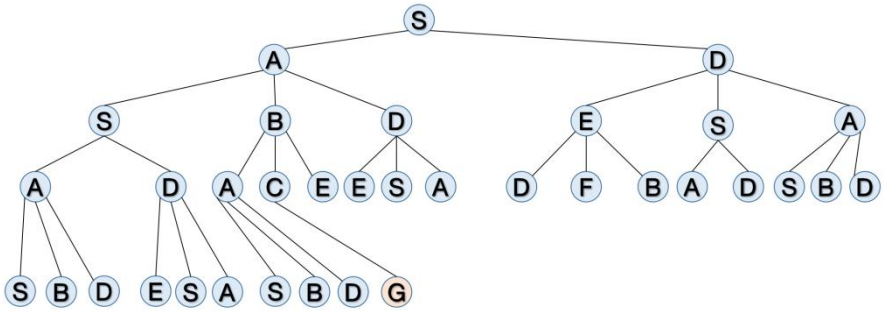
Problem assignment 1

Due: Wednesday, September 18, 2019

Problem 1

Part a.

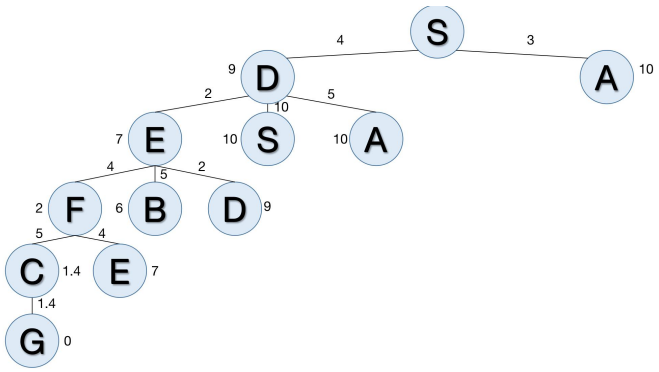


The nodes will be expanded as:

S-A-D-S-B-D-E-S-A-A-D-A-C-E-E-S-A-D-F-B-A-D-S-B-D-S-B-D-E-S-A-S-B-D-G

The path would be: S-A-B-C-G. The cost is 12.4. It's the optimal path.

Part b.

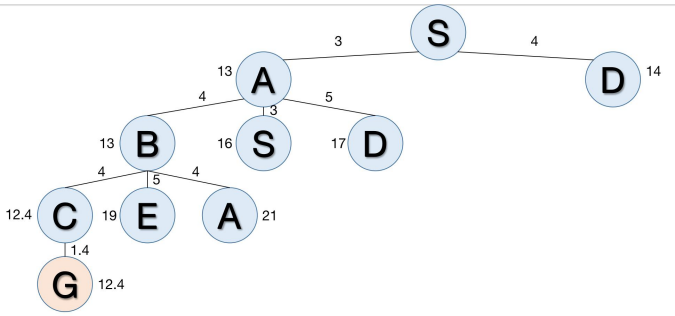


The nodes will be expanded as:

S-D-A-E-S-A-F-B-D-C-E-G

The path would be S-D-E-F-C-G. The cost is 16.4. It's not the optimal path.

Part c.



The nodes will be expanded as:

S-A-D-B-S-D-C-E-A-G

The path would be S-A-B-C-G. The cost is 12.4. It's the optimal path.

Problem 2

Part a.

Yes, the bidirectional A* is complete.

The bidirectional A* algorithm is based on two A* search expanded from beginning and end note. So, we only need to prove the completeness of A* search.

For A* search, the algorithm won't stuck into infinite loops.

Let's suppose there are two nodes A and B, B is generated from A, the start node would be S, the distance between A and B is $D_{A,B}$

For A, the $f(A) = g(A) + h(A)$

For B, the $f(B) = g(B) + h(B) = g(A) + D_{A,B} + h(B)$.

Then, when B expanding the nodes, the A generated by B would be:

$f'(A) = g(B) + h(B) = g(A) + D_{A,B} + h(B) + h(A) > g(A) + h(A)$

So for next steps, the second A won't be generated first. So the A* won't stuck into infinite loops..

Thus, the A* search is complete, and the bidirectional A* is also complete.

Part b.

Yes, the bidirectional A* is optimal.

The start node is S, the end node is E.

Let's suppose there's a sub-optimal node of state G, and the optimal state is G'.

$f(G) = f_S(G) + f_E(G) = g_S(G) + g_E(G)$

Thus, before reaching the G', at least from one side, there would be a middle state M, let's suppose it's on the forward direction.

Since G' is optimal and G is sub-optimal, $g_S(G') < g_S(G)$.

M is the the middle state, and the heuristic is admissible, so $f_S(M) \leq g_S(G')$.

Therefore, $f_S(M) < g_S(G)$.

Thus, when expand nodes, state M would be considered first than state G, which means the algorithm won't choose a sub-optimal result.

If the node of state is on the backward direction, the situation is also the same.

Problem 3

Part a. See main3a.py.

Part b. See main3b.py

Part c. See heuristic1.py

Part d. See heuristic2.py

Part e.

Uniform cost search

	Nodes generated	Nodes expanded	Maximum length of queue	Length of solution path

Example 1	82	29	53	3
Example 2	1383	500	83	6
Example 3	5623	1962	3661	8

Uniform cost search with elimination of repeats

	Nodes generated	**Nodes expanded**	**Maximum length of queue**	**Length of solution path**
Example 1	45	16	22	3
Example 2	235	84	105	6
Example 3	491	173	216	8
Example 4	1951	723	775	10
Example 5	84813	31591	30611	18

A* with misplaced tile heuristic

	Nodes generated	**Nodes expanded**	**Maximum length of queue**	**Length of solution path**
Example 1	13	4	9	3
Example 2	16	6	10	6
Example 3	25	9	16	8
Example 4	74	27	40	10
Example 5	4106	1523	1602	18

A* with Manhattan distance heuristics

	Nodes generated	**Nodes expanded**	**Maximum length of queue**	**Length of solution path**
Example 1	13	4	9	3
Example 2	25	9	15	6
Example 3	30	11	19	8
Example 4	50	18	25	10
Example 5	1030	384	365	18

The A* with Manhattan distance heuristics generally have the best statistics, especially when the search area becomes bigger. That's because the A* search with Manhattan distance heuristics has the most reasonable heuristics functions.

I suggest to choose the Manhattan distance heuristics. For misplaced tile heuristics, for most of the time, it would under-estimate the remaining steps to the goal state. Therefore, when generating nodes, some shortcuts which couldn't lead to the target state might be added to the queue and expanded first. However, for

Manhattan distance heuristics, it calculate the need to move steps for each tiles between current state and the goal state, which is close to the real state movements.

The A* search could workout without state repeats elimination. First, it's based on the breath-first search strategy. If we let heuristic=0, the A* would degenerate to breath-first search. Then, let's suppose there are two nodes A and B in the A* search tree, B is generated from A, the start node would be S, the distance between A and B is $D_{A,B}$

For A, the $f(A) = g(A) + h(A)$

For B, the $f(B) = g(B) + h(B) = g(A) + D_{A,B} + h(B)$.

Then, when B expanding the nodes, the A generated by B would be:

$f'(A) = g(B) + h(B) = g(A) + D_{A,B} + h(B) + h(A) > g(A) + h(A)$

So for next steps, the second A won't be generated first. So the A* won't stuck into infinite loops.

So the search tree would expand the new node first and won't stuck in the infinite loop. Thus, the A* would anyway reach the goal state.

Yes, h_3 is a admissible heuristic.

For h_1 and h_2, since they are admissible heuristic, we can know that h_1 and h_2 is smaller than the true distance h* from node to the goal, in expression:

$h_1(n) \leq h^*(n)$ $h_2(n) \leq h^*(n)$

Therefore, $h_3(n) = (h_1(n)+h_2(n))/2 \leq h^*(n)/2 + h^*(n)/2 = h^*(n)$.

From the lecture, we've already proved that if we don't over-estimate the distance to the goal, the heuristic is admissible. Since $h_3 \leq h^*$, the h_3 is a admissible heuristic.