**Guidelines – Read Carefully!** Please check each problem for problem-specific instructions. For the submission of this machine problem assignment, you should create a top-level folder named with your NETID, (e.g., XYZ007 for a single person group and XYZ007-ABC999 for a two-person group). Then, for each machine problem in a given assignment, you should create a sub-folder for that problem named with the problem number. For example, for this MP set, which has two problems, you should have the following folder structure if you are a single person group:

xyz007
xyz007/SPR
xyz007/SPR/visualize.py
xyz007/SPR/spr.py

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 XYZ007.zip or XYZ007-ABC999.zip. For this particular assignment, the folder structure is already created for you in the accompanied zip file; you just need to rename the top folder as instructed.

You are required to write your program adhering to Python 3.7 standards. Specifically, we will grade only using python 3.7.4. Beside the default libraries supplied in the standard Python distribution, you may use ONLY numpy, matplotlib for this MP assignment.

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 point.**

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 is 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 spr.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. Do not change code (function signature) that are provided to you for grading purpose. If you do, you may receive 0 credit.

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

   `findReflexVertices(polygons)`

   to identify all reflex vertices. The vertices should be returned as a list (see `spr.py` for more details). You may assume that there are only polygons (no lines or points) .

2. **Compute roadmap edges and their lengths [30 points].** You are to check for each

 

pair of reflex 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 reflex 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 reflex vertices returned from the function

`findReflexVertices()`

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

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

   to add the start and goal vertices to the roadmap so later the uniform-cost search function could complete the search. Your function should return a list of vertex labels as the path. You may assume the start and goal vertices are not inside of a polygon.

4. **Implement a 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.

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).

 