Nova Southeastern University
College of Computing and Engineering

Assignment 1

CISC 670 Artificial Intelligence

Fall 2022

Due date: 11/5/2022 11:59 PM ET

Total points: 100

Part 1. Text Reading:

Introduction to Artificial Intelligence (Chap. 1)
Intelligent Agents (Chap. 2)
Searching (Chaps. 3, 4)

Part 2. Problems:

2.1 Depth-first & breadth-first search [40 points]

The textbook shows algorithms for Depth-First Search (DFS) and Breadth-First Search (BFS) applied to tree structures. Now we extend the algorithms to graph structures.

In our definition, BFS is a graph search algorithm that begins from a node (a "root" node) and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until all the nodes within the graph are explored.

DFS is a graph search algorithm that starts from a node (a "root" node) and explores as far as possible along each branch before backtracking (going back to a node that has been explored before). For an undirected and connected graph, such a process could produce a Depth-First Search Tree, in which the starting node serves as the root of the tree. Whenever a new unvisited node is explored for the first time during the DFS search process, it is attached to the DFS tree as a child to the node from which it is being reached.

Please note that our BFS and DFS algorithms are defined in a traditional way. The algorithms shown in the textbook try to find a goal node. In our algorithms, we are trying to explore *all* the nodes within the graph.

Please use the Figure 1 and answer the questions. The figure shows an undirected graph and all the edges have equal costs (e.g., the lengths of all edges are equal). **For simplicity, when we**

have a number of alternate nodes to explore, we will select them in *alphabetical* order. The problems can be easily solved by hand.

Assume node ***a*** is the starting point (the "root" node) from which the search starts.

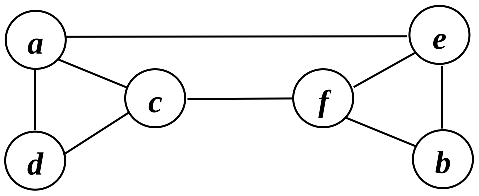

Figure 1

1) Apply the BFS algorithm and show the output (list all the nodes by the order they are being explored) [15 points]

2) Apply the DFS algorithm and show the output (list all the nodes by the order they are being explored) [15 points]

3) Show the output from the previous question in the form of a DFS tree. Edges in a DFS tree are a subset of the original edge set from which a node is visited for the first time during the DFS searching process. [10 points]

2.2: A* Search [60 points]

Note: This is a programming problem. You are required to write your own program to solve the problem, but it is free to choose any programming language you feel comfortable with. You are not required to include your source code in your submission. Please include a brief discussion on how the experiments are conducted (e.g., the data structures used, parameters you choose for the algorithms, etc), and how you get your solutions for each of the questions. Please note that you need to describe important assumptions based on which your solutions can be derived.

The problem is described as follows. Consider an idealization of a problem where a robot has to navigate its way around obstacles. The goal is to find the shortest distance between two points on a plane that has convex polygonal obstacles. Figure 2 shows an example scene with eight polygonal obstacles where the robot has to move from the point start to the point end. Convince yourself that the shortest path from one polygon vertex to any other in the scene consists of straight-line segments joining some of the vertices of the polygon. (Note that the start and the end goal points may be considered polygons of size 0).

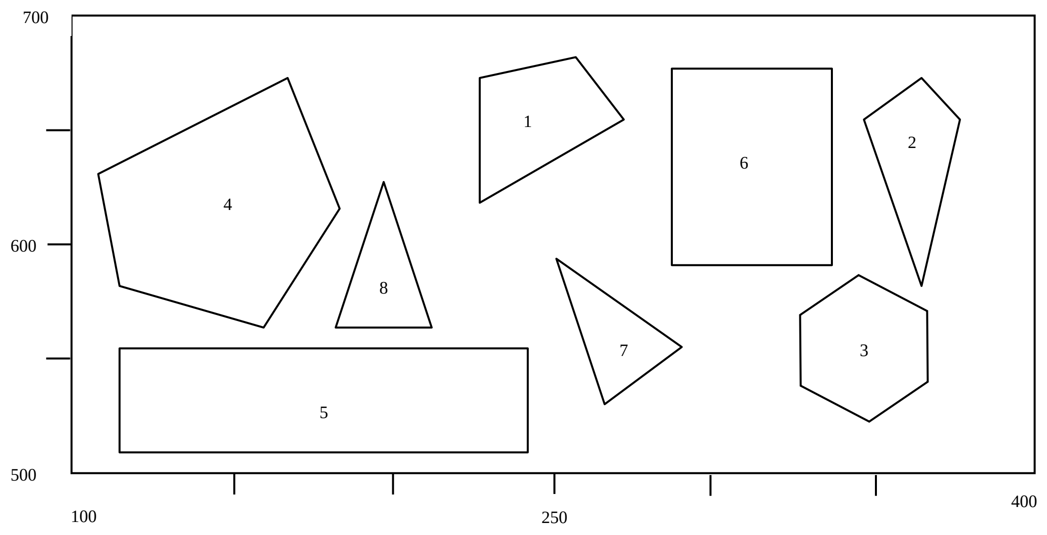

Figure 2

Polygon 1: ((220, 616), (220, 666), (251, 670), (272, 647))
Polygon 2: ((341, 655), (359, 667), (374, 651), (366, 577))
Polygon 3: ((311, 530), (311, 559), (339, 578), (361, 560), (361, 528), (336, 516))
Polygon 4: ((105, 628), (151, 670), (180, 629), (156, 577), (113, 587))
Polygon 5: ((118, 517), (245, 517), (245, 577), (118, 557))
Polygon 6: ((280, 583), (333, 583), (333, 665), (280, 665))
Polygon 7: ((252, 594), (290, 562), (264, 538))
Polygon 8: ((198, 635), (217, 574), (182, 574))

Note: This figure is for illustration purpose only. Positions of the polygons inside the figure may not reflect their actual coordinates.

1) Suppose the state space (i.e., a set of possible locations for the robot) consists of all possible positions *(x, y)* in the plane. How many possible states are there? How many paths are there to the goal? Provide assumptions for your answer. [10 points]

2) Based on your solution for question 1), can you find a better (and reasonable) state space for the problem? If the answer is yes, how large is the state space? If the answer is no, provide justifications why you think this is a good state space to implement your searching algorithm. [10 points]

3) Implement an algorithm to find the shortest path from the start node to the end node using an A* (A-star) heuristic search. Use the straight-line distance to the end node as a heuristic function. **Show your pseudo code for this algorithm**. Is this an admissible heuristic function? Why or why not? [15 points]

Hint: Define the necessary functions to implement the search problem. This should include a function that takes a vertex as input and returns the set of vertices that can be reached in a straight line from the given vertex. You may need to implement a function that detects whether two-line segments intersect (the algorithm is attached). The problem can be solved using shortest path algorithms, but you are required to use A*.

4) Present the solutions for the following pair of starting point and ending point using the A* algorithm you implemented. Show the optimal path. [15 points]

Start: (115, 655) End: (380, 560)

5) Is it possible to solve the problem using a breadth-first or a depth-first search algorithm? If the answer is yes, then briefly discuss your solutions. Otherwise please explain. [10 points]

Additional Resources in A*:

http://www.edenwaith.com/products/pige/tutorials/a-star.php

http://en.wikipedia.org/wiki/A*_search_algorithm

http://theory.stanford.edu/~amitp/GameProgramming/