Benjamin Horn

Artificial Intelligence

Homework 1

PROBLEM 1 From Node A to Node J Total Cost: 21

Uniform Cost Search with no check to see if the node was in the closed list before adding it to open list or closed list.

A: Order Removed From Open List: Given in (Node, Path Cost, Parent)

([('A', 0, 'n/a'), ('D', 7, 'A'), ('B', 12, 'D'), ('A', 14, 'D'), ('C', 15, 'B'), ('D', 17, 'B'), ('F', 17, 'C'), ('B', 18, 'C'), ('E', 18, 'D'), ('B', 19, 'A'), ('C', 19, 'F'), ('C', 21, 'B'), ('D', 21, 'A'), ('F', 21, 'C'), ('J', 21, 'F')])

B: Open List when Goal Found: Given in [Node, Path Cost, Parent]

[B, 22, C] [B, 22, D] [C, 22, B] [C, 23, F] [D, 23, B] [F, 23, C] [A, 24, D] [B, 24, C] [D, 24, B] [F, 24, B] [E, 25, C] [J, 25, F] [B, 26, D] [G, 27, D] [A, 28, D] [C, 28, E] [E, 28, D] [H, 28, E] [J, 28, E] [B, 29, F] [D, 29, E] [E, 29, C] [F, 30, B] [A, 31, B] [E, 31, C] [F, 31, B] [E, 32, D] [B, 33, A] [B, 33, F] [A, 37, B] [G, 37, D] [A, 38, B] [G, 41, D]

B: Closed List when Goal Found: Given in (Node, Path Cost, Parent)

([('A', 0, 'n/a'), ('D', 7, 'A'), ('B', 12, 'D'), ('A', 14, 'D'), ('C', 15, 'B'), ('D', 17, 'B'), ('F', 17, 'C'), ('B', 18, 'C'), ('E', 18, 'D'), ('B', 19, 'A'), ('C', 19, 'F'), ('C', 21, 'B'), ('D', 21, 'A'), ('F', 21, 'C'), ('J', 21, 'F')])

C: Total Nodes Added to Open List: 48

D: Total time each node appears as the current node in the final open list:

{'A': 5, 'C': 3, 'B': 7, 'E': 5, 'D': 3, 'G': 3, 'F': 4, 'I': 0, 'H': 1, 'J': 2}

E: Total time each node appears as the current node in the final closed list:

{'A': 2, 'C': 3, 'B': 3, 'E': 1, 'D': 3, 'G': 0, 'F': 2, 'I': 0, 'H': 0, 'J': 1}

PROBLEM 2 From Node A to Node J Total Cost: 3

Uniform Cost Search with no check to see if the node was in the closed list before adding it to open list or closed list where all weights are 1. This is how bfs algorithm works. If this was done like problem 1 it would result in 100 additions to the open list due to the cost of everything being 1.

A: Order Removed From Open List: Given in (Node, Path Cost, Parent)

([(('A', 0, 'n/a'), ('B', 1, 'A'), ('D', 1, 'A'), ('C', 2, 'B'), ('E', 2, 'D'), ('F', 2, 'B'), ('G', 2, 'D'), ('H', 3, 'E'), ('J', 3, 'E')])

B: Open List when Goal Found: Given in [Node, Path Cost, Parent] [(J, 3, F), (J, 4, H)]

B: Closed List when Goal Found: Given in (Node, Path Cost, Parent)

([('A', 0, 'n/a'), ('B', 1, 'A'), ('D', 1, 'A'), ('C', 2, 'B'), ('E', 2, 'D'), ('F', 2, 'B'), ('G', 2, 'D'), ('H', 3, 'G'), ('J', 3, 'E')])

C: Total Nodes Added to Open List: 15

D: Total time each node appears as the current node in the final open list:

{'A': 0, 'C': 0, 'B': 0, 'E': 0, 'D': 0, 'G': 0, 'F': 0, 'I': 0, 'H': 0, 'J': 2}

E: Total time each node appears as the current node in the final closed list:

{'A': 1, 'C': 1, 'B': 1, 'E': 1, 'D': 1, 'G': 1, 'F': 1, 'I': 0, 'H': 1, 'J': 1}

PROBLEM 3

False, because only the first time a node is selected off the top of the open list, it will be the lowest cost path. But can be false in any other case. When B is selected off the open list in problem 1, it has a cost of 18, but B was previously selected at a lower path cost of 12. A second example from problem 1 is when D is selected at a cost of 17, it has a previous lower path cost of 7. The third example is having F selected at 17, and then later selected off the top of the open list at a cost of 21. These are just 3 examples proving the statement false.

PROBLEM 4 From Node A to Node J Total Cost: 21

A: Order Removed From Open List: Given in (Node, Path Cost, Parent)

([('A', 0, 'n/a'), ('D', 7, 'A'), ('B', 12, 'D'), ('C', 15, 'B'), ('F', 17, 'C'), ('E', 18, 'D'), ('B', 19, 'A'), ('J', 21, 'F')])

B: Open List when Goal Found: Given in [Node, Path Cost, Parent]

[F, 24, B] [E, 25, C] [G, 27, D] [H, 28, E] [J, 28, E]

B: Closed List when Goal Found: Given in (Node, Path Cost, Parent)

([('A', 0, 'n/a'), ('D', 7, 'A'), ('B', 12, 'D'), ('C', 15, 'B'), ('F', 17, 'C'), ('E', 18, 'D'), ('B', 19, 'A'), ('J', 21, 'F')])

C: Total Nodes Added to Open List: 13

D: Total time each node appears as the current node in the final open list:

{'A': 0, 'C': 0, 'B': 0, 'E': 1, 'D': 0, 'G': 1, 'F': 1, 'I': 0, 'H': 1, 'J': 1}

E: Total time each node appears as the current node in the final closed list:

{'A': 1, 'C': 1, 'B': 2, 'E': 1, 'D': 1, 'G': 0, 'F': 1, 'I': 0, 'H': 0, 'J': 1}

b. (10) Differences and similarities for parts a through e.

1. Order Removed From Open List.
   1. In problem 1, using the original algorithm, the order removed from the open list is much longer. This is because we only add items to the open list in the latter algorithm if they are not on the closed list. This check on the closed list makes the open list much shorter. Also, there will be no duplicates entries in the algorithm in problem 4 because it is checking the closed list before it adds entries. So problem 4 has 8 entries removed from the open list, with no duplicates where problem 1 has 15 nodes removed, and does contain duplicates.
2. 2 Part Question
   1. Open List when Goal Found.
      1. The answer for B is similar to A when the goal node is found. It is much shorter and does not contain duplicates. This is because duplicates are not added and therefore it is a shorter list to pop nodes off of, making it much smaller.
   2. Closed List when Goal Found.
      1. The closed list is much shorter than the one from problem 1, but the set in problem 4 is included in problem 1’s closed list. It is shorter because we are checking neighboring nodes and not adding them to the open list if they are already in the closed list.
3. Total Nodes Added to Open List
   1. In problem 1, more nodes are added to the open list, 48 vs just 13 in problem 4. This is because we check whether the node is already in the closed list before we add it to the open list.
4. Total time each node appears as the current node in the final open list
   1. In problem 4 this frequency will be way less. Because we do check the closed to make sure it is not re added. This is shown here {'A': 0, 'C': 0, 'B': 0, 'E': 1, 'D': 0, 'G': 1, 'F': 1, 'I': 0, 'H': 1, 'J': 1}. Where in the first problem, a node can be added multiple time and appear many times as the current node in the list. Shown here {'A': 5, 'C': 3, 'B': 7, 'E': 5, 'D': 3, 'G': 3, 'F': 4, 'I': 0, 'H': 1, 'J': 2}.
5. Total time each node appears as the current node in the final closed list:
   1. Here it is possible for both algorithms to have duplicates as neither have a check if a node is currently in the closed list before it is potentially re-added to the closed list. So both list contain duplicates for example in problem 1’s final closed list is {'A': 2, 'C': 3, 'B': 3, 'E': 1, 'D': 3, 'G': 0, 'F': 2, 'I': 0, 'H': 0, 'J': 1} where problem 4’s final closed list is {'A': 1, 'C': 1, 'B': 2, 'E': 1, 'D': 1, 'G': 0, 'F': 1, 'I': 0, 'H': 0, 'J': 1}, where only B shows up more than once.

c. (10) What are the main advantages and disadvantages of using the modified version of the algorithm?

The advantages of using the modified version of the algorithm over the original version is that it takes fewer iterations to finish than the original. It checks if nodes are already in the open list before adding them to the open list, making the open list much shorter. The disadvantage of the modified algorithm compared to the original if that you have to check to if a node is in the closed list and that could be a time/ performance cost.