karim khoja - 301376869
bhakti bhanushali - 30144839
one late day claimed

CMPT 310
Assignment 1

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*>.

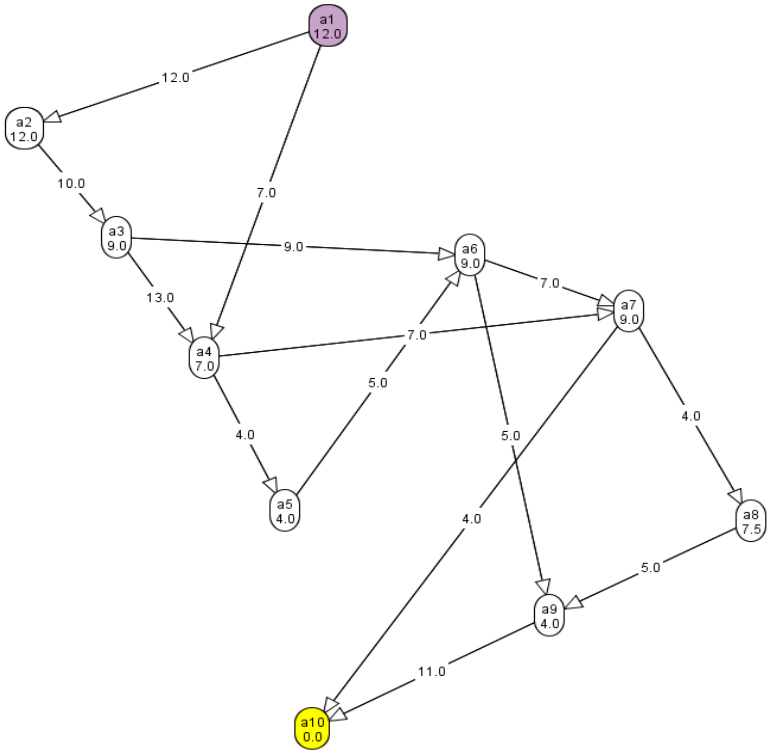


Figure 1: Graph

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.

Node explored	path
a1	<a1, a2>, <a1, a4>
a2 (<a1, a2>)	<a1, a2, a3>, <a1, a4>
a4(<a1, a4>)	<a1, a4, a5>, <a1, a4, a7>, <a1, a2, a3>
a3(<a1, a2, a3>)	<a1, a2, a3, a4>, <a1, a2, a3, a6>, <a1, a4, a5>, <a1, a4, a7>
a5(<a1, a4, a5>)	<a1, a4, a5, a6>, <a1, a2, a3, a4>, <a1, a2, a3, a6>, <a1, a4, a7>
a7 (<a1, a4, a7>)	<a1, a4, a7, a8>, <a1, a4, a7, a10>, <a1, a2, a3, a6>, <a1, a4, a5, a6>, <a1, a2, a3, a4>
a4(<a1, a2, a3, a4>)	<a1, a4, a7, a8>, <a1, a4, a7, a10>, <a1, a2, a3, a6>, <a1, a4, a5, a6>, <a1, a2, a3, a4, a5>, <a1, a2, a3, a4, a7>
a6(<a1, a2, a3, a6>)	<a1, a2, a3, a4, a5>, <a1, a2, a3, a4, a7>, <a1, a4, a5, a6>, <a1, a4, a7, a8>, <a1, a4, a7, a10>, <a1, a2, a3, a6, a7>, <a1, a2, a3, a6, a9>
a6(<a1, a4, a5, a6>)	<a1, a2, a3, a4, a5>, <a1, a2, a3, a4, a7>, <a1, a4, a5, a6, a7>, <a1, a4, a5, a6, a9>, <a1, a4, a7, a8>, <a1, a4, a7, a10>, <a1, a2, a3, a6, a7>, <a1, a2, a3, a6, a9>
a8(<a1, a4, a7, a8>)	<a1, a2, a3, a4, a5>, <a1, a2, a3, a4, a7>, <a1, 4, a5, a6, a7>, <a1, a4, a5, a6, a9>, <a1, a4, a7, a8, a9>, <a1, a4, a7, a10>, <a1, a2, a3, a6, a7>, <a1, a2, a3, a6, a9>
a10(<a1, a4, a7, a10)	Goal node reached

(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.

Node explored	Cost	Path
a1	7 12	<a1, a4> <a1, a2>
a4 <a1, a4, a5>	11 14 12	<a1, a4, a5> <a1, a4, a7> <a1, a2>
a5< a1, a4, a5>	16 14 12	<a1, a4, a5, a6> <a1, a4, a7> <a1, a2>
a2<a1, a2>	22 14 16	<a1, a2, a3> <a1, a4, a7> <a1, a4, a5, a6>
a7<a1, a4, a7>	18 18 22 16	<a1, a4, a7, a10> <a1, a4, a7, a8> <a1, a2, a3> <a1, a4, a5, a6>
a6<a1, a4, a5, a6>	21 23 18 18 22	<a1, a4, a5, a6, a9> <a1, a4, a5, a6, a7> <a1, a4, a7, a10> <a1, a4, a7, a8> <a1, a2, a3>
a8<a1, a4, a7, a8>	23 21 18 22	<a1, a4, a7, a8, a9> <a1, a4, a5, a6, a9> <a1, a4, a7, a10> <a1, a2, a3>
a10<a1, a4, a7, a10>	18	Goal node reached

(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)

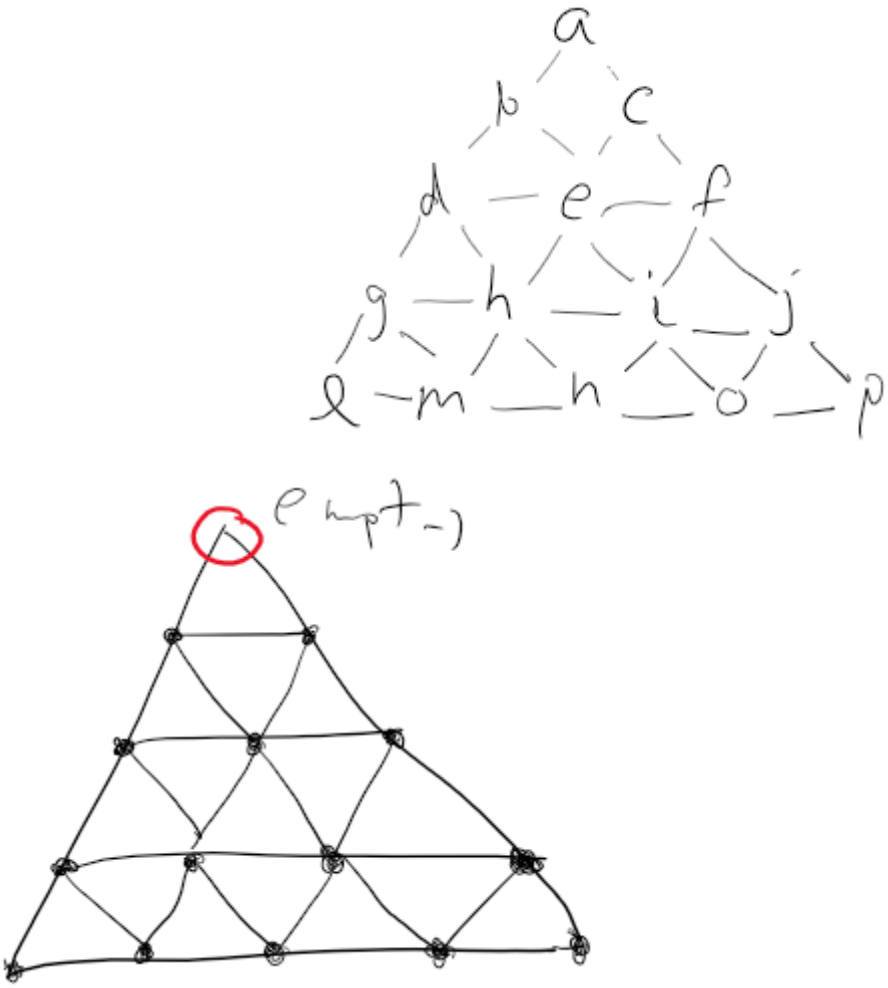
Iteration	Node(s)	Actual path cost	Heuristic cost	Total cost	Path(s)
0 (<a1>)	a1	0	12	12	<a1>
1(<a1, a4>)	a2 a4	12 7	12 7	24 14	<a1, a2> <a1, a4>
2(<a1, a4, a5>)	a7 a5 a2	14 11 12	9 4 12	23 15 24	<a1, a4, a7> <a1, a4, a5> <a1, a2>
3(<a1, a4, a7>)	a6 a7 a2	16 14 12	9 9 12	25 23 24	<a1, a4, a5, a6> <a1,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

(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* algorithm is the most efficient because it explores the least number of nodes. This is because it is an informed search algorithm that considers the estimated length of future paths and makes the most optimal choice.

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?
Each node would be the current representation of the seashell game. It would have properties like -> state, child and parent. State is the arrangement of the seashells in

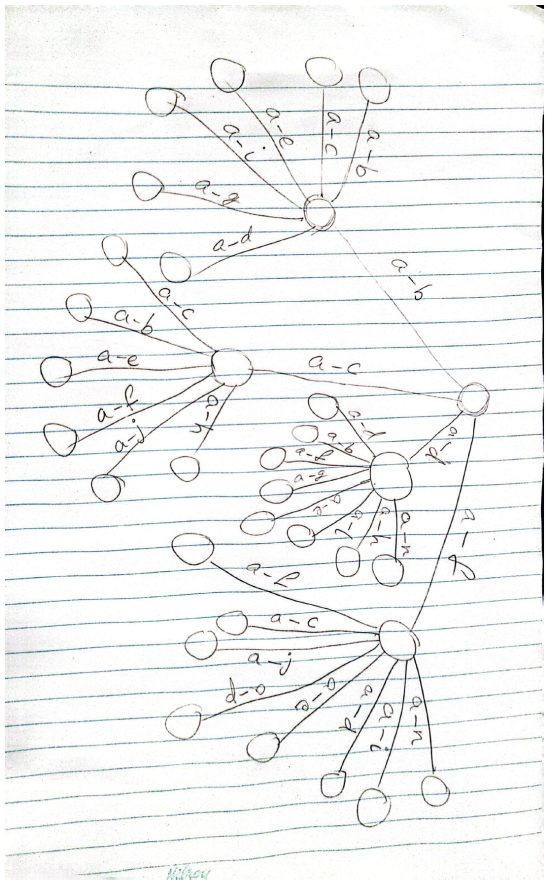
the game, parent is the node that the current node is a successor of and node child is the node after making a certain action. Each state can be represented as a tuple of 0's and 1's where 0 represents closed seashells and 1 represents open seashell and a * for the empty space.

(b) [2 points] In your representation, what is the goal node?
The goal node is that state of the game where all the seashells are open which would be a tuple of 1's and one blank spot represented by a " * "
<11111111111*111>

(c) [3 points] How would you represent the arcs?
Arcs give a connection between the parent and children nodes. Each node would have a pointer to their parent and child. Parent is the previous state of the node, before making an action while child is the state after making a specific action, these pointers act as arcs. For instance, a node (state) can be <1101*0011000000> where 1's represent open seashells and 0's represent closed seashells whilst * is the open space. One possible move would can result the representation to be <110101*11000000>

(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.
Since each seashell can either be open, closed or an empty space and there are 15 spots the game has 3^15 states.

(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).



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

We would use A* search algorithm to find the goal node. This is because the solution is located deep within the tree as all the shells have to be opened. DFS is avoided since there's the possibility of cycles in the graph. A* with the appropriate heuristic function like the number of open cells would give us the best solution without getting stuck in cycles.

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

Yes, we would use cycle checking because theres the possibility that you obtain the same output after a specific set of moves. In order to avoid such a situation wher the algorithm is stuck in an infinite loop, cycle-checking is required.

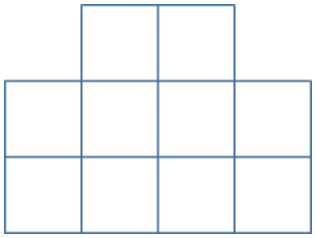
Ques 3. [25 marks] Programming 1

For this question, you get a chance to play with some heuristics. We have provided you with the code needed for this activity. There are two files:

- search.py
- Util.py

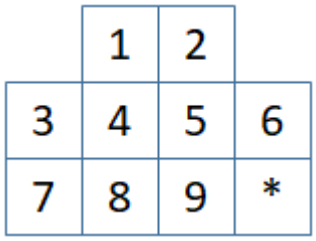
These files are available on canvas. Go through the code and understand it, including the Problem class that it inherits from.

In search.py, we have added a new StagePuzzle class, which is a modified version of the EightPuzzle class. We call our modified puzzle as StagePuzzle since our puzzle is like a champion stage, as shown below.



StagePuzzle

The desired final state of our puzzle is as follows:



StagePuzzle Final State

(a) [5 marks] Write a function called `make_rand_StagePuzzle()` that returns a new instance of a `StagePuzzle` problem with a random initial state that is solvable. Note that `StagePuzzle` has a method called `check_solvability` that you should use to help ensure your initial state is solvable.

(b) [5 marks] Write a function called `display(state)` that takes a `StagePuzzle` state (i.e. a tuple that is a permutation of (0, 1, 2, …, 9)) as input and prints a neat and readable representation of it. 0 is blank and should be printed as a `*` character.

For example, if `state` is `(0, 3, 2, 1, 8, 7, 4, 6, 5, 9)`, then `display(state)` should print:

```
 *3
2187
4659
```

(c) [15 marks] Create 8 (more would be better!) random StagePuzzle instances (using your code from above) and solve each of them using the algorithms below. Each algorithm should be run on the exact same set of problems to make the comparison fair.

For each solved problem, record:

- the total running time in seconds
- the length (i.e. number of tiles moved) of the solution
- that total number of nodes that were *removed* from the frontier

You will probably need to modify the A* function named "astar_search" in the provided code to get all this data.

Note:

- The time it takes to solve random StagePuzzle instances can vary from less than a second to hundreds of seconds. So solving all these problems might take some time!

- The result function in StagePuzzle class does not necessarily check the requested action's feasibly before doing it. It is your responsibility to check the actions feasibilities before doing them. You can also edit the StagePuzzle class and add new functions if you need them.

- Make sure to add the comments in the code.

The algorithms you should test are:

- A*search using the misplaced tile heuristic (this is the default heuristic in the StagePuzzle class)
- A* search using the Manhattan distance heuristic. Please implement your version of the Manhattan heuristic.
    - Be careful: there is an incorrect Manhattan distance function in tests/test_search.py. So, don't use that!
- A*search using the max of the misplaced tile heuristic and the Manhattan distance heuristic

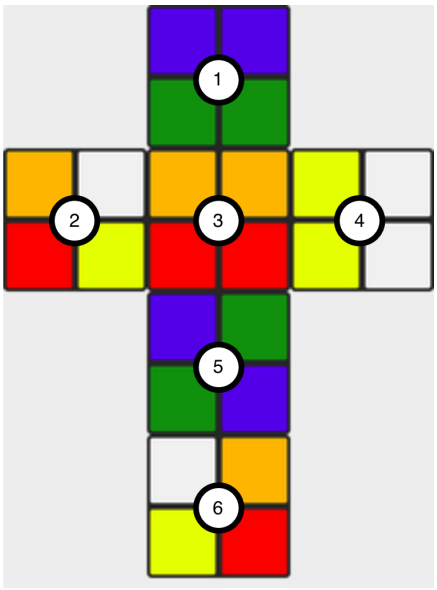
Summarize all your results in a single table for comparing the three heuristic functions you used. Based on your data, which algorithm is the best? Explain how you came to you conclusion.

Initial State	Total Run Time			Solution length			Nodes Expanded		
	Manhattan	Misplaced	Max	Manhattan	Misplaced	Max	Manhattan	Misplaced	Max
[1,4,9,5,7,6,3,2,0,8]	0.479397058	12.49738216	1.045835972	25	25	25	2209	12781	3376
[6,4,3,1,8,9,7,2,5,0]	0.006847143	0.055254936	0.008797884	18	18	18	149	695	234
[1,6,2,4,8,9,0,3,7,5]	0.007771015	0.092089176	0.011961222	19	19	19	211	1002	280
[1,2,3,5,0,4,6,7,8,9]	0.106408834	1.419753075	0.159577847	22	22	22	932	4205	1256
[4,9,3,6,0,2,7,8,5,1]	1.410352945	26.50601172	3.07457304	26	26	26	3921	18513	5818
[4,5,8,3,6,9,1,2,7,0]	0.3771348	13.84648705	0.472485781	26	26	26	2057	13319	2360
[6,4,3,8,9,0,2,7,1,5]	0.08732605	7.945097923	0.226443052	25	25	25	921	10260	1574
[1,7,5,4,8,2,3,9,0,6]	0.103672028	3.355480671	0.269722223	23	23	23	938	6365	1616
Mean times	0.322363734	8.214694589	0.658674628	23	23	23	1417.25	8392.5	2064.25

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) [20 marks] In this problem, you need to input a 2*2 rubik's cube and solve it using A* searching algorithm. The transition between the states is the same as real moves on a rubik's cube (ie. in each step, one of the facets can be turned clockwise or counterclockwise and there are 12 choices in total). Consider the cube as the following image.

The colors are numbered as Orange=1, Green=2, White=3, Blue=4, Red=5, and Yellow=6. You input the facets of the initial state of the cube with the following order:



For example, the input for the above state is:

4 4 2 2
1 3 5 6
1 1 5 5
6 3 6 3
4 2 2 4
3 1 6 5

You need to output the moves by specifying the number of the facet and its direction (ie. clockwise vs counterclockwise). For example:

Turn Facet#1 Clockwise
Turn Facet#2 Counterclockwise
…

After finding the solution, you should also print the number of explored nodes and depth of the found goal node. Use the following heuristic function for the A* algorithm.

h(n) = 4 * (number of facets with 4 different colors)
+ 2 * (number of facets with 3 different colors)
+ number of facets with 2 different colors

For instance, the value of the heuristic function for the previous example is 12.

Below is a sample output:

```
cube colors:
4 4 2 2
1 3 5 6
1 1 5 5
6 3 6 3
4 2 2 4
3 1 6 5
[[2, 2, 2, 2], [6, 6, 6, 6], [5, 5, 5, 5], [3, 3, 3, 3], [4, 4, 4, 4], [1, 1, 1, 1]]
Produced Nodes: 18191
Expanded Nodes: 4027
Answer Depth: 10
Max Memory: 18191

Solution:
turn facet#1 anticlockwise
turn facet#3 anticlockwise
turn facet#2 anticlockwise
turn facet#1 clockwise
turn facet#2 anticlockwise
turn facet#2 anticlockwise
turn facet#1 anticlockwise
turn facet#2 clockwise
turn facet#3 clockwise
turn facet#1 anticlockwise
```

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

No, the solutions found are not optimal because the heuristic function is not admissible. The estimated cost returned by the function is sometimes higher than the actual cost which makes it inadmissible. H(n) is an overestimate of the length of the shortest path from a node to the goal node.

Note:

For specific help related to Python - Drop-in TAs programming OH.