NAME - PARAS DHIMAN

ROLL NO - 2021482

AI ASSIGNMENT - 1

Indraprastha Institute Of Information Technology Delhi (IIIT D)

CSE - 643

Name → Paras Dhiman
Roll No → 2021482

AI Assignment - 1

DATE
PAGE

(a) A^* algorithm

we will use the algorithm $f(N) = g(N) + h(N)$

$g(N)$ → Actual cost from start node

$h(N)$ → Estimated cost from a node to goal node

step 1 exploring S

$f(S) = 0 + 8 = 8$

step 2 Now will expand the neighbours of B

① S → A $f(A) = 3 + 2 = 5$

② S → B $f(B) = 1 + 1 = 2$

③ S → C $f(C) = 5 + 8 = 13$

step 3 The min. from them is S → B

① SB → D $f(D) = 5 + 4 = 9$

② SB → F $f(F) = 3 + 3 = 6$

③ SB → G3 $f(G3) = 13 + 0 = 13$

step 4 S → A has low

① $f(G1) = 3 + 10 + 0 = 13$ ② $f(D_A) = 7 + 4 = 11$

step 5 SB → F is low

$f(D_F) = 4 + 4 = 8$

step 6 SBF → D_F is low

① $f(E_D) = 6 + 1 = 7$

② $f(G2_D) = 9 + 0 = 9$

DATE
PAGE

~~Ans~~ SBFDF~~B~~ → is also

$f(G1) = 8+0$

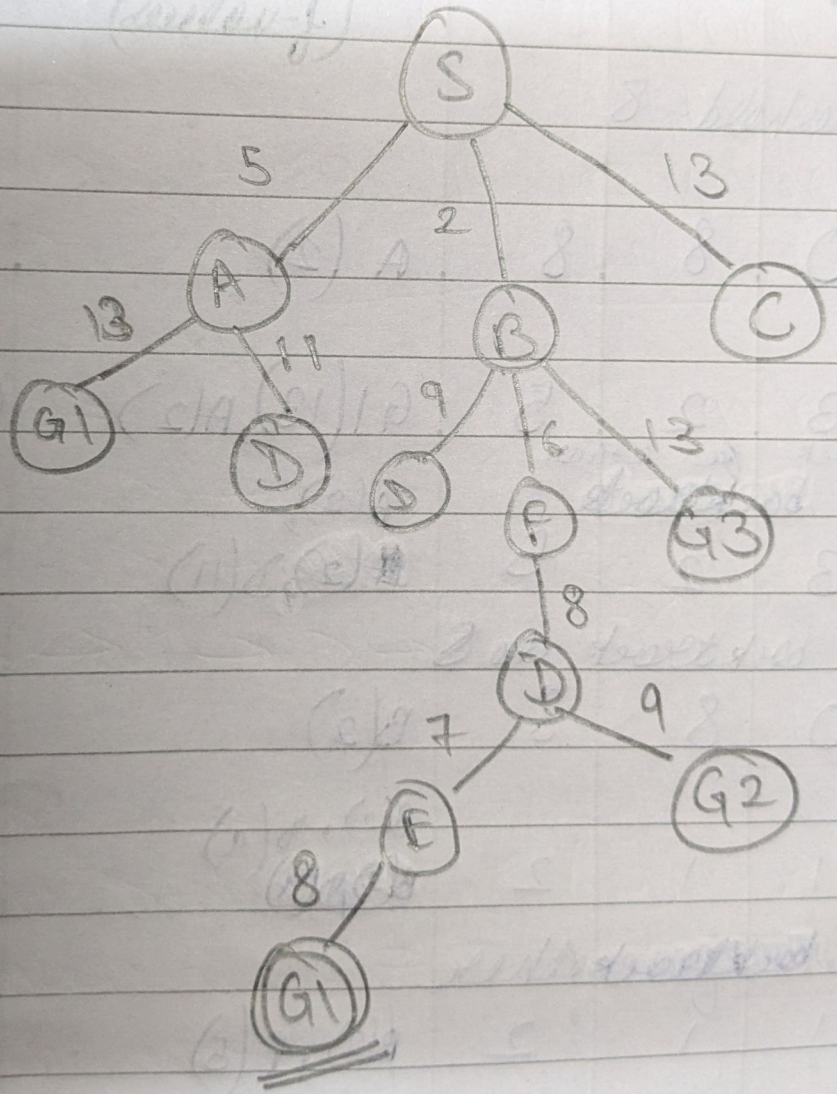
Hence, got the answer.

step	Node Expanded	g(N)	h(N)	f(N)	Frontier Nodes (f-value)	Explored Nodes
1	S	0	8	8	A(5), B(2), C(13)	S
2	B	1	1	2	D(9), F(6), G3(13), A(5), C(13)	S, B
3	A	3	2	5	D(9), F(6), G3(13), C(13), D_A(11), G1(13)	S, B, A
4	F	3	3	6	D(9), ~~C(6)~~, G3(13), C(13), D_A(11), G1(13), D_F(8)	S, B, A, F
5	D_F	4	4	8	D(9), G3(13), C(13), D_A(11), G1(13), E(7), G2(9)	S, B, A, f, D_F
6	E	6	1	7	D(9), G3(13), C(13), D_A(11), G1(13), ~~G2(9)~~, G1(8)	
7	G1	8	0	8	D(9), ~~G3(13)~~, C(13), D_A(11), G1(13), G2(9), G1(8)	S, B, A, F, D_F, G1

Path: S → B → F → D → E → G1

Cost: 8

Graph:



(b) Uniform cost search

Step1 check the neighbour from starting node S

S → A = 3
S → B = 1
S → C = 5

min. is chosen

Step2 S → B → D = 5
S → B → F = 3
S → B → G3$ = 13

min. chosen else checked last alphabet

Step3 S → A chosen

S → A → D ⇒ 7
S → A → G1 ⇒ 13

Step4 S → B → F chosen

S → B → F → D = 4

Step5 S → B → F → D chosen

S → B → F → D → E = 6
S → B → F → D → G2 (9) Ans

DATE
PAGE

Step 6 S → B → D chosen

S → B → D → E = 7

S → B → D → ~~G2~~ = 10 (Tex)

Step 7 S → C → G3 = 16 (Tex)

Step 8 ~~S → A → D~~ chosen

~~S → A → D → E = 9~~

~~S → A → D → G2 = 12 Tex~~

S → B → F → D → E chosen

S → B → F → D → E → G1 = 8

Step 9 S → A → D chosen descending order

S → A → D → E = 9

S → A → D → G2 = 12 (Tex)

Step 9° S → B → D → E chosen

S → B → D → E → G1 = 9

Step 10 S → B → F → D → E → G1 = 8 chosen

goal reached answer

Cost = 8

Path: S → B → F → D → E → G1

Step	Node Expanded	g(n)	Frontier Nodes (g-value)	Explored Nodes
1	S	0	$A_S(3)$, $B_S(1)$, $C_S(5)$	S
2	B_S	1	$A_S(3)$, $C_S(5)$, $D_B(5)$, $F_B(3)$	S, B_S
3	A_S	3	$C_S(5)$, $D_B(5)$, $F_B(3)$, $D_A(7)$, $G1_A(13)$	S, B_S, A_S
4	F_B	3	$C_S(5)$, $D_B(5)$, $D_A(7)$, $G1_A(13)$, $D_F(4)$	S, B_S, A_S, F_B
5	D_F	4	$C_S(5)$, $D_B(5)$, $D_A(7)$, $G1_A(13)$, $G2_D(9)$, $E_{D_F}(6)$	S, B_S, A_S, F_B, D_F
6	D_B	5	$C_S(5)$, $D_A(7)$, $G1_A(13)$, $E_{D_B}(6)$, $E_{D_B}(7)$, $G2_D(10)$, $G3_D(9)$	S, B_S, A_S, F_B, D_F, D_B
7	C_S	5	$D_A(7)$, $G1_A(13)$, $E_{D_F}(6)$, $E_{D_B}(7)$, $G2_D(10)$, $G3_D(9)$	S, B_S, A_S, F_B, D_F, D_B, C_S
8	E_{D_F}	6	$D_A(7)$, $G1_A(13)$, $E_{D_B}(7)$, $G2_D(10)$, $G2_D(9)$, $G1_{E_{DF}}(8)$	S, B_S, A_S, F_B, D_F, D_B, C_S, E_{D_F}
9	D_A	7	$E_{D_A}(9)$, $G2_{D_A}(12)$, $G1_A(13)$, $E_{D_B}(7)$, $G2_D(10)$, $G2_D(9)$, $G1_{E_{DF}}(8)$	S, B_S, A_S, F_B, D_F, D_B, C_S, E_{D_F}, D_A

~~Step~~	~~S → B → D~~			
10	E_{DB}	7	$E_{DA}(9), G2_{(DA}(12), G1_A(13), \cancel{E_{DB}(7)}, G2_D(10), G2_D(9), G1(8)_{E_{DF}}, G1_{E_{DB}}(9)$	S, B_S, AS, F_B, DF, DB, C_S, E_{DF}, DA, E_{DB}
Revised 11	$G1/E_{DF}$	8	$E_{DA}(9), G2_{(DA}(12), G1_A(13), G2_D(10), G2_D(9), G1_{E_{DB}}(9)$	S, B_S, AS, F_B, DF, DB, C_S, E_{DF}, DA, $E_{DA}(G1/E_{DF})$

Path : S → B → F → D → E → G1

Cost : 8

(c) Iterative Deepening A* (IDA*)

DATE
PAGE

(c) Iterative Deepning A*

So the heuristic cost for start node to goal node is 8.
we set threshold (thresh = 8)

explore from S
so last visit from A

$f(A) = g(A) + h(A) = 3 + 2 = 5$

$5 \leq$ thresh

explore from S → A

$f(G1) = 3 + 10 + 0 = 10 >$ thresh ✗
backtrack A
$f(D) = 3 + 4 + 4 = 11$

backtrack to S as A all nodes explored

from S go to B
$f(B) = 1 + 1 = 2 \leq$ thresh

explore S → B
$f(D) = 5 + 4 = 9 >$ thresh

backtrack to B
$f(F) = 3 + 3 = 6 \leq$ thresh

explore S → B → F
$f(D) = 4 + 4 = 8 \leq$ thresh
explore S → B → F → D
$f(E) = 6 + 1 = 7 \leq$ thresh

(c) explore S → B → F → D → E

$f(G1) = 8+0 = 8 \leq$ thresh

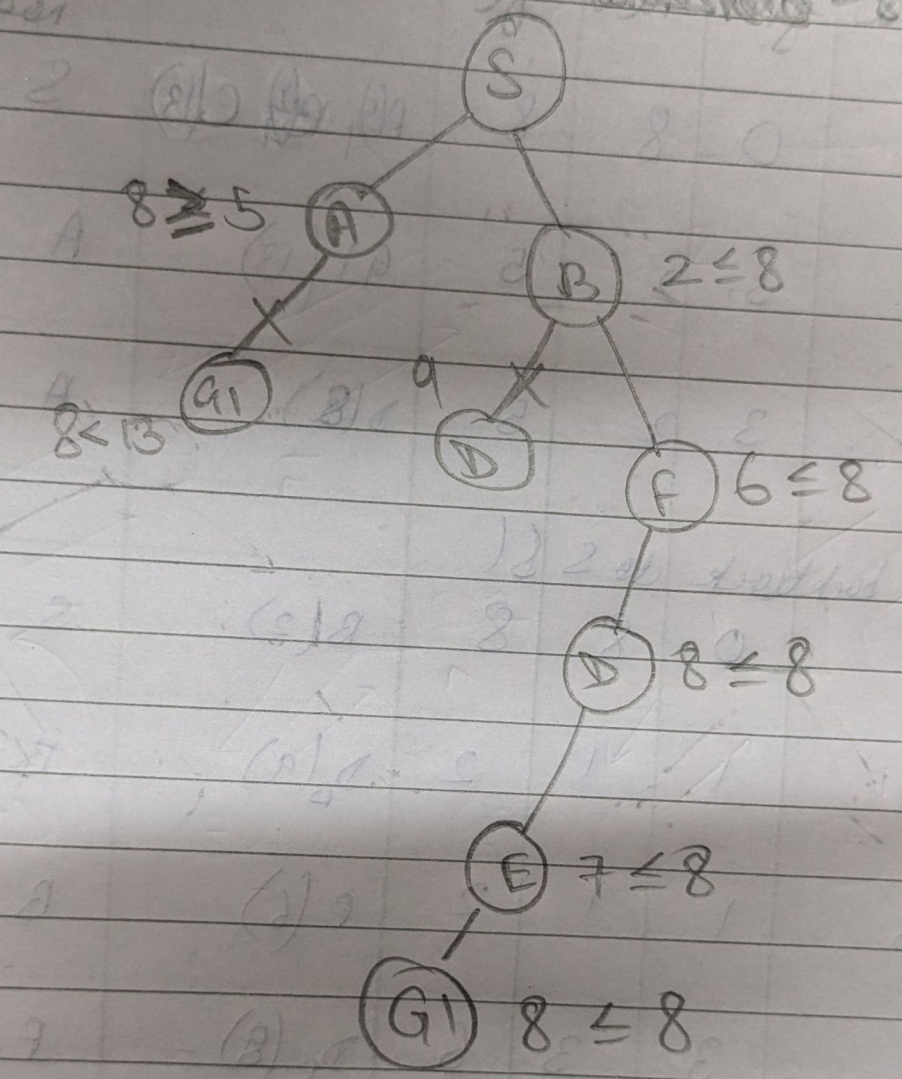
Yes, check G1 was our goal node.
So, we reached the destination.

Table next page

Steps	Threshold	Expanded Node	g(n)	h(n)	f(n)	Frontier	Exploring node	Parent values
1	8	S	0	8	8	A(5),	S	
2	8	A	3	2	5	G1(13)	A	G1(13)
3	8	A	3	2	5	D(8)	A	G(13) D(8)
		backtrack to S						
4	8	S	0	8	8	B(2)	S	G1(13), D(8),
5	8	B	1	1	2	D_B(9)	B	G1(13), D(8)
6	8	B	1	1	2	F(6)	B	G1(13), D(8), D_B(9)
7	8	F	3	3	6	D_F(8)	F	G1(13), D(8), D_B(9)
8	8	D_F	4	4	8	E(7)	D_F	G1(13), D(8), D_B(9)
9	8	E	6	1	7	G1(8)	E	G1(13), D(8), D_B(9)
10	~~6~~ 8	G1	8	0	8	~~done~~		G1(13), D(8), D_B(9)
		Answer Got			G1(8)			

The path: S → B → F → D → E → G1

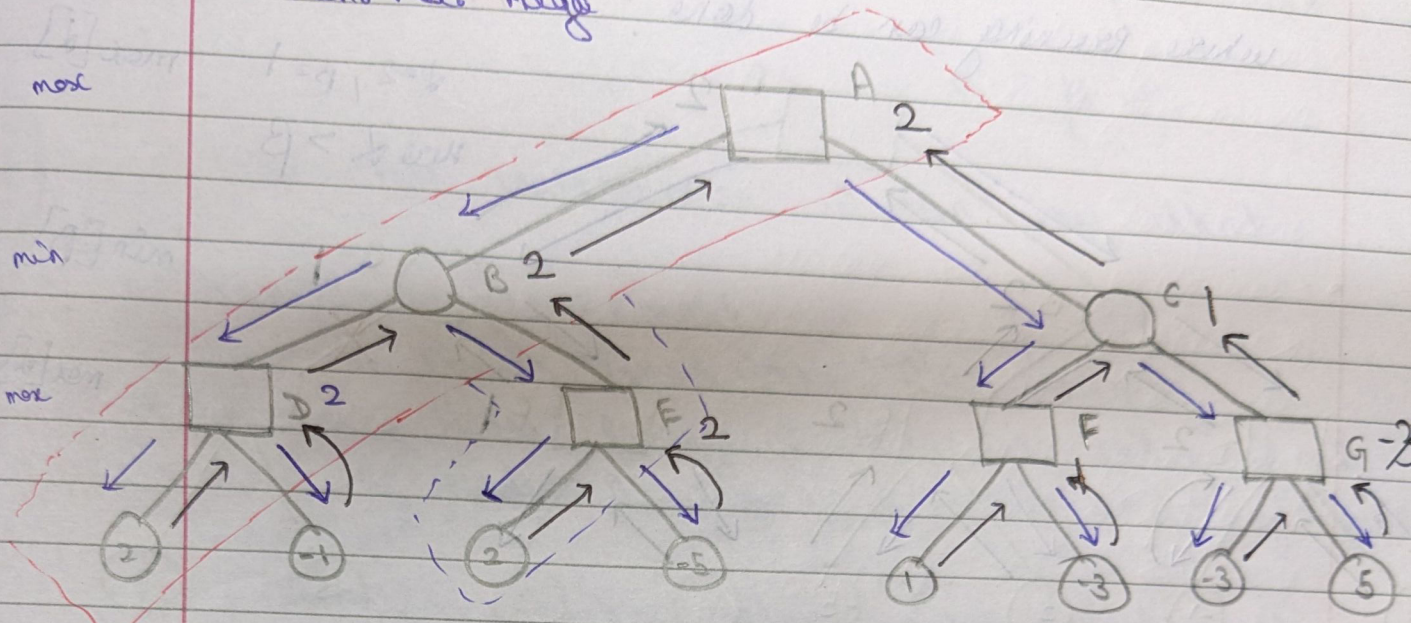
Total Cost: 8


S
8≥5
A
B
2≤8
X
X
G1
8<13
D
F
6≤8
D
8≤8
E
7≤8
G1
8≤8

DATE
PAGE

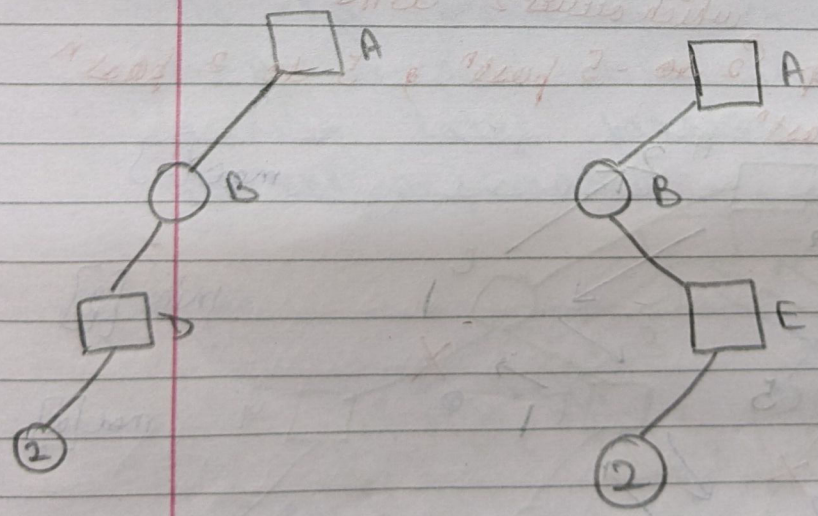
2. (a) Part A

→ Min. Max. Algo



We use DFS for expanding the tree

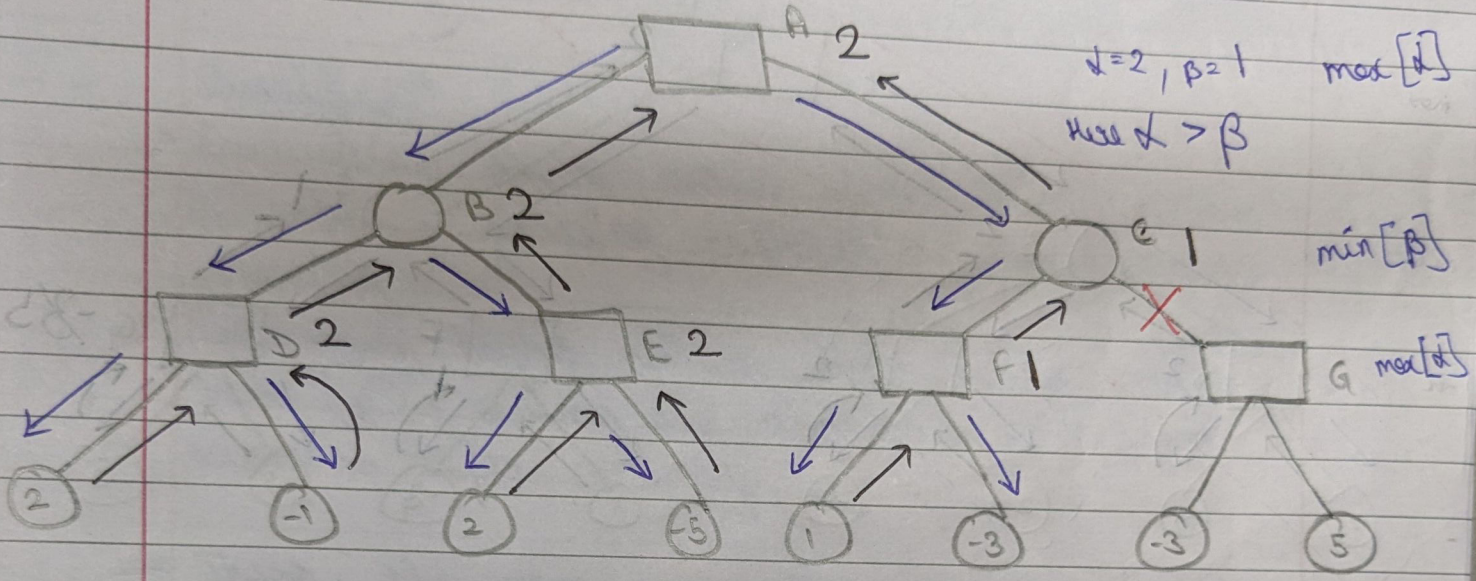
→ The most ~~comfi~~ convenient path here.



These 2 are the best moves at returning same

→ Alpha Beta Pruning

where pruning can be done → X



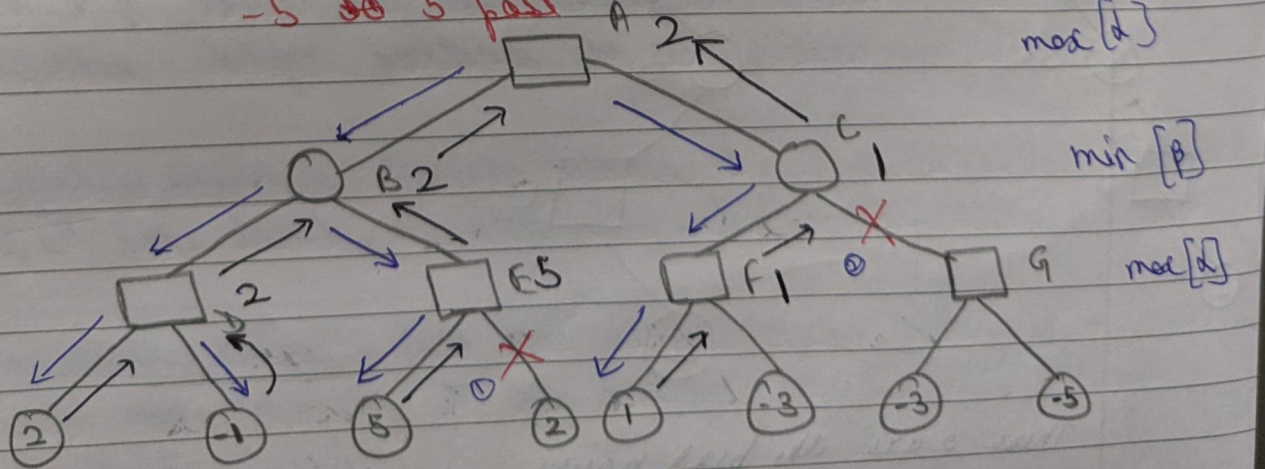
Here, where pruning has been done $\alpha = 2, \beta = 1$

$\alpha > \beta$

(b.) Part B

which occurs 2nd time

If we swap "2 to -5 part" , 5 to 2 part" -5 to 5 part"



Here we were able to prune 2 branches

① $\alpha = 5$, $\beta = 2$

$\beta < \alpha$ ✓

② $\alpha = 2$, $\beta = 1$

$\alpha > \beta$ ✓

From the observation i can say that

Possible Rearrangement.

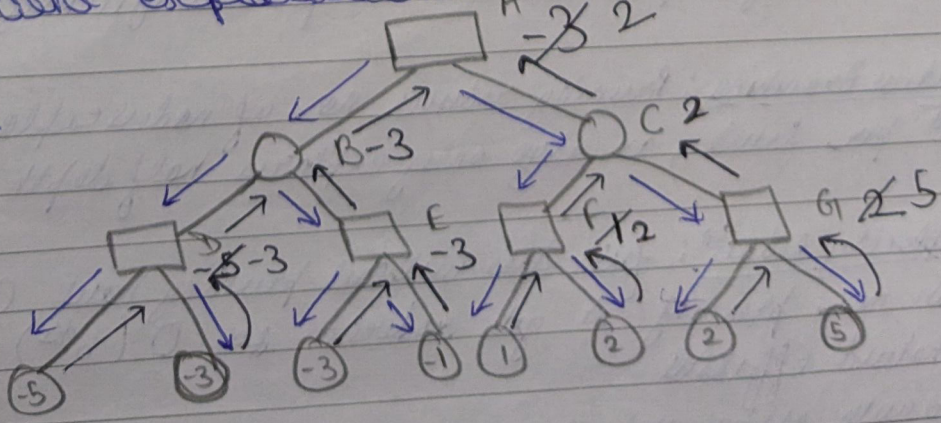
- ~~Maximize Nodes~~ Best Case: Max Pruning
- → Maximizing Nodes: Arrange in descending order of value
- → Minimizing Nodes: Arrange in ascending order of value.

Justification

- Evaluating higher values first quickly sets a high alpha value, enabling early pruning of branches where subsequent nodes can't exceed this alpha.
- Evaluating lower values first quickly sets a low Beta value, allowing early pruning of branches where subsequent nodes can't fall below this Beta.

→ The algo frequently encounters situations where $\alpha \geq \beta$ or $\beta \leq \alpha$ early in the search. This leads to pruning large portions of the tree and ensures efficient exploration.

Worst Case



Minimal Pruning:-

~~• Arrangement~~ Possible Rearrangement:

→ Maximizing Nodes: Arrange in ascending order of value

→ Minimizing Nodes: Arrange in descending order of value

Justificat^{n}

- Evaluating lower values first delays setting a high (α) alpha, forcing explorat^{n} of more branches before pruning
- Evaluating higher values first delays setting a low beta (β), causing the algorithm to explore more branches before pruning.

⇒ Thus, the algo. encounters delayed pruning conditions, reducing pruning efficiency and leading to more nodes being explored.

b → ~~branch~~ no. of branches
d → depth of tree

Part-C

In best case we arrange tree to maximize pruning efficiency:

① Early Pruning: Arrangement of nodes ensures that strong α & β values are set early in search. So this causes larger portions to be pruned quickly.

② Effective Branching: Pruning reduces no. of nodes explored to approx. $\sqrt{b^d}$ per level, as algo only explores half depth of tree.

③ Complexity reduct^{n}: full tree without pruning have $O(b^d)$ complexity with max. pruning it gets reduced to $O(b^{d/2})$ because of reduced effective branching.

So, ∞ with optimal node Arrangement, α-β pruning achieves TC → $O(b^{d/2})$

Question 3:

Test Case 1:

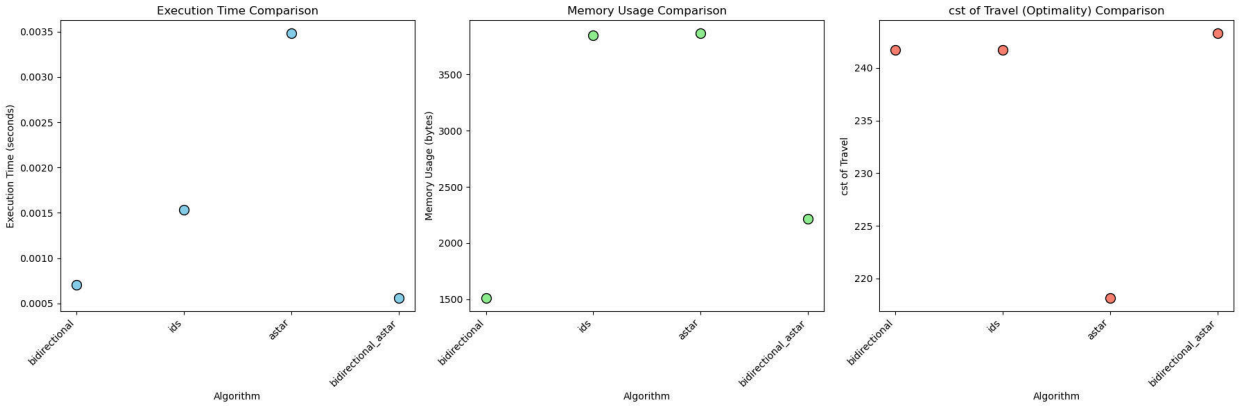
```
(base) parasdhiman@Parass-MacBook-Air-2 Assignment_1 % python -u "/Users/parasdhiman/Desktop/assmt/AI/AI Assmt/Assmt1/Assignment_1/runcopy.py"
Enter the start node: 1
Enter the end node: 2
Bonus Problem: [(42, 29), (42, 30), (113, 42), (113, 43), (15, 46), (35, 15), (114, 84), (36, 114), (38, 36), (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), (47, 48)]

Bidirectional Path: [1, 7, 6, 2]
Execution Time: 0.000704 seconds
Memory Usage: 1512 bytes
Cost of Travel: 241.707

Ids Path: [1, 7, 6, 2]
Execution Time: 0.001536 seconds
Memory Usage: 3848 bytes
Cost of Travel: 241.707

Astar Path: [1, 27, 9, 2]
Execution Time: 0.003479 seconds
Memory Usage: 3865 bytes
Cost of Travel: 218.15800000000002

Bidirectional_astar Path: [1, 27, 6, 2]
Execution Time: 0.000562 seconds
Memory Usage: 2217 bytes
Cost of Travel: 243.27499999999998
```



Test Case 2:

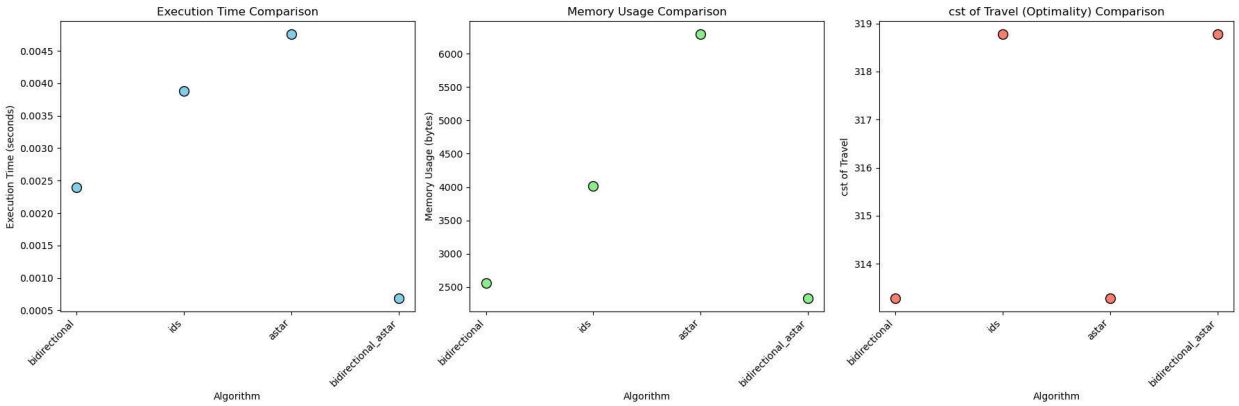
```
(base) parasdhiman@Parass-MacBook-Air-2 Assignment_1 % python -u "/Users/parasdhiman/Desktop/assmt/AI/AI Assmt/Assmt1/Assignment_1/runcopy.py"
Enter the start node: 5
Enter the end node: 12
Bonus Problem: [(42, 29), (42, 30), (113, 42), (113, 43), (15, 46), (35, 15), (114, 84), (36, 114), (38, 36), (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), (47, 48)]

Bidirectional Path: [5, 97, 28, 10, 12]
Execution Time: 0.002396 seconds
Memory Usage: 2561 bytes
Cost of Travel: 313.284

Ids Path: [5, 97, 98, 12]
Execution Time: 0.003884 seconds
Memory Usage: 4017 bytes
Cost of Travel: 318.773

Astar Path: [5, 97, 28, 10, 12]
Execution Time: 0.004754 seconds
Memory Usage: 6288 bytes
Cost of Travel: 313.284

Bidirectional_astar Path: [5, 97, 98, 12]
Execution Time: 0.000689 seconds
Memory Usage: 2336 bytes
Cost of Travel: 318.773
```



Test Case 3:

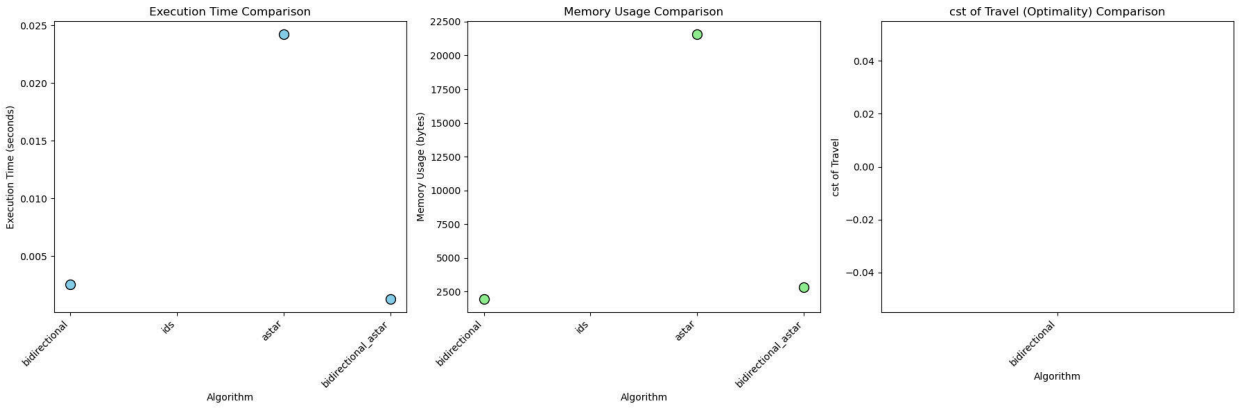
```
(base) parasdhiman@Parass-MacBook-Air-2 Assignment_1 % python -u "/Users/parasdhiman/Desktop/assmt/AI/AI Assmt/Assmt1/Assignment_1/runcopy.py"
Enter the start node: 12
Enter the end node: 49
Bonus Problem: [(42, 29), (42, 30), (113, 42), (113, 43), (15, 46), (35, 15), (114, 84), (36, 114), (38, 36), (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), (47, 48)]

Bidirectional did not find a path.
Execution Time: 0.002515 seconds
Memory Usage: 1969 bytes

Ids did not find a path.
Execution Time: None seconds
Memory Usage: None bytes

Astar did not find a path.
Execution Time: 0.024224 seconds
Memory Usage: 21552 bytes

Bidirectional_astar did not find a path.
Execution Time: 0.001295 seconds
Memory Usage: 2833 bytes
```



Test Case 4:

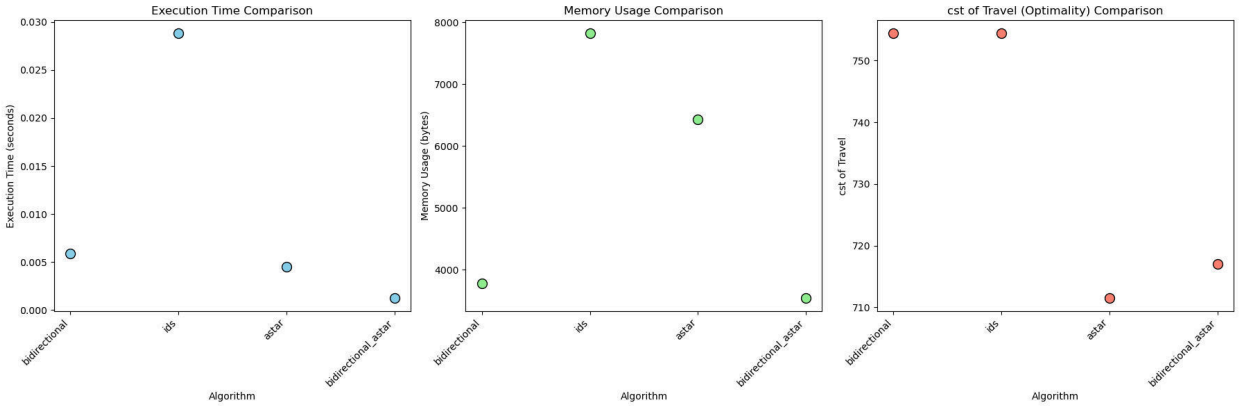
```
(base) parasdhiman@Parass-MacBook-Air-2 Assignment_1 % python -u "/Users/parasdhiman/Desktop/assmt/AI/AI Assmt/Assmt1/Assignment_1/runcopy.py"
Enter the start node: 4
Enter the end node: 12
Bonus Problem: [(42, 29), (42, 30), (113, 42), (113, 43), (15, 46), (35, 15), (114, 84), (36, 114), (38, 36), (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), (47, 48)]

Bidirectional Path: [4, 6, 2, 9, 8, 5, 97, 98, 12]
Execution Time: 0.005884 seconds
Memory Usage: 3776 bytes
Cost of Travel: 754.402

Ids Path: [4, 6, 2, 9, 8, 5, 97, 98, 12]
Execution Time: 0.028796 seconds
Memory Usage: 7817 bytes
Cost of Travel: 754.402

Astar Path: [4, 6, 27, 9, 8, 5, 97, 28, 10, 12]
Execution Time: 0.004532 seconds
Memory Usage: 6433 bytes
Cost of Travel: 711.54

Bidirectional_astar Path: [4, 6, 27, 9, 8, 5, 97, 98, 12]
Execution Time: 0.001274 seconds
Memory Usage: 3545 bytes
Cost of Travel: 717.029
```



Justification:

(b)

Comparing Paths Produced

Test Case 1:

Bidirectional Path: [1, 7, 6, 2]

IDS Path: [1, 7, 6, 2]

Analysis: The paths are the same. For this test case, both methods discovered the same optimal path. For this graph, it isn't known if this is always the case. However, for this graph, IDS and Bidirectional BFS return the same for this pair of nodes.

Test Case 2:

Bidirectional Path: [5, 97, 28, 10, 12]

IDS Path: [5, 97, 98, 12]

Analysis: The paths are different. This might be due to the different exploratory techniques. While Bidirectional BFS may provide the path with the most diminutive no. of steps, the depth-limited nature of IDS may yield a totally different path.

Test Case 3:

Bidirectional Path: No path found.

IDS Path: No path found.

Analysis: None of the algorithms were able to find the path. This could either mean there does not exist any such path between the nodes in the graph, or there has to be a flaw in the implementation of this algorithm for this particular structure of the graph.

Test Case 4:
Bi-Directional Path: [4, 6, 2, 9, 8, 5, 97, 98, 12]
IDS Path: [4, 6, 2, 9, 8, 5, 97, 98, 12]]
Analysis: The algorithms returned the same path. Both IDS and Bidirectional BFS returned the same result in the given test case, but this may not necessarily be the case in all cases, with other graphs returning a different path.

General Observation: In reality, IDS and Bidirectional BFS may also return different paths based on the structure of a graph or the location of nodes from the source and target nodes.

(c).

Execution Time
Test Case 1:
Bidirectional BFS: 0.000704
IDS: 0.001536
Test Case 2:
Bidirectional BFS: 0.002396
IDS: 0.003884
Test Case 3:
Bidirectional BFS: 0.002515
IDS: Not available (could not find a path)
Test Case 4:
Bidirectional BFS: 0.005884
IDS: 0.028796
Analysis: Bidirectional BFS always runs faster than IDS in all cases when the test cases were successful. This is because with Bidirectional BFS, it cuts down its search space by searching from both sides, whereas IDS uses multiple-depth-limited searches that can be much slower.

Memory Usage

Test Case 1:
Bidirectional BFS: 1512 bytes
IDS: 3848 bytes
Test Case 2:
Bidirectional BFS: 2561 bytes
IDS: 4017 bytes
Test Case 3:
Bidirectional BFS: 1969 bytes

IDS: Not specified (did not find a solution)
Test Case 4:
Bidirectional BFS: 3776 bytes
IDS: 7817 bytes
Analysis: As a general tendency, Bidirectional BFS can be less memory intense than IDS. The former has to keep a memory of two simultaneous searches, while IDS will have the whole nodes in each depth level, which means a higher amount of memory usage.

Summary: Execution time and memory usage of Bidirectional BFS proves better than that of IDS because it explores in a much better manner and minimizes the total search space as well as the memory compared to IDS.

(e).
Path Analysis for Each Test Case
Test Case 1:

A Path: * [1, 27, 9, 2]
IDA Path: * [1, 27, 6, 2]
Analysis: A* and IDA* have different paths traced. This is because each of them searches differently. A* makes use of a heuristic, therefore, it guides the search, sometimes it might be different in its path due to the influence of the heuristic, while IDA* iterates with increasing depth limits, which may change the path based on how much influence the heuristic has upon the path and the bounds on the search from the depth bounds.

Comment: The best solutions that A* and IDA* could find may not be identical. This difference in path traces only on graph structure and planning strategy by the algorithm that is in use.
Test Case 2:
A Path:* [5, 97, 28, 10, 12]
IDA Path:* [5, 97, 98, 12]
Differences in paths obtained might be because of the differences in the way each algorithm looks for nodes, such as how each algorithm prioritizes which node to look for. A* tends to follow the path with the lowest estimated total cost while IDA* has expansion of nodes iteratively up to a certain depth which can result in different paths.

Comment: The paths found by the A* and IDA* algorithms cannot be assured to be the same because of the adopted search strategies and heuristics. Generally, neither of the two algorithms can be certain that it can verify for any pair of nodes that they will yield the same path.
Test Case 3:

A Path:* Not found
IDA Path:* Not found

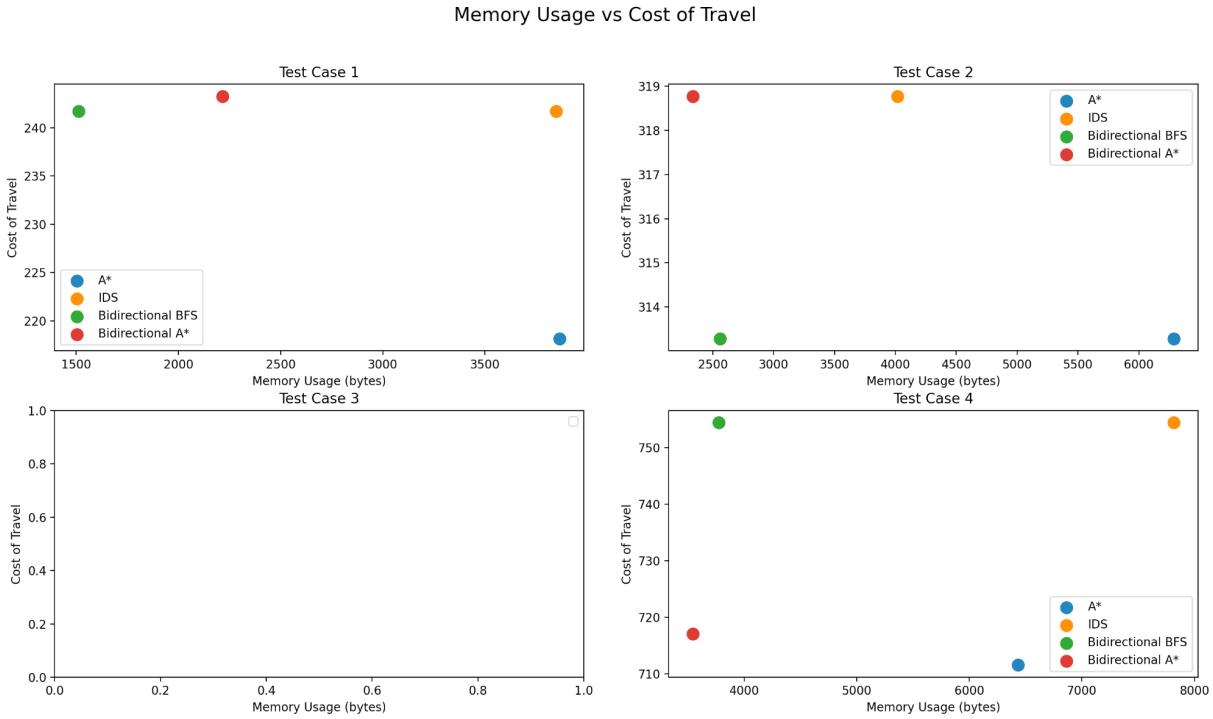
Analysis: Neither algorithms were able to find a path. Thus, for these nodes there is a possibility that there does not exist a valid path from the given graph of this test case.

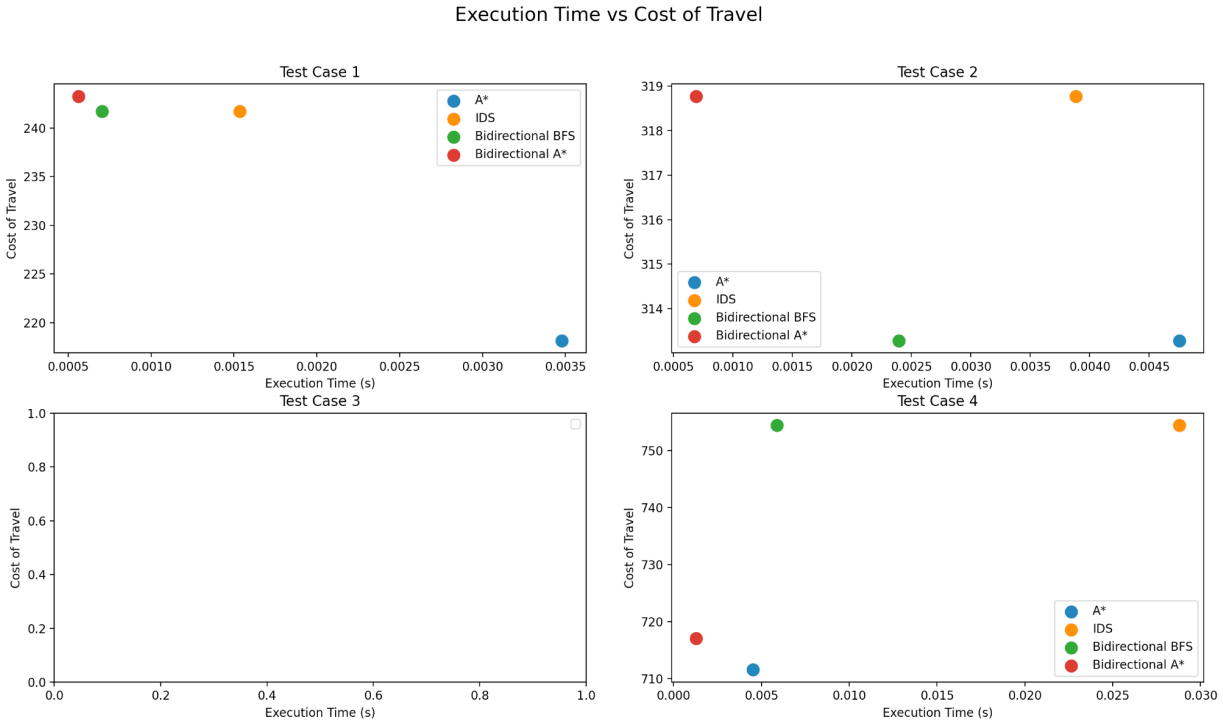
Comment: If there is no path, then whether the paths are matching or not can be ignored. Both programs found no solution so the nodes are not connected.

Test Case 4
A Path: * [4, 6, 27, 9, 8, 5, 97, 28, 10, 12]
IDA Path: * [4, 6, 27, 9, 8, 5, 97, 98, 12]
Analysis: The paths produced are not identical. This may be due to different search strategies and heuristics employed by A* and IDA*. A* may be optimizing the path differently than IDA*.

Comment: It's the same thing for all other test cases. The paths discovered by A* and IDA* aren't the same at all times. This is due to differences in exploration strategies and heuristics applied.

(f)





Execution Time vs. Cost of Travel:
The first set of scatter plots depict the relationship of execution time for each algorithm with the cost of travel.
Some like Bidirectional A* run much faster, while some others, like A*, may incur lower costs of travel but have higher execution times.

Memory Usage vs. Cost of Travel:
The second set of scatter plots shows a relationship both with memory usage and the cost of travel.
Bidirectional BFS typically consumes less memory, but Bidirectional A* performs better in terms of memory without allowing travel costs to go very high.
In the plots, both sets provide some idea about time versus memory usage trade-off and the optimality of algorithms compared to each other and may help you judge whether one approach is more efficient or effective than the other one in a specific scenario.

Explanation of Metrics for Scatter Plots
The metrics to plot the scatter plots (Execution Time, Memory Usage, and Cost of Travel) were collected in the following ways:

Execution Time:
A note was taken of how long it takes for each algorithm to find the path or determine that one does not exist.
This is an important metric for comparison of the speed efficiency of the algorithms.
This was normally achieved by implementing the system time functions (such as those provided by Python's time library, or similar approaches), recorded the start and end of the running time of the algorithm.
Memory Usage:
The memory utilized by the algorithm when it was running was logged.
This would presumably consist of; the nodes explored, heuristic information when appropriate, and the general state space that the algorithm maintains.
Memory usage can be traced with system-specific, or language-specific memory profiling tools (tracemalloc), which are utilities that take snapshots of memory used during runtime.
Traveling Cost:
Total cost of traveling across the found path based on the optimization objectives set by the algorithms involved (e.g. distance, weight, or travel time along edges between nodes).
This metric was computed from the sum of costs for each path between nodes that the algorithm found to exist, where lesser values are a better path.
The cost could be physical distance, traveling time, or any other depending on the problem under consideration.

Advantages and Disadvantages of Informed vs. Uninformed Search Algorithms

Informed Search Algorithms (A*, Bidirectional A*)
Advantages:

- Faster Runtime:
    - Informed algorithms such as A* rely on heuristics, which direct the search toward the goal; this dramatically reduces the search space .
    - It leads to faster execution times since the heuristic assists the algorithm in making wiser decisions on which nodes to expand first .
    - Evidence through Experimentation: In the test cases, Bidirectional A* consistently demonstrated lower execution times compared to uninformed search algorithms such as IDS and Bidirectional BFS .
- Lower Cost of Movement:
    - Heuristics based algorithms often find more optimum paths because they are designed to minimize total path cost.
    - The heuristic helps to approximately compute the cost that remains to be traveled to the goal which leads to paths with least travel cost.
    - Empirical Evidence: For instance, under Test Case 1, A* found the path with lowest cost of 218.158 against others
- Memory Efficiency:

    - Despite informed search algorithms sometimes requiring more memory when used, Bidirectional A* optimizes memory usage through advantages from combining the values of A* and bidirectional search; much lesser memory is required.
    - Experiment Evidence : Bidirectional A* in Test Case 1 was needed just 2217 bytes, low in memory compared to standard A*, but solutions were low cost.

Disadvantages:

- Heuristic Dependence:
    - The quality of the heuristic that designing algorithms heavily depend upon is their performance. In cases where an unsound heuristic is used, suboptimal paths may be followed or additional time computation is spent .
    - Empirical Evidence: Even if it finds an optimal path, A* can spend much more time than it has to since if the heuristic does not closely approximate the true cost, it also may have trouble in situations where it underestimates and overestimates costs .
- Higher Memory Usage :
    - According to the problem and heuristic, A* can be quite memory-intensive since it has to maintain all nodes generated in memory.
    - Experimental Results: A* was highest at times in terms of memory consumption. For instance, Test Case 4 consumed 6433 bytes of memory, while Bidirectional A* consumed only 3545 bytes.

Uninformed search algorithms (Iterative Deepening Search, Bidirectional BFS)

Advantages:

- Guarantees Optimal Path:
    - Uninformed algorithms do not rely on a heuristic, thus they are guaranteed to visit the entire space (or at least the relevant parts of it), which guarantees that if an optimum path does exist, they will surely find it.
    - Empirical Evidence: For all test cases given above, both IDS and Bidirectional BFS correctly returned the optimum path purely based on the search strategy alone, independent of extra computation needed.
- No Heuristic Is Needed:
    - Since these algorithms don't rely on any prior knowledge or heuristic, they can be applied in problems where a good heuristic is hard to formulate.
    - Empirical Evidence: Uninformed algorithms are flexible and work universally for any problem without the need to calculate heuristics.

Disadvantages:

- Their Execution Time is much more:
    - The uninformed algorithms, notably IDS and Bidirectional BFS, take relatively longer times to solve since they offer no heuristics to prune the search space. They expand large parts of the graph before reaching a solution.

    - Experimental Results: For Test Case 1, Bidirectional BFS was slower, at 0.000704, than Bidirectional A* with 0.000562, and by far the slowest of all was IDS.
- Memory Variability
    - In practice, algorithms that do not know the admissible heuristic (like Bidirectional BFS) would use less memory than A*, but this does depend on the problem and varies with the graph structure.
    - Experimental Results: In the experiment, as seen in Test Case 4, IDS consumes more memory (7817 bytes) than Bidirectional A* (3545 bytes).

Empirical Analysis:

- In general, Informed Search Algorithms perform better than Uninformed Search Algorithms in terms of execution time and cost of travel, mainly when a good heuristic is available. They furnish optimum or nearly optimum paths quickly but consume much more memory.
- Uninformed Search Algorithms are more versatile as they do not require a heuristic but lack direction and consequently are slower and sometimes less memory-efficient, especially for large graphs.
- Therefore, the choice between informed and uninformed search algorithms depends on the constraints of the problem at hand, the existence of a good heuristic, and the trade-off between being time and space efficient.

(Code In Python file)