# D – Mayan Writing

Luo Tsz Fung {pepper1208}

2025-05-16

# Background

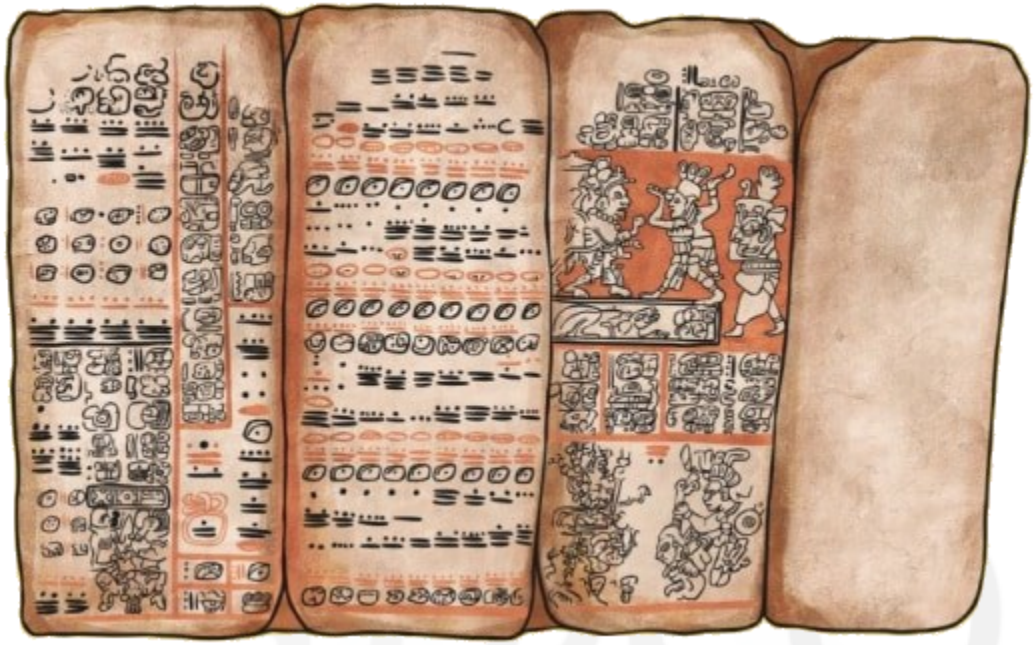
Problem Idea by `pepper1208`, IOI 2006

Preparation by `pepper1208`, `rina__owo`

# Problem Restatement

Find the number of occurrence of permutation of *W* in *S*.

Seems simple? ☺



# Subtasks

- Attempts: 9
- Max: 72
- First solved by **no one!**

# Subtasks Constraints

For all test cases,

$$1 \leq N \leq 3000$$

$$1 \leq M \leq 3 \times 10^6$$

# Solution 1.1: Brute Force

- Although naïve, brute force is often the first approach for us.
- Store all distinct permutation of *W*. Then, for each *W*, check the number of its occurrence in *S*.
- Time complexity: O(N! * M)
- Expected score: **36** (6/16 checkpoints)

# Solution 1.2: Brute Force + KMP

- “I have learnt KMP in Problem B! I will overkill this problem!”
- Time complexity: O(N! + M)
- Expected score: **36** (6/16 checkpoints)

- No use.
- You should analyze the time complexity of your algorithm before implementing it, especially for algorithm with significant time and effort.

# Solution 1.3: Brute Force Optimized

- You should realize that any solution that actually tries to match every possible permutation of *W* within *S* is doomed to failure. Even with a small alphabet the total number of permutations for *W* is very big.
- An optimal solution should take into account the fact that we are looking for *every* possible permutation of *W*. In other words, we do not care for the **order of the characters**.

# Solution 1.3: Brute Force Optimized

- Therefore, sort *W* first. All permutation of *W* must have the same result after sorting.
- Then, compare *W* to different partition of *S* with sorting.
- Time complexity: O(M * N log N)
- Expected score: **72** (12/16 checkpoints)

# Solution 1.4: Brute Force + Frequency Array

- We would like to omit the “N log N” factor for sorting.
- Therefore, we can use a frequency array technique to store the appearance of characters in *W* or in the partition of *S*.

# Solution 1.4: Brute Force + Frequency Array

- Starting from every position in *S*, look for a match by checking the number of times each character appears in *W* with maintaining two frequency arrays. If an error is found, stop an advance to the next position in *S*.
  - Error 1: There exists a character that does not appear in *W* in the window.
  - Error 2: There exists a character that the number of it appears in the window exceeds the number of times that character appears in *W*.
- Time complexity: $O(M^2)$
- Expected score: **50 – 65** (8 – 10/16 checkpoints, depends on implementation)
- Worse… ☹

# Solution 2.1: ?

- Have you notice something?
- When checking every position of *S*, the checking process keep “sliding rightwards in one direction”.
- We would not check the characters when they have checked.
- This leads us to think about the **two-pointers approach**, which is a fancy name for using a **queue** / **sliding window** to approach this problem.

# Solution 2.1: Two-pointers approach

- Every time a character is pushed or popped from the queue check every character in the alphabet to catch an error condition. One frequency array is enough to store the frequency of characters appear in *W*.
- Time complexity: O(M * Σ), where Σ is the number of possible characters
- Expected score: **70** (11/16 checkpoints)

# Solution 2.2: Two-pointers approach optimized

- How could we optimize the two-pointers approach?
- For two-pointers, it is unavoidable to check the whole *S*.
- Therefore, we can optimize the Σ factor.

# Solution 2.2: Two-pointers approach optimized

- Whenever you push or pop a character from the queue, the entry for that character must be checked in both vectors.
- Why not use two frequency arrays again?

# Solution 2.2: Two-pointers approach optimized

- After an error condition is reached, characters are popped from the queue until the error condition is cleared.
- Whenever you push or pop a character from the queue, the entry for that character must be checked in both frequency arrays.
- If at any time the length of the queue is equal to the length of *W* without having an error condition, the you have found a match!

# Solution 2.2: Two-pointers approach optimized

- Therefore, the Σ factor can be nicely eliminated.
- Time complexity: O(M)
- Expected score: **100 AC!** (16/16 checkpoints)

# Takeaways

- Although brute force is naïve, it is often the first choice for you. It may also lead you to the right path of optimizing the brute force algorithm.
- DRAW! In this question, it is better for you to simulate the searching process to find out the “shifting rightwards” fashion.
- Learn more algorithm and tools for you to optimize your solution, like eliminating some factor(s) occur in the time complexity using data structures or suitable algorithms. If you cannot think of two-pointers / queue with the right method, you cannot achieve full solution.