Họ tên: Đỗ Thành Nhơn
MSSV: 1512387
Lớp: CNTN2015

CƠ SỞ TRÍ TUỆ NHÂN TẠO
BÁO CÁO ĐỒ ÁN 1
BÀI TẬP 2

Giảng viên lí thuyết: Lê Hoài Bắc
Giảng viên hướng dẫn thực hành:
Lê Ngọc Thành
Nguyễn Ngọc Thảo
Nguyễn Hài Minh

Các mốc thực hiện đồ án:

Giai đoạn	Công việc
1	Tìm hiểu các thuật toán tìm kiếm. Tìm hiểu ngôn ngữ lập trình hỗ trợ giao diện đồ hoạ (C# .NET)
2	Thiết kế cấu trúc dữ liệu lưu trữ và thuật toán sử dụng.
3	Tiến hành cài đặt.
4	Viết các Test case, tiến hành kiểm tra và sửa lỗi (nếu có).
5	Viết báo cáo.

I. Mô tả thiết kế lưu trữ bản đồ và các thông tin liên quan.

1. Lưu trữ bản đồ.

- Ta chỉ lưu trữ toạ độ các đỉnh của đa giác và toạ độ của hai đỉnh cần tìm đường đi.
- Trong quá trình thêm các đỉnh, ta sẽ lưu tại 4 giá trị mô tả hình chữ nhật nhỏ nhất có thể bao hết các đa giác và hai điểm cần tìm đường đi để xác định kích thước bản đồ.

2. Thiết kế các lớp đối tượng.

```
class Line
{
    // Hệ số của phương trình đường thẳng
    Int64 a;
    Int64 b;
    Int64 c;
    // Hàm tính UCLN của 2 số
    private Int64 getUCLN(Int64 p1, Int64 p2);
    // Constructor
    public Line(KeyValuePair<int, int> p1, KeyValuePair<int, int> p2);
    // Tính giá trị f(x. y)
    public Int64 CalculatePoint(KeyValuePair<int, int> p);
    // Kiểm tra đường thẳng cắt nhau
    public bool LineIntersect(KeyValuePair<int, int> p1, KeyValuePair<int, int> p2);
    public bool LineIntersect1(KeyValuePair<int, int> p1, KeyValuePair<int, int> p2);
    // TÍnh khoảng cách điểm tới đường thẳng
    public double DistancePointToLine(KeyValuePair<int, int> p);

    ~Line();
}
```

```
class Polygon
{
    // Danh sách các dỉnh
    private List<KeyValuePair<int, int>> arr;
    // Constructor
    public Polygon();
    // Constructor
    public Polygon(List<KeyValuePair<int, int>> p);
    // Constructor
    public Polygon(Polygon p);
    // Destructor
    ~Polygon();
    // Thêm đỉnh vào đa giác
    public void AddVertex(int x, int y);
    // Kiểm tra đường thẳng cắt đa giác
    public bool LineIntersectPolygon(KeyValuePair<int, int> p1, KeyValuePair<int, int> p2);
    // Tính diện tích tam giác
    private Int64 TriangleSquare(KeyValuePair<int, int> p1, KeyValuePair<int, int> p2, KeyValuePair<int, int> p3);
    // Tính diện tích đa giác
    private Int64 PolygonSquare();
    // Kiểm tra điểm nằm trên đa giác
    public bool PointInsidePolygon(KeyValuePair<int, int> p);
    // Kiểm tra 2 đa giác giao nhau
    public bool PolygonIntersectPolygon(Polygon p);
    // Khoảng cách ngắn nhất từ đa giác tới đường thẳng
    public double getMinDistanceLineToPolygon(Line l);
    // Di chuyển đa giác
    public void MovePolygon(int dx, int dy);
    // Lấy danh sách đỉnh
    public List<KeyValuePair<int, int>> getPoints();
    // Vẽ đa giác
    public void DrawPolygon(Graphics gp);
}
```

```
class Graph
{
    // Điểm đầu và kết thúc
    private KeyValuePair<int, int> source;
    private KeyValuePair<int, int> goal;
    // Danh sách đa giác
    private List<Polygon> arrPolygons;
    // Danh sách các đỉnh của đồ thị
    private HashSet<KeyValuePair<int, int>> arrVertexs;
    // Mảng đánh dấu đường đi
    int[] trace;
    // Constructor
    public Graph();
    // Constructor
    public Graph(KeyValuePair<int, int> s, KeyValuePair<int, int> g, List<Polygon> arr)
    // Destructor
    ~Graph();
    // Tính khoảng cách 2 điểm
    private double getPointDistance(KeyValuePair<int, int> p1, KeyValuePair<int, int> p
    // Kiểm tra điểm có thuộc đa giác
    bool PointInsidePolygons(KeyValuePair<int, int> p);
    // Tìm những điểm xung quanh đỉnh đa giác hợp lệ
    void getCandidateVertex();
    // Kiểm tra đường thẳng nói 2 điểm
    private bool LinePointInt(KeyValuePair<int, int> p1, KeyValuePair<int, int> p2);
    // Tính trọng số giữa 2 đỉnh
    private double getWeight(KeyValuePair<int, int> p1, KeyValuePair<int, int> p2);
    // Xây dựng độ thị
    private double[,] BuildGraph();
    // Tính heiristic
    private double[] getHeuristic();
    // THuật toán tìm đường
    public void Astar();
    // Tìm đường khi vật cản di chuyển
    public void Move();
    // Tìm đường đi từ mảng đánh dấu
    private Point[] getPath();
    // Hoán vị
    private void swap(ref int a, ref int b);
    // Thuật toán midpoint
    public void midpointline(Point p1, Point p2, Graphics gp);
    // Vẽ bản đồ là kết quả
    public void Draw(Graphics gp);
    // Đọc file input
    public void SetInput(string fileName);
    // Xuất file output
    public void SaveOutput();
}
```

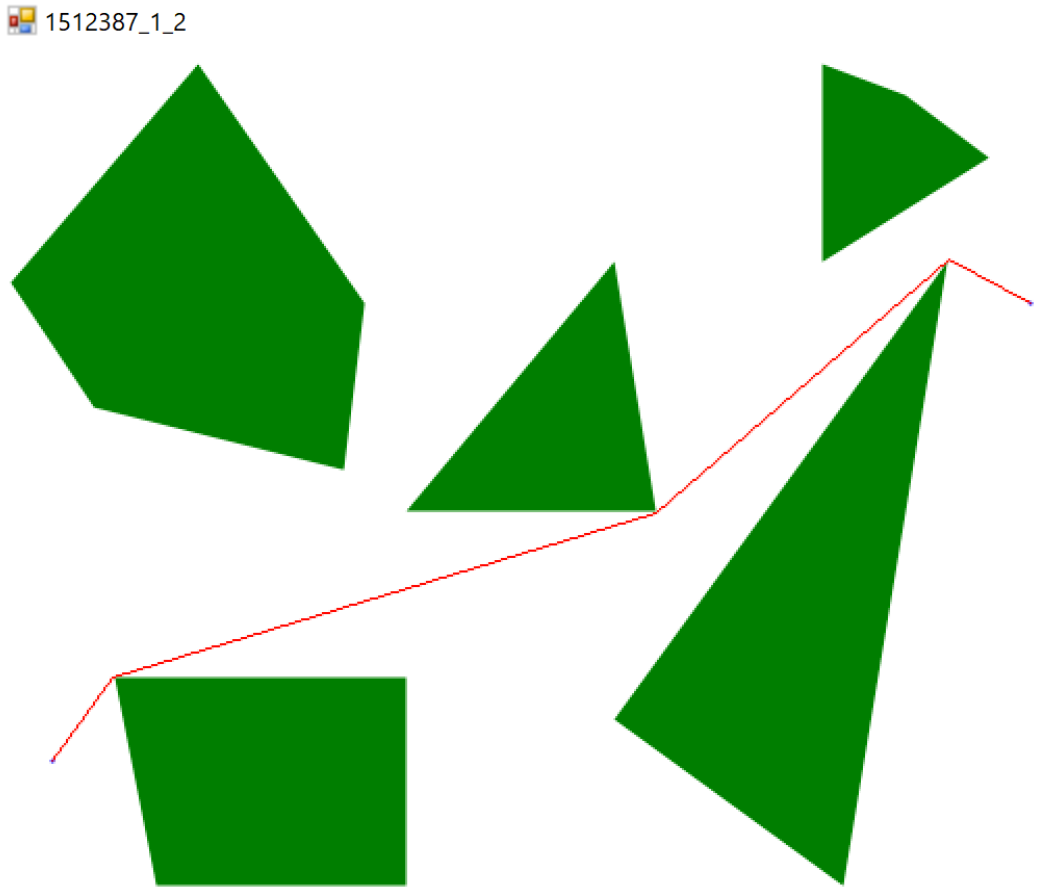
```
public class Vertex : IComparable<Vertex>
{
    // Đỉnh
    public int v;
    // Chi phí f = g + h
    public double f;
    // Constructor
    public Vertex(int _v, double _f);
    // Destructor
    public Vertex(Vertex p);
    // Tiêu chí so sánh để dùng SortSet
    public int CompareTo(Vertex other);
}
```

3. Thuật toán sử dụng và mã giả.
   - ❖ Ý tưởng:
     - Từ một đồ thì vô hướng xác định với số lượng đỉnh hữu hạn, ta luôn có thể tìm được đường đi ngắn nhất từ đỉnh s tới đỉnh g (nếu có) thông qua việc đi qua các cạnh nối các đỉnh trong đồ thị.
     - Nếu ta coi mỗi đỉnh của đa giác và 2 điểm cần tìm đường đi là đỉnh của đồ thị, các cạnh được xây dựng sao cho không cắt bất kì vào một đa giác nào. Thì hoàn toàn có thể tìm được đường đi ngắn nhất (nếu có) giữa 2 đỉnh trên đồ thị này.
       Tuy nhiên thuật toán trên gặp 2 vấn đề như sau:
     - Ta không được phép đi lên đỉnh cũng như cạnh của các đa giác.
     - Toạ độ tại mỗi điểm (pixel) phải là số nguyên.
   - ❖ Giải pháp:
     - Tại mỗi đỉnh của đa giác, ta xét 8 đỉnh xung quanh, nếu đỉnh mà không nằm trong bất kì một đa giác nào thì ta đánh dấu lại. Sau đó ta xây tự đồ thị từ những điểm vừa đánh dấu. Cạnh của đồ thị được nối khi tồn tại một đường đi toạ độ nguyên và không cắt bất kì đa giác nào giữa 2 đỉnh.
     - Khi đi theo đường chéo từ điểm $A(x1, y2)$ tới điểm $B(x2, y2)$, nếu $dx = |x1 - x2| \neq |y1 - y2| = dy$ thì trên đường thẳng sẽ có những điểm (pixel) có toạ độ là số thực. Khi đó để cho toạ độ các điểm là số nguyên thì ta phải đi $min(dx, dy)$ bước theo đường chéo và đi $max(dx, dy) - min(dx, dy)$ bước theo phương ngang (dọc). Ta sẽ sử dụng thuật toán vẽ đường thẳng MidPoint để tính toạ độ là số nguyên của tất cả những điểm giữa hai điểm A và B cho trước.
     - Áp dụng thuật toán tìm đường đi A* trên đồ thị đã xây dựng được.
     - Vấn đề chạy Real-Time khi các đa giác di chuyển. Ta xem điểm vừa mới đi được là điểm bắt đầu (s), di chuyển ngẫu nhiên các đa giác và sử dụng thuật toán A*. Quá trình kết thúc khi s trùng với g hoặc không tồn tại đường đi.

```
public void Astar()
    {
        // Xây dựng đồ thị
        double[,] a = this.BuildGraph();
        // Tính giá trị hèuistic
        double[] heuristic = this.getHeuristic();
        // Khai báo mảng lưu giá trị f=g+h và khởi tạo giá trị
        double[] d = new double[n];
        for (int i = 0; i < n; ++i)
            d[i] = double.MaxValue;
        trace[n - 2] = n - 2;
        d[n - 2] = 0;
        // Dùng SortSet cảu C# có chức năng tương tự Priority_Queue
        SortedSet<Vertex> arr = new SortedSet<Vertex>();
        // Thêm đỉnh đầu vào SortedSet
        arr.Add(new Vertex(n - 2, heuristic[n - 2]));
        // Trong khi SortedSet chưa rỗng
        while (arr.Count > 0)
        {
            // Lấy đỉnh có giá trị f nhỏ nhất ra
            Vertex tmp = arr.First();
            arr.Remove(tmp);
            int u = tmp.v;
            double w = tmp.f - heuristic[u];
            if (Math.Abs(w - d[u]) > 0.0001)
                continue;
            // Nếu đó là đỉnh đích thì dừng quá trình duyệt
            if (u == n - 1)
                return;
            // Duyệt các đỉnh kề với u
            for (int v = 0; v < n; ++v)
            {
                // Nếu tại v, đường đi từ u đến v ngắn hơn thì cập nhật
                if (a[u, v] > 0 && d[v] > d[u] + a[u, v])
                {
                    d[v] = d[u] + a[u, v];
                    arr.Add(new Vertex(v, d[v] + heuristic[v]));
                    trace[v] = u;
                }
            }
        }
    }
```

Ví dụ:

```
5
5 100 5 180 120 170 200 50 170 10 110
3 300 100 200 220 320 220
4 400 5 440 20 480 50 400 100
3 300 320 410 400 460 100
4 60 300 80 400 200 400 200 300
500 120
30 340
```



II. Hướng dẫn chạy chương trình.

Ngôn ngữ sử dụng: C# .NET.

Phiên bản Visual Studio: 2017.

Khi chạy chương trình, cần để file Input ở thư mục cùng cấp tới file exe tới tên là input.txt.

Câu trúc file input.txt:

Dòng đầu chứa số nguyên n: số đa giác.

N dòng tiếp theo, mỗi dòng chứa số nguyên m là các số đỉnh của đa giác là m cặp số xi, yi là toạ độ của mỗi đỉnh.

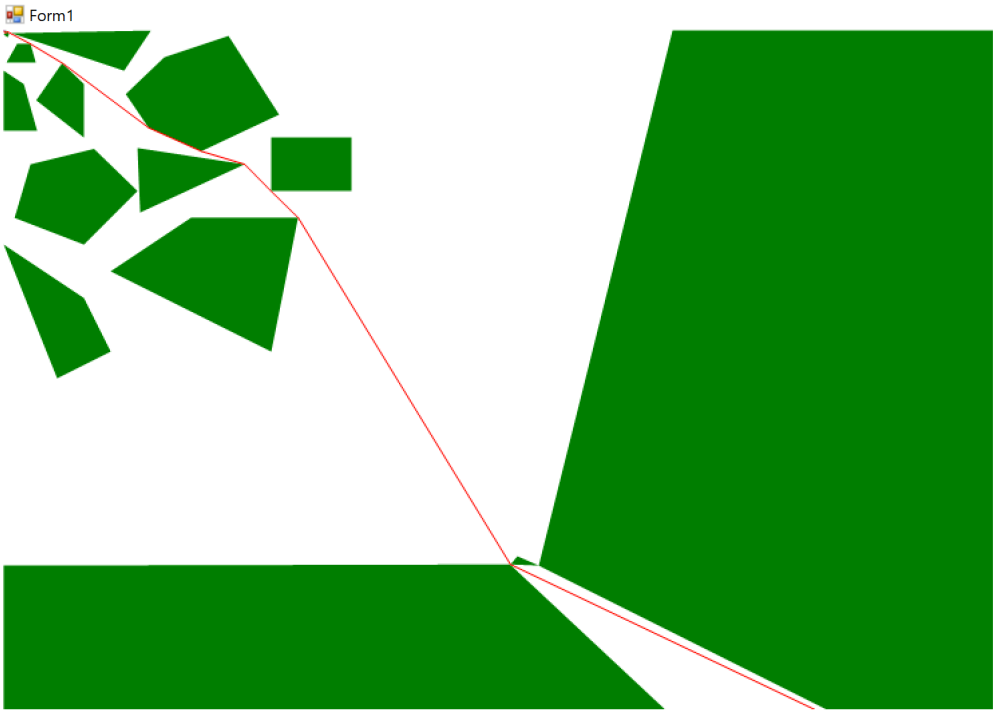
Hai dòng cuối cùng, mỗi dùng chứa 1 cặp số x, y là toạ độ của điểm đầu và điểm đích.

Khi chương trình kết thúc sẽ xuất ra file output.txt ghi lại kết quả:

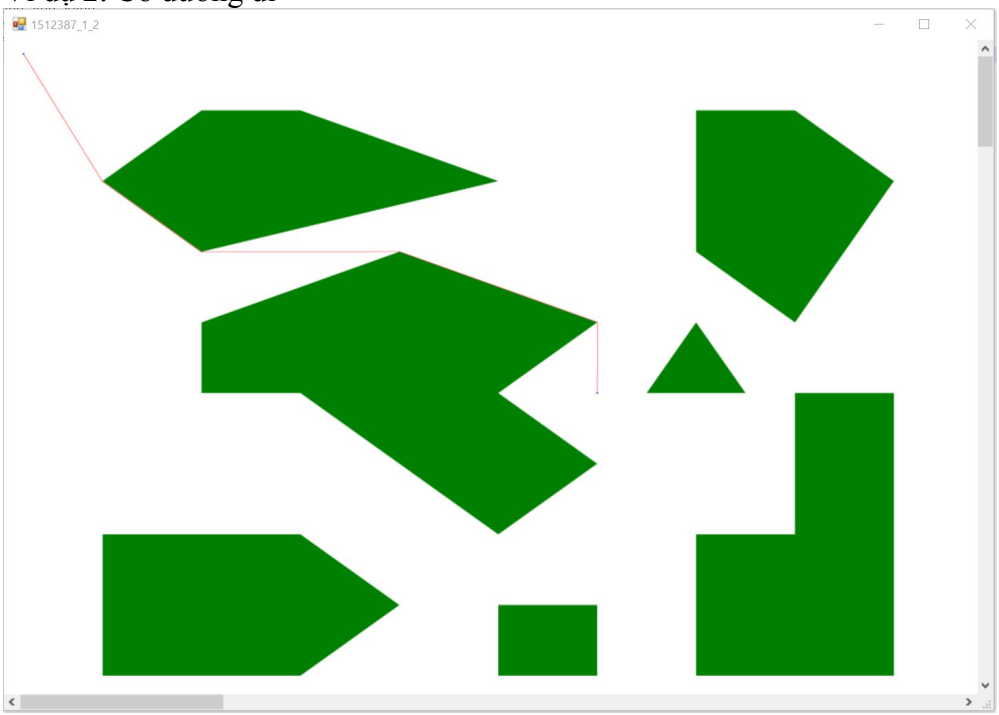
Dòng đâu: "YES" nếu có đường đi hoặc "NO" nếu không tồn tại đường đi.

Dòng thứ 2: Tổng quãng đường tính theo khoảng cách Euclide.

Ví dụ 1: Có đường đi



Ví dụ 2: Có đường đi



Ví dụ 3: Không có đường đi