AI Assignment - Search Algorithms

Ritika Thakur | 2022408

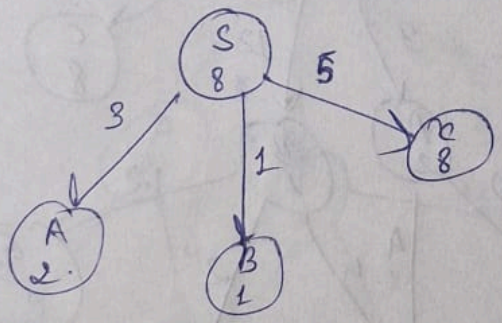
Question 1:

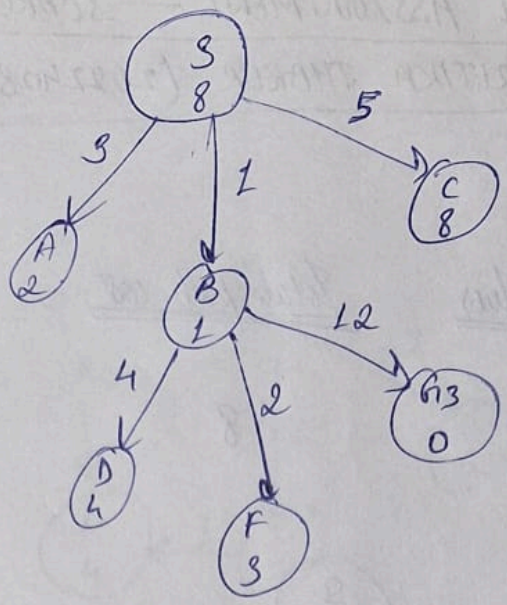
AI ASSIGNMENT — SEARCH ALGORITHMS

RITIKA THAKUR (2022408)

Q1. @ A*

Frontier	f-values	Total path cost	Path
S 8	S = 8	8	S
B 2 A 5 C 13	A = 5 B = 2 C = 13	2	S → B
A 5 F 6 D 9 C 13 G_3 13	D = 9 F = 6 G_3 = 13	5	S → A





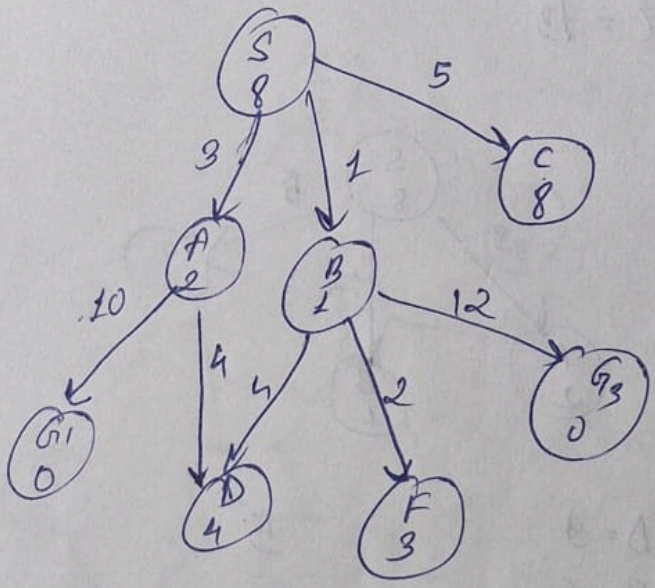
F	6
D	9
C	13
G_1	13
G_3	13

$G_1 = 13$

$D = 9$

6

S → B → F

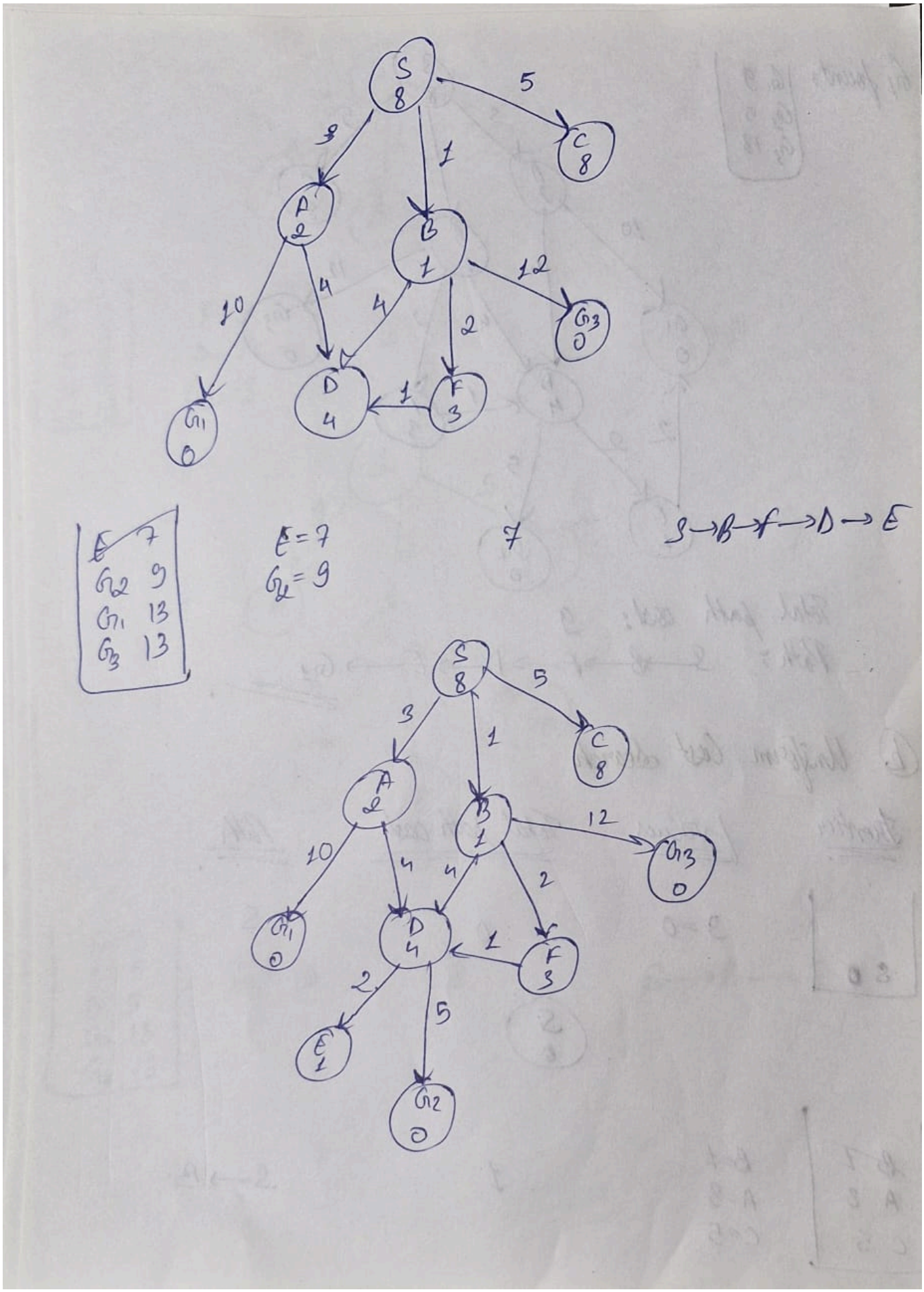


D	8
C	13
G_1	13
G_3	13

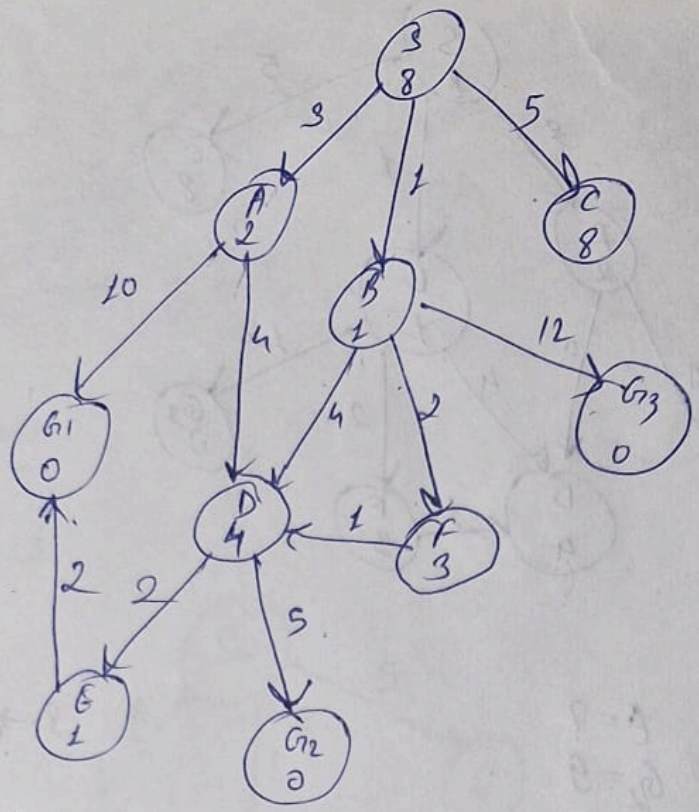
$D = 8$

8

S → B → F → D


S
8
5
3
1
C
8
A
2
B
1
12
10
4
4
2
G3
0
G1
0
D
4
1
F
3
E = 7
G2 = 9
7
S → B → F → D → E
E 7
G2 9
G1 13
G3 13
S
8
5
3
1
C
8
A
2
B
1
12
10
4
4
2
G3
0
G1
0
D
4
1
F
3
2
5
E
1
G2
0

G_1 found: | G_1 9 | G_2 9 | G_3 13 |



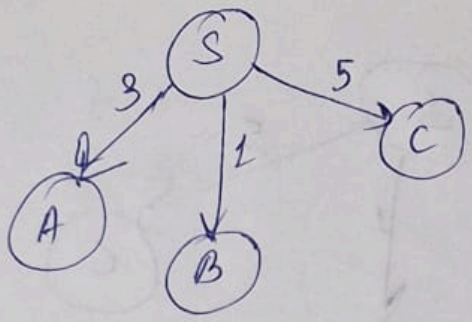
Total path cost: 9

Path: S → B → F → D → E → G_1

(b) Uniform Cost Search

Frontier	f-values	Total path cost	Path
S 0	S = 0	0	S
B 1 A 3 C 5	B = 1 A = 3 C = 5	1	S → B

S
8

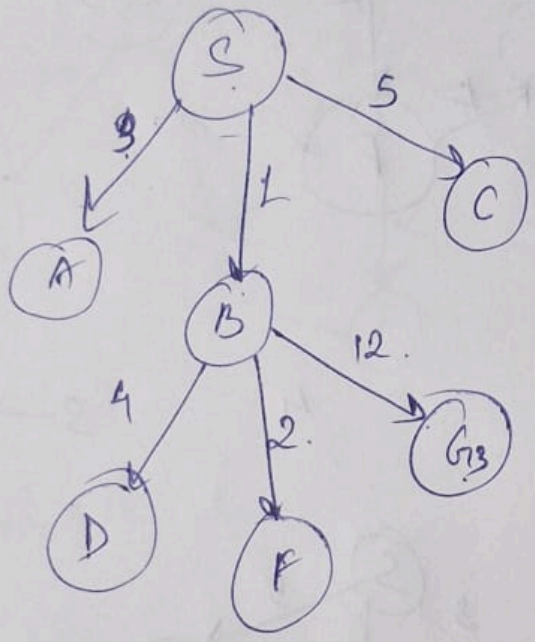


~~A~~	~~3~~
F	8
C	5
D	5
G_3	13

F=3
D=5
G_3=13

3

S → A

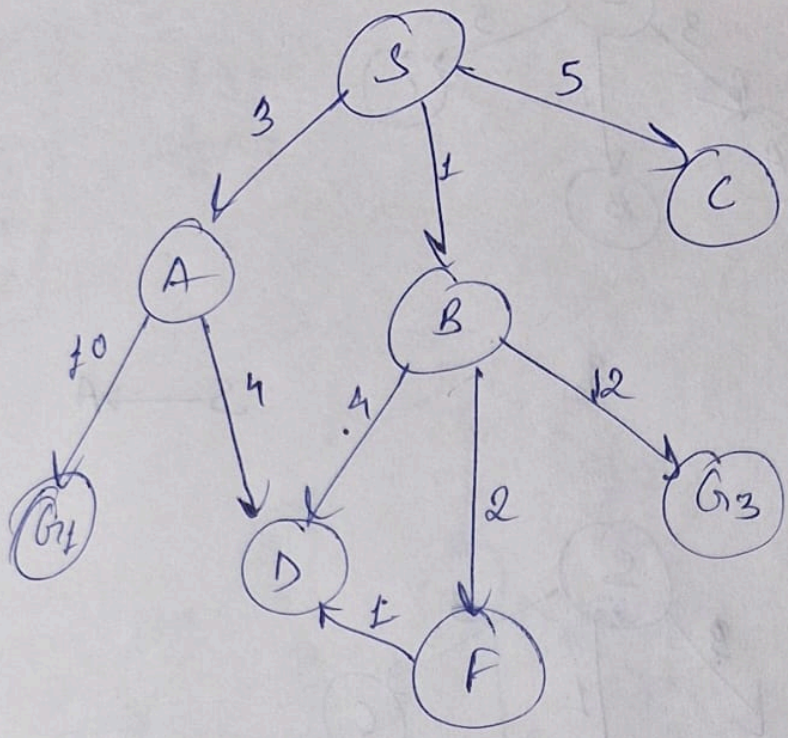


~~F~~	~~3~~
C	5
D	5
G_1	13
G_3	13

G_1 = 13

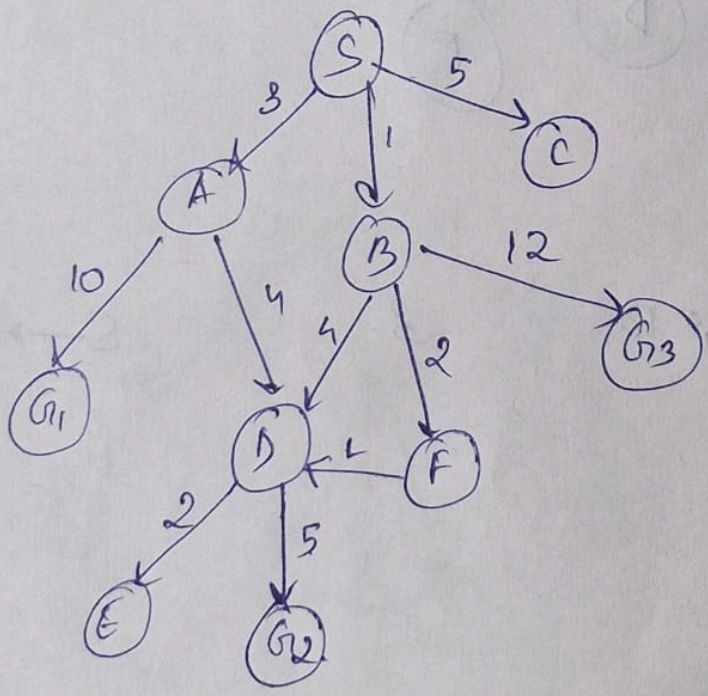
3

S → B → F



~~D~~	4
C	5
G_1	13
G_3	13

D=4. 4 S → B → F → D



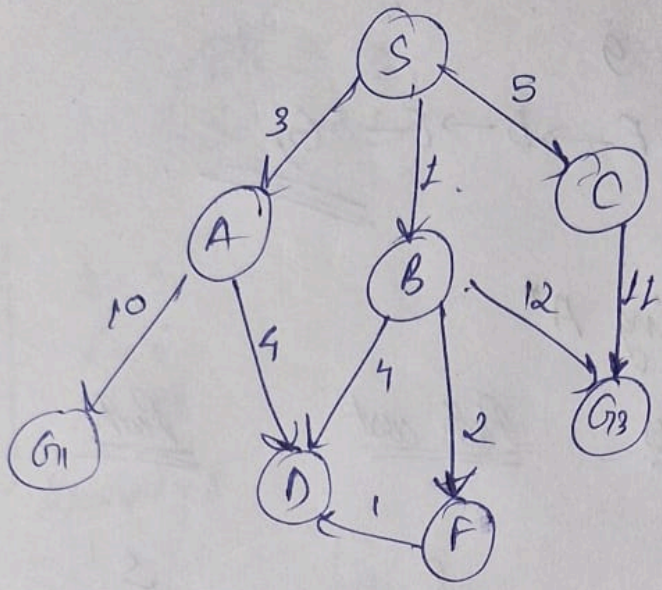
~~C~~	5
E	7
G_1	13
G_2	9
G_3	13

$E = 6$

$G_2 = 9$

5

$S \rightarrow C$

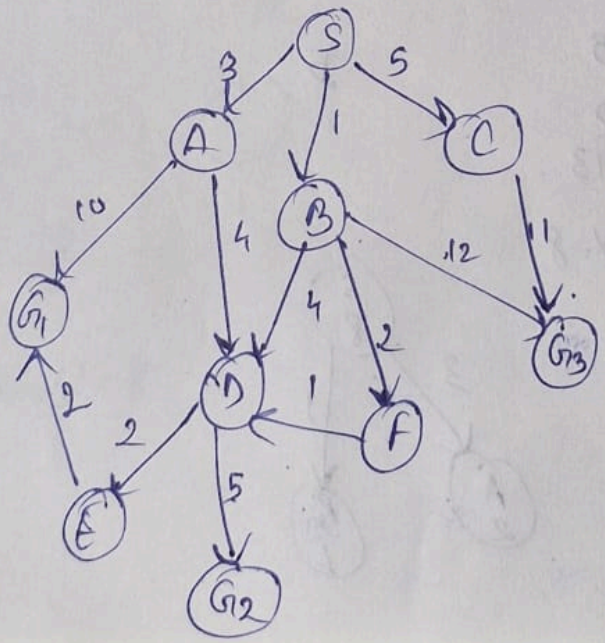


~~E~~	~~7~~
G_2	9
G_1	13
G_3	13

$G_3 = 13$

6

$S \rightarrow B \rightarrow \cancel{F} \rightarrow D \rightarrow E$



G_1	9
G_2	9
G_3	13

$G_1 = 9$ 9 $S \to B \to F \to D \to E \to G_1$

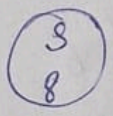
Path found.

Total path cost = 9

Path : $S \to B \to F \to D \to E \to G_1$

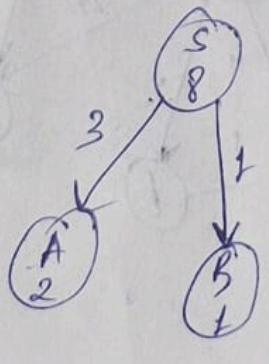
© Iterative Deepening A*

Frontier (Visited)	f-values	Path cost	Path
S	S = 8	8	S
	Threshold = 8		



C 13	A = 5 B = 2 C = 13	5	S → A

Threshold = 8

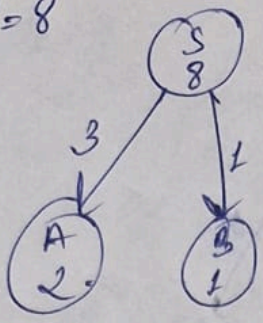


D 11
C 13
G_1 13

$D = 11$
$G_1 = 13$
threshold = 8

5

$S \rightarrow A$

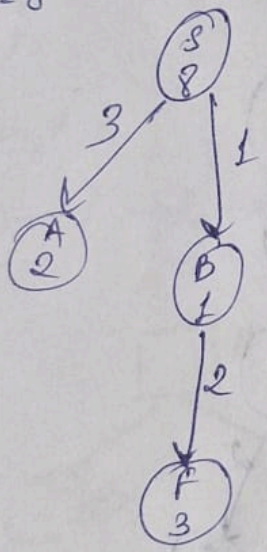


D 9
C 13
G_1 13
G_3 13

$D = 9$
$F = 6$
$G_3 = 13$
threshold = 8

2

$S \rightarrow B$

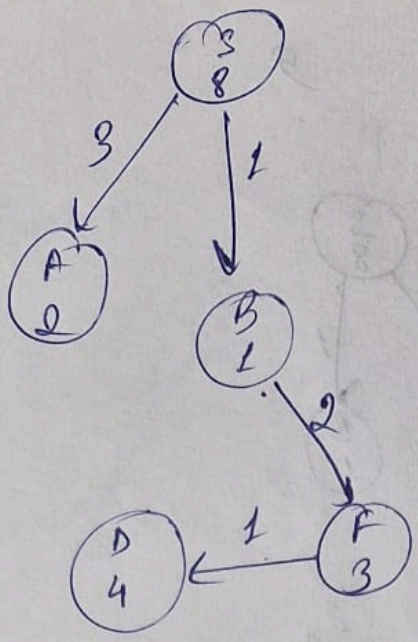


C 13
G_1 13
G_3 13

$D = 8$
threshold = 8

6

$S \rightarrow B \rightarrow F$



G_2	9
C	13
G_1	13
G_3	13

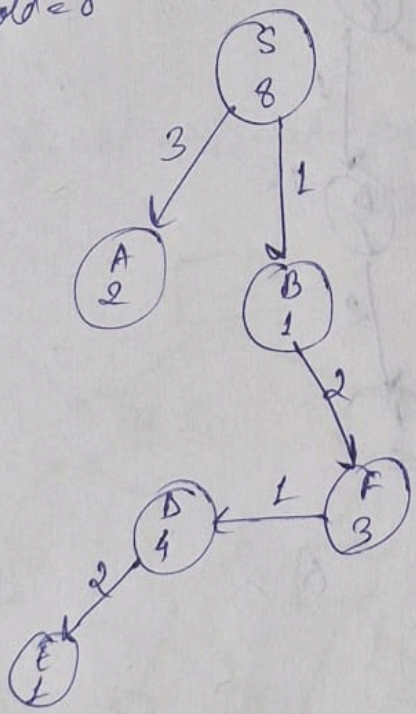
$E = 7$

$G_2 = 9$

Threshold = 8

8

$S \rightarrow B \rightarrow F \rightarrow D$



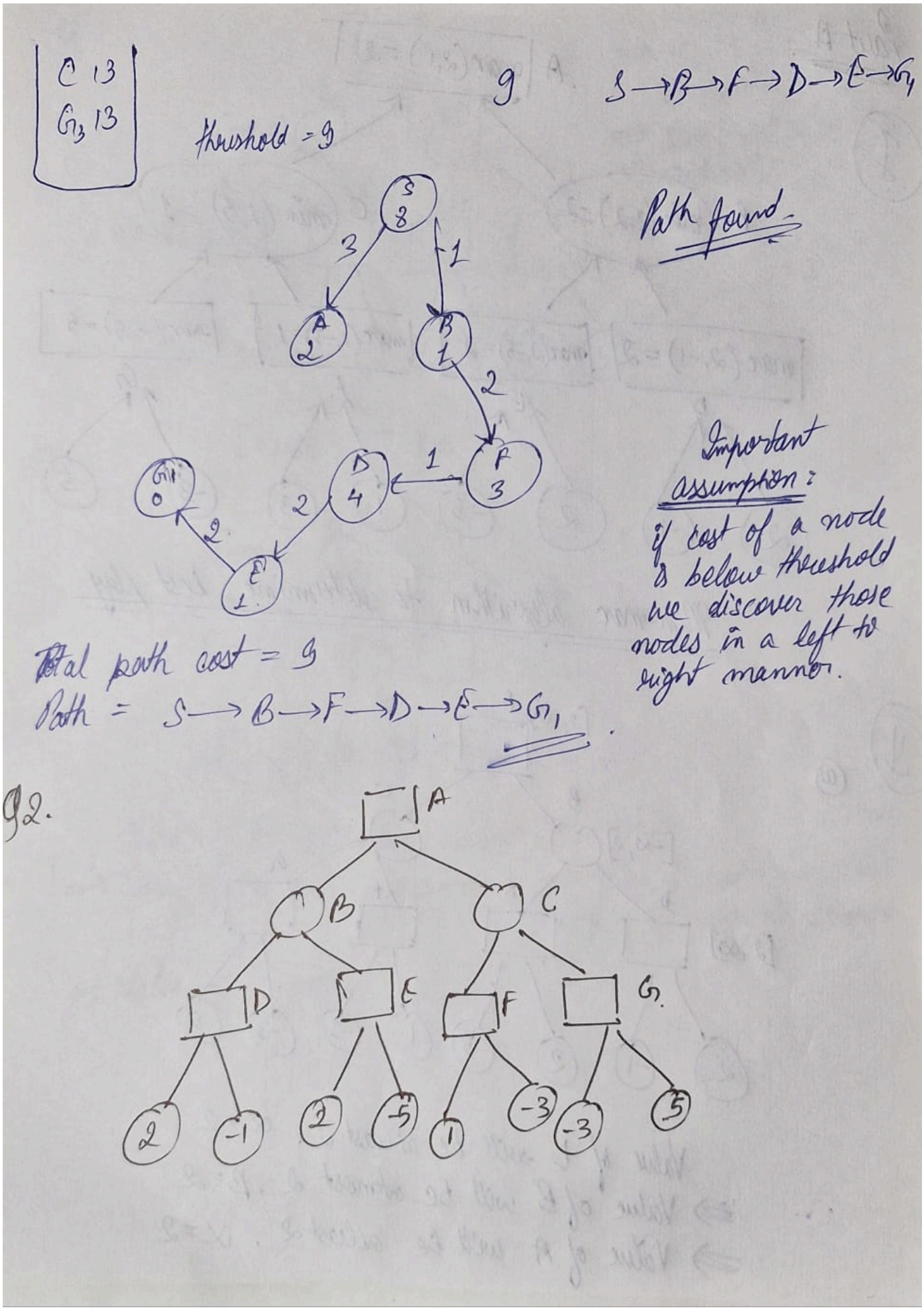
G_1	9
G_2	9
C	13
G_3	13

$G_1 = 9$

Threshold = 8

7

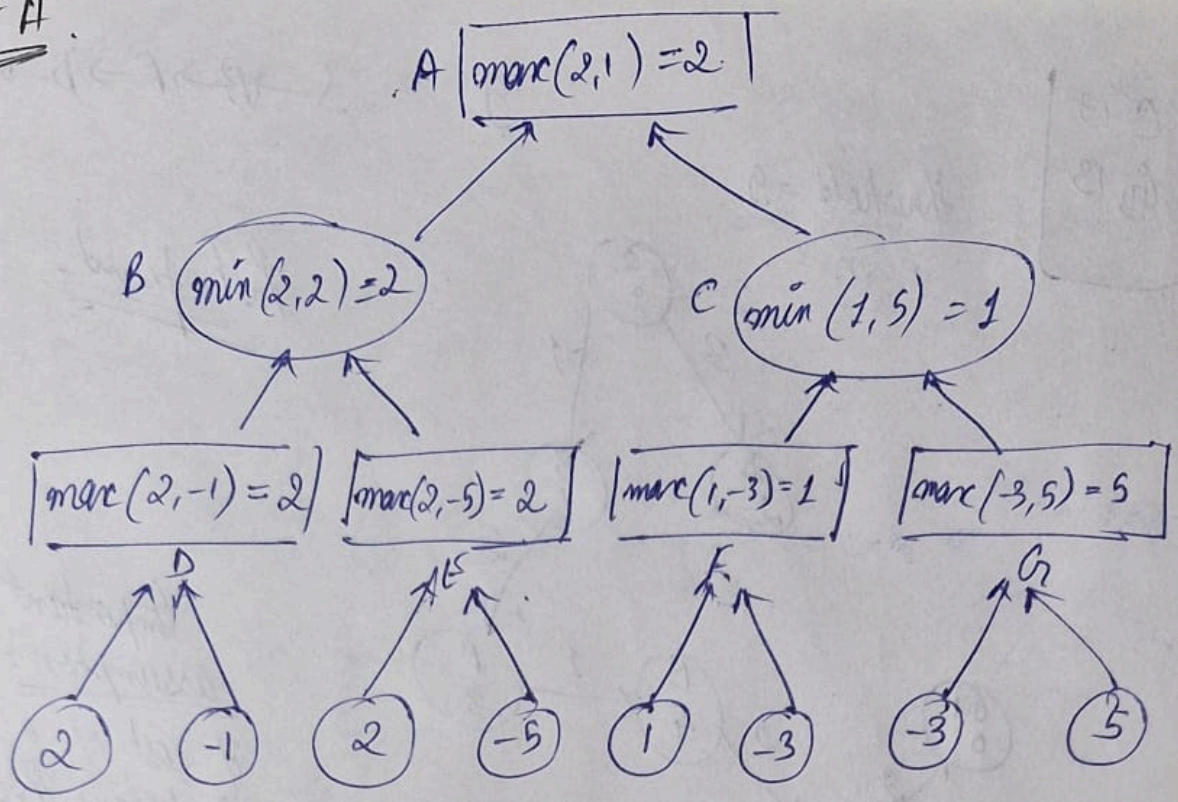
$S \rightarrow B \rightarrow F \rightarrow D \rightarrow E$


C 13
G3 13
Threshold = 9
9
S → B → F → D → E → G1
Path found.
S 8
A 2
B 1
F 3
D 4
E 1
G1 0
3
1
2
1
2
2
Important assumption:
if cost of a node is below threshold we discover those nodes in a left to right manner.
Total path cost = 9
Path = S → B → F → D → E → G1
Q2.
A
B
C
D
E
F
G
2
-1
2
-5
1
-3
-3
5

Question 2:

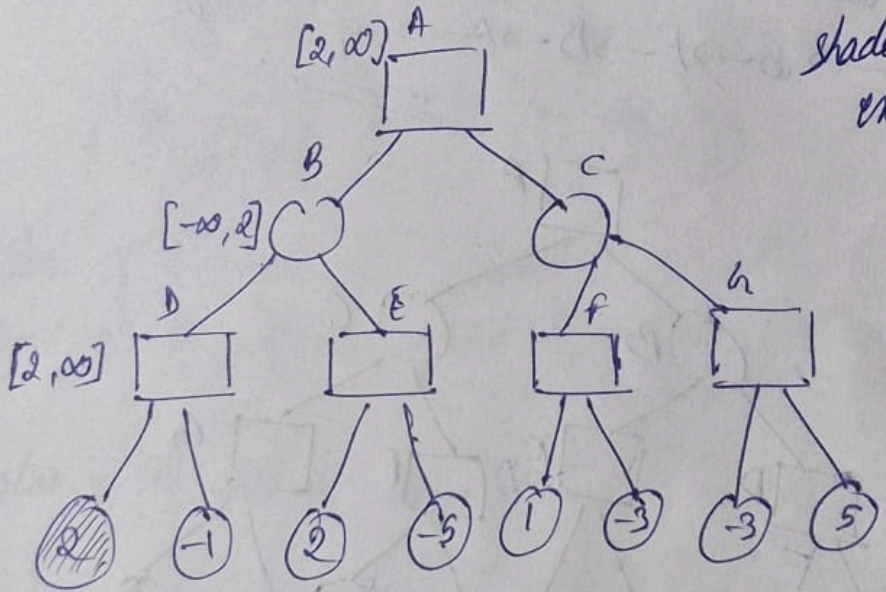
Part A.

(1)



Min-max algorithm to determine best play

(2) (a)



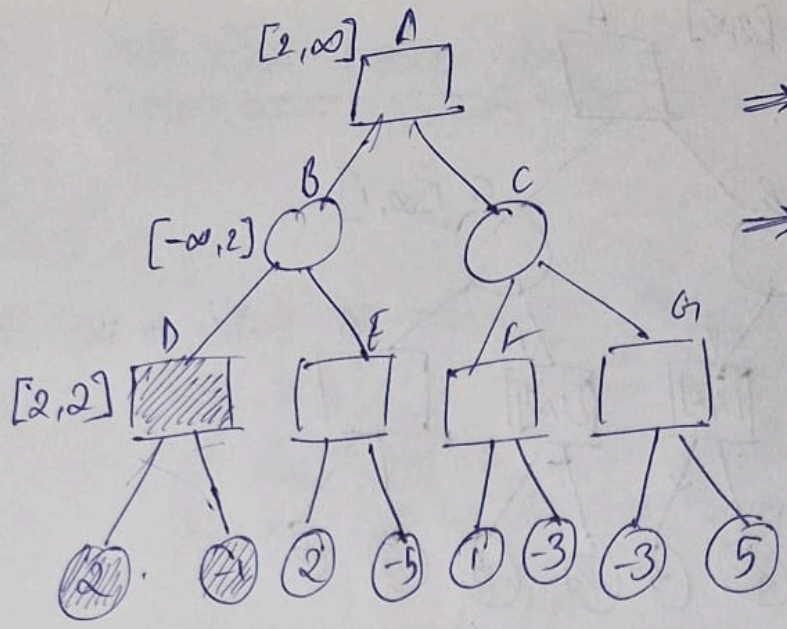
shaded nodes: explored nodes

Value of D will be atleast 2. $\alpha = 2$

$\Rightarrow$ Value of B will be atmost 2. $\beta = 2$

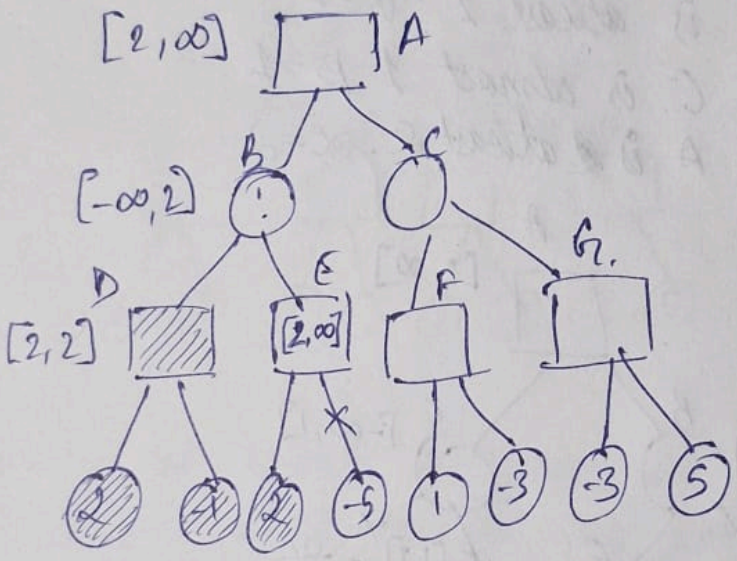
$\Rightarrow$ Value of A will be atleast 2. $\alpha = 2$

(b)



Value of D is 2
⇒ Value of B will be atmost 2. $\beta = 2$
⇒ Value of A will be atleast 2. $\alpha = 2$.

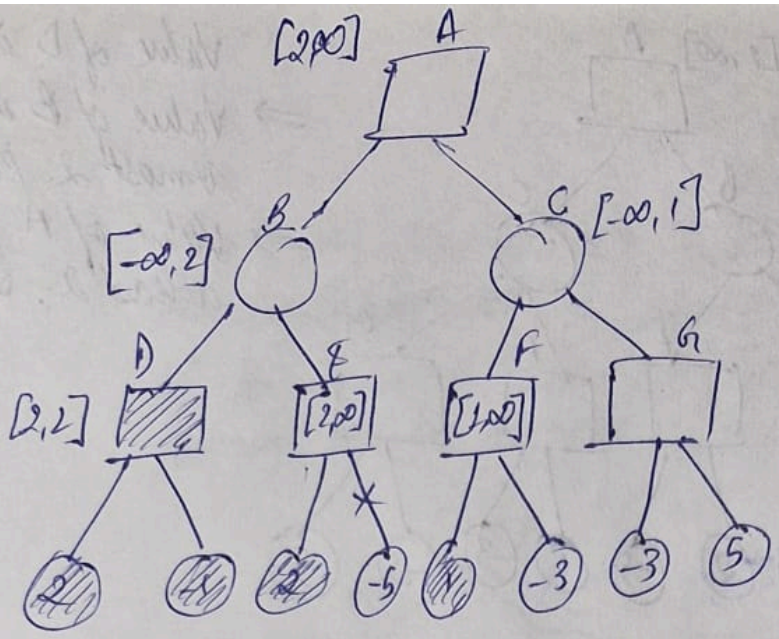
(c)



Value of E will be atleast 2. $\alpha = 2$.
⇒ Value of B will be atmost 2 ⇒ We do not need to explore the second branch of E.
$\beta = 2$.

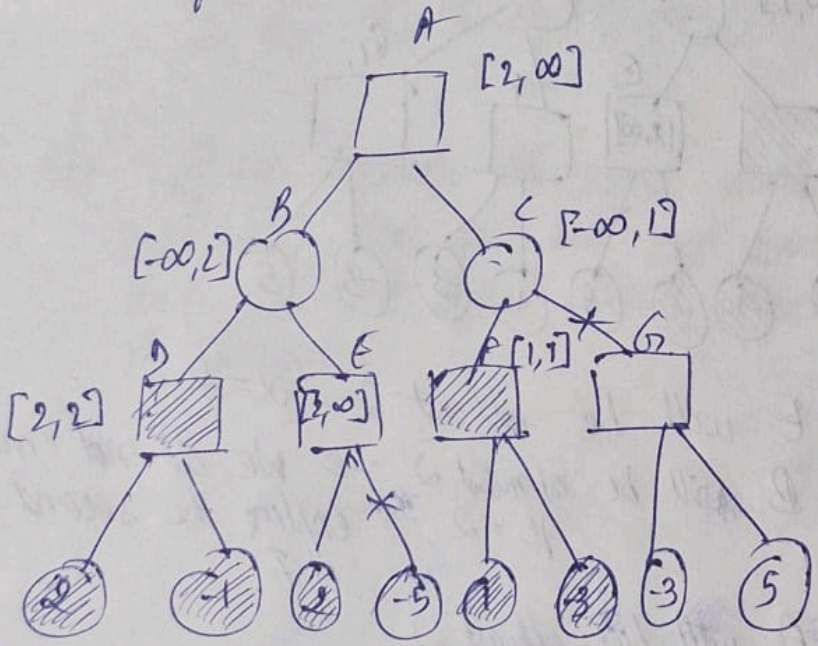
⇒ Value of A will be atleast 2.
$\alpha = 2$.

(d).



Value of F is atleast 1 $\alpha = 1$
$\Rightarrow$ Value of C is atmost 1 $\beta = 1$
$\Rightarrow$ Value of A is atleast 2. $\alpha = 2$

(e)

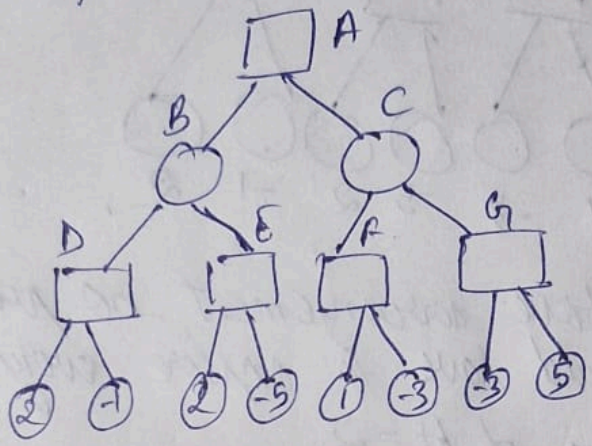


Value of F is 1.
$\Rightarrow$ Value of C is atmost 1 $\beta = 1$
$\Rightarrow$ Value of A is atleast 2. $\alpha = 2$.
$\Rightarrow$ If value of $G_1 < 1 \Rightarrow C < 1 \Rightarrow A = \max(B, C) = 2$.
$\Rightarrow$ If value of $G_1 > 1 \Rightarrow C = 1 \Rightarrow A = \max(B, C) = 2$.

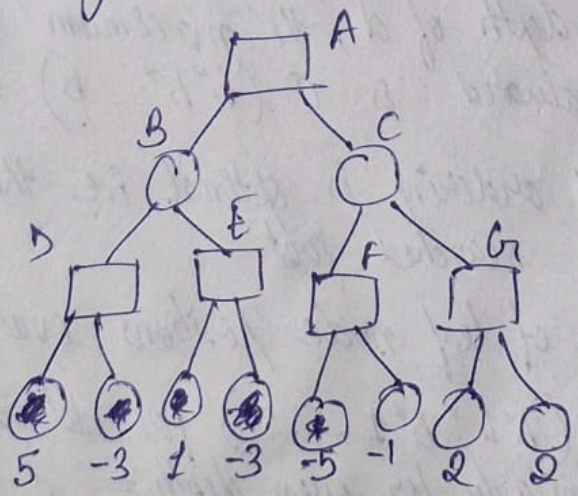
(a). ⇒ Final value for A = 2.

Part B

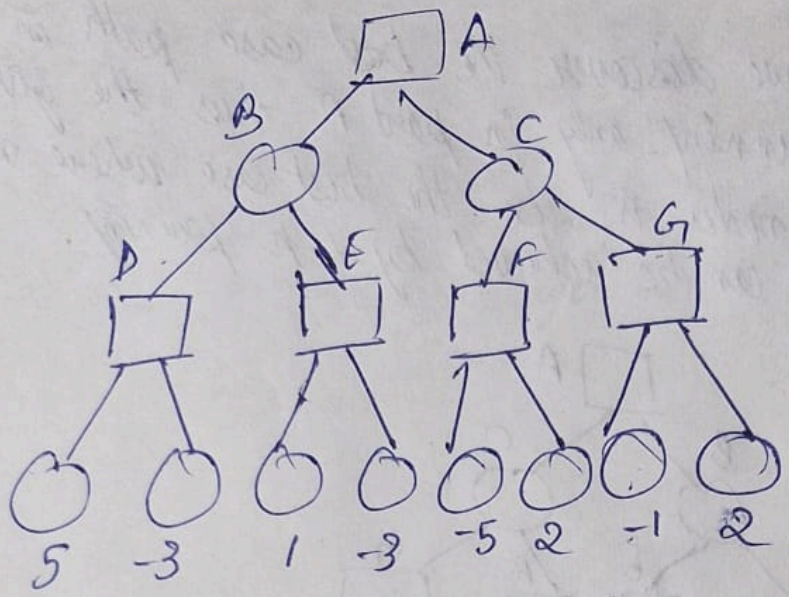
Best Case : Since we discover the best case path in the very beginning only in part A thus the given order of leaf nodes is also the best case where maximum pruning can be achieved by α-β pruning.



Worst Case : If we reorder the leaves such that the leaf nodes which gives us our best path are at the end and the just before them then we minimize the pruning.



In the given arrangement the only branch that can be pruned is G. This arrangement will give value of A = 1.



Now, in the above arrangement no pruning is possible, we would have to explore every single possibility. Value of A = 2.

Thus, this is the worst case scenario.

Part C

With an average or constant branching factor b and a search depth of d, the maximum no. of leaf node positions evaluated is $O(b*b*...b) = O(b^d)$

If the move ordering is optimal i.e. the best moves are always searched first

$\Rightarrow$ No. of leaf node positions evaluated

$\approx O(b*1*b*1...*b)$ for odd depth and similarly for even depth. $= O(b^{(d/2)})$

⇒ The search can go twice as deep with same amount of computation.

⇒ $b^*1^*b^*1 \ldots ^*b$ can be explained by the fact that all the first player's moves must be studied to find the best one, but for each only the best second player's move is needed to refute all but first (and best) first player move ⇒ α-β ensures no other second player moves need to be considered.

Question 3:

(a) Implemented in code_2022408.py

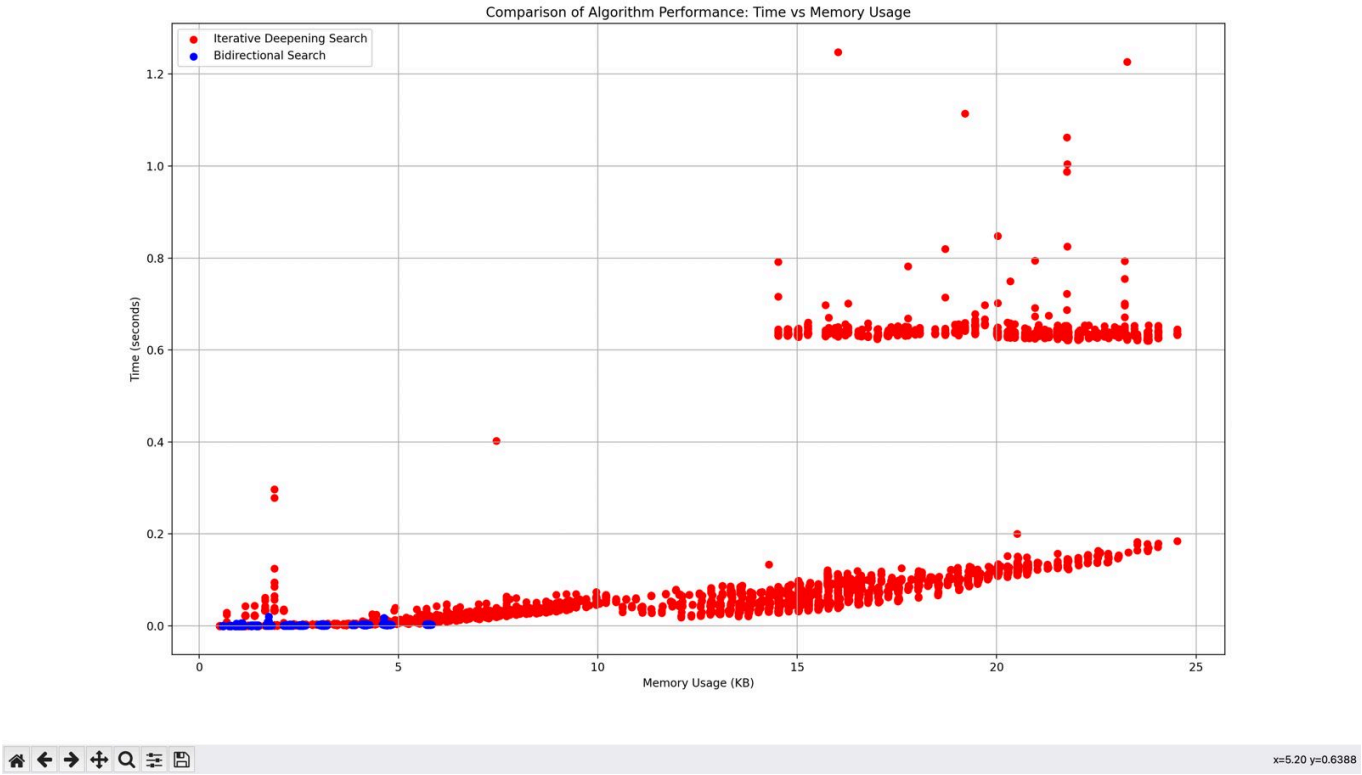
(b) We notice that the path obtained to travel from u to v for Iterative Deepening Search and Bidirectional Search is the same for all public test cases. However, it will not always be identical for all possible test cases. This is because during Iterative Deepening Search, the path is obtained by performing a depth-first search with a depth limit. The path obtained is the first path that is found by the algorithm. On the other hand, Bidirectional Search is a graph search algorithm that starts the search from both the source and the destination nodes. The algorithm stops when the two searches meet. The path obtained is the shortest path between the source and the destination nodes. The path obtained by the two algorithms will be the same when the first path found by Iterative Deepening Search is the shortest path between the source and the destination nodes. This will not always be the case, as the path obtained by Iterative Deepening Search may not be the shortest path between the source and the destination nodes. Hence, the path obtained by the two algorithms will not always be identical for all possible test cases.

(c)

```
--------------------------------------------------
Total time for all pairs:
Iterative Deepening Search: 1247.6375143527985 seconds
Bidirectional Search: 21.58349299430847 seconds
Total memory for all pairs:
Iterative Deepening Search: 122847.1513671875 KB
Bidirectional Search: 38515.9140625 KB
2024-09-17 04:13:52.799 Python[52491:3274110] +[IMKClient subclass]: chose IMKClient_Modern
2024-09-17 04:13:52.799 Python[52491:3274110] +[IMKInputSession subclass]: chose IMKInputSession_Modern
```

The above figure shows the total time and space taken by Iterative Deepening Search and Bidirectional Search for the public test cases. We notice that the time and space taken by Iterative Deepening Search is greater than the time and space taken by Bidirectional Search for all public test cases. This is because Iterative Deepening Search is a depth-first search algorithm that performs a depth-first search with a depth limit. The algorithm continues to increase the depth limit until the goal node is found. This results in the algorithm exploring a large number of nodes, which increases the time and space complexity of the algorithm. On the other hand, Bidirectional Search is a graph search algorithm that starts the search from both the source and the destination nodes. The algorithm stops when the two searches meet. This results in the algorithm exploring a smaller number of nodes, which decreases the time and space complexity of the algorithm. Hence, the time and space taken by Iterative Deepening Search is greater than the time and space taken by Bidirectional Search for all public test cases.

Below is a scatter plot showing the same:



(d) Implemented in code_2022408.py

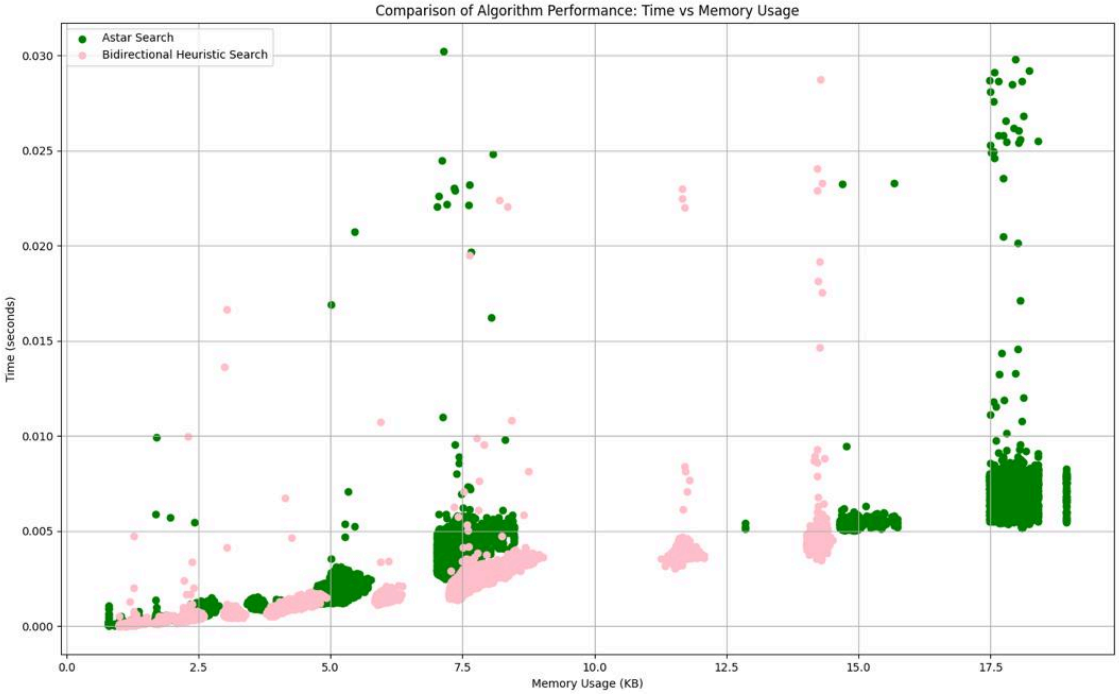
(e) We notice that the path obtained to travel from u to v is not identical for all public test cases for A star Search and Bidirectional Heuristic Search. This is because A star Search is a graph search algorithm that uses a heuristic function to estimate the cost of reaching the goal node from the current node. The algorithm uses this heuristic function to guide the search towards the goal node. On the other hand, Bidirectional Heuristic Search is a graph search algorithm that starts the search from both the source and the destination nodes. The algorithm stops when the two searches meet. The path obtained is the shortest path between the source and the destination nodes. The path obtained by the two algorithms will be the same when the heuristic function used by A star Search is admissible and consistent. This will not always be the case, as the heuristic function used by A star Search may not be admissible and consistent. Hence, the path obtained by the two algorithms will not always be identical for all possible test cases.

```
Total time for all pairs:
Astar Search: 62.095661640167236 seconds
Bidirectional Heuristic Search: 31.50393557548523 seconds
Total memory for all pairs:
Astar Search: 146223.791015625 KB
Bidirectional Heuristic Search: 99196.2265625 KB
2024-09-17 03:46:48.806 Python[47241:3255040] +[IMKClient subclass]: chose IMKClient_Modern
2024-09-17 03:46:48.806 Python[47241:3255040] +[IMKInputSession subclass]: chose IMKInputSession_Modern
2024-09-17 03:46:59.295 Python[47241:3255040] The class 'NSSavePanel' overrides the method identifier.  This method is implemented by class 'NSWindow'
~/Downloads/Assignment1 2
```

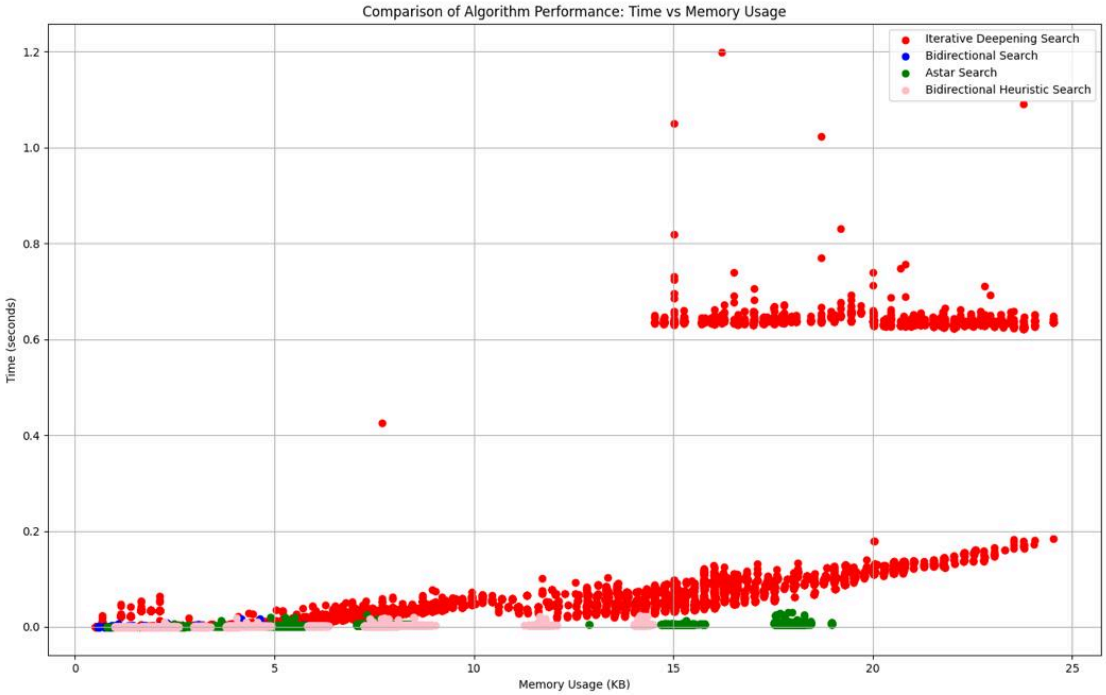
The above figure shows the total time and space taken by A star Search and Bidirectional Heuristic Search for the public test cases. We notice that the time and space taken by A star Search is greater than the time and space taken by Bidirectional Heuristic Search for all public test cases. This is because A star Search is a graph search algorithm that uses a heuristic function to estimate the cost of reaching the goal node from the current node. The algorithm uses this heuristic function to guide the search towards the goal node. The time and space complexity of the algorithm depends on the heuristic function used. On the other hand, Bidirectional Heuristic Search is a graph search algorithm that starts the search from both the source and the destination nodes. The algorithm stops when the two searches meet. This results in the algorithm exploring a smaller number of nodes, which decreases the time and space complexity of the algorithm. Hence, the time

and space taken by A star Search is greater than the time and space taken by Bidirectional Heuristic Search for all public test cases.

Below is a scatter plot showing the same:



(f) Below is a scatter plot comparing all the abpve discussed uninformed and informed search algorithms:



The total time and space taken by the above discussed uninformed and informed search algorithms are as follows:

The uninformed search algorithms are Iterative Deepening Search and Bidirectional Search. The informed search algorithms are A star Search and Bidirectional Heuristic Search.
IDS takes the largest amount of time making it least optimal out of all four algorithms. Bidirectional Heuristic and Bidirectional Search take approximately the same amount of time, followed by A star Search which is not greater by a large margin but only a small one.
A star Search takes the most amount of space with IDS not far behind making these te least efficient. Bidirectional Search takes the least amount of space making it the most efficient and Bidierectional Heuristic Search lies comfortably between IDS and Bidirectional Search.
The metric to obtain the scatter plot is the total amount of time and space taken by all four algirthms for each pair of nodes. The x-axis represents the total amount of space taken by the algorithms and the y-axis represents the total amount of time taken by the algorithms. The scatter plot shows the total time and space taken by the algorithms for each pair of nodes.

Benefits of using informed search algorithms over uninformed search algorithms:

1. Informed search algorithms use a heuristic function to estimate the cost of reaching the goal node from the current node. This heuristic function guides the search towards the goal node, which helps in reducing the time and space complexity of the algorithm.
2. Informed search algorithms are more optimal than uninformed search algorithms as they use additional information about the problem domain to guide the search towards the goal node.

Limitations of using informed search algorithms over uninformed search algorithms:

1. Informed search algorithms may not always find the optimal path between the source and the destination nodes. This is because the heuristic function used by the algorithm may not be admissible and consistent.
2. Informed search algorithms may take more space than uninformed search algorithms. This is because the algorithm needs to store additional information about the problem domain to guide the search towards the goal node.

(g) Implemented in code_2022408.py