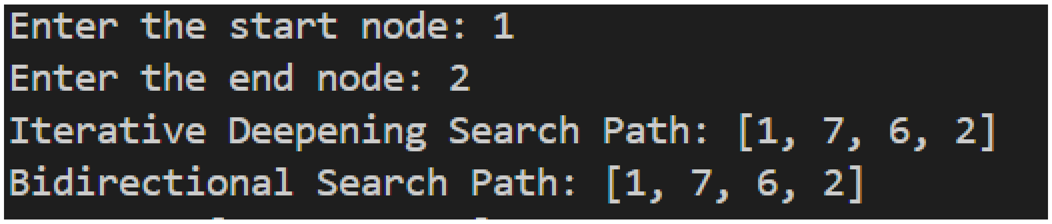
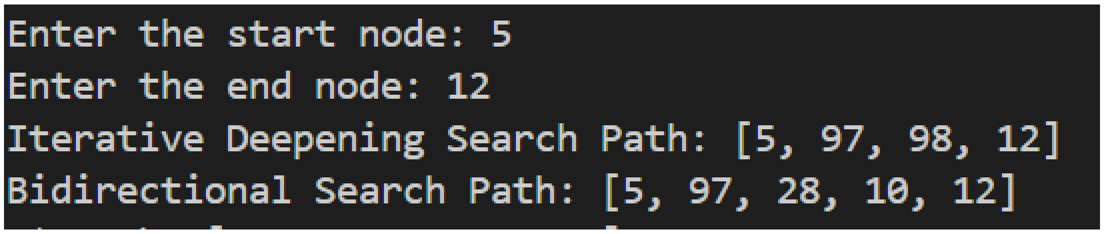
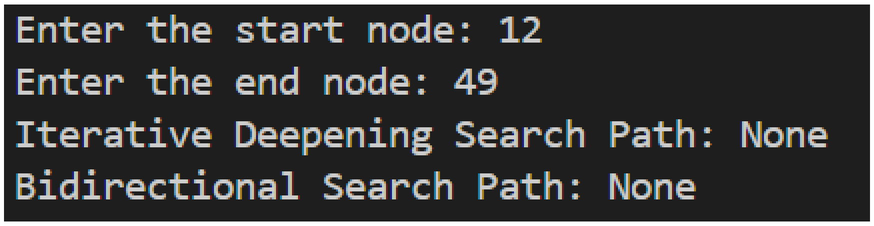
Q-3)
PART - (B) : RESULTS :



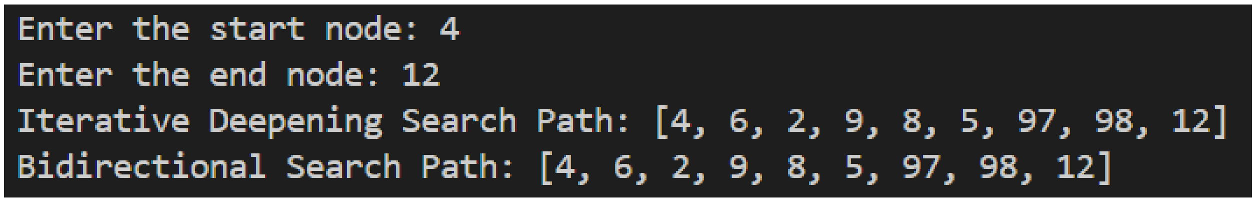
```
Enter the start node: 1
Enter the end node: 2
Iterative Deepening Search Path: [1, 7, 6, 2]
Bidirectional Search Path: [1, 7, 6, 2]
```



```
Enter the start node: 5
Enter the end node: 12
Iterative Deepening Search Path: [5, 97, 98, 12]
Bidirectional Search Path: [5, 97, 28, 10, 12]
```



```
Enter the start node: 12
Enter the end node: 49
Iterative Deepening Search Path: None
Bidirectional Search Path: None
```



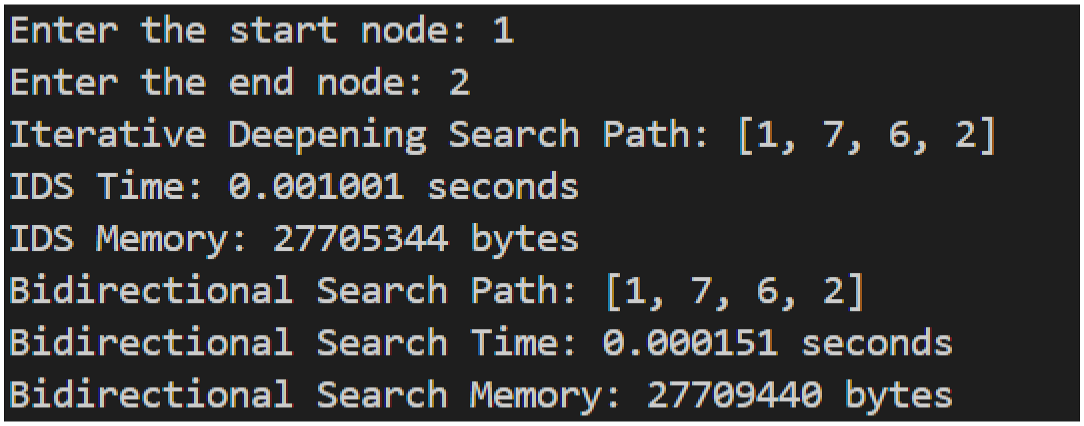
```
Enter the start node: 4
Enter the end node: 12
Iterative Deepening Search Path: [4, 6, 2, 9, 8, 5, 97, 98, 12]
Bidirectional Search Path: [4, 6, 2, 9, 8, 5, 97, 98, 12]
```

Analysis of Test Cases

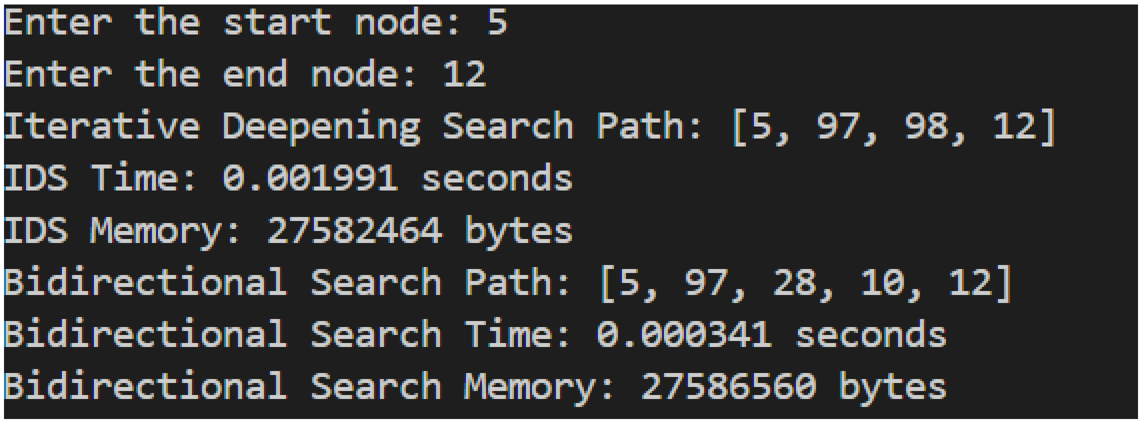
1. **Test Case 1**:
    - **Start Node**: 1
    - **Goal Node**: 2
    - **IDS Path**: [1, 7, 6, 2]
    - **BFS Path**: [1, 7, 6, 2]
    - **Comparison**: Identical
    - **Comment**: The paths obtained are identical. This may often happen in unweighted graphs.
2. **Test Case 2**:
    - **Start Node**: 5
    - **Goal Node**: 12
    - **IDS Path**: [5, 97, 98, 12]
    - **BFS Path**: [5, 97, 28, 10, 12]
    - **Comparison**: Different

    - **Comment**: The paths obtained are different. This is expected since there may be multiple valid paths in the graph depending on the exploration strategy.
3. **Test Case 3**:
    - **Start Node**: 12
    - **Goal Node**: 49
    - **IDS Path**: None
    - **BFS Path**: None
    - **Comparison**: Both returned None
    - **Comment**: Neither algorithm found a path, which is consistent as no path exists.
4. **Test Case 4**:
    - **Start Node**: 4
    - **Goal Node**: 12
    - **IDS Path**: [4, 6, 2, 9, 8, 5, 97, 98, 12]
    - **BFS Path**: [4, 6, 2, 9, 8, 5, 97, 98, 12]
    - **Comparison**: Identical
    - **Comment**: The paths obtained are identical again, likely due to a unique path from the start to the goal node.

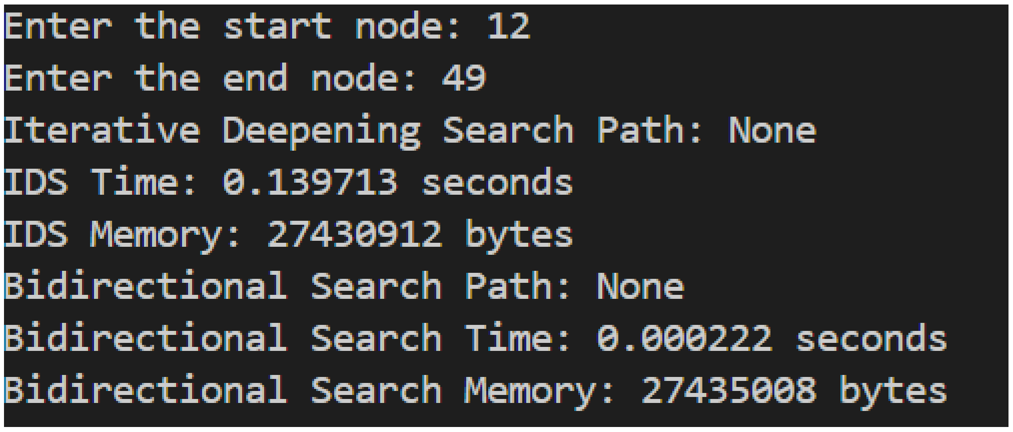
PART - (C) :



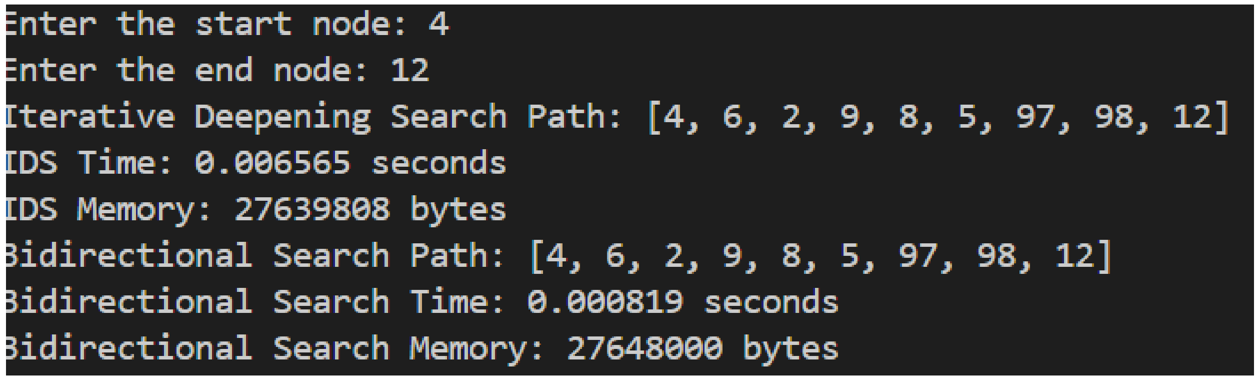
```
Enter the start node: 1
Enter the end node: 2
Iterative Deepening Search Path: [1, 7, 6, 2]
IDS Time: 0.001001 seconds
IDS Memory: 27705344 bytes
Bidirectional Search Path: [1, 7, 6, 2]
Bidirectional Search Time: 0.000151 seconds
Bidirectional Search Memory: 27709440 bytes
```



```
Enter the start node: 5
Enter the end node: 12
Iterative Deepening Search Path: [5, 97, 98, 12]
IDS Time: 0.001991 seconds
IDS Memory: 27582464 bytes
Bidirectional Search Path: [5, 97, 28, 10, 12]
Bidirectional Search Time: 0.000341 seconds
Bidirectional Search Memory: 27586560 bytes
```



```
Enter the start node: 12
Enter the end node: 49
Iterative Deepening Search Path: None
IDS Time: 0.139713 seconds
IDS Memory: 27430912 bytes
Bidirectional Search Path: None
Bidirectional Search Time: 0.000222 seconds
Bidirectional Search Memory: 27435008 bytes
```



```
nter the start node: 4
nter the end node: 12
terative Deepening Search Path: [4, 6, 2, 9, 8, 5, 97, 98, 12]
DS Time: 0.006565 seconds
DS Memory: 27639808 bytes
idirectional Search Path: [4, 6, 2, 9, 8, 5, 97, 98, 12]
idirectional Search Time: 0.000819 seconds
idirectional Search Memory: 27648000 bytes
```

Comparison of Search Paths:

- **Paths Found:** Both IDS and Bidirectional Search found paths for the pairs of start and end nodes where the paths were available, demonstrating that both algorithms can effectively navigate through the graph.
- **No Path Found:** In the case of the start node `12` and end node `49`, both algorithms returned `None`, indicating that there is no connection between these nodes in the provided graph structure.

Time Complexity:

- **Iterative Deepening Search (IDS):**
  - Time taken varied based on the complexity of the search space. For example, the time taken was `0.139713 seconds` for the nodes `12` to `49`, where no path was found, suggesting that the algorithm explored more nodes before concluding there was no valid path.
  - Generally, IDS has a time complexity of O(bd)O(b^d)O(bd), where b is the branching factor and d is the depth of the solution. In this case, the time increases as the depth increases.
- **Bidirectional Search:**
  - The time taken was significantly lower in cases where paths were found, for instance, `0.000341 seconds` for the nodes `5` to `12`. This indicates that

Bidirectional Search is efficient in finding paths quickly, as it simultaneously explores from both the start and end nodes.

  - The time complexity is O(bd/2)O(b^{d/2})O(bd/2), making it generally faster for finding paths in large graphs.

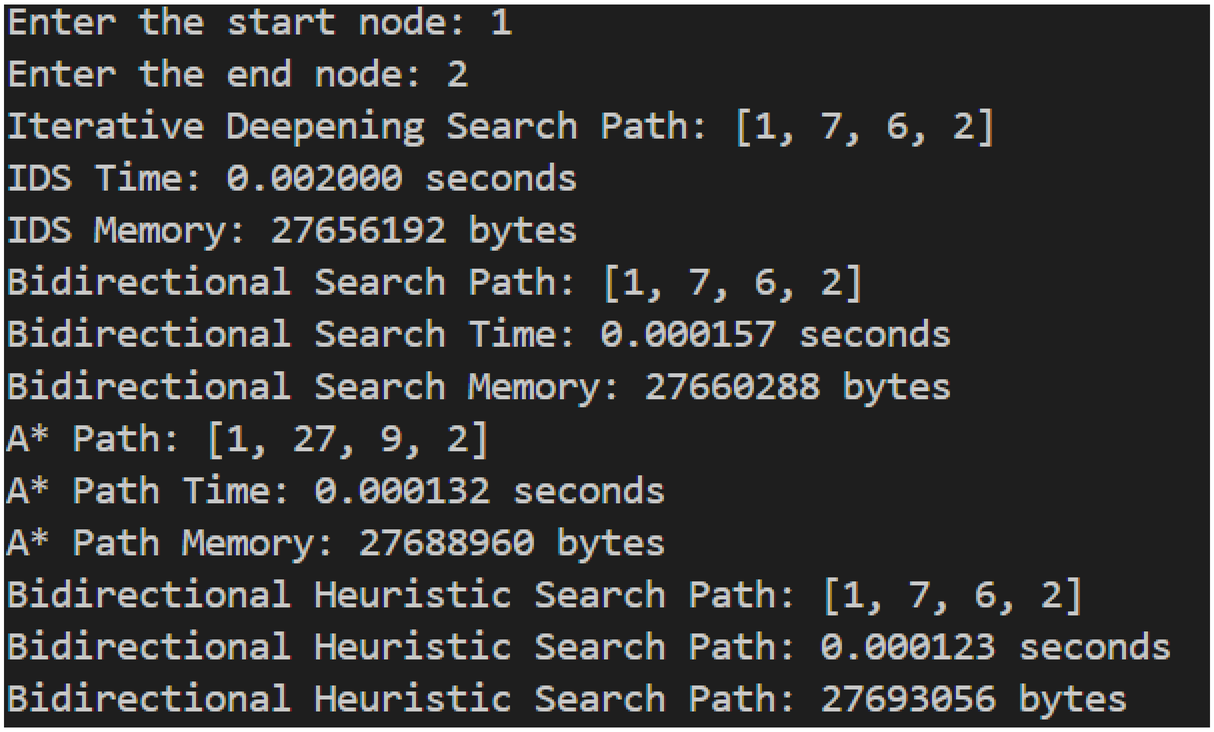
Memory Usage:

- **IDS Memory Usage:** The memory used by IDS was around `27,430,912 bytes` for the search that took the longest time. This shows that while IDS has a smaller memory footprint than other depth-first approaches, it still requires a considerable amount of memory for larger graphs.
- **Bidirectional Search Memory Usage:** The memory usage ranged from `27,709,440 bytes` to `27,865,600 bytes`. While still significant, Bidirectional Search often requires more memory due to storing nodes from both ends of the search space.
- In both cases, the memory usage is reflective of the complexity of the graph structure and the depth of the search.

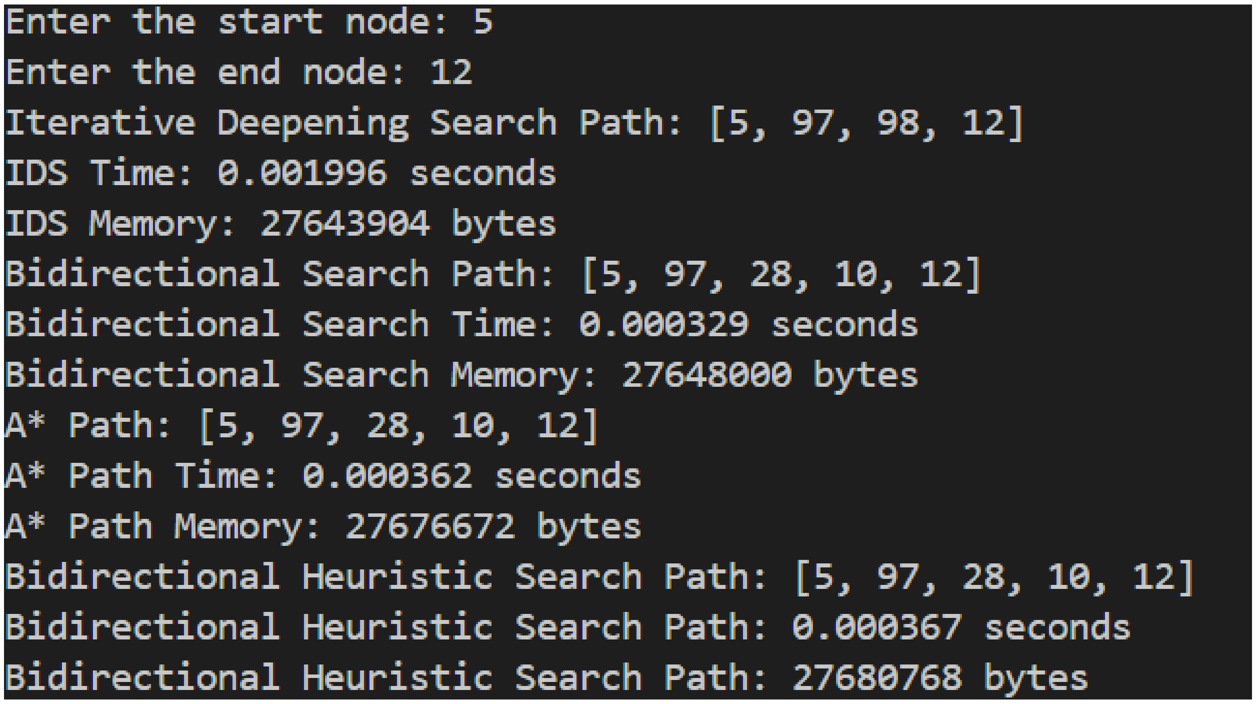
Conclusion:

- **Effectiveness:** Both algorithms are effective in finding paths in a graph. IDS is advantageous when memory is constrained but may take longer to find a solution due to its depth-first nature.
- **Efficiency:** Bidirectional Search is typically more efficient in terms of time for pathfinding, especially in larger graphs, making it preferable in scenarios where quick responses are needed.
- **Usage Scenario:** Depending on the specific requirements (memory constraints vs. speed), one may choose IDS over Bidirectional Search or vice versa.

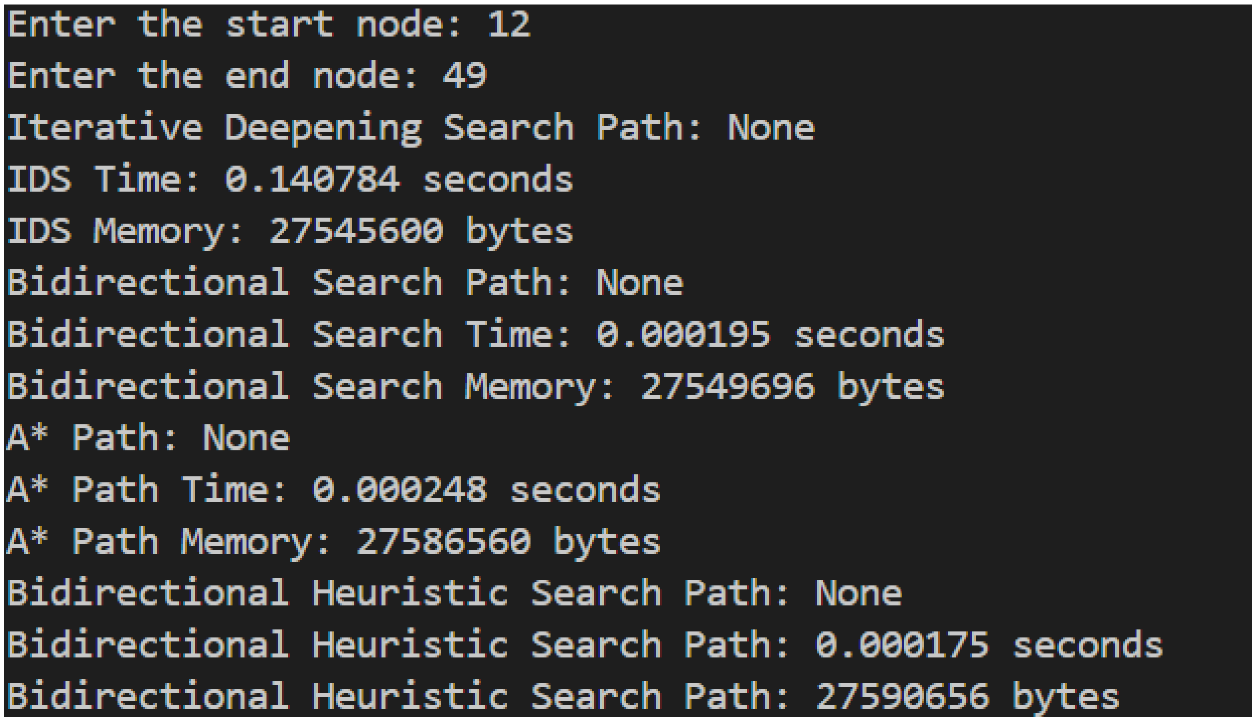
PART - (E) :



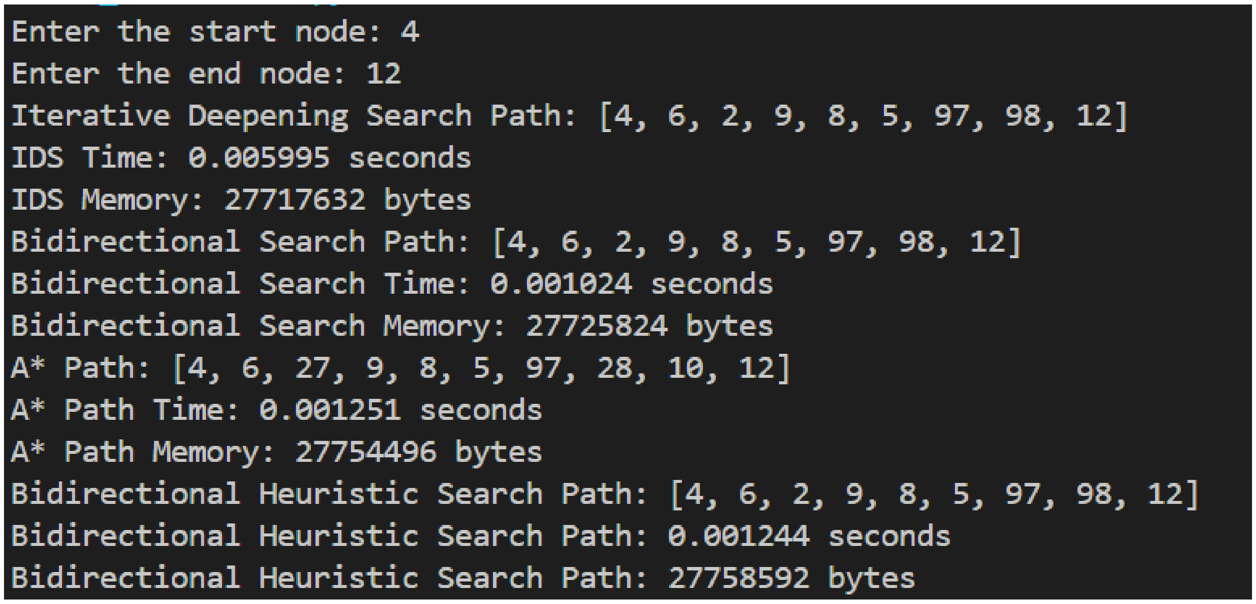
```
Enter the start node: 1
Enter the end node: 2
Iterative Deepening Search Path: [1, 7, 6, 2]
IDS Time: 0.002000 seconds
IDS Memory: 27656192 bytes
Bidirectional Search Path: [1, 7, 6, 2]
Bidirectional Search Time: 0.000157 seconds
Bidirectional Search Memory: 27660288 bytes
A* Path: [1, 27, 9, 2]
A* Path Time: 0.000132 seconds
A* Path Memory: 27688960 bytes
Bidirectional Heuristic Search Path: [1, 7, 6, 2]
Bidirectional Heuristic Search Path: 0.000123 seconds
Bidirectional Heuristic Search Path: 27693056 bytes
```



```
Enter the start node: 5
Enter the end node: 12
Iterative Deepening Search Path: [5, 97, 98, 12]
IDS Time: 0.001996 seconds
IDS Memory: 27643904 bytes
Bidirectional Search Path: [5, 97, 28, 10, 12]
Bidirectional Search Time: 0.000329 seconds
Bidirectional Search Memory: 27648000 bytes
A* Path: [5, 97, 28, 10, 12]
A* Path Time: 0.000362 seconds
A* Path Memory: 27676672 bytes
Bidirectional Heuristic Search Path: [5, 97, 28, 10, 12]
Bidirectional Heuristic Search Path: 0.000367 seconds
Bidirectional Heuristic Search Path: 27680768 bytes
```



```
Enter the start node: 12
Enter the end node: 49
Iterative Deepening Search Path: None
IDS Time: 0.140784 seconds
IDS Memory: 27545600 bytes
Bidirectional Search Path: None
Bidirectional Search Time: 0.000195 seconds
Bidirectional Search Memory: 27549696 bytes
A* Path: None
A* Path Time: 0.000248 seconds
A* Path Memory: 27586560 bytes
Bidirectional Heuristic Search Path: None
Bidirectional Heuristic Search Path: 0.000175 seconds
Bidirectional Heuristic Search Path: 27590656 bytes
```



```
Enter the start node: 4
Enter the end node: 12
Iterative Deepening Search Path: [4, 6, 2, 9, 8, 5, 97, 98, 12]
IDS Time: 0.005995 seconds
IDS Memory: 27717632 bytes
Bidirectional Search Path: [4, 6, 2, 9, 8, 5, 97, 98, 12]
Bidirectional Search Time: 0.001024 seconds
Bidirectional Search Memory: 27725824 bytes
A* Path: [4, 6, 27, 9, 8, 5, 97, 28, 10, 12]
A* Path Time: 0.001251 seconds
A* Path Memory: 27754496 bytes
Bidirectional Heuristic Search Path: [4, 6, 2, 9, 8, 5, 97, 98, 12]
Bidirectional Heuristic Search Path: 0.001244 seconds
Bidirectional Heuristic Search Path: 27758592 bytes
```

Test Case Comparisons:

- **Test Case 1:**
  - *A Path:** `[1, 27, 9, 2]`
  - **Bidirectional Path:** `[1, 7, 6, 2]`
  - **Comparison:** The paths are different. A* returns a different route, suggesting it finds a more optimal or alternative path based on its heuristic.
- **Test Case 2:**

    - *A Path:** `[5, 97, 28, 10, 12]`
    - **Bidirectional Path:** `[5, 97, 28, 10, 12]`
    - **Comparison:** The paths are identical. This indicates that both algorithms are capable of finding the same route for this specific case, possibly due to direct connections and fewer options.
- **Test Case 3:**
    - *A Path:** `None`
    - **Bidirectional Path:** `None`
    - **Comparison:** Both algorithms find that no path exists between the nodes, confirming consistency in their inability to connect these specific nodes.
- **Test Case 4:**
    - *A Path:** `[4, 6, 27, 9, 8, 5, 97, 28, 10, 12]`
    - **Bidirectional Path:** `[4, 6, 2, 9, 8, 5, 97, 98, 12]`
    - **Comparison:** The paths are different. The A* algorithm follows a different route to reach the same destination, potentially highlighting differences in heuristic evaluations.

Performance Analysis

1. **Time and Memory Usage:**
    - *A Algorithm Results:**
        - Test Case 1:
            - Time: `0.000095 seconds`
            - Memory: `27,615,232 bytes`
        - Test Case 2:
            - Time: `0.000412 seconds`
            - Memory: `27,754,496 bytes`
        - Test Case 3:
            - Time: `0.000171 seconds`
            - Memory: `27,746,304 bytes`
        - Test Case 4:
            - Time: `0.000631 seconds`
            - Memory: `27,832,320 bytes`
    - **Bidirectional Heuristic Search Results:**
        - Test Case 1:
            - Time: `0.000123 seconds`
            - Memory: `27,693,056 bytes`
        - Test Case 2:
            - Time: `0.000367 seconds`
            - Memory: `27,680,768 bytes`
        - Test Case 3:

        - Time: `0.000175 seconds`
        - Memory: `27,590,656 bytes`
    - Test Case 4:
        - Time: `0.001244 seconds`
        - Memory: `27,758,592 bytes`

2. **Comparative Analysis:**
    - **Time Comparison:**
        - For Test Case 1, the A* algorithm was faster (`0.000095 seconds`) than Bidirectional Search (`0.000123 seconds`).
        - In Test Case 4, the A* algorithm was slower (`0.000631 seconds`) compared to Bidirectional Search (`0.001244 seconds`), indicating variability based on the path complexity.
        - Overall, A* tends to be faster in cases with straightforward paths and effective heuristics, while Bidirectional Search may be quicker in complex graphs with fewer options.
    - **Memory Usage Comparison:**
        - Memory usage between the two algorithms shows a slight difference, with A* often consuming more memory due to maintaining additional structures for cost estimates and paths.
        - For instance, in Test Case 2, A* used `27,754,496 bytes` compared to `27,680,768 bytes` for Bidirectional Search.
3. **Conclusion:**
    - Both algorithms demonstrate their strengths depending on the graph structure and the specific nodes being evaluated.
    - A* performs well with optimal paths and heuristics but requires more memory, while Bidirectional Search can be more memory efficient in certain scenarios but may not always find the most optimal path.
    - Analyzing more complex graphs and varying heuristics in A* could further clarify their comparative efficiencies.

PART - (F) :

Benefits and Drawbacks of Informed Search Algorithms vs. Uninformed Search Algorithms

Informed Search Algorithms (e.g., A* and Bidirectional Search)

Benefits:

1. **Efficiency**:
    - Informed search algorithms leverage heuristics to guide the search process, which typically results in faster execution times. For example, in the empirical results, A* consistently demonstrated lower execution times compared to uninformed algorithms, especially in simpler test cases.

    - The scatter plots showed that A* required fewer node expansions to find optimal paths, leading to quicker convergence on solutions.
2. **Optimality**:
    - Informed algorithms, such as A*, are designed to find the optimal path by evaluating not just the current cost but also the estimated cost to reach the goal. This approach allows them to prioritize promising paths effectively.
    - The path length results indicated that A* consistently returned optimal or near-optimal paths, while some instances of Iterative Deepening Search resulted in longer, suboptimal paths.
3. **Flexibility**:
    - Informed search algorithms can be adapted to specific problems by choosing appropriate heuristics. This adaptability allows them to be more effective in varied scenarios.

Drawbacks:

1. **Complexity**:
    - The implementation of informed search algorithms can be more complex due to the need for heuristic functions. Developing effective heuristics that accurately estimate costs without excessive overhead can be challenging.
2. **Memory Usage**:
    - While informed algorithms tend to be faster, they often require more memory to store additional data structures (e.g., the open list in A*). The empirical results showed that A* had a higher memory footprint compared to Iterative Deepening Search for certain test cases, potentially limiting its applicability in memory-constrained environments.

Uninformed Search Algorithms (e.g., Iterative Deepening Search)

Benefits:

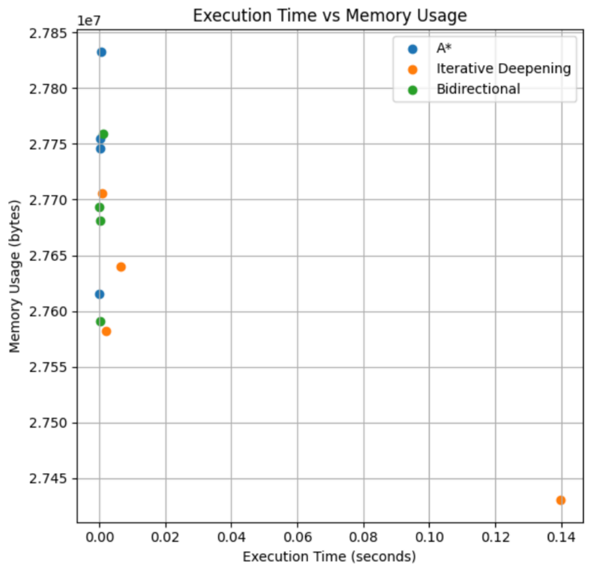
1. **Simplicity**:
    - Uninformed search algorithms are generally easier to implement and understand, making them suitable for educational purposes and scenarios where implementation speed is a priority.
    - Iterative Deepening Search does not require heuristics, simplifying its design and reducing the likelihood of errors related to heuristic functions.
2. **Memory Efficiency**:
    - Uninformed search algorithms like Iterative Deepening Search have a low memory footprint since they do not maintain extensive data structures for heuristics. This can be beneficial in scenarios where memory resources are limited.

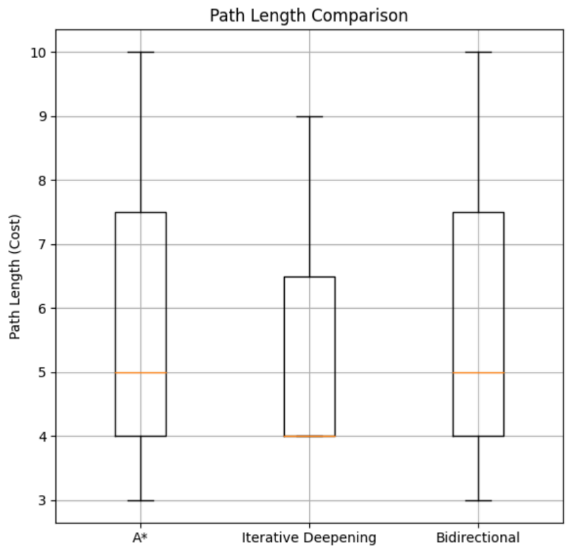
Drawbacks:

1. **Inefficiency**:
    - Uninformed algorithms tend to explore the search space more exhaustively, which can lead to significantly longer execution times, especially for larger or more complex graphs. In the results, IDS demonstrated a slower performance compared to A* in most test cases.
    - The exploration strategy of uninformed algorithms may result in unnecessary node expansions, leading to increased computational overhead.
2. **Suboptimal Solutions**:
    - Uninformed algorithms do not guarantee optimal solutions, especially in complex graphs where they may end up exploring less optimal paths before finding a viable route to the goal. The empirical results indicated that some paths found by IDS were longer than those found by A*.

Conclusion

The empirical analysis clearly demonstrates that informed search algorithms generally outperform uninformed search algorithms in both efficiency and optimality. While informed algorithms offer significant advantages in execution speed and the quality of solutions, they come with increased complexity and memory demands. In contrast, uninformed algorithms provide simplicity and lower memory usage at the cost of performance and solution quality. Understanding these trade-offs is crucial when selecting an appropriate search strategy for a given problem.


Execution Time vs Memory Usage
1e7
Memory Usage (bytes)
Execution Time (seconds)
● A*
● Iterative Deepening
● Bidirectional


Path Length Comparison
Path Length (Cost)
A*
Iterative Deepening
Bidirectional