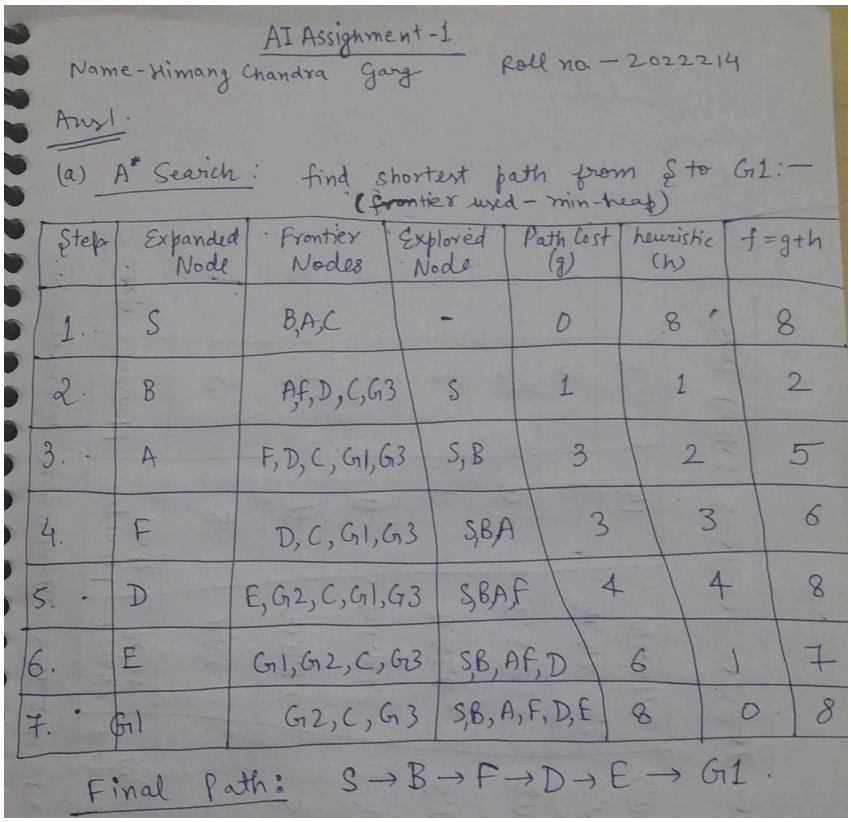
REPORT

Assignment - 1

Name - Himang Chandra Garg
Roll number - 2022214

Ans 1.



AI Assignment-1

Name-Himang Chandra Garg Roll no – 2022214

Ans1.

(a) A* Search: find shortest path from S to G1:–
(frontier used – min-heap)

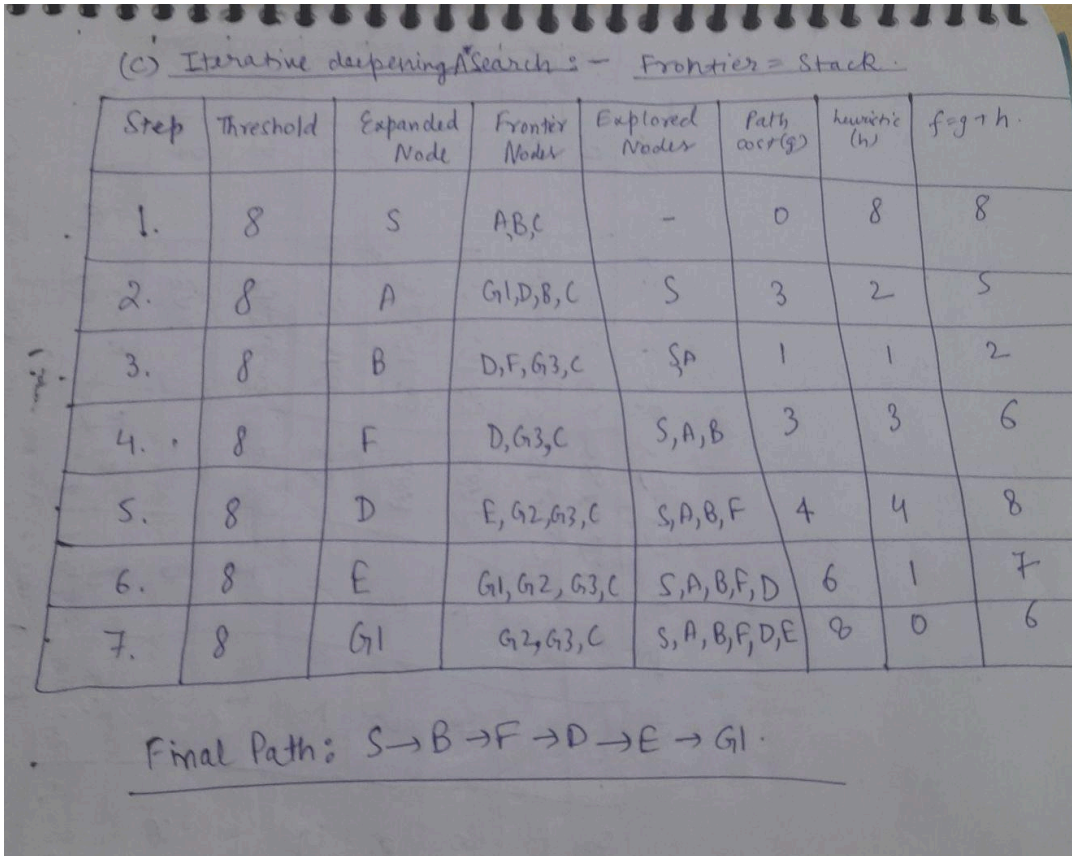
Step	Expanded Node	Frontier Nodes	Explored Node	Path Cost (g)	heuristic (h)	f=g+h
1.	S	B,A,C	-	0	8	8
2.	B	A,f,D,C,G3	S	1	1	2
3.	A	F,D,C,G1,G3	S,B	3	2	5
4.	F	D,C,G1,G3	S,B,A	3	3	6
5.	D	E,G2,C,G1,G3	S,B,A,F	4	4	8
6.	E	G1,G2,C,G3	S,B,Af,D	6	1	7
7.	G1	G2,C,G3	S,B,A,F,D,E	8	0	8

Final Path: S → B → F → D → E → G1.

(b) Uniform Cost Search:- Frontier used- Min heap.

Step	Expanded Node	Frontier Nodes	Explored Nodes	Path Cost (g).
1.	S	B, A, C	–	0
2.	B	A, F, C, D, G3	S	1
3.	A	F, G, D, G1, G3	S, B	3
4.	F	D, C, G1, G3	S, B, A	3
5.	D	G, E, G2, G1, G3	S, B, A, F	4
6.	C	E, G2, G1, G3	S, B, A, F, D	5
7.	E	G1, G2, G3	S, B, A, F, D, C	6
8.	G1	G2, G3	S, B, A, F, D, C, G1	8

final Path: S → B → F → D → E → G1



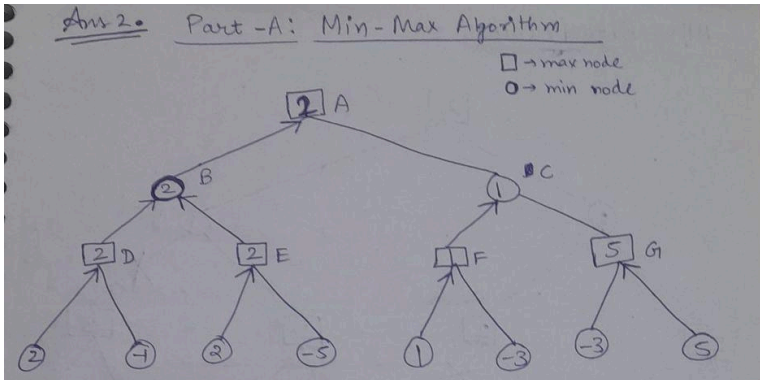
(c) Iterative deepening Search :- Frontier = Stack.

Step	Threshold	Expanded Node	Frontier Nodes	Explored Nodes	Path cost (g)	heuristic (h)	f=g+h.
1.	8	S	A,B,C	-	0	8	8
2.	8	A	G1,D,B,C	S	3	2	5
3.	8	B	D,F,G3,C	S,A	1	1	2
4.	8	F	D,G3,C	S,A,B	3	3	6
5.	8	D	E,G2,G3,C	S,A,B,F	4	4	8
6.	8	E	G1,G2,G3,C	S,A,B,F,D	6	1	7
7.	8	G1	G2,G3,C	S,A,B,F,D,E	8	0	6

Final Path: S → B → F → D → E → G1.

Ans 2.

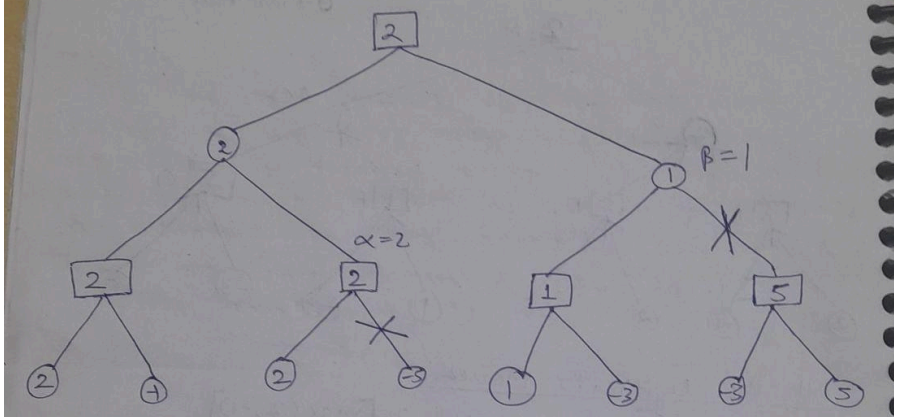
Ans 2. Part -A: Min-Max Algorithm



All the legitimate Moves:-

① Max node D chooses 2 [max (2,-1)]
② " " E chooses 2 [max (2,-5)]
③ Min node B chooses 2. [min (2,2)].
④ Max node F chooses 1. [max (1,-3)].
⑤ Max node G chooses 5. [max (5,-3)].
⑥ min node C chooses 1. [min (1,5)].
⑦ max node A chooses 2. [max (1,2)].

Alpha Beta Pruning: -

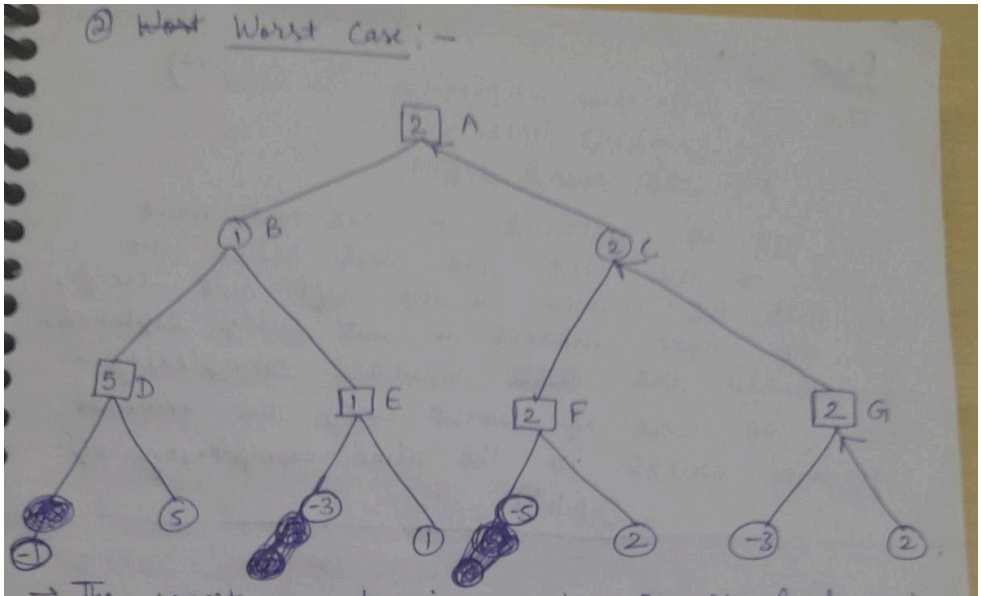


Part B:

① Best Case:-

- The best case pruning can be achieved by the order of leaf nodes shown above. Here the most optimal path is in the left subtree itself and hence no further pruning can be possible as we have pruned 3 leaf nodes completely.

② Worst Worst Case :-



→ The worst case pruning order of leaf nodes is shown above. Here the best path is present on the right most side of the tree hence no pruning be done. ~~Moreover~~ Also in every subtree the path from where the answer comes from is present on the right side.

Part (c) :→

The best case time complexity is $O(b^{d/2})$ here, b = branching factor.
d = look ahead depth.

This can be explained as the best move shifts to the left side and hence the dfs find the answer on the left side itself. and the right subtree is not fully explored. the reduces the ~~cost~~ overall complexity. For the case of worst case, the minmax algorithm works in the time complexity of $= O(b^d)$.

Ans 3.

(a) Done in python file.
(b)

Path obtained using my algorithm:

```
PS C:\Users\himan\Desktop\AI-Assignments> python -u "c:\Users\himan\Desktop\AI-Assignments\Assignment_1\code_RollNumber.py"
Enter the start node: 1
Enter the end node: 2
Iterative Deepening Search Path: [1, 27, 9, 2]
Bidirectional Search Path: [1, 7, 6, 2]
PS C:\Users\himan\Desktop\AI-Assignments> python -u "c:\Users\himan\Desktop\AI-Assignments\Assignment_1\code_RollNumber.py"
Enter the start node: 5
Enter the end node: 12
Iterative Deepening Search Path: [5, 97, 98, 12]
Bidirectional Search Path: [5, 97, 98, 12]
PS C:\Users\himan\Desktop\AI-Assignments> python -u "c:\Users\himan\Desktop\AI-Assignments\Assignment_1\code_RollNumber.py"
Enter the start node: 12
Enter the end node: 49
Iterative Deepening Search Path: []
Bidirectional Search Path: []
PS C:\Users\himan\Desktop\AI-Assignments> python -u "c:\Users\himan\Desktop\AI-Assignments\Assignment_1\code_RollNumber.py"
Enter the start node: 4
Enter the end node: 12
Iterative Deepening Search Path: [4, 6, 27, 9, 8, 5, 97, 98, 12]
Bidirectional Search Path: [4, 6, 2, 9, 8, 5, 97, 98, 12]
PS C:\Users\himan\Desktop\AI-Assignments>
```

The paths obtained using both algorithms are different for sample tests 1 and 4.

Path given in sample test case:
For ids-

```
#   Test Case 1:
#     - Start node: 1, Goal node: 2
#     - Return: [1, 7, 6, 2]

#   Test Case 2:
#     - Start node: 5, Goal node: 12
#     - Return: [5, 97, 98, 12]

#   Test Case 3:
#     - Start node: 12, Goal node: 49
#     - Return: []

#   Test Case 4:
#     - Start node: 4, Goal node: 12
#     - Return: [4, 6, 2, 9, 8, 5, 97, 98, 12]
```

For bbfs-

```
#   Test Case 1:
#     - Start node: 1, Goal node: 2
#     - Return: [1, 7, 6, 2]

#   Test Case 2:
#     - Start node: 5, Goal node: 12
#     - Return: [5, 97, 98, 12]

#   Test Case 3:
#     - Start node: 12, Goal node: 49
#     - Return: []

#   Test Case 4:
#     - Start node: 4, Goal node: 12
#     - Return: [4, 6, 2, 9, 8, 5, 97, 98, 12]
```

The paths may be different because of the choice of graph traversal. I have used stack as a frontier for iterative deepening search and queue as a frontier for bidirectional breadth-first search. But the depth of the path should always be the same for ids.

©

Due to large execution time I have calculated time and memory for first 20 nodes only.

ids:

```
Iterative Deepening Search Path: [19, 73, 39, 20, 57, 12, 98, 35, 15]
Start Node: 19, End Node: 16
Iterative Deepening Search Path: []
Start Node: 19, End Node: 17
Iterative Deepening Search Path: [19, 75, 18, 55, 52, 51, 45, 17]
Start Node: 19, End Node: 18
Iterative Deepening Search Path: [19, 75, 18]
Start Node: 19, End Node: 19
Iterative Deepening Search Path: [19]
Execution Time: 13.309344291687012 seconds
Memory Used: 0.16246795654296875 MB
PS C:\Users\himan\Desktop\AI-Assignments>
```

Execution Time: 13.309344291687012 seconds
Memory Used: 0.16246795654296875 MB

Bbfs:

```
Bidirectional Search Path: [19]
Execution Time: 28.100208282470703 seconds
Memory Used: 0.2323760986328125 MB
```

Execution Time: 28.100208282470703 seconds
Memory Used: 0.2323760986328125 MB

Generally, the execution time is larger for ids in the case of large graphs but since I have only tested on the first 20 nodes the execution time is more for bbfs.

The Memory used is almost similar in both algorithms.

(d) Done in python file.

(e)

(b).

Paths obtained using my code:

```
PS C:\Users\himan\Desktop\AI-Assignments> python -u "c:\Users\himan\Desktop\AI-Assignments\Assignment_1\code_RollNumber.py"
Enter the start node: 1
Enter the end node: 2
A* Path: [1, 27, 9, 2]
Bidirectional Heuristic Search Path: [1, 27, 9, 11, 2]
PS C:\Users\himan\Desktop\AI-Assignments> python -u "c:\Users\himan\Desktop\AI-Assignments\Assignment_1\code_RollNumber.py"
Enter the start node: 5
Enter the end node: 12
A* Path: [5, 97, 28, 10, 12]
Bidirectional Heuristic Search Path: [5, 97, 28, 10, 12]
PS C:\Users\himan\Desktop\AI-Assignments> python -u "c:\Users\himan\Desktop\AI-Assignments\Assignment_1\code_RollNumber.py"
Enter the start node: 12
Enter the end node: 49
A* Path: []
Bidirectional Heuristic Search Path: []
PS C:\Users\himan\Desktop\AI-Assignments> python -u "c:\Users\himan\Desktop\AI-Assignments\Assignment_1\code_RollNumber.py"
Enter the start node: 4
Enter the end node: 12
A* Path: [4, 6, 27, 9, 8, 5, 97, 28, 10, 12]
Bidirectional Heuristic Search Path: [4, 6, 27, 9, 8, 5, 20, 57, 12]
PS C:\Users\himan\Desktop\AI-Assignments>
```

The paths obtained using both the algorithms are different for sample test 1 and 4.

Paths given for A*:

```
# Sample Test Cases:

#   Test Case 1:
#     - Start node: 1, Goal node: 2
#     - Return: [1, 7, 6, 2]

#   Test Case 2:
#     - Start node: 5, Goal node: 12
#     - Return: [5, 97, 28, 10, 12]

#   Test Case 3:
#     - Start node: 12, Goal node: 49
#     - Return: []

#   Test Case 4:
#     - Start node: 4, Goal node: 12
#     - Return: [4, 6, 27, 9, 8, 5, 97, 28, 10, 12]
```

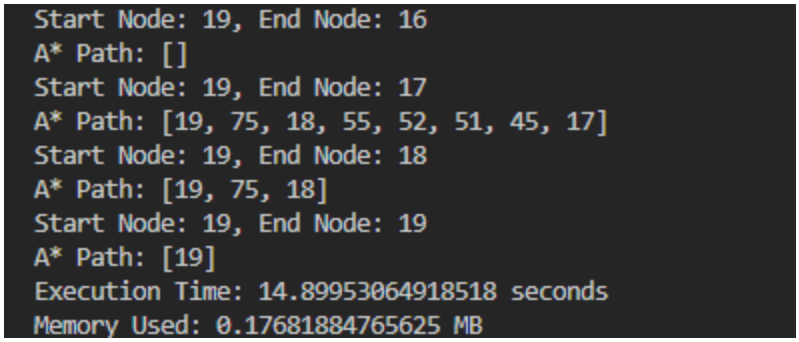
Paths given for Bidirectional A*:

```
# Sample Test Cases:

#   Test Case 1:
#     - Start node: 1, Goal node: 2
#     - Return: [1, 7, 6, 2]

#   Test Case 2:
#     - Start node: 5, Goal node: 12
#     - Return: [5, 97, 98, 12]

#   Test Case 3:
#     - Start node: 12, Goal node: 49
#     - Return: []

#   Test Case 4:
#     - Start node: 4, Goal node: 12
#     - Return: [4, 34, 33, 11, 32, 31, 3, 5, 97, 28, 10, 12]
```

The paths obtained with these algorithms may be different in the case when more than one node has the same value of f=path cost + heuristic, then the algorithm will have 2 paths and hence may result in different paths.

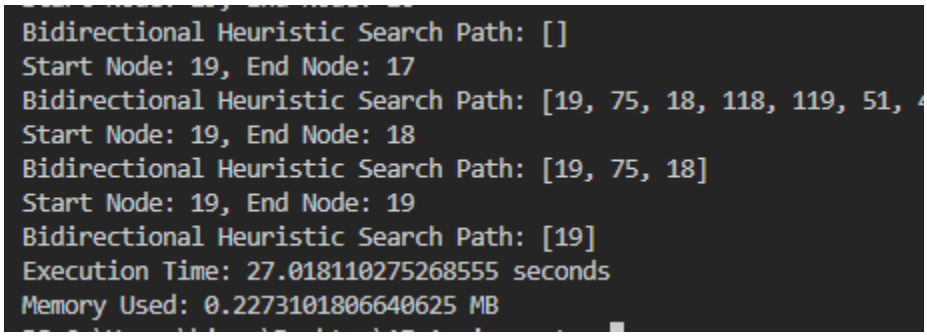
© due to less time left I have only run for the first 20 nodes.

A* :



Execution Time: 14.89953064918518 seconds
Memory Used: 0.17681884765625 MB

B A*:



Execution Time: 27.018110275268555 seconds
Memory Used: 0.2273101806640625 MB

The time taken and memory used for bidirectional A* is more in this case because the number of nodes were less. If we run the algorithms for complete graph the time taken in bidirectional A* would definitely be less.

(g) Done in python file.