CSCI 561: Foundations of Artificial Intelligence
Summer 2017

Homework #1

Due on June 2, 2017 at 11:59 PM

Written Assignment (20%)

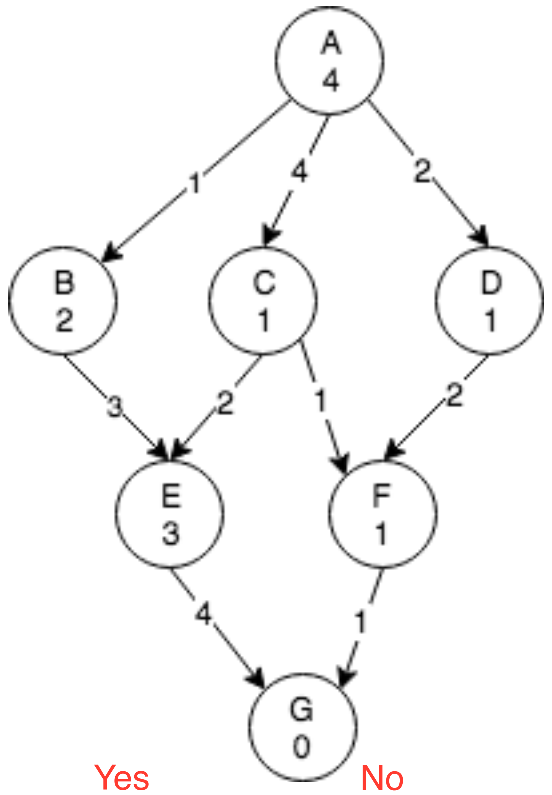
1) True/False (4.5%)

a. A perfectly rational tic-tac-toe-playing agent never loses. True
b. A perfectly rational chess-playing agent never loses. False
c. IDS always performs better than DFS. False
d. UCS is a special case of A* search. True
e. DFS is optimal in finite state space. False
f. Doubling a computer's speed allows us to double the depth of a tree search in the same amount of time. False
g. If both h1(s) and h2(s) are admissible heuristics, then $h3(s) = \alpha.h1(s) + (1-\alpha).h2(s)$ $(0 < \alpha < 1)$ is also admissible. True
h. Simulated annealing always performs better than hill climbing. False
i. Alpha-beta pruning with a good evaluation function always yield the same result as minimax algorithm. True

2) For each of the following intelligent agent activities, determine the task environment properties. (4%)

Task	Fully Observable	Single agent	Deterministic	Episodic	Static	Discrete
Playing Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Text Translation	Fully	Single	Deterministic	Episodic	Static	Discrete
Online Advertising	Partially	Multi	Stochastic	Episodic	Dynamic	Continuous
Cooking	Fully	Single	Deterministic	Sequential	Dynamic	Continuous

3) Consider the following graph. The search begins at A and the goal node is G. The cost of each edge is shown on it and the heuristic value of each node is inside it.



Yes No

a) Is the heuristic function admissible? Is it consistent? (1%)
b) Using graph search, for each of the following search algorithms, show the order in which nodes get expanded and the path found. In case of a tie, pick in alphabetical order.
   I. UCS (2%) Expanded: A -> B -> D -> C -> E -> F-> G, Path found: A -> D -> F -> G
   II. Greedy Best-First (2%) Expanded: A -> C -> D -> F -> G, Path found: A -> C -> F -> G
   III. A* (2%) Expanded: A -> B -> D -> C -> F -> G, Path found: A -> D -> F ->G

4) We have a list of ingredients and a list of scored tuples of them. Each scored tuple specifies how well the two ingredients go together: for example, (salt, beef) can have a score of 10, while (salt, juice) can have a score of -6. We want to make a dish from the available ingredients with the highest possible score.

a) Formulate the problem as a local search problem and determine how we can solve it. (2.5%)
b) Now assume scored tuples can have a score of 1, 0, or -1. If the score is 1, it means they should go together, and if it is 0, it means they should not be used together. Also, 0 means we can use them with or without each other. We want to make at most 5 dishes using all ingredients subject to the scored tuples constraints. Formulate the problem as a CSP. (2%)

Submission Instructions:

You will use Crowdmark to upload your answers to the written assignment. The link which you need to use, will be emailed to you.

a) Use a vector to represent the dish [0, 0, 1, 0, 1, 0…0]
0: The ingredient is not in the dish.
1: The ingredient is in the dish.
Initially, we generate a random vector. Then, we get its child by flipping each pos in the vector.
Select the one with highest score for the nect.

b) Set each ingredient as a variable. I1, I2, I3…In
Domain: {1,2,3,4,5} means this ingredient belongs to which dish
Restriction: Ii != Ij if (Ii, Ij) = -1

Programming Assignment (80%)

A delivery drone is an unmanned aerial vehicle (UAV) utilized to transport packages, food or other goods. In this programming assignment, you will need to implement the following search algorithms to deliver goods from a starting position S to a destination position D in an undirected weighted graph G:

- Breadth-first search
- Depth-first search
- Uniform-cost search using the amount of fuel needed as cost

The drone has an initial amount of fuel, F, and cannot refill until it reaches the destination. The nodes in G represent the places where the drone can travel to. If an edge exists between node A and node B, the drone can travel from A to B. The weight of an edge corresponds to the amount of fuel needed to traverse that edge. So, the route selection will be restricted by the amount of fuel that the drone has. For example, when the drone is in node A, and there exists a path between A and B, the drone can go to B if and only if it has the necessary amount of fuel. Otherwise, B will be considered unreachable from A.

If a path from S to D exists under the fuel constraint, you should return the path; otherwise you should return “No Path”.

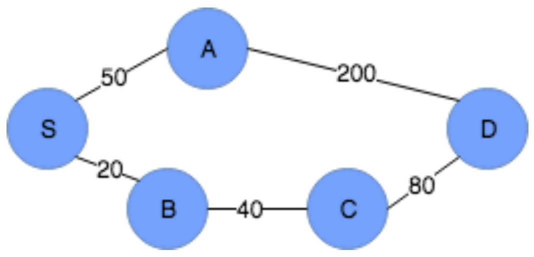


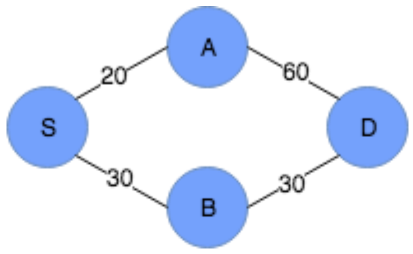
Figure 1. Sample Travelling Graph.

Note: When the costs of two or more nodes are equal, you need to make sure these nodes are popped out of the frontier in alphabetical order. This will resolve ambiguity and ensure that there is only one correct output for each input.

Sample Input and Output:

The input file will contain the search algorithm, the initial fuel amount, the starting and destination positions, and the graph representation. The output file should contain the found path (if one exists, otherwise “No Path”) and the remaining amount of fuel at the destination. We will now see this in greater detail with a few examples.

For BFS: Input and Output



Input File:

```
BFS
70
S
D
S: A-20, B-30
A: S-20, D-60
B: S-30, D-30
D: A-60, B-30
```

Output File:

```
S-B-D 10
```

In the input file, the first line indicates the algorithm to be used, which is either BFS or DFS or UCS. The second line is the initial fuel amount. In the above example, it is 70. The third and fourth lines are the start and goal node names, respectively. The rest represents the graph. For instance, in the above example "S: A-20, B-30" means node S has two neighbors, A and B, the cost between S and A is 20, and the cost between S and B is 30. You could assume the input file is always in the correct format.

The returned path in the above example should be "S-B-D", because with BFS, from S, both A and B are reachable and get added to the frontier. If we expand the nodes in alphabetical order, we expand A first and the remaining fuel at A will be 70-20=50. Note that because the remaining fuel at A is less than the amount of fuel needed to go to D from A, D does not get added to the frontier at this point. Then we expand B and the remaining fuel at B will be 70-30=40. Now we can add D to the frontier. Lastly, we extract D and the search will be done.

For DFS: Input and output

Input File 1:

```
DFS
270
S
D
S: A-50, B-20
A: S-50, D-200
B: S-20, C-40
C: B-40, D-80
D: A-200, C-80
```

Output File 1:
```
      S-A-D 20
```

Input File 2:
```
      DFS
      100
      S
      D
      S: A-50, B-20
      A: S-50, D-200
      B: S-20, C-40
      C: B-40, D-80
      D: A-200, C-80
```
Output File 2:
```
      No Path
```

Note that the graph in these two input files is the same as figure 1.

For UCS: Input and output

Input File:
```
      UCS
      270
      S
      D
      S: A-50, B-20
      A: S-50, D-200
      B: S-20, C-40
      C: B-40, D-80
      D: A-200, C-80
```
Output File:
```
      S-B-C-D 130
```

Note: Node names can be any strings consisting of only alphanumerical characters [a-zA-Z0-9]. All test cases will follow the same format, including the placement of whitespace and new-line characters. A node can have one or more connecting edges. All edge costs and the initial fuel amount will be integers greater than or equal to zero.

Grading Notice:

Please follow the instructions carefully. Any deviations from the instructions will lead your grade to be zero for the assignment. If you have any doubts, please use the discussion board on Piazza. Do not assume anything that is not explicitly stated.

- You must use **PYTHON** (Python 2.7) to implement your code. You must not use any other Python libraries besides the default libraries provided by the Python 2.7 environment. You have to implement any other functions or methods by yourself.
- You need to create a file named "**hw1cs561s17.py**". The command to run your program will be as follows: *(When you submit the homework on labs.vocareum.com, the following command will be executed automatically by the grading script.)*
  ```
      python hw1cs561s17.py –i <inputFile>
  ```
  where *<inputFile>* is the filename of the input file that your program needs to read.
- The generated output file needs to be named as "**output.txt**".
- You will use labs.vocareum.com to submit your code. Please refer to http://help.vocareum.com/article/30-getting-started-students to get started with the system. Please only upload your code to the "/work" directory. Don't create any subfolders or upload any other files.
- If we are unable to execute your code successfully, you will not receive any credits.
- You will get partial credit based on the percentage of test cases that your program gets right for each task.
- Your program should handle all test cases within a reasonable time (not more than a few seconds for each sample test case). The complexity of test cases is similar to, but not necessarily the same as, the ones provided in the assignment description.
- The deadline for this assignment is **June 2, 2017 at 11:59 PM PST**. **No late homework will be accepted.** Any late submissions will not be graded. Any emails for late submission will be ignored.