# □ Freedom Trail Problem Documentation

## 1. Problem Statement

Given a circular ring (ring) engraved with lowercase letters and a key, your task is to spell the key in order by rotating the ring. Each rotation (clockwise or counterclockwise by one position) costs 1 step, and pressing the center button to select a character also costs 1 step. You must determine the minimum number of steps required to spell the key.

Constraints:

- 1 <= ring.length, key.length <= 100
- ring and key contain only lowercase letters.
- Every character in key is guaranteed to appear in ring.

## 2. Intuition

The problem revolves around rotating a circular ring to align characters. Since characters may appear multiple times, we must explore all possible matching positions in the ring for each key character to find the most optimal path. This calls for a recursive strategy with memoization to avoid recalculating subproblems.

## 3. Key Observations

- The ring is circular, so we use:min(abs(i - j), len(ring) - abs(i - j)) to compute the rotation steps.
- Each character in the key may appear multiple times in the ring.
- This is a state transition problem, best solved using Dynamic Programming.
- We must keep track of the current ring position and the current key index to calculate minimum steps recursively.

## 4. Approach

- Preprocessing:
  - Build a map from each character to all indices it appears in the ring.
- Recursive Function:
  - Define dfs(index, curr_pos) which returns the minimum steps to spell key[index:] starting from ring position curr_pos.
- Memoization:
  - Use lru_cache to store already computed states for faster lookup.
- Transition:
  - For each position of the current character in the ring:
    - Calculate rotation steps from curr_pos.
    - Add 1 step to press the center button.
    - Recur for the next key character.
- Return:
  - Start the recursion from index = 0, curr_pos = 0.

## 5. Edge Cases

- The first character in key is already at position 0: No rotation needed.
- Repeating characters in the ring or key: Must consider all options for optimality.
- Long ring and key: Handled via memoization to avoid TLE.

## 6. Complexity Analysis

 Time Complexity:

- Let n = len(ring), m = len(key)
- For each key index (m), we might check each ring index (n), resulting in:
  - $O(m * n^2)$ in the worst case (with multiple appearances of each character).
- With memoization, we reduce redundant calculations.

💾 Space Complexity:

- O(m * n) for memoization cache.
- O(n) for the dictionary mapping characters to indices.
- O(m + n) auxiliary call stack space (recursive depth).

## 7. Alternative Approaches

- BFS (Breadth-First Search):
  - Simulate all possible rotations using a queue.
  - More complex and less efficient than memoized DFS for this problem.
- Bottom-Up DP:
  - Convert the recursive approach into tabular form.
  - Possible but harder to implement due to circular ring logic.

## 8. Test Cases

```
# Example 1
    print(Solution().findRotateSteps("godding", "gd"))  # Output: 4

# Example 2
    print(Solution().findRotateSteps("godding", "godding"))  # Output: 13

# Additional Cases
    print(Solution().findRotateSteps("abcde", "ade"))  # Output: 6
    print(Solution().findRotateSteps("aaaaa", "aaa"))  # Output: 3
    print(Solution().findRotateSteps("abc", "cba"))    # Output: 7
```

## 9. Final Thoughts

- The key to this problem is recognizing the circular nature of the ring and the possibility of multiple positions for each character.
- Recursive DP with memoization is optimal due to overlapping subproblems and manageable input size.
- This type of problem is excellent practice for state modeling and understanding how to optimize rotational/cyclic operations efficiently.