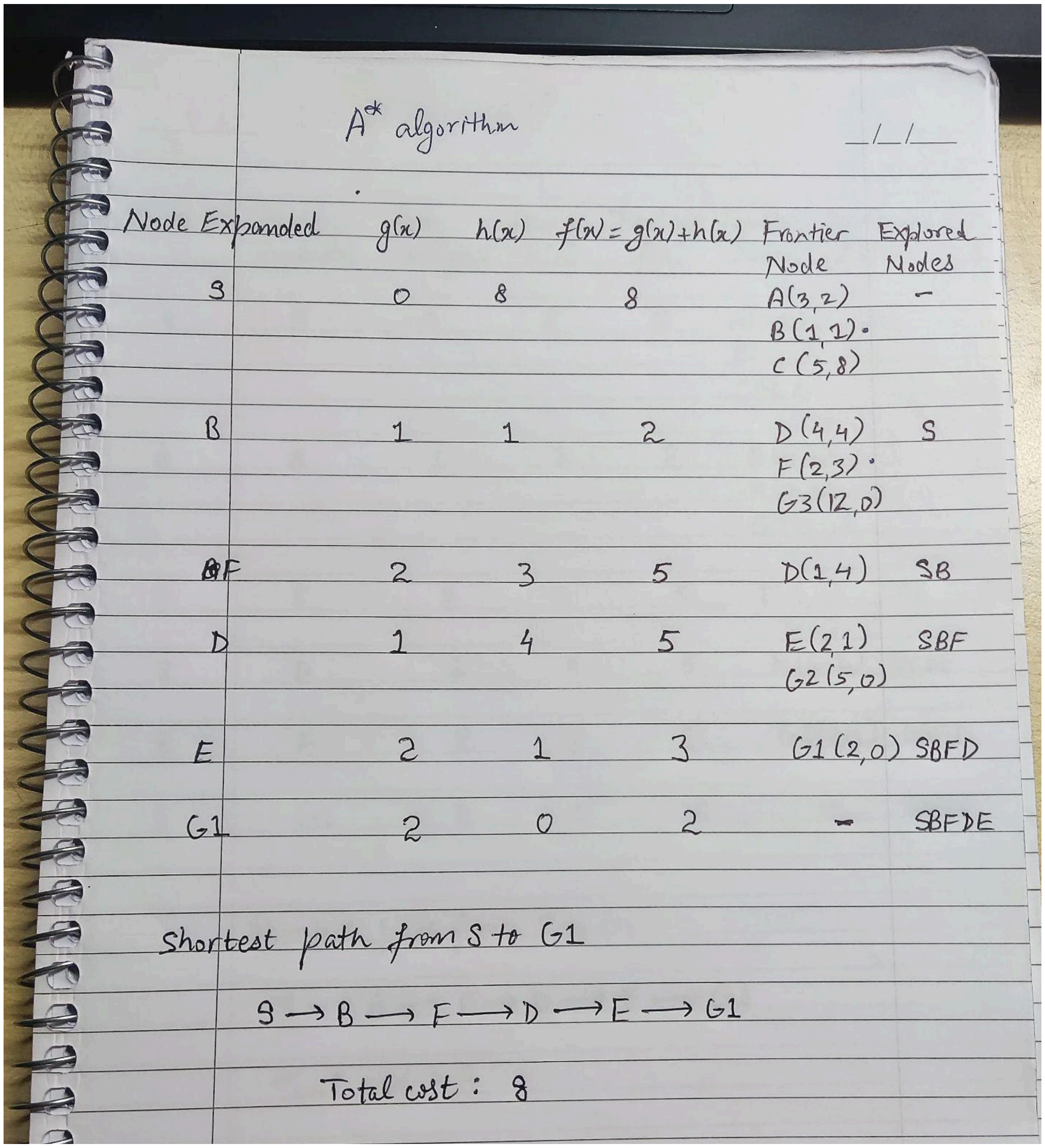
AI ASSIGNMENT – Search Algorithms

Saumil Lakra,2021097

Q1:

1. **A***



A* algorithm

__/__/__

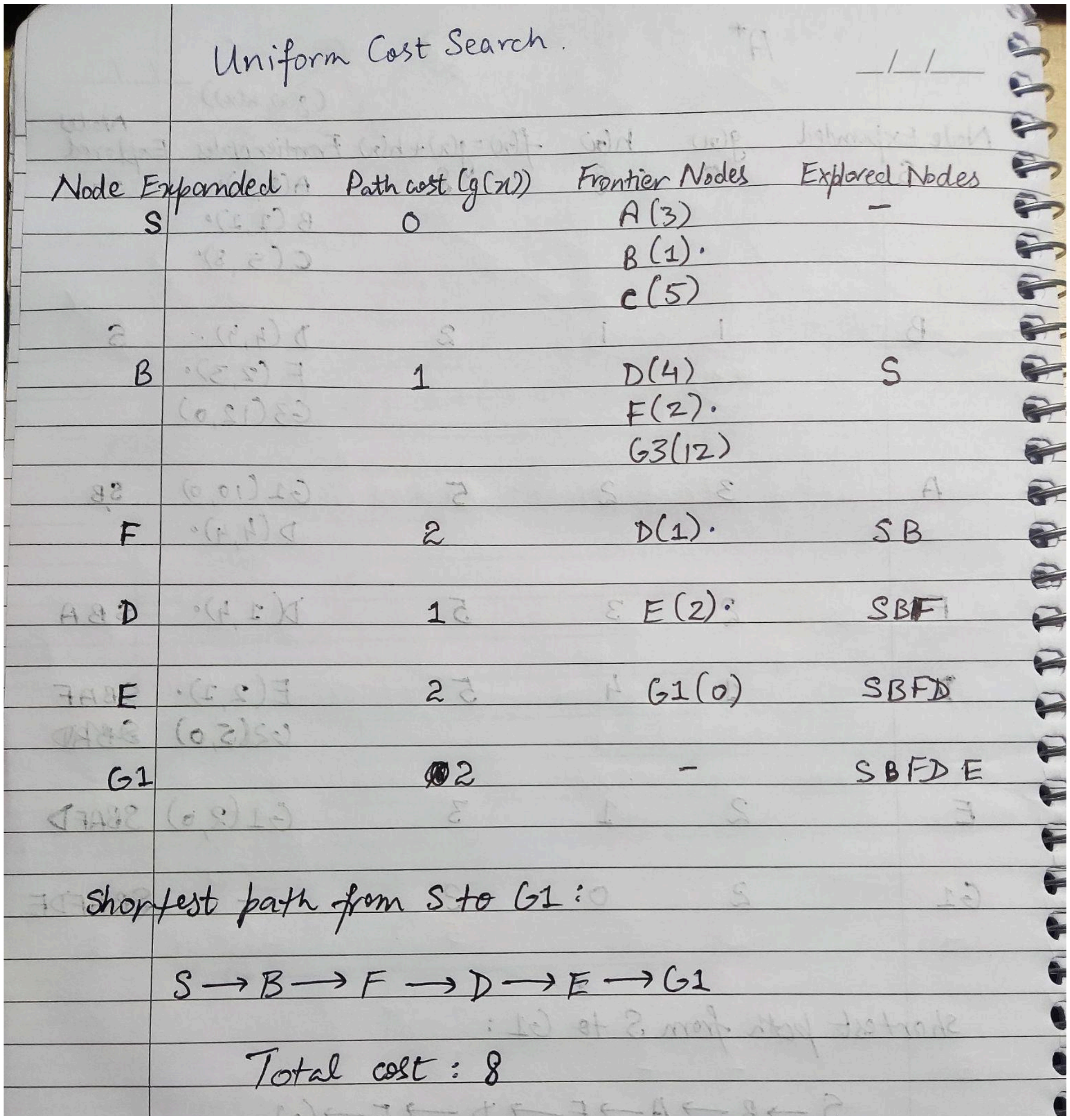
Node Expanded	$g(x)$	$h(x)$	$f(x) = g(x)+h(x)$	Frontier Node	Explored Nodes
S	0	8	8	A(3,2) B(1,1)· C(5,8)	–
B	1	1	2	D(4,4) F(2,3)· G3(12,0)	S
~~B~~F	2	3	5	D(1,4)	SB
D	1	4	5	E(2,1) G2(5,0)	SBF
E	2	1	3	G1(2,0)	SBFD
G1	2	0	2	–	SBFDE

Shortest path from S to G1

$S \rightarrow B \rightarrow F \rightarrow D \rightarrow E \rightarrow G1$

Total cost : 8

2. Uniform cost search



Uniform Cost Search.

Node Expanded	Path cost (g(n))	Frontier Nodes	Explored Nodes
S	0	A (3) B (1). C (5)	-

B	1	D(4)
		F(2).
		G3(12)
			S

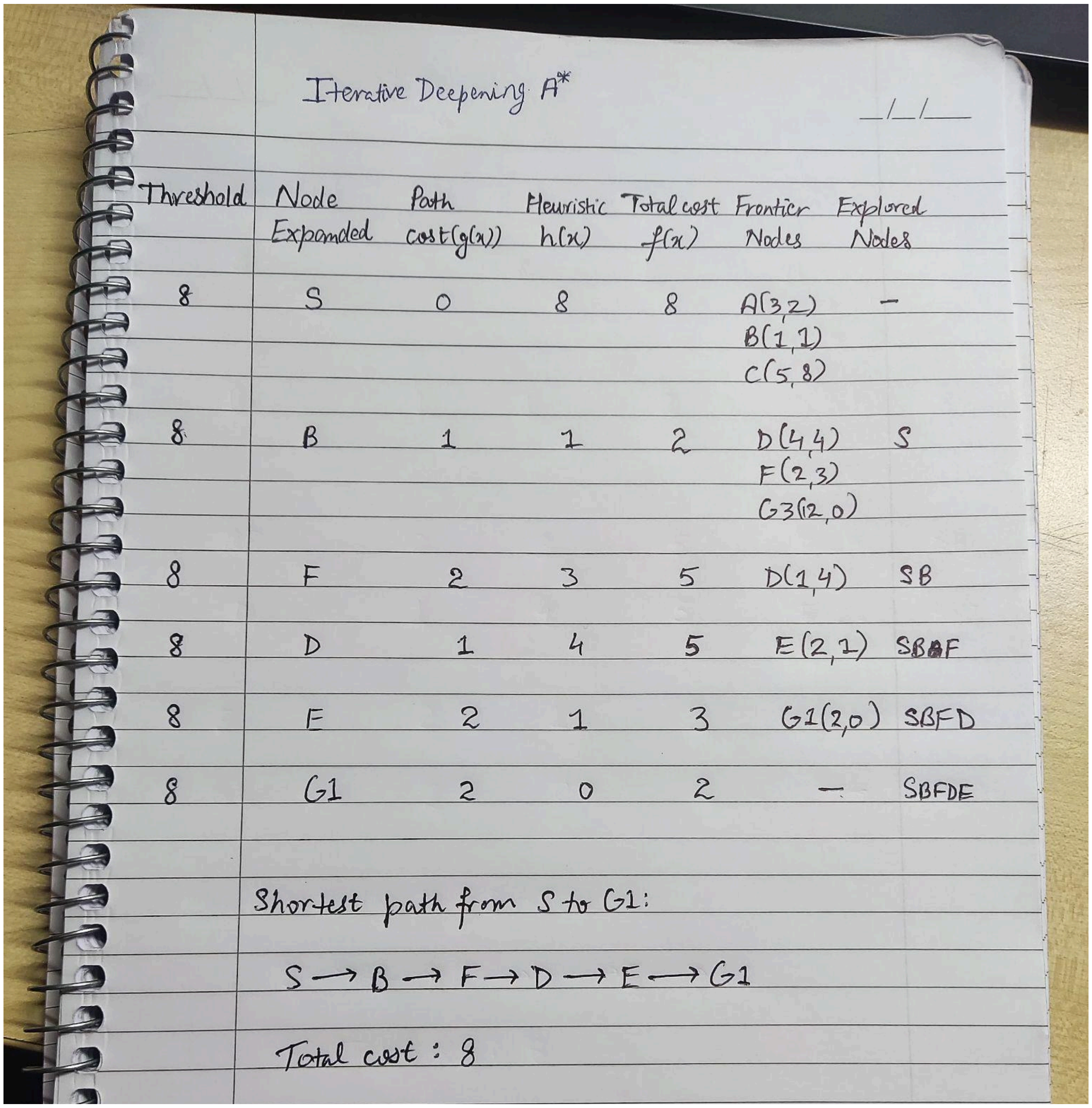
F	2	D(1).	S B
D	1	E (2).	SBF
E	2	G1(0)	SBFD
G1	2	-	S B F D E

Shortest path from S to G1:

$S \rightarrow B \rightarrow F \rightarrow D \rightarrow E \rightarrow G1$

Total cost: 8

3. Iterative Deepening A*



Iterative Deepening A*

Threshold	Node Expanded	Path cost(g(x))	Heuristic h(x)	Total cost f(x)	Frontier Nodes	Explored Nodes

8	S	0	8	8	A(3,2)
					B(1,1)
					C(5,8)
						–

8	B	1	1	2	D(4,4)
					F(2,3)
					G3(12,0)
						S

8	F	2	3	5	D(1,4)	SB
8	D	1	4	5	E(2,1)	SBF
8	E	2	1	3	G1(2,0)	SBFD
8	G1	2	0	2	–	SBFDE

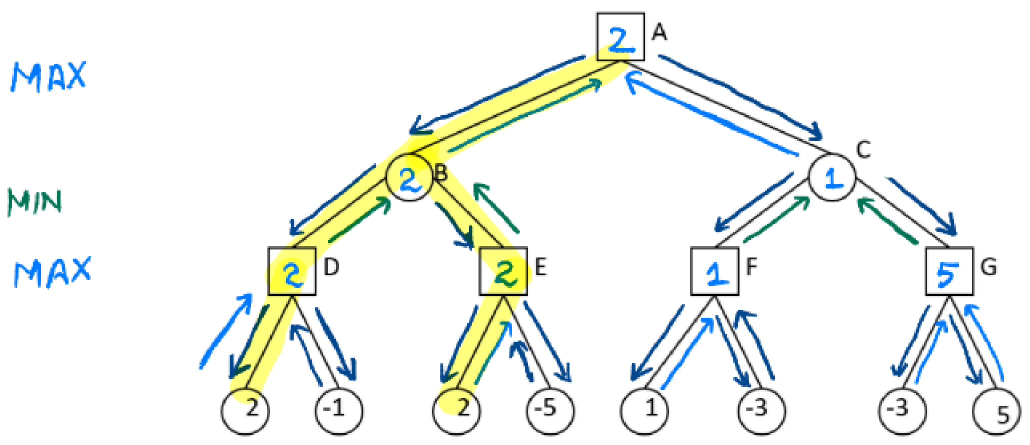
Shortest path from S to G1:

$S \longrightarrow B \longrightarrow F \longrightarrow D \longrightarrow E \longrightarrow G1$

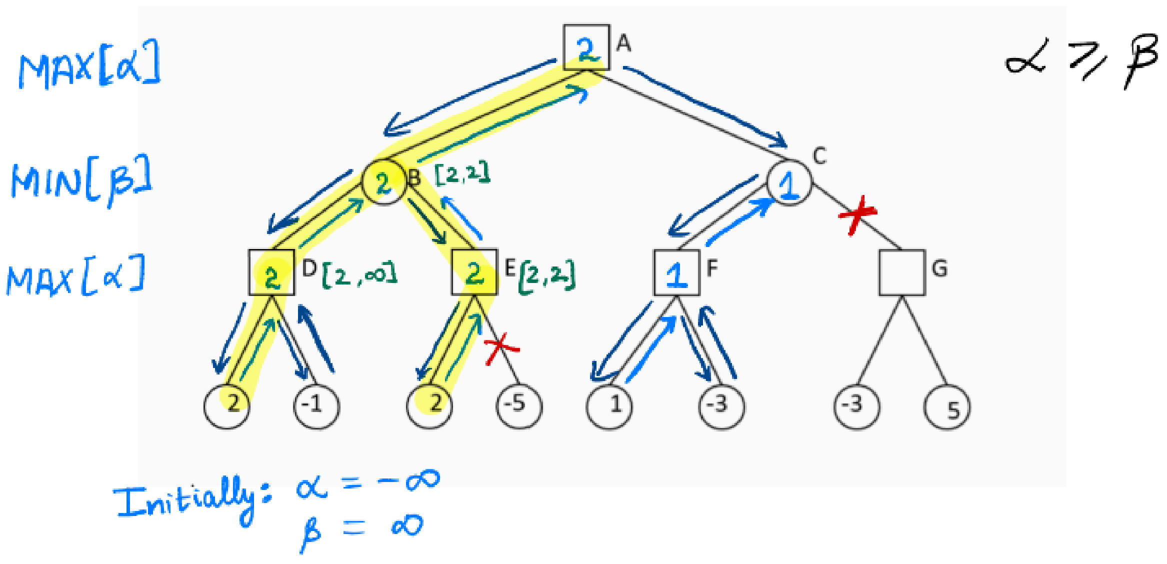
Total cost : 8

Q2.

(a) Part A



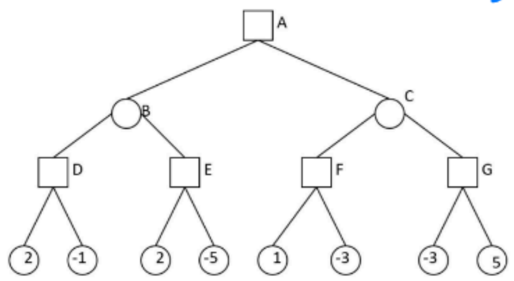
Min-max Algorithm



Alpha Beta Pruning

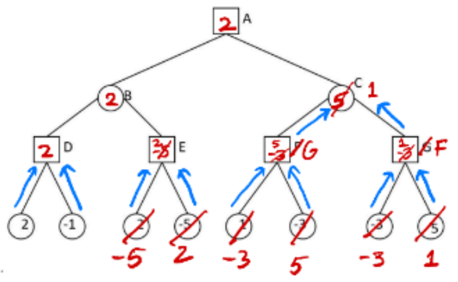
(b). Part B

(b) Part B



Best case: The given game tree is the best case for maximum pruning achieved by alpha-beta pruning. Even if we rearrange the leaf nodes or internal nodes, this is the best case. No more pruning can be achieved.

Worst case: Worst case scenario in this case will be if we exchange F & G node. And interchange leaf nodes of E & F.



Here I have avoided dfs traversal arrows to avoid confusion.

To avoid pruning of right edge of E, I interchanged the leaf nodes. Now after visiting -5, the algo has to go to the right edge i.e. 2. Now interchanging F & G positions & switching nodes of F ensures to update 5, we go to rightmost G as well which gives 1.

(c) Part C

In the best case, the pruning algorithm reduces the number of nodes to be evaluated by the minimax algorithm. In the best case, best scores/moves for the players come from the leftmost node which is evaluated first in the dfs algorithm. The left traversal already gives the best scores, so the right tree is not fully explored. For a complex graph, the complexity reduces drastically. Worst case of the algorithm is O(b^{d}) and the best case is O($b^{d/2}$).

Q3.

(b) Test case 1 doesn't give the same answer, rest of the cases give the same answer. The path obtained from both the algorithms may or may not be the same, since in a weighted graph there may be more than one path that has the minimum cost. This is because the IDS algorithm is DFS based whereas Bidirectional BFS is a BFS based algorithm. DFS based algo goes into the maximum given depth first and then explores the rest of the graph, on the other hand, BFS based algorithm explores in a level-wise manner.

(c) Iterative Deepening Search:
Memory used: 3.19921875 MiB
Time taken: 274.06890869140625 seconds

Bidirectional Search:
Memory used: 2.1796875 MiB
Time taken: 35.72912931442261 seconds

Memory consumption by Iterative Deepening Search is mores since it is DFS based algorithm, it maintains a recursion stack and maintains it across the depth. Bidirectional Search is more efficient because since it uses BFS algo it only needs to keep track of frontier nodes of a particular node.

Bidirectional Search is more efficient because it runs the search from both the start node and the end node. Both try to find each other in the graph. But in ids only the start node tries to find the end node. IDS conducts its search from the root node again and again for each new depth value.

(e)
All the test cases are matching for both the algorithms. But they may not always be the same. There may exist 2 paths with the same cost. This is because of the algorithm both the methods are built on. A* is a modified version of the dijkstra algorithm which uses heuristic function and Bidirectional Heuristic Search uses BFS as well as the heuristic function from both the start node and end node. Bidirectional Heuristic Search may find a different intermediate node while traversing the graph which gives the optimal path but doesn't exist in the path given by A* algorithm but both still give the same optimal cost for the path.

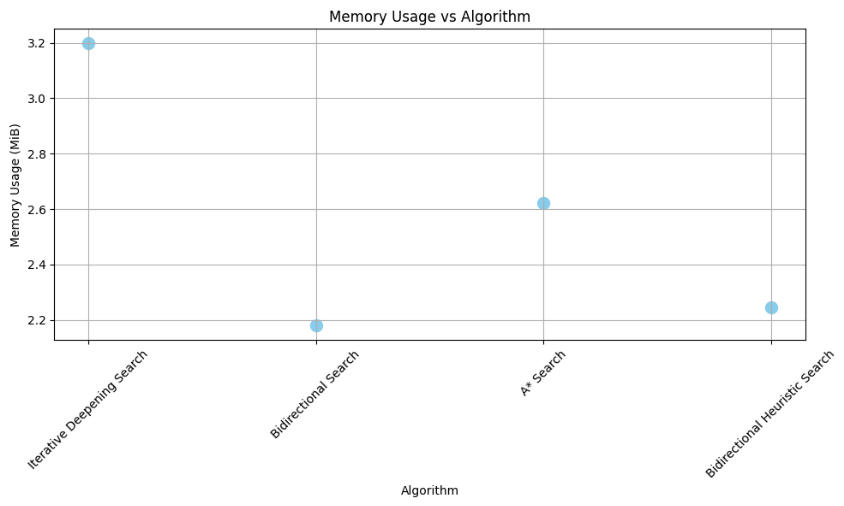
A* Search:
Memory used: 2.62109375 MiB
Time taken: 42.56570792198181 seconds

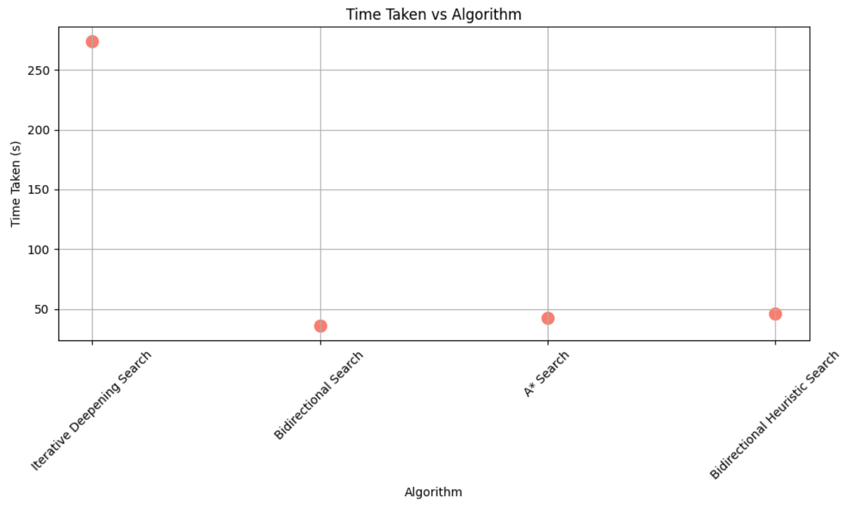
Bidirectional Heuristic Search:
Memory used: 2.24609375 MiB
Time taken: 46.15577006340027 seconds

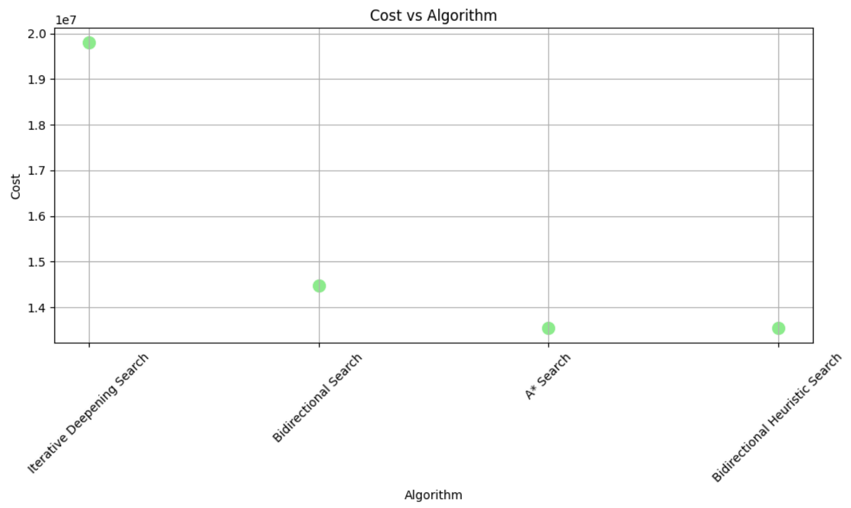
Bidirectional Heuristic Search is more memory efficient time than A* search because it concurrently runs from both start node and end node using bfs algorithm. It only stores frontier nodes of the given node. One the other hand, A* search runs only from the start node and stores nodes in its recursion stack.

Here, A* search algorithm is more efficient than Bidirectional Heuristic Search because it is the modified version of dijkstra algorithm.

(f)


Memory Usage vs Algorithm
Memory Usage (MiB)
3.2
3.0
2.8
2.6
2.4
2.2
Iterative Deepening Search
Bidirectional Search
A* Search
Bidirectional Heuristic Search
Algorithm


Time Taken vs Algorithm
Time Taken (s)
250
200
150
100
50
Iterative Deepening Search
Bidirectional Search
A* Search
Bidirectional Heuristic Search
Algorithm


Cost vs Algorithm
1e7
Cost
2.0
1.9
1.8
1.7
1.6
1.5
1.4
Iterative Deepening Search
Bidirectional Search
A* Search
Bidirectional Heuristic Search
Algorithm

Iterative Deepening Search:
Memory used: 3.19921875 MiB
Time taken: 274.06890869140625 seconds
Cost: 19803790.626988005

Bidirectional Search:
Memory used: 2.1796875 MiB
Time taken: 35.72912931442261 seconds
Cost: 14482908.243998483

A* Search:
Memory used: 2.62109375 MiB
Time taken: 42.56570792198181 seconds
Cost: 13540779.809997834

Bidirectional Heuristic Search:
Memory used: 2.24609375 MiB
Time taken: 46.15577006340027 seconds
Cost: 13555830.6589979

Metrics in the graph are time, memory and cost for each path. To calculate the time I used python's time library to find the start time and end time of the whole algorithm and their difference gave me the time algorithm took. To cost I made a function that takes in a the list of paths and returns the sum of the cost for all the paths for a particular algorithm .To find time I used psutil library of python.

Informed search algorithms give the most optimal cost for the path of pairs and are also time efficient w.r.t uninformed algorithms. Informed search algorithms are more complex and use heuristic functions for better estimation of the cost. Memory wise it is evident that dfs based algorithms(IDS and A*) are more time consuming than their BFS(bidirectional search and bidirectional heuristic search) counterparts. IDS algorithms takes most overall time because it searches the node again and again(if not found) for each new depth from the source node. So the time complexity increases dramatically for this case.

Drawback of using informed search algorithms over uninformed search algorithms is that they are more complex depending mainly on the quality of the heuristic function. A complex heuristic function may lead to increased time and memory consumption.

Benefits of using informed search algorithms over uninformed search algorithms is that they give the most optimal path heuristic helps to find the cost in a more efficient way. It gives the most probable path such that the start node can reach the end node. This ensures that there are less paths explored.

(g) **BONUS:**

Solution:

Bonus Problem: [[47, 48], [49, 47], [0, 49], [42, 29], [42, 30], [113, 42], [113, 43], [15, 46], [35, 15], [114, 84], [36, 114], [38, 36], [87, 88], [69, 124], [41, 70], [45, 17], [89, 90], [51, 50], [39, 40], [73, 72], [19, 100], [106, 107], [108, 109], [108, 112], [111, 108], [111, 110], [106, 111], [75, 106], [55, 56], [12, 57], [53, 54], [95, 96], [53, 95], [14, 53], [14, 99]]