CMPT 310
Assignment 1

Amogh Madhavan - 301359656
Carmen Choo - 301573484

October 5, 2022

Question 1 [25 marks]: Search Algorithms

Consider the problem of finding the shortest path from *a1* to *a10* in the following directed graph. The edges are labelled with their costs. The nodes are labelled with their heuristic values. Expand neighbours of a node in alphabetical order, and break ties in alphabetical order as well. For example, suppose you are running an algorithm that does not consider costs, and you expand a; you will add the paths <*a,b*> and <*a,e*> to the frontier in such a way that <*a,b*> is expanded before <*a,e*>.

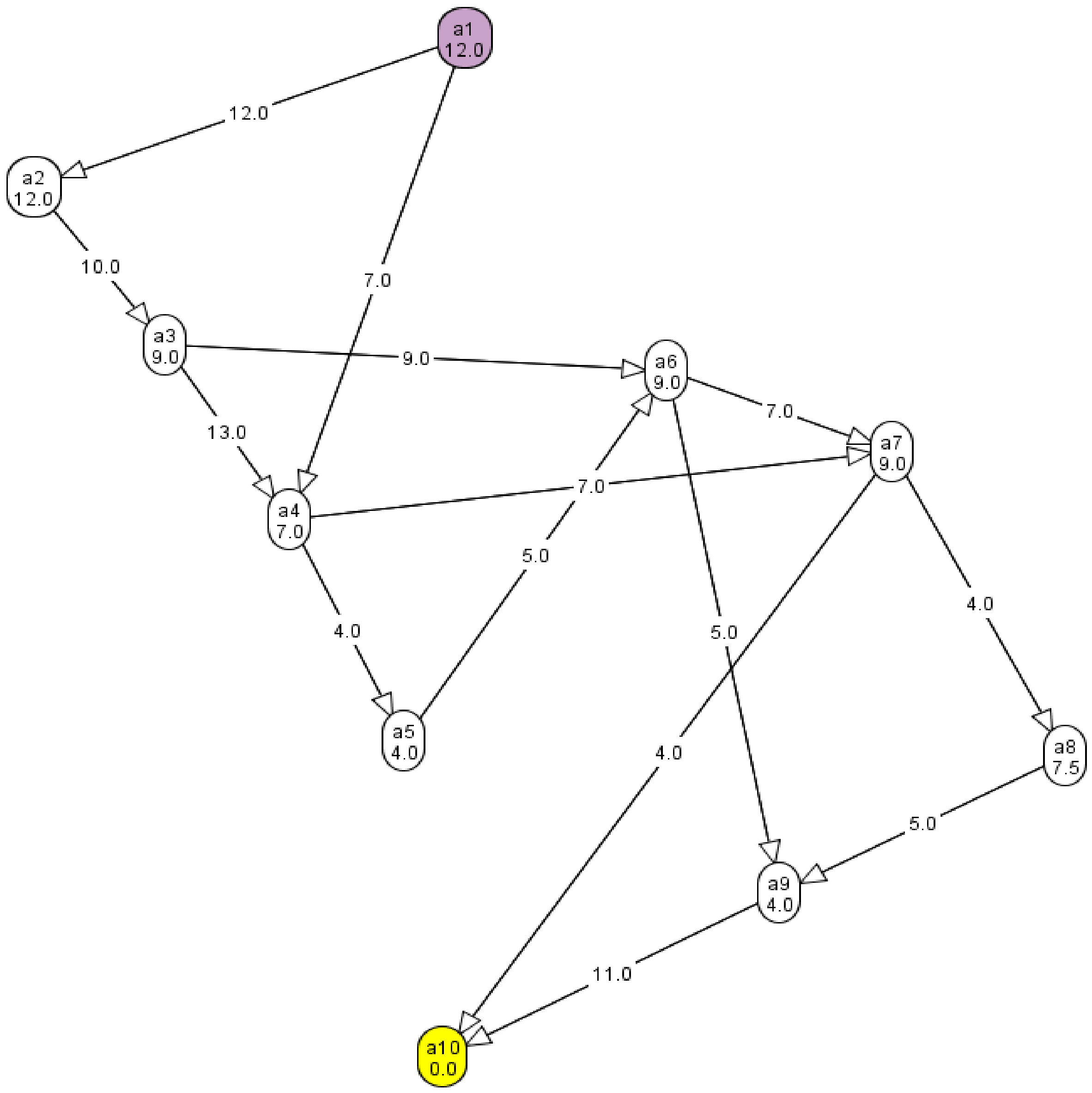


Figure 1: Graph

Note: You may need more rows than given in the table.

(a) [4 marks] Use **breadth-first search** to find the shortest path from *a1* to *a10*, show all paths explored for each step. First two steps are given as an example.

Node explored	path
a1	<a1, a2>, <a1, a4>
a2 (<a1, a2>)	<a1, a2, a3>, <a1, a4>
a4(<a1,a4>)	<a1,a2,a3>, <a1,a4,a5>, <a1,a4,a7>
a3(<a1,a2,a3>)	<a1,a4,a5>, <a1,a4,a7>, <a1,a2,a3,a4>, <a1,a2,a3,a6>
a5(<a1,a4,a5>)	<a1,a4,a7>, <a1,a2,a3,a4>, <a1,a2,a3,a6>, <a1,a4,a5,a6>
a7(<a1,a4,a7>)	<a1,a2,a3,a4>, <a1,a2,a3,a6>, <a1,a4,a5,a6>, <a1,a4,a7,a8>, <a1,a4,a7,a10> FOUND

(b) [6 marks] Use **lowest-cost-first search** to find the shortest path from *a1* to *a10*, show all paths explored, and their costs, for each step. First step is given as an example.

Node explored	Cost	Path
a1	7	<a1, a4>
	12	<a1, a2>
a4	11	<a1, a4, a5>
	12	<a1, a2>
	14	<a1, a4, a7>
a5	12	<a1, a2>
	14	<a1, a4, a7>
	16	<a1, a4, a5, a6>
a2	14	<a1, a4, a7>
	16	<a1, a4, a5, a6>
	22	<a1, a2, a3>
a7	16	<a1, a4, a5, a6>
	22	<a1, a2, a3>
	18	<a1, a4, a7, a8>
	18	<a1, a4, a7, a10> FOUND

(c) [12 marks] Use **A* search** to find the shortest path from *a1* to *a10*, show all paths explored, and their values of f(n), for each step.

(Note that f(n) = g(n) + h(n), where g(n) is the cost of the path from the start node to the current node n, h(n) is the heuristic value of the current node n).

First step is given as an example.

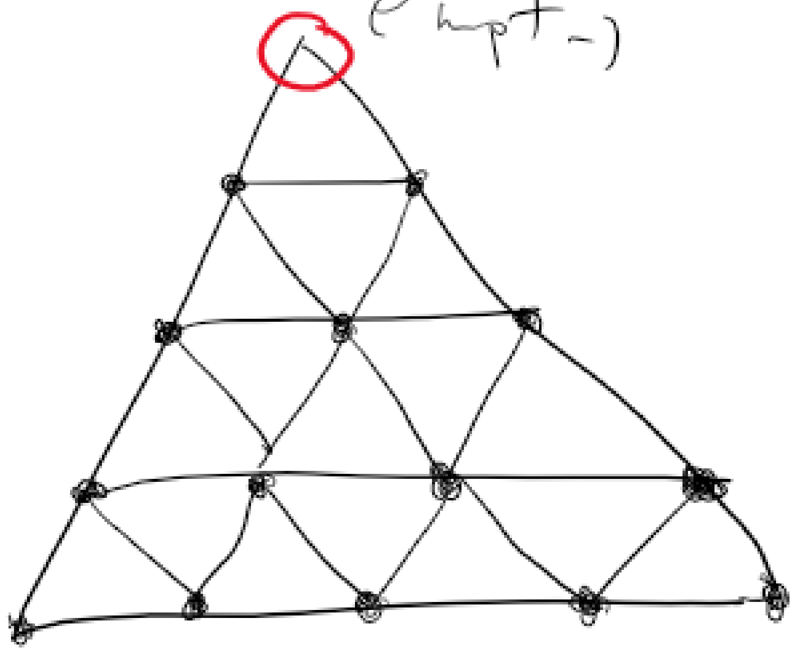
Iteration	Node(s)	Actual path cost	Heuristic cost	Total cost	Path(s)
0 (<a1>)	a1	0	12	12	<a1>
1 (<a1,a4>)	a4	7	7	14	<a1,a4>
	a2	12	12	24	<a1,a2>
2 (<a1,a4,a5>)	a5	11	4	15	<a1,a4,a5>
	a7	14	9	23	<a1,a4,a7>
	a2	12	12	24	<a1,a2>
3<a1,a4,a7>	a7	14	9	23	<a1,a4,a7>
	a2	12	12	24	<a1,a2>
	a6	16	9	25	<a1,a4,a5,a6>
4	a2	12	12	24	<a1,a2>
	a6	16	9	25	<a1,a4,a5,a6>
	a10	18	0	18 FOUND	<a1,a4,a7,a10>
	a8	18	7.5	25.5	<a1,a4,a7,a8>

(d) [3 marks] Out of BFS, LCFS, A*, which search algorithm do you think is most efficient for the above example? Justify your reasoning.

A* is the most efficient algorithm, since the algorithm involve the admissible heuristic value which is the underestimation of cost required for each node to reach the goal note, it's able to reduce the number of path we need to explore as we'll always choose the path which has the smallest f(n), meanwhile BFS does not take into account the cost of each path whereas LCFS only take into account the cost of each path but doesn't estimate the remaining distance for the current path to reach the goal node.

Question 2: [25 marks] Game

For this activity, you are going to play the Seashell game. The dots are closed seashells. The rules of the game are to use the seashells to jump over other seashells like in checkers, which in turn will open the seashell that has been jumped over. The seashells can also be moved into the empty spot adjacent to it but will not open any shells. The goal is to do this for all the seashells to win the game.



You can play the game here: https://www.novelgames.com/en/seashell/

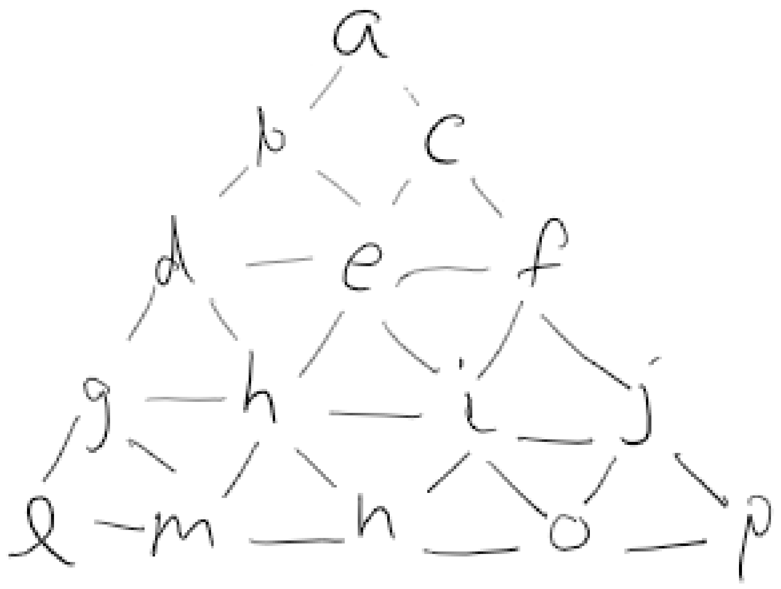


Fig 2. Seashell board

Now, you are going to represent seashells as a search problem. (Use the labels provided in Fig 2 for referring to spaces on the board).

a. [4 points] How would you represent a node/state?

- **a, b, c, d, e, f, g, h, i, j, l, m, n, o, p where each alphabet represents a position on the seashell board.**
- **Each alphabet (a to p, except k) ∈ { x, 0, 1 }.**
- **x denotes an empty space, 0 denotes a closed seashell and 1 denotes an open seashell. In other words, a = x denotes that position a is an empty space, b=0 means that position b has a closed seashell, c=1 means that position c has an open seashell.**

b. [2 points] In your representation, what is the goal node?

- **Goal node happens when 14 out of 15 alphabets = 1, and the remaining alphabet = x. In other words 14 out of 15 of the positions will have an open seashell and the remaining one will be an empty space.**

c. [3 points] How would you represent the arcs?

- **Each arc is represented by a swap (alphabet1, alphabet2) where there exists an edge in the seashell board between alphabet and alphabet2. For this to happen, alphabet1 and alphabet2 must have different conditions { x, 0, 1 }. Then, we will swap the condition of alphabet1 with alphabet2 {x,0,1}. For example alphabet1=x and alphabet2=0, swap(alphabet1, alphabet2) will give us alphabet1=0 (Closed seashell) and alphabet2=x (Empty space).**

- **If there is a seashell in between both the alphabets, we will represent the arcs by another function called jump(alphabet1, alphabet2). If this happens, on top of swapping the condition of alphabet1 with alphabet2 {x,0,1}, we will also need to change the condition of the alphabet in between from 0 to 1 or 1 to 0. For example alphabet1=x and alphabet2=0, jump(alphabet1, alphabet2) will give us alphabet1=0 (Closed seashell) and alphabet2=x (Empty space). On top of that let's say there is another alphabet3 = 0 in between, after we perform the jump function, alphabet3 = 1.**

d. [3 points] How many possible board states are there? Note: this is not the same as the number of "valid" or "reachable" game states, which is a much more challenging problem.

- **Each position will contribute to n+2 possible states where n is the number of edges/ arcs a position/ alphabet has in the seashell board. In this case, alphabet a has 2 edges(a->b & a->c). Therefore we will have n+2 = 4 possible states if position a is empty and we were to move position a (swap a->b, swap a->c , swap a->d, swap a->f) and other positions are tabulated below. This will give us a total of 88 possible states.**

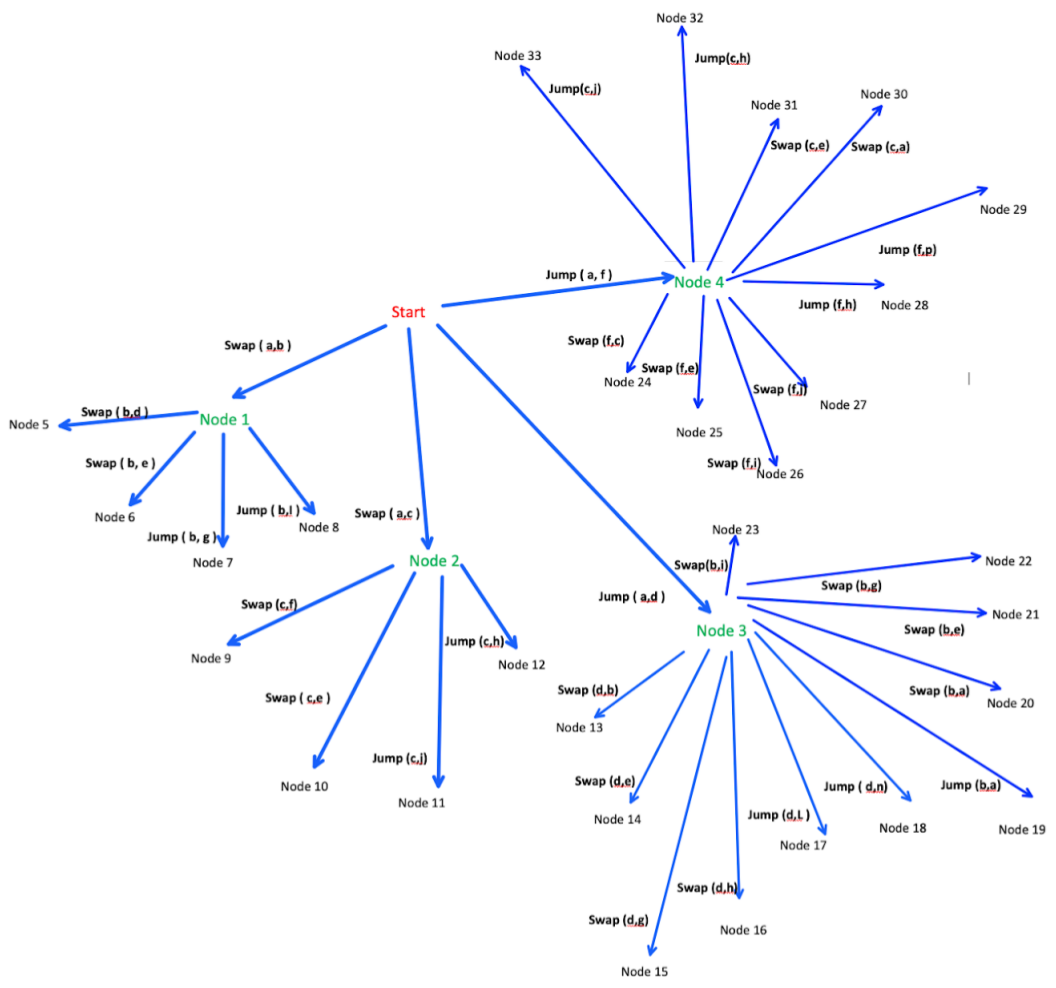
Position/ Alphabet	Number of Edges (n)	Possible States (n+2)
a	2	4
b	3	5
c	3	5
d	4	6

e	6	8
f	4	6
g	4	6
h	6	8
i	6	8
j	4	6
l	2	4
m	4	6
n	4	6
o	4	6
p	2	4

e. [6 marks] Write out the first three levels (counting the root as level 1) of the search tree based on the labels in Figure 3. (Only label the arcs; labelling the nodes would be too much work).

Level	Parent Node	Nodes	Arcs from parent node to target node
1	-	Start	-
2	Start	n1 n2 n3 n4	Swap (a,b) Swap (a,b) Jump (a,d) Jump (a,f)
3	n1	n5 n6 n7 n8	Swap (b,d) Swap (b,e) Jump (b,g) Jump (b,i)

	n2	n9	Swap (c,f)
		n10	Swap (c,e)
		n11	Jump (c,j)
		n12	Jump (c,h)
	n3	n13	Swap(d,b)
		n14	Swap(d,e)
		n15	Swap(d,g)
		n16	Swap(d,h)
		n17	Jump (d,f)
		n18	Jump(d,l)
		n19	Jump(d,n)
		n20	Swap (b,a)
		n21	Swap (b,e)
		n22	Jump (b,g)
		n23	Jump (b,i)
		n24	
	n4	n25	Swap (f,c)
		n26	Swap (f,e)
		n27	Swap (f,i)
		n28	Swap (f,j)
		n29	Jump (f,h)
		n30	Jump (f,p)
		n31	Swap (c,a)
		n32	Swap (c,e)
		n33	Jump (c,h)
		n34	Jump (c,j)



f. [3 marks]What kind of search algorithm would you use for this problem? Justify your answer.

BFS, since there are many solutions that could lead us to the goal node, we only want the shortest path.

g. [4 marks] Would you use cycle-checking? Justify your answer.

Yes, we will need to make sure that we do not generate the same state if it's already a part of our path from start to the goal node, this makes sure that we will not be trapped inside a cycle and this could reduce the search space. To do this, we will add all possible states into a list and the next time we generate new states, we are going to make sure that it does not already exist in the list to avoid walking through a state more than once.

Q3.

The code is in the file search2.py, all of the code is at the end of the file. There is some code that was used for testing purposes and to fill the table below, that is reflected through the comments.

A = Misplaced Tile Heuristic, B = Manhattan Distance, C = Max

Initial State	**Total Running Time**			**Solution Length**			**Removed Frontier Nodes**		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
(6,4,3,1,8,9,7,2,5,0)	9.5367431640625e-07s	9.5367431640625e-07s	1.1920928955078125e-06s	**18**	**18**	**18**	**695**	**124**	**154**
(1,3,4,2,5,6,7,8,9,0)	3.519962774589658e-07s	1.8400169210508466e-07s	1.539956429041922e-07s	**22**	**22**	**22**	**4511**	**4654**	**4604**
(1,3,4,2,5,6,7,8,0,9)	4.029934643767774e-07s	1.8200080376118422e-07s	1.629960024729371e-07s	**21**	**21**	**21**	**3603**	**3879**	**3830**
(1,2,3,4,5,6,9,7,0,8)	3.00002284348011e-07s	1.3799581211060286e-07s	1.2599775800481439e-07s	**17**	**17**	**17**	**677**	**1857**	**1845**
(1,2,3,4,5,6,9,7,8,0)	2.5599729269742966e-07s	2.0700099412351847e-07s	9.799987310543656e-08s	**18**	**18**	**18**	**859**	**2266**	**2243**
(3,2,4,1,5,6,7,8,9,0)	2.6899942895397544e-07s	1.5999830793589354e-07s	1.2300006346777081e-07s	**24**	**24**	**24**	**8134**	**5505**	**5456**
(1,2,3,4,5,6,7,8,0,9)	4.069952410645783e-07s	2.770029823295772e-07s	2.4199835024774075e-07s	**1**	**1**	**1**	**2**	**2**	**2**
(2,3,1,4,5,6,7,8,9,0)	8.220013114623725e-07s	4.899993655271828e-07s	1.5499972505494952e-07s	**20**	**20**	**20**	**1405**	**1742**	**1727**

Summary of the results (using the average values)

Heuristic	**Total Run Time**	**Solution Length**	**Removed Frontier Nodes**
Misplaced Tile	4.70E-07s	**18**	**2486**
Manhattan Distance	3.24E-07s	**18**	**2504**
Max	2.82E-07s	**18**	**2483**

Overall, the which is best heuristic really depends on the initial state, there were some states that made solving it easier using the misplaced tile heuristic as compared to the Manhattan distance and vice versa. The simpler the initial state is to solve, misplaced tile would be a better heuristic to use and the more complex, Manhattan would be more efficient, but the max honestly is better than misplaced in a complex situation but worse than Manhattan but if I were to pick one, it would be max because it is sort of the best of both worlds, its efficient enough in any situation. And based on the summary, Max has the best average score for both the running time and the removed frontier nodes, so that is one we would pick as the best.

Question 4: [25 marks] Programming II - You can do (a) in any programming language. TAs would be able to help you with python only.

(a) [5 marks] Do you think that the solutions found by this algorithm are always optimal? why?

Of course if you would have looked at our code file, you would have figured out that we didn't actually get the solution because it was simply too complicated but to answer the question nonetheless, we think the reason why the solutions found by this algorithm are the most efficient and always optimal is because it relies on an open list and a closed list to find a path that is both optimal and complete towards the goal. So theoretically should you feed the algorithm admissible heuristics, it will always get you an optimal solution. Of course, if the algorithm doesn't get overestimated costs, it won't be optimal anymore but under perfect circumstances, we think that this algorithm always produces optimal results