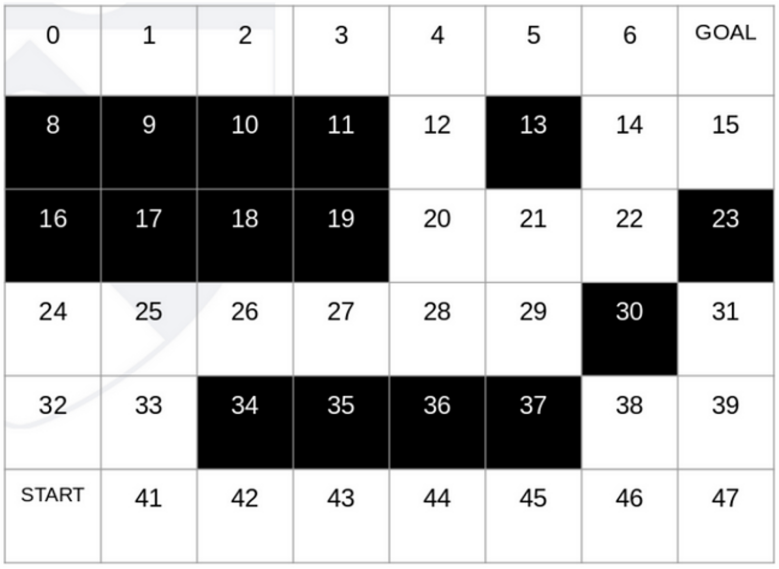
CSE 4/568 Programming Assignment 5
Name: . Antarpreet Kaur
UB Name: antarpre
UBID: 50486342

Task 1



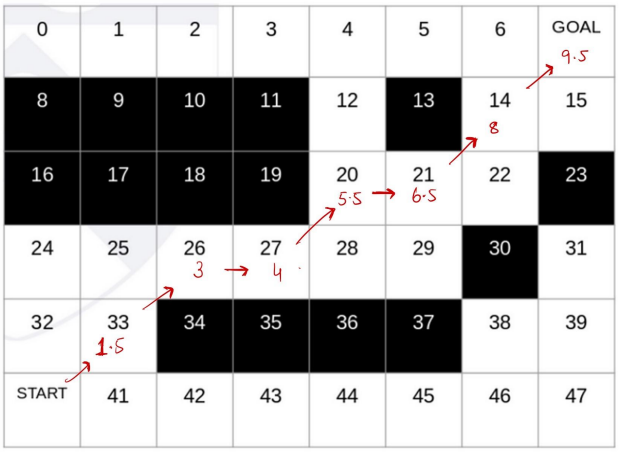
1. For figure 1, write out the steps for Djikstra's algorithm (8-connected). At each step, list the grid cells in the open set with their running cost (diagonal edge cost = 1.5, horizontal edge cost = 1)and the grid cells in the visited set. Write the final path as a list of grid cell ids.

   Given, Horizontal edge cost = 1
   Diagonal edge cost = 1.5
   Cost Calculation :

Iterations	Open Set	Cost	Visited Set
Iter 1	32	1	start
	33	1.5	start
	41	1	start
Iter 2	24	2	32
	24	2.5	33
	25	2.5	32

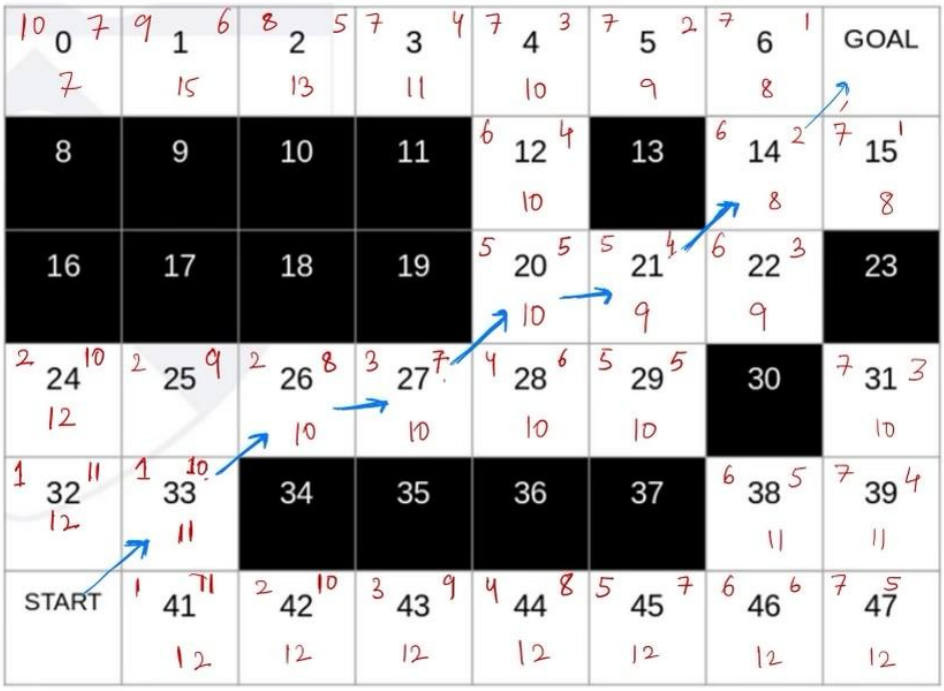
	26	3	33
	42	2	41
Iter 2	24	2	32
	25	25	33
	26	3	33
	42	2	41
	3	8	12
	4	75	12
	5	8	12
	6	9	14
	15	9	22
	31	8	38
	39	7.5	38
	47	7	46
	Goal	9.5	14
Iter 8	2	9	3
Iter 9	1	10	2
Iter 10	0	11	1

We will consider the lowest cost for the optimal path. Comparing the costs above, the optimal path is:
Start → 33 → 26 → 27 → 28 → 21 → 14 → Goal
Total Cost = 9.5



2. For figure 1, write out the steps for A* algorithm (8-connected, assume uniform cost or each action). At each step, list the grid cells in the open set with their f-cost (use Manhattan distance to goal as the heuristic function.) and the grid cells in the visited set. Write the final path as a list of grid cell ids.

   Upper left = h_cost
   Upper right = g_cost
   Low_center = f_cost



g(n) = 1 for all the direction
h(n) = abs (node.x-goal. x) + abs (node. y - goal.y)
Goal = (0,7)

Grid Cell	h_cost
33	| 4 - 0 | + |1 - 7| = 10
32	| 4 - 0| + |0 - 7| = 10
38	| 4 - 0| + |6 - 7| = 5
39	4
41	11
42	10

43	9
44	8
45	7
46	6
47	5
24	10
25	9
26	8
27	7
28	6
29	5
31	3
20	5
21	4
22	3
12	4
14	2
15	1
0	7
1	6
2	5
3	4
4	3
5	2
6	1

g_cost (Start to Current)

Tile Number	g-cost
32	1
33	1
41	1
24	2
25	2
26	2
42	2
27	3
43	3
28	4
44	4
45	5
20	5,6
29	5
46	6
21	6,5
12	6
22	6
38	6
14	7,6
15	7
31	7
39	7
47	7
4	7
3	7

5	7
6	8,7
2	8
1	9
0	10

Tile Number	f-cost
32	12
33	11
41	12
42	12
43	12
44	12
45	12
46	12
47	12
38	11
39	11
31	10
24	12
25	11
26	10
27	10
28	10
29	10
20	10
21	10,9
22	9

12	10
14	9,8
15	8
0	17
1	15
2	13
3	11
4	10
5	9
6	9,8

After calculating the f_cost for all the grid cells, we look for the cells with the lowest f_cost.
If f_cost value is same for any two neighbors, we check for l h_cost values. If in case, the h_cost values are also the same, we check for high g_cost.
According to these steps, the optimal path is:
Start → 33 → 26 → 27 → 20 → 21 → 14 → Goal

3. Write one main difference between A* and Djikstra's algorithm.
   A key difference between A* and Dijkstra's algorithm lies in how they select the next node to visit while searching. Dijkstra's algorithm only considers the cost of reaching each node from the starting node and ignores the cost of reaching the goal node. It chooses the next node to visit based on the lowest cost found so far, potentially exploring nodes that are not on the path to the goal and making the search longer.
   In contrast, A* uses a heuristic function to estimate the cost of reaching the goal node from each node on the graph. It combines this estimated cost with the actual cost of reaching each node from the starting node to calculate an "f-score" for each node. A* chooses the next node to visit based on the lowest f-score, prioritizing nodes that are more likely to be on the path to the goal. This approach can make the search more efficient by avoiding exploration of paths that are unlikely to lead to the goal.

4. Which data structure can be used to augment the performance of both of these algorithms.

   A priority queue can be used to augment the performance of both A* and Dijkstra's algorithm.
   In Dijkstra's algorithm, a priority queue can be used to efficiently select the node with the lowest cost from the set of unvisited nodes, improving the overall performance of the algorithm.
   Similarly, in A*, a priority queue can be used to keep track of the nodes with the lowest f-score, ensuring that the algorithm always considers the most promising nodes first and avoids exploring less likely paths.
   By using a priority queue to manage the search process, both algorithms can be made more efficient and effective at finding the shortest path in a graph.

5. Which algorithm (A* or Djikstra's) returns an optimal path?
   Both A* and Dijkstra's algorithms can find the shortest path from the start node to the goal node, resulting in an optimal path.
   Dijkstra's algorithm is guaranteed to find the shortest path in a graph as long as the edge weights are non-negative. A* is a more advanced search algorithm that uses a heuristic function to direct the search towards the goal node. When the heuristic function is admissible, meaning it never overestimates the true cost to the goal, A* is guaranteed to find the optimal path under the same conditions as Dijkstra's algorithm.
   Overall, both algorithms are well-known for their ability to find the shortest path in a graph and return an optimal path.