200316059 İkram Celal KESKİN

200316011 M.Sina ERTUĞRUL

200316045 Mert KARDAŞ

NQueens Search Algorithm Report

A – Development Environment

We developed and tested the code in a Python 3.x environment. The operating system used as Windows 10. The code was written and executed in Visual Studio Code, a popular IDE for Python development. The CPU properties include an AMD Ryzen 3 3250U with Radeon Graphics processor.

B – Problem Formulation

The problem addressed in the code is the N-Queens problem, where N queens are placed on an N×N chessboard in such a way that no two queens threaten each other. The problem is formulated as a search problem using the SimpleAI library. Here are the key components:

- **State Specification**: The state is represented as a string of length N, where each character corresponds to the row position of the queen in the respective column.
- **Initial State**: The initial state is either manually entered by the user or generated randomly. The validity of the state is ensured through the _is_valid method.
- **Possible Actions**: The possible actions represent moving a queen to a different row within its column.
- **Transition Model**: The result method generates a new state by applying an action to the current state.
- **Goal Test**: The goal is reached when there are no attacking pairs of queens on the board.
- **Path Cost**: The cost of each step is constant, set to 1.

C- Testing

Please check results.txt, We only tested some of these because, program crashed cause of large iterations

D – Discussion

Completeness and Optimality

1. A (A-Star):*

Completeness: A* is complete if the heuristic function used is both admissible and consistent. If these conditions are met, A* is guaranteed to find a solution if one exists.

Optimality: A* is optimal given that the heuristic function is admissible, meaning it never overestimates the cost to reach the goal. In such cases, A* will find the shortest path to the goal.

2. Breadth-First Search (BFS):

Completeness: BFS is complete; it will always find a solution if one exists, provided that the search space is finite.

Optimality: BFS guarantees finding the shallowest goal, so it is optimal when all step costs are uniform.

3. Depth-First Search (DFS):

Completeness: DFS is not complete as it may get stuck in infinite loops or traverse indefinitely in infinite spaces.

Optimality: DFS is not optimal because it doesn't necessarily find the shortest path. It could reach a solution that is not the most optimal in terms of path length.

4. Limited Depth-First Search:

Completeness: Similar to DFS, limited depth-first search is not complete due to the same limitations of getting stuck in infinite loops or infinite spaces.

Optimality: Similar to DFS, it is not optimal as it might not find the shortest path.

5. Iterative Limited Depth-First Search:

Completeness: Iterative limited depth-first search is complete when the depth limit increases incrementally; however, if there's an infinite path, it won't terminate.

Optimality: It is not necessarily optimal as it can return a suboptimal solution depending on the iteration where the goal is found.

6. Uniform Cost Search:

Completeness: Uniform Cost Search is complete if the cost of each step is greater than some positive constant.

Optimality: Uniform Cost Search is optimal when all step costs are non-negative, as it guarantees finding the shortest path.

7. Greedy Best-First Search:

Completeness: Greedy Best-First Search is not guaranteed to be complete. It can get stuck in loops or go down paths that seem optimal but may not lead to a solution.

Optimality: It is not necessarily optimal as it makes decisions based solely on heuristic information, which may not always lead to the globally optimal solution.

Time and Space Complexity

1. *A (A-Star):**
    - **Time Complexity:** In the worst case, the time complexity of A* is exponential. However, it tends to perform well in practice due to the use of heuristic functions, often resulting in much better performance than brute-force search algorithms.
    - **Space Complexity:** The space complexity of A* is also exponential in the worst case, mainly due to storing and managing the priority queue or open/closed lists.
2. **Breadth-First Search (BFS):**
    - **Time Complexity:** For a graph with *b* branches and *d* depth, the time complexity of BFS is *O*(*b*^*d*) since it explores all nodes at a given depth before moving to the next level.
    - **Space Complexity:** BFS stores all nodes at a given level, so the space complexity is *O*(*b*^*d*) in the worst case due to storing all nodes at the maximum depth.
3. **Depth-First Search (DFS):**
    - **Time Complexity:** In the worst case, the time complexity of DFS is exponential, *O*(*b*^*m*), where *b* is the branching factor and *m* is the maximum depth.
    - **Space Complexity:** The space complexity of DFS is linear with respect to the maximum depth of the search tree, *O*(*bm*), due to the use of a stack or recursion to keep track of nodes.
4. **Limited Depth-First Search:**
    - **Time Complexity:** Similar to DFS, the time complexity of limited depth-first search can be exponential, *O*(*b*^*m*), depending on the branching factor and maximum depth.
    - **Space Complexity:** The space complexity is also linear with respect to the maximum depth *O*(*bm*), similar to DFS.
5. **Iterative Limited Depth-First Search:**

- **Time Complexity:** The time complexity is contingent on the depth limit and the number of iterations. In the worst case, it can be $O(b^m)$, similar to DFS.
- **Space Complexity:** The space complexity remains $O(bm)$, similar to DFS.

6. **Uniform Cost Search:**
    - **Time Complexity:** The time complexity of Uniform Cost Search can be exponential in the worst case if the cost is not bounded.
    - **Space Complexity:** The space complexity is exponential in the worst case due to storing all expanded nodes.
7. **Greedy Best-First Search:**
    - **Time Complexity:** The time complexity can vary widely based on the heuristic function and the quality of its estimation. In the worst case, it can be exponential.
    - **Space Complexity:** The space complexity is exponential due to the storage of all expanded nodes.

Conclusion

In conclusion, when evaluating search algorithms, considerations of completeness, optimality, time complexity, and space complexity are crucial. A* stands out for its optimality under admissible heuristic conditions, while BFS guarantees completeness but may suffer from higher space complexity. DFS and its variants are limited by completeness and optimality issues but might have lower space requirements. Uniform Cost Search offers optimality under certain cost conditions. Greedy Best-First Search relies heavily on heuristics and may sacrifice optimality for efficiency. Each algorithm has its strengths and weaknesses, making the choice dependent on specific problem constraints, trade-offs between time and space efficiency, and characteristics of the search space. Understanding these complexities aids in selecting the most suitable algorithm for a given problem.