

ai assignment 02

21F-9184,21F-9080



**Q2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Time complexity** | **Space complexity** | **Path Length** |
| Breadth-first search | O(b^d) | O(b^d) | Shortest |
| Depth-first search | O(b^m) | O(bm) | Largest |
| Uniform Cost Search | O(b^(1 + C\*/ε)) | O(b^(1 + \*/ε)) | Shortest |
| Greedy Best-First Search | O(b^m) | O(b^m) | Shortest |
| A\* Search | O(b^d) | O(b^d) | Shortest |
|  |  |  |  |

Where:

* *b* is the branching factor of the search tree.
* *d* is the depth of the goal node.
* *m* is the maximum depth of the search tree.
* *C\** is the cost of the optimal solution.
* *ε* is the smallest positive step cost.

ii. **Maximum Time Complexity**:

Depth-first search (\( O(b^m) \)) has the potential to take the maximum time complexity since it explores the deepest paths first, which might lead to traversing the entire tree.

iii. **Maximum Space Complexity**: Both Breadth-first search and Uniform Cost Search have the same worst-case space complexity \( O(b^d) \), where \( d \) is the depth of the goal node. Thus, they may take the maximum space complexity depending on the size of the search tree.

iv. **Longest Path**: Depth-first search may return the longest path since it explores one branch of the search tree as deeply as possible before backtracking.

v. **Shortest Path Length**: Breadth-first search, Uniform Cost Search, Greedy Best-First Search, and A\* Search aim to find the shortest path. Among them, Breadth-first search and A\* Search often return the shortest path length, as they systematically explore all possible paths.

vi. **Time Efficiency**: Breadth-first search and A\* Search are generally more time-efficient, as they ensure finding the optimal solution with the shortest path length.

vii. **Space Efficiency**: Greedy Best-First Search is more space-efficient than the other algorithms since it only needs to store nodes on the fringe, unlike Breadth-first search which stores all nodes at a given depth.

**Q4**

**Analysis of Informed Search Methods**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Time Complexity** | **Space Complexity** | **Path Length** |
| Greedy Best-First Search (h1) | O(b^m) | O(b^m) | Shortest |
| Greedy Best-First Search (h2) | O(b^m) | O(b^m) | Shortest |
| A\* Search (h1) | O(b^d) | O(b^d) | Shortest |
| A\* Search (h2) | O(b^d) | O(b^d) | Shortest |

Where:

* *b* is the branching factor of the search tree.
* *m* is the maximum depth of the search tree.
* *d* is the depth of the goal node.

ii. **Maximum Time Complexity**: A\* Search has the potential to take the maximum time complexity among the given algorithms because it combines both the heuristic function and the cost function to guide the search.

iii. **Maximum Space Complexity**: All algorithms have the same worst-case space complexity *O*(*bm*), where *m* is the maximum depth of the search tree. Thus, they may take the maximum space complexity depending on the size of the search tree.

iv. **Longest Path**: All algorithms aim to find the shortest path, so none of them would return the longest path.

v. **Shortest Path Length**: Both A\* Search with either heuristic function (h1 or h2) and Greedy Best-First Search with either heuristic function (h1 or h2) aim to find the shortest path length.

vi. **Time Efficiency**: Greedy Best-First Search may be more time-efficient than A\* Search since it only considers the heuristic function and doesn't involve the additional cost function.

vii. **Space Efficiency**: Both Greedy Best-First Search and A\* Search have the same space complexity, so neither algorithm is more space-efficient than the other.

viii. **Comparison of Heuristic Functions (h1 and h2)**:

* Both heuristic functions ℎ1*h*1 (Manhattan distance) and ℎ2*h*2 (Number of misplaced tiles) are admissible, meaning they never overestimate the cost to reach the goal.
* If both heuristic functions are admissible, it's preferable to choose the one that provides better guidance to the search algorithm. In the case of the eight puzzle problem, the Manhattan distance heuristic (ℎ1*h*1) usually provides better guidance as it considers the actual distance of tiles from their goal positions, giving a more accurate estimation of the remaining cost to reach the goal state.