Guidelines – Read Carefully! Please check each problem for problem-specific instructions. You are provided a zip file containing a single folder named xyz007 with the following folders and files:

xyz007
xyz007/SPR
xyz007/SPR/visualize.py
xyz007/SPR/spr.py
xyz007/group.txt

You should first rename the folder name xyz007 with your NETID. If you are working in a team, the folder should be named with both group members' NETIDs, e.g., NETID01NETID02. For assurance, also put the NETID(s) in the file group.txt. When you are ready to submit, remove any extra files (e.g., some python interpreter will create .pyc files) that are not required and zip the entire folder. The zip file should also be named with your NETID as NETID.zip (or NETID01NETID02.zip for groups with two members).

You are required to write your program adhering to Python 2.7 standards. In particular, DO NOT use Python 3.x. Beside the default libraries supplied in the standard Python distribution, you may use ONLY numpy and matplotlib libraries.

As mentioned in class, you may form groups of up to two people. Only a single student needs to submit per group.

Problem 1 [110 points]. Shortest Path Roadmap algorithm for a translating square.

You will be implementing the Shortest Path Roadmap algorithm for a point robot. The robot resides in a region with the lower left corner being $(0, 0)$ and the upper right corner being $(10, 10)$. There are multiple polygonal obstacles in the region. An example of the obstacles are provided in the file `env_01.txt`. In the file, each line represents a list of clockwise arranged x-y coordinates that define a polygonal obstacle. To see a visualization of the environment, you may run the following code:

```
python visualize.py env_01.txt
```

You are also given a skeleton file `spr.py` to work with. The current code takes in as arguments a file describing the polygonal obstacles, and coordinates for the start and goal configurations. For example, you should be able to run the command

```
python spy.py env_01.txt 1.0 2.0 3.0 4.0
```

You are to implement the Shortest Path Roadmap algorithm following the steps listed below.

1. **Compute the reflexive vertices [20 points]**. You are to implement the function

   `findReflexiveVertices(polygons)`

   to identify all reflexive vertices. The vertices should be returned as a list (see `spr.py` for more details).

2. **Compute roadmap edges and their lengths [30 points]**. You are to check for each pair of reflexive vertices whether the edge between them should be part of the shortest path roadmap. As the valid edges are identified, you are to build the roadmap (graph). To store the map, first assign each reflexive vertex a label (e.g., $1, 2, 3, \ldots$). You can then represent the roadmap as adjacency lists. As the outcome, you should provide two dictionaries. The first dictionary should map a vertex label to vertex coordinates, e.g.,

`{1: [5.2,6.7], 2: [9.2,2.3], ...}`

In the above dictionary, vertex 1 has a coordinate of $(5.2, 6.7)$ and vertex 2 has a coordinate of $(9.2, 2.3)$. The second dictionary should map a vertex label to a list of vertices that are adjacent to that vertex and also store the lengths of the edges, e.g.,

`{1: [[2, 5.95], [3, 4.72]], 2: [[1, 5.95], [5, 3.52]], ...}`

In the above dictionary, vertex 1 is adjacent to vertices 2 and 3. Vertex 2 is adjacent to verices 1 and 5. The distance between vertices 1 and 2 is 5.95 and the distance between vertices 1 and 3 is 4.72. The distance between vertices 2 and 5 is 3.52. Your code should go in the function

`computeSPRoadmap(polygons, reflexVertices)`

The argument `reflexVertices` is the list of reflexive vertices returned from the function `findReflexiveVertices()`

3. **Implement an uniform-cost search algorithm [30 points]**. You are to implement the uniform-cost search algorithm that we have covered in class to work on the adjacency list from the previous step. Note that this is a standard uniform-cost search algorithm that should work with any edge-weighted graph. Your code should go in the function

   `uniformCostSearch(adjListMap, start, goal)`

   where `adjListMap` has the same structure as the adjacency list mentioned in the previous task, I.e., it has the format

   `{1: [[2, 5.95], [3, 4.72]], 2: [[1, 5.95], [5,3.52]], ...}`

   The arguments `start` and `goal` are the vertex labels of the start and goal.

4. **Putting things together [20 points]**. You are to now add the start and the goal to the roadmap graph (i.e., the adjacency list) and perform the search. You need to implement the function

   `updateRoadmap(adjListMap, x1, y1, x2, y2)`

   to add the start and goal vertices to the roadmap and the call the uniform-cost search function to complete the search. Your function should return a list of vertex labels as the path.

5. **Visualization bonus [10 points]**. Through modifying the visualization code supplied in `visualize.py`, draw the roadmap (using green lines, including the start and the goal) and the path that you computed (using red lines).

