Họ tên: Phạm Văn Thuận

MSSV: 6351071068
Lớp: CQ.63.CNTT

Cài đặt thuật toán Hill Climbing

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace HillClimbing
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.OutputEncoding = Encoding.UTF8;
            Console.WriteLine("=== THUẬT TOÁN HILL CLIMBING ===");
            Console.WriteLine();

            Graph graph = InputGraph();

            Console.WriteLine("\n=== Đồ thị đã nhập ===");
            graph.PrintGraph();
            Console.WriteLine();
```

```
    Console.Write("Nhập node bắt đầu: ");
    string startNode = Console.ReadLine().Trim().ToUpper();

    Console.Write("Nhập node đích: ");
    string goalNode = Console.ReadLine().Trim().ToUpper();

    if (!graph.HasNode(startNode) || !graph.HasNode(goalNode))
    {
        Console.WriteLine("Node bắt đầu hoặc node đích không tồn tại trong đồ thị!");
        Console.ReadKey();
        return;
    }

    Console.WriteLine();

    HillClimbing(graph, startNode, goalNode);

    Console.WriteLine("\nNhấn phím bất kỳ để kết thúc...");
    Console.ReadKey();
}

static Graph InputGraph()
{
    Graph graph = new Graph();

    Console.WriteLine("NHẬP THÔNG TIN ĐỒ THỊ:");
```

```
Console.Write("Số lượng node: ");
int nodeCount = int.Parse(Console.ReadLine());

Console.WriteLine("\nNhập các node và giá trị heuristic:");
for (int i = 0; i < nodeCount; i++)
{
    Console.Write($"Tên node {i + 1}: ");
    string nodeId = Console.ReadLine().Trim().ToUpper();

    Console.Write($"Giá trị heuristic của node {nodeId}: ");
    double heuristic = double.Parse(Console.ReadLine());

    graph.AddNode(nodeId, heuristic);
}

Console.Write("\nSố lượng cạnh: ");
int edgeCount = int.Parse(Console.ReadLine());

Console.WriteLine("\nNhập các cạnh và trọng số:");
for (int i = 0; i < edgeCount; i++)
{
    Console.Write($"Cạnh {i + 1} - Node bắt đầu: ");
    string fromNode = Console.ReadLine().Trim().ToUpper();

    Console.Write($"Cạnh {i + 1} - Node kết thúc: ");
```

```
        string toNode = Console.ReadLine().Trim().ToUpper();

        Console.Write($"Trọng số cạnh {fromNode} -> {toNode}: ");
        double cost = double.Parse(Console.ReadLine());

        if (graph.HasNode(fromNode) && graph.HasNode(toNode))
        {
            graph.AddEdge(fromNode, toNode, cost);
        }
        else
        {
            Console.WriteLine("Node không tồn tại trong đồ thị! Vui lòng nhập lại.");
            i--;
        }
    }

    return graph;
}

static void HillClimbing(Graph graph, string startNodeId, string goalNodeId)
{
    Console.WriteLine("Thuật toán Hill Climbing:");

    List<Node> openList = new List<Node> { graph.GetNode(startNodeId) };
    HashSet<string> visited = new HashSet<string>();
    Dictionary<string, string> cameFrom = new Dictionary<string, string>();
```

```
        int step = 1;

        Console.WriteLine($"Bước 0: L = [{startNodeId}]");

        while (openList.Count > 0)
        {
            Node currentNode = openList[0];
            openList.RemoveAt(0);

            Console.WriteLine($"Bước {step}: u = {currentNode.Id}");
            step++;

            if (currentNode.Id == goalNodeId)
            {
                Console.WriteLine($"Bước {step}: u == GOAL -> Thành công (success)");
                PrintPath(cameFrom, goalNodeId, startNodeId);
                return;
            }

            visited.Add(currentNode.Id);

            List<Edge> neighbors = graph.GetNeighbors(currentNode.Id);
            Console.WriteLine($"Bước {step}: Neighbors of {currentNode.Id} =
[{string.Join(", ", neighbors.Select(e => e.ToNodeId))}]");
            step++;
```

```
            List<Node> tempList = new List<Node>();

            foreach (Edge edge in neighbors)
            {
                Node neighbor = graph.GetNode(edge.ToNodeId);

                if (!visited.Contains(neighbor.Id) && !openList.Contains(neighbor) &&
!tempList.Contains(neighbor))
                {
                    tempList.Add(neighbor);
                    cameFrom[neighbor.Id] = currentNode.Id;
                }
            }

            tempList = tempList.OrderBy(n => n.Heuristic).ToList();

            openList.InsertRange(0, tempList);

            Console.WriteLine($"Bước {step}: L = [{string.Join(", ", openList.Select(n =>
n.Id))}] (với các node hàng xóm đã sắp xếp ở đầu)");
            step++;
        }

        Console.WriteLine("Thất bại (failure) - không tìm thấy đường đến đích");
    }
```

```
    static void PrintPath(Dictionary<string, string> cameFrom, string goalNode, string startNode)
    {
        if (!cameFrom.ContainsKey(goalNode) && goalNode != startNode)
        {
            Console.WriteLine("Không thể xây dựng đường đi!");
            return;
        }

        List<string> path = new List<string>();
        string current = goalNode;

        while (current != null)
        {
            path.Add(current);
            if (current == startNode) break;
            current = cameFrom.ContainsKey(current) ? cameFrom[current] : null;
        }

        path.Reverse();

        Console.WriteLine($"Đường đi: {string.Join(" -> ", path)}");
    }
}

class Graph
```

```
{
    private Dictionary<string, Node> nodes = new Dictionary<string, Node>();
    private Dictionary<string, List<Edge>> edges = new Dictionary<string,
List<Edge>>();

    public void AddNode(string id, double heuristic)
    {
        nodes[id] = new Node(id, heuristic);
        edges[id] = new List<Edge>();
    }

    public void AddEdge(string fromNodeId, string toNodeId, double cost)
    {
        edges[fromNodeId].Add(new Edge(fromNodeId, toNodeId, cost));
    }

    public Node GetNode(string id)
    {
        return nodes[id];
    }

    public List<Edge> GetNeighbors(string nodeId)
    {
        return edges[nodeId];
    }
```

```
    public bool HasNode(string id)
    {
        return nodes.ContainsKey(id);
    }

    public void PrintGraph()
    {
        Console.WriteLine("Các node và giá trị heuristic:");
        foreach (var node in nodes.Values)
        {
            Console.WriteLine($"Node {node.Id}: h = {node.Heuristic}");
        }

        Console.WriteLine("\nCác cạnh và trọng số:");
        foreach (var nodeId in edges.Keys)
        {
            foreach (var edge in edges[nodeId])
            {
                Console.WriteLine($"{edge.FromNodeId} -> {edge.ToNodeId}: cost =
{edge.Cost}");
            }
        }
    }
}

class Node
```

```
{
    public string Id { get; private set; }
    public double Heuristic { get; private set; }

    public Node(string id, double heuristic)
    {
        Id = id;
        Heuristic = heuristic;
    }

    public override bool Equals(object obj)
    {
        if (obj is Node other)
        {
            return Id == other.Id;
        }
        return false;
    }

    public override int GetHashCode()
    {
        return Id.GetHashCode();
    }
}

class Edge
```

```
    {
        public string FromNodeId { get; private set; }
        public string ToNodeId { get; private set; }
        public double Cost { get; private set; }

        public Edge(string fromNodeId, string toNodeId, double cost)
        {
            FromNodeId = fromNodeId;
            ToNodeId = toNodeId;
            Cost = cost;
        }
    }
}
```