School of Computer Science and Applied Mathematics
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

COMS3005A: Advanced Analysis Algorithms
Assignment 1 (Part 1)

Sheng Yan Lim

Semester II, 2019

1 Aim

This assignment is intended to give you some exposure to the experimental nature of Computer Science - specifically the concept of measuring the performance of an algorithm and relating these measurements to the theoretical analysis of the said algorithm.

2 Group Registration

Visit Moodle to register your group. A reminder that the maximum number of students per group is 2.

3 Assignment Topic: A^* search

In this assignment, you are required to study the A^* search algorithm for path finding, perform theoretical and empirical analysis by implementing the algorithm with the use of probabilistic roadmap graphs.

3.1 Probabilistic Roadmaps (PRMs)

PRMs are a standard approach to path planning; that is, finding a path from a start point to a goal while avoiding obstacles. In this assignment, by applying what you know and following the guides in this document, you will implement your own version of the PRM algorithm.

3.1.1 PRM Algorithm

The algorithm for creating a PRM can be split into 6 steps, 5 algorithmic ones and one pre-processing step. These steps can be summarised as follows:

1. Create a map with obstacles, a start point and an end (goal) point.
2. Generate n random points on the map.
3. Check if the points collide with any of the obstacles and remove them if they do.
4. Connect each point with its k-nearest neighbouring points (create a graph).
5. Check if the edges between points collide with any obstacles and remove them if they do.
6. Find a path connecting the start and end point.

The following describes each of the above steps in more detail.

Step 1:

Consider a 2-dimensional map. We begin by adding our start point, end point and some rectangular obstacles with coordinates describing the top left and bottom right corners. A sample plot of a map is shown below in Figure 1. The descriptions of the obstacles are as follow:

```
20,10;20,50
20,50;90,50
30,30;40,40
70,10;75,30
```

Note that a map displayed is not necessary for the running of the algorithm and is simply for visualisation purposes. Constructing a full map would decrease efficiency and should not be done during empirical and complexity analysis.

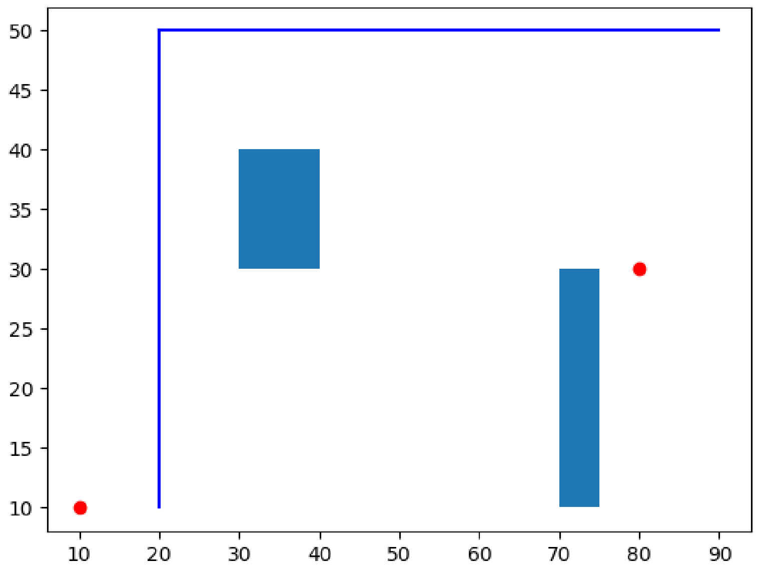

Figure 1: Map displaying start point, end point and obstacles

Step 2:

Step 2 entails sampling or generating n random points in the range of the map. These random points can be drawn from a uniform distribution and can be generated without consideration of obstacles (for now). Figure 2 shows the result of sampling the map. In this case, there are 300 points randomly generated across the map.

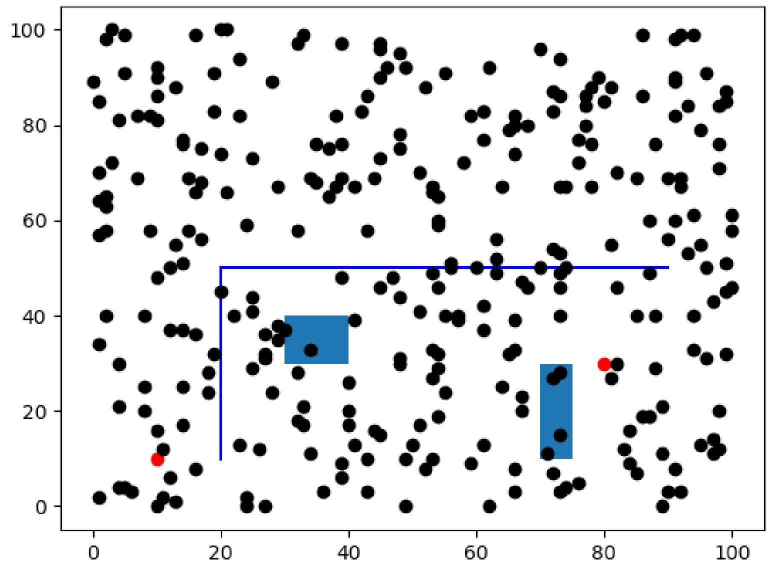

Figure 2: Generate sample points

Step 3:

Next we check if any of the random samples overlap with any of the obstacles. If they do, remove them. The resultant map would be as follows:

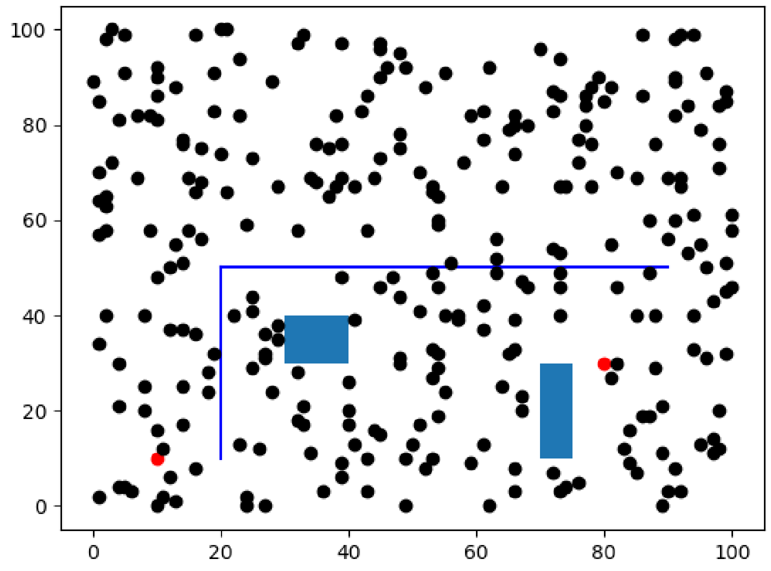

Figure 3: Remove samples colliding with obstacles

Step 4:

Connect every remaining dot with its k-nearest neighbours. You would do this using the k-Nearest Neighbour (kNN) algorithm and the result would look as follows:

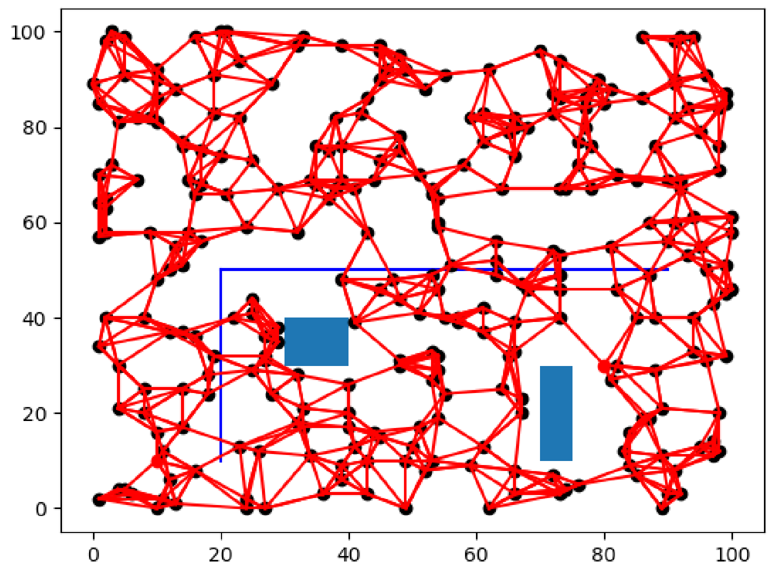

Figure 4: k-Nearest Neighbours

Note that in this implementation the points are connected without consideration of whether they collide; fusing steps 4 and 5 is easier and more computationally efficient.

Step 5:

We now remove the edges that collide with obstacles. This results in a graph with vertices and edges that do not collide with any obstacles. Note that the graph is now not guaranteed to be connected. However, if an appropriate n is chosen, it is highly probable that the graph will contain a path from the start to the end point. Figure 5 shows the result of removing colliding edges.

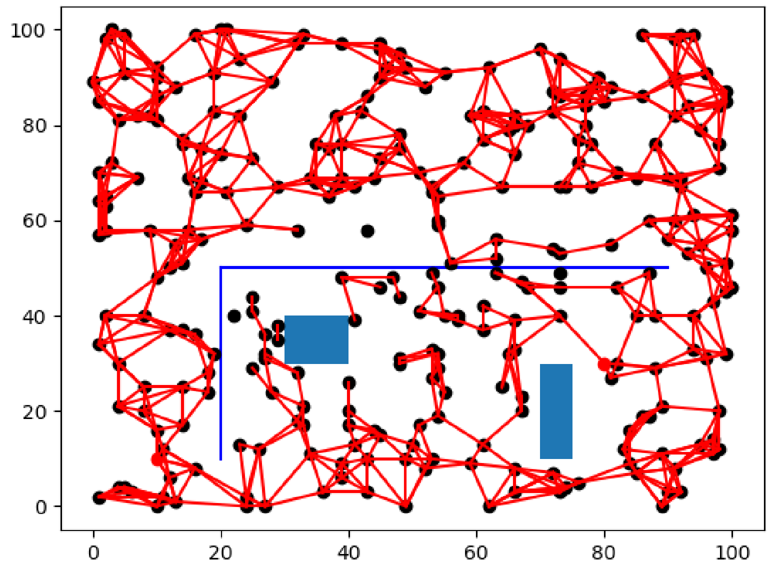

Figure 5: Remove edges colliding with obstacles

(Hint: use line intersections)

Step 6:

Finally, we find a path through the graph. This can be done using DFS, BFS or A*. We should be left with the shortest safe path available in the graph between the start and end points, which can be smoothed as is necessary. In our case, the shortest path through the graph is as shown below.

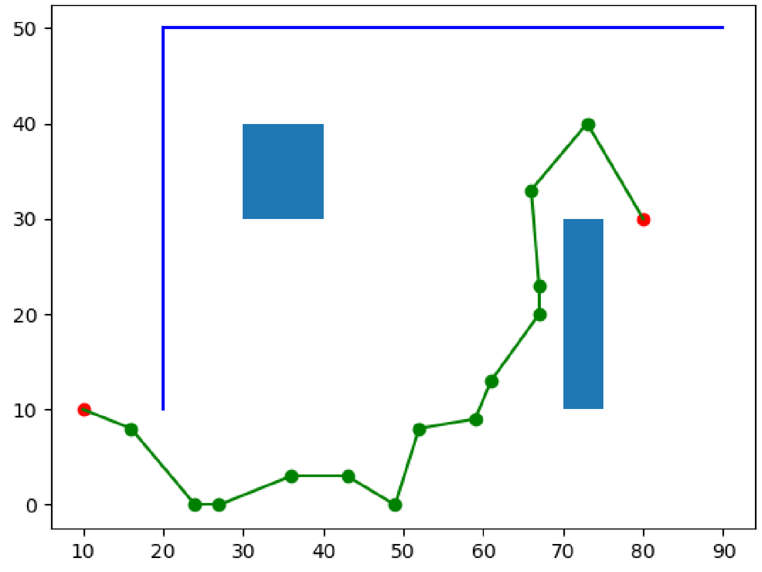

Figure 6: Shortest Path

4 Instructions

This assignment are broken down into three tasks.

4.1 Task 1

Deadline: 7th August 2019 at 23:59

You are required to construct the kNN connected graph using PRM as described above and submit your solution for marking.

Input

The first line of the input consists of k, p, n and d where k is the number of neighbours, p is the number of obstacles, n is the number of sample points and d is the dimension of the $d \times d$ map. The next two lines consist of the start and end points. The following p lines contain the top left and bottom right corner points of rectangular obstacles. The last n lines contain the sample points. All points are separated by a semicolon and the $x - y$ points is separated by a comma.

```
k p n d
start point
end point
obstacle point
...
obstacle point
sample point
...
sample point
```

Output

Output an adjacency matrix of the kNN connected graph where 1 indicates an edge exists between two points, 0 otherwise. Note that the start and end points must be included in the adjacency matrix. They should appear as node 0 and 1 respectively.

Sample Input

```
5 2 12 25
2,15
18,1
5,12;6,17
12,3;15,4
1,1
1,12
2,3
3,2
17,20
21,10
21,14
22,5
3,13
16,6
```

```
10,11
7,8
```

Sample Output

```
0 0 0 1 1 1 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1 1 1 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 1 1
1 0 0 0 1 1 0 0 0 0 0 0 1 1
0 0 1 1 0 1 0 0 0 0 0 0 1 1
0 0 1 1 1 0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 1 1 0 0 0 1 1
0 1 0 0 0 0 1 0 1 1 0 0 1 0
0 1 0 0 0 0 1 1 0 1 0 0 1 0
0 1 0 0 0 0 0 1 1 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 1 0 0 0 0 0 1
1 0 0 1 1 1 0 0 0 0 0 0 1 0
```

4.2 Task 2

Deadline: 14th August 2019 at 23:59

Implement

1. breadth first search for the start node and output the traverse sequence.
2. dijkstra's algorithm to find the shortest path from the start point to the end point. Use the manhattan distance function to calculate the distance between each point.

Input

The input contains the start and end points on the first two lines and n the number of points, followed by the adjacency matrix.

Output

Output the path determined by breadth first search as a sequence of points starting from the start point on the first line. In the next line, output the shortest path sequence calculated using dijkstra's algorithm starting from the start point to the end point. Lastly output the distance of the above shortest path on the third line.

Sample Input

```
2,15
18,1
14
0 0 0 1 1 1 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1 1 1 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 1 1
1 0 0 0 1 1 0 0 0 0 0 0 1 1
0 0 1 1 0 1 0 0 0 0 0 0 1 1
0 0 1 1 1 0 0 0 0 0 0 0 1 1
```

```
0 0 0 0 0 0 0 1 1 0 0 0 1 1
0 1 0 0 0 0 1 0 1 1 0 0 1 0
0 1 0 0 0 0 1 1 0 1 0 0 1 0
0 1 0 0 0 0 0 1 1 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 1 0 0 0 0 0 1
1 0 0 1 1 1 0 0 0 0 0 0 1 0
```

Sample Output

```
0 3 4 5 13 12 2 7 1 6 8 9
0 3 12 7 1
38
```

4.3 Task 3

Deadline: 30th August 2019 at 23:59

See part 2 of the assignment.

5 Submissions

For each task, you are required to submit a zipped file containing the main java program and any other relevant files. The name of the main java program has to be *Program.java* not *Main.java* in order for the marker to mark correctly. There are no restrictions on naming the zipped file.

Note that using code downloaded from the web is a form of **plagiarism**. Refer to the general course outline for more information