Neamt", "Nome Apagado", 87, "GO");

romeniaMapProblem.AddChildToParent("Nome Apagado", "Neamt", 87, "BACK");

}

**[TestMethod]**

public void **SearchTreeFromAradToBucareste**()

{

var romeniaMapProblem = new RomeniaMapProblem("Arad", "Bucareste", \_stateSpaces);

**AddNodesToProblem**(ref romeniaMapProblem);

romeniaMapProblem.**SearchTree**();

Assert.AreEqual(romeniaMapProblem.SolutionNode.State, "Bucareste");

Assert.AreEqual(romeniaMapProblem.SolutionNode.ParentNode.State, "Fagaras");

Assert.AreEqual(romeniaMapProblem.SolutionNode.Action, "GO");

Assert.AreEqual(romeniaMapProblem.Depth, 3);

}