BACK TRACKING METHOD

CSA0695

DESIGN AND ANALYSIS OF ALGORITHMS FOR OPEN ADDRESING TECHNIQUES

NAME:G.Siva Shankar Reddy

REG NO:192211486

SUPERVISOR:
Dr.R.Dhanalakshmi

PROBLEM STATEMENT:

- There is an 8 x 8 chessboard containing n pieces (rooks, queens, or bishops). You are given a string array pieces of length n, where pieces[i] describes the type (rook, queen, or bishop) of the ith piece. In addition, you are given a 2D integer array positions also of length n, where positions[i] = [ri, ci] indicates that the ith piece is currently at the 1-based coordinate (ri, ci) on the chessboard. When making a move for a piece, you choose a destination square that the piece will travel toward and stop on. A rook can only travel horizontally or vertically from (r, c) to the direction of (r+1, c), (r-1, c), (r, c+1), or (r, c-1).
- A queen can only travel horizontally, vertically, or diagonally from (r, c) to the direction of (r+1, c), (r-1, c), (r, c+1), (r, c-1), (r+1, c+1), (r+1, c-1), (r-1, c+1), (r1, c-1).
- A bishop can only travel diagonally from (r, c) to the direction of (r+1, c+1), (r+1, c-1), (r-1, c+1), (r-1, c-1). You must make a move for every piece on the board simultaneously. A move combination consists of all the moves performed on all the given pieces. Every second, each piece will instantaneously travel one square towards their destination if they are not already at it. All pieces start traveling at the 0th second. A move combination is invalid if, at a given time, two or more pieces occupy the same square. Return the number of valid move combinations. Notes:
- No two pieces will start in the same square. You may choose the square a piece is already on as its destination. If two pieces are directly adjacent to each other, it is valid for them to move past each other and swap positions in one second.
- Example 1: Input: pieces = ["rook"], positions = [[1,1]]
- Output: 15 Explanation: The image above shows the possible squares the piece can move to

ABSTRACT:

- This problem explores the movement of multiple chess pieces (rooks, queens, and bishops) on an 8x8 chessboard.
- We aim to determine how many valid move combinations exist such that no two pieces occupy the same square at the same time. The movement of each piece depends on its type and must follow chess rules.
- The solution evaluates all possible movement scenarios for the given pieces and ensures that the combinations where two or more pieces collide are excluded.

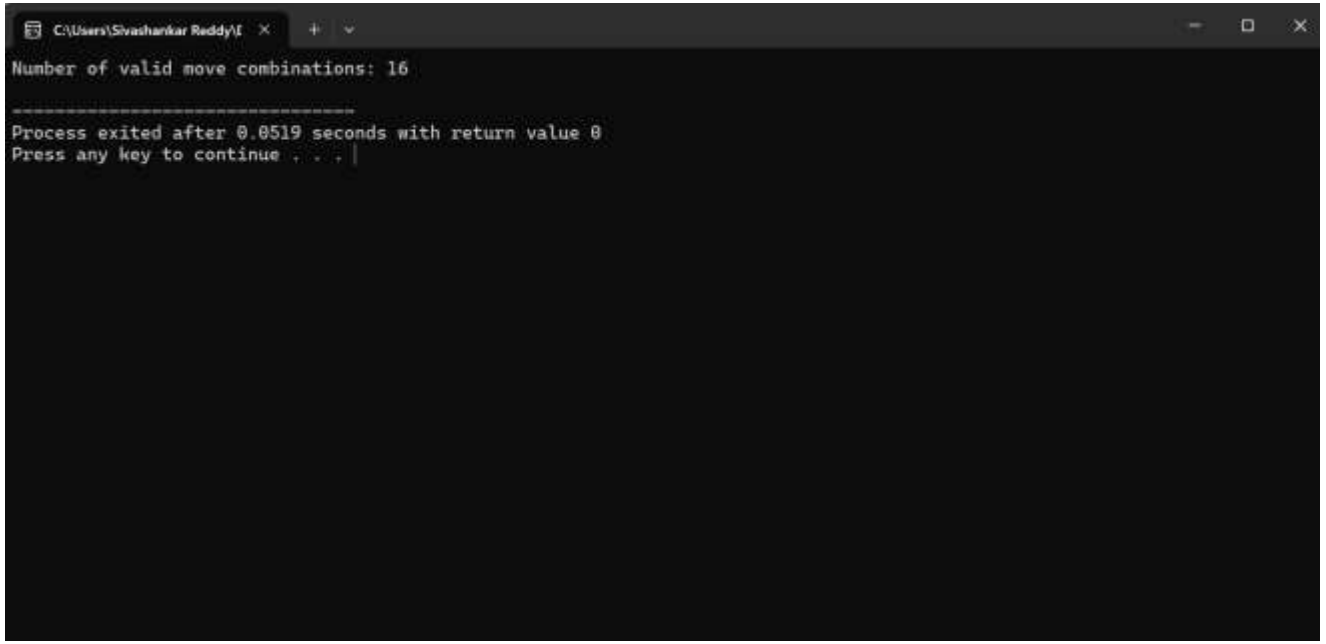
INTRODUCTION

- This problem explores the simultaneous movement of multiple pieces—rooks, queens, and bishops—on an 8x8 chessboard. Each piece can move according to its respective rules: a rook moves horizontally or vertically, a queen can move in any direction, and a bishop only moves diagonally.
- The goal is to determine how many valid move combinations exist where no two pieces occupy the same square at the same time.

KEY FEATURES:

- Key features of this problem include the constraints that tasks must fit within the session Time limit, tasks can be reordered, and each task must be completed within one session.
- The solution explores how to efficiently schedule tasks into these sessions, optimizing for the fewest number of sessions while ensuring no session exceeds the time limit. The input guarantees that no task is larger than the session Time.

OUTPUT


C:\Users\Sivashankar Reddy\E
Number of valid move combinations: 16

Process exited after 0.0519 seconds with return value 0
Press any key to continue . . .

COMPLEXITY ANALYSIS

- **Time Complexity :** The time complexity for each piece's move generation is O(k) where k is the number of valid squares it can move to. For each of the pieces, depending on whether it is a rook, bishop, or queen, the complexity varies slightly:
- Rook: O(2 * BOARD_SIZE) as it can move horizontally or vertically.
- Bishop: O(4 * BOARD_SIZE) as it can move diagonally.
- Queen: O(6 * BOARD_SIZE) as it combines rook and bishop moves. Hence, the time complexity for n pieces is O(n * BOARD_SIZE).

- **Space Complexity**:The space complexity is primarily O(BOARD_SIZE^2) due to the use of a chessboard array (8x8) to track the positions and movements of pieces.
- Additionally, if recursion or backtracking is used, the space complexity may increase depending on the depth of the recursion stack, which could add an extra O(n) where n is the number of pieces. Overall, the space complexity is typically O(BOARD_SIZE^2 + n).

CASES

BEST CASE

- In the best case scenario, each piece already has a destination that it can reach without colliding with any other pieces.
- This results in fewer checks for collisions and a faster simulation of moves.
- The best case occurs when there are very few pieces (n is small) and they have no conflicting movement paths. The time complexity in this case is approximately **O(n * BOARD_SIZE)**.

WORST CASE

- The worst case arises when there are many pieces (n is large) and their movement paths frequently intersect, requiring many collision checks and backtracking.
- In this scenario, the algorithm needs to explore multiple potential move combinations, leading to a time complexity approaching **O(n * BOARD_SIZE^n)**, where all possible moves must be checked for each piece.

AVERAGE CASE

- In the average case, there are a moderate number of pieces, and their movement paths sometimes overlap.
- The algorithm needs to explore a balanced number of move combinations and perform a moderate number of collision checks.
- The time complexity in this case lies between the best and worst cases, typically around **O(n * BOARD_SIZE^2)**.

FUTURE SCOPE

- The future scope for this problem is vast and can involve several interesting extensions. One possibility is to introduce additional chess pieces like knights or pawns, each with their own movement rules, increasing the complexity of collision detection.
- Another area of exploration could involve obstacles or blocked paths on the chessboard, further complicating piece movement.
- Expanding the board to larger dimensions or introducing a variable-sized chessboard would also create more challenging scenarios.
- Time-limited moves or step constraints could add a dynamic element, while multi-player or AI-driven strategies might make the problem more interactive and applicable to gaming or simulations..

CONCLUSION:

- In conclusion, solving the problem of determining valid move combinations for multiple chess pieces on an 8x8 board demonstrates an interesting application of chess mechanics in algorithm design.
- It requires careful consideration of each piece's unique movement abilities and the need to prevent collisions during simultaneous moves. Efficient algorithms, such as recursive backtracking, help explore possible move combinations and ensure that no two pieces end up on the same square.
- By exploring various move combinations and constraints, this problem provides insight into dynamic systems and collision detection. Overall, it offers a compelling challenge for optimizing chess-based movement strategies while adhering to game rules.