**TCSS 435**

**Homework Assignment 1**

1. **(10 Points) Dijkstra’s Shortest Path Algorithm:**

Consider the following algorithm to find the shortest distance between two cities **(S -> G)**

6.0

3.1

3.97

0.5

5.2

4.6

1.5

2.4

4.8

1.1

Source

Step 1: Maintain a list of cities C that you have visited so far.

Cache the total path-cost g(c) and the predecessor city p(c) for every city c in C

Step 2: Maintain a list of neighboring cities F (the fringe) that are not in C. C ∩ F = Ø

Cache the total path-cost g(f) and the predecessor city p(f) for every city f in F

Cities in F are ordered according to their total path cost g(f)

Step 3: At every iteration of the algorithm, starting at S, visit the city in the fringe that has the

lowest path-cost. Add it to C and remove it from F

Step 4: Add all new neighboring cities (including their g(f) and p(f)) into the fringe that are not

already in C. (It is possible to have copies in F with different predecessors).

Step 5: If more than 1 copy of a city is in the fringe, only retain the one with lowest g(f).

Delete all other copies.

Step 6: IF you reach G, reconstruct the path from S -> G and report g(G) and the path.

Answer the following questions:

* 1. [2 points] Work out the first 3 steps of the Dijkstra’s algorithm and Uniform Cost Search.

This means: visit and expand the first 3 nodes starting at 0.

Use the graph above. “0” is the source and “6” is the goal.

We can think of this problem as all nodes having a value that is set to infinity to start. As paths are taken we update the node values to the shortest distance to get to the node.

**Step One**:

Calculate Node 0 to Node 1 which is 1.1. Update the value at node 1 from infinity to 1.1.

Calculate Node 0 to Node 2 which is 4.8. Update the value at node 2 from infinity to 4.8.

We can visualize this by coloring the line 0-->1, the line 0-->2, and node 0 red.

Red nodes mean we have visited and expanded this node already. Red paths mean we shouldn’t calculate this path (0-->1 represents the line in between node 0 and node 1) distance again.

We pick to expand Node 1 next because 1.1 is < 4.8.

**Step Two**:

Calculate Node 1 to Node 3 which is 3.5. Update the value at node 3 from infinity to 3.5.

Calculate Node 1 to Node 2 which is 5.7. Do not update the shortest distance (value) to node two (4.8) with the new value since it is bigger (5.7)

We can visualize this by coloring the line 1-->3, the line 1-->2, and node 1 red.

We pick to expand Node 3 next because 3.5 is less than 4.8.

**Step Three**:

Calculate Node 3 to Node 2 which is 4. Update the value at node 2 from 4.8 to 4.

Calculate Node 3 to Node 4 which is 5. Update the value at node 2 from infinity to 5.

Calculate Node 3 to Node 6 which is 7.47. Update the value at node 6 from infinity to 7.47.

We can visualize this by coloring the line 3-->2, the line 3-->4, the line 3-->6, and node 3 red.

We pick to expand Node 2 next because 4 is less than the following: 7.47 (Node 6) and 5 (Node 4). We compare the value of all nodes that aren’t red but have a red path to them to find which node to expand next.

* 1. [2 points] Is this algorithm an instance of an informed or an uninformed search algorithm? (Explain)

**Uninformed search algorithm.** From page 87 of our book, (uninformed search algorithms) “rely on current path cost alone, rather than an estimate of the distance to the goal”. Dijkstra’s Shortest Path Algorithm makes no guesses about the distance/route between the current state and the goal state. It is a version of breadth first search (according to our book) and BFS is a uninformed search as well. Also, there is no heuristic.

* 1. [2 Points] Is this algorithm an instance of a tree search or graph search algorithm? (Explain)

**Graph search algorithm**. All expanded states are recorded.

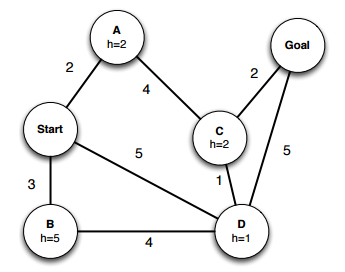
* 1. [2 points] Is this algorithm complete? (Explain)

**Yes**. Assuming best solution has a finite cost and minimum arc (weight) cost is positive (the distances in between the nodes aren’t negative). From page 85 in our book: “Uniform-cost search expands the least-cost leaf node first**.** It **is complete**, and unlike breadth-first search **is optimal** even when operators have differing costs. Its space and time complexity are the same as for breadth-first search.”

* 1. [2 points] Is this algorithm optimal? (Explain)

**Yes**. It always returns the cheapest solution according to the weights of the graph. Dijkstra’s is also an instance of Uniform Cost Graph Search and the book says that is optimal as stated above.

1. **(15 Points) Search algorithms in action:**



For each of the following graph search strategies, work out the order in which states are expanded as well as the path returned by graph search. In all cases, assume ties resolve in such a way that states with earlier alphabetical order are expanded first. The start and the goal state are S and G, respectively. Remember in graph search, a state is expanded only once.

* 1. [3 points] Depth-first search

Let Start = S and Goal = G

Step 1: (S, A), (S, B), (S, D)

Expand A next (pick based on alphabetical order)

Step 2: (S, A, C)

Expand C next (only choice for this depth)

Step 3: (S, A, C, D), (S, A, C, G)

Declare Success since we found G.

**Return Path (S, A, C, G)**

* 1. [3 points] Breadth-first search

This algorithm always expands the shallowest node in the search tree.

Step 1: (S, A), (S, B), (S, D)

S expanded because it is the start. Expand A next because of alphabetical tie breaker.

Step 2: (S, A, C)

Expand B next because it is the shallowest in the tree.

Step 3: (S, B, D)

Expand D next because it is the shallowest in the tree.

Step 4: (S, D, C), (S, D, G)

We can set are algorithm up to declare success at this point (or if we want to when we try to expand G which would mean we would need more steps for this answer)

**Return Path (S, D, G)**

* 1. [3 points] Uniform cost search

Step 1: Fill S with red and draw red on lines from S to A, S to B, and S to D.

Update value of A to 2, value of B to 3, and value of D to 5.

Expand A next because it has the lowest total cost.

Step 2: Fill A with red and draw red on the line from A to C. Update value of C to 6.

Expand B next because it has the lowest total cost.

Step 3: Fill B with red and draw red on lines from B to D. Compare value of 7 to current value of D (5). Keep old value.

Expand D next because it has the lowest total cost.

Step 4: Fill D with red and draw red on the lines from D to C and D to G. Update value of G to 10. C keeps value of 6 since it is a tie.

Expand c next because it has the lowest total cost.

Step 5: Fill C with red. Draw red on line from C to G. Update value of G to 8 since it is better than 10.

Next Expand G.

Step 6: Nothing to expand. Return Path from S to G.

**Return Path (S, A, C, G)**

Or if you have your code set to take the tied route (S, D, C, G).

* 1. [3 points] Greedy search with heuristic h shown on the graph.

Step 1: (S🡪D) since h = 1 < h= 2 < h= 5

All nodes (and heuristic h values) not expanded yet are on the fringe at this point.

Step 2: (S🡪D🡪G) since G has a no heuristic (zero). 0 < 2 <= 2 < 5

**Return Path (S🡪D🡪G)**

* 1. [3 points] A\* search with the same heuristic.

Backwards cost + heuristic cost to choose expansion.

Step 1: S to B has total cost of 8, S to D has a total cost of 6, and S to A has total cost of 4. Go to A.

Step 2: (S🡪A🡪C) has total cost of 8. Tied with (S🡪B) which is still on the fringe. Expand D since it is lower than both of those which are 8.

Step 3: (S🡪D🡪C) has total cost of 8. (S🡪D🡪G) has total cost of 10. (S🡪D🡪B)has total cost of 14.

Expand B.

Step 4: (S🡪B🡪D) has total cost of 8. No need to add to fringe since we only expand a state once.

Step 5: Expand C since it is 8 in the fringe which is the lowest (and the last node to expand!)

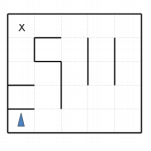
(S🡪A🡪C🡪G) since we are dequeing from the fringe according to alphabet if there is a tie.

(S🡪A🡪C🡪G) has total cost of 10.

**Return Path (S🡪A🡪C🡪G)**

1. **(25 points) Search and Heuristics**

Imagine a car-like agent wishes to exit a maze like the one shown below:



The agent is directional and at all times faces some direction *d ϵ (N, S, E, W).* With a single action, the agent can either move forward at an adjustable velocity v or turn. The turning actions are left and right, which change the agent's direction by 90 degrees. Turning is only permitted when the velocity is zero (and leaves it at zero). The moving actions are fast and slow. Fast increments the velocity by 1 and slow decrements the velocity by 1; in both cases the agent then moves a number of squares equal to its NEW adjusted velocity. Any action that would result in a collision with a wall crashes the agent and is illegal. Any action that would reduce v below 0 or above a maximum speed Vmax is also illegal. The agent's goal is to find a plan which parks it (stationary)on the exit square using as few actions (time steps) as possible.

Example: if the agent shown were initially stationary, it might first turn to the east using (right), then move one square east using fast, then two more squares east using fast again. The agent will of course have to slow to turn.

Answer the following questions:

1. [4 points] If the grid is M by N, what is the size of the state space? Justify your answer. You should assume that all configurations are reachable from the start state.

We must consider direction, speed, x coordinate and y coordinate. There are M \* N possible spots for the agent to be occupying on the maze. There are four different directions our agent can face while in these spots (N,E,S,W). There is also a set of speeds from 0 to Vmax.

So, there are **M\*N \* 4 \* (1 + Vmax) state spaces.**

1. [4 points] What is the maximum branching factor of this problem? You may assume that illegal actions are simply not returned by the successor function. Briefly justify your answer.

**The maximum branching factor is 3.** 3 happens when the agent is still, it can rotate 90 degree right or left and increase its speed.

1. [4 points] Is the Manhattan distance from the agent's location to the exit's location admissible? Why or why not?

A heuristic is admissible if it never over estimates the cost to reach a goal (Page 97 of our book). h(n) <= h\*(n) where h(n) is the estimate and h\*(n) is the actually cost to get to the goal.

**Manhattan distance is not admissible**. The agent can move faster than one in some scenarios and thus beat the Manhattan distance. It can do this by speeding up to Vmax and then slowing when it starts to come near the goal. If it does this it will reach the goal in less “moves” / turns / steps / time than there are squares. An example of this is a 1x5000 board. The agent could speed up to Vmax in the middle of the board and conquer much more than 1 square a step. This example proves that the Manhattan distance is not admissible.

1. [4 points] State and justify a non-trivial admissible heuristic for this problem which is not the Manhattan distance to the exit.

I think that **number of turns** to get the agent to face the goal would work. This will work because the agent can more or less jump to the next turn by increasing it’s speed to the number required to meet the next turn/goal.

1. [3 points] If we used an inadmissible heuristic in A\* tree search, could it change the completeness of the search?

**No**. A\* search visits every possible node from the fringe. It will always find a path if there is one, therefore it is complete.

Completeness has to do with finding a path, not finding the most efficient path.

1. [3 points] If we used an inadmissible heuristic in A\* tree search, could it change the optimality of the search?

**Yes**. Given an inadmissible heuristic A\* tree search can find a suboptimal route/path to the goal.

1. [3 points] Give a general advantage that an inadmissible heuristic might have over an admissible one.

It might be easier for us (a person) to come up with/ think of an inadmissible heuristic than an admissible heuristic.

A second advantage is it might be able to compute a solution quicker than an optimal solution. (This advantage depends on your situation /application of the problem)