STRING ALGORITHMS

STRINGS DEFINITIONS AND TRICKS

- Size of a **string**
- Reversing a **string**
- All permutations of a **string**
- **String** concatenation
- **Vector** methods work on **strings**

PALINDROMIC STRING

- It is a **string** that reads the same if reversed
- Examples are:
  - “aba”
  - “a”
  - “feeceef”
  - “080”
  - “abcddcba”

FASTEST WAY TO CHECK A PALINDROME

1. Go through all elements to the half of the sequence
2. Check if the element equals the element on the opposite sides

- In other words:

$$Check\ for\ all\ i < \frac{|S|}{2}\ if\ S[i] == S[|S| - i\ - 1]\ then\ S\ is\ a\ Plaindrome$$

COUNT ALL PALINDROME SUBSTRINGS

- You have a **string S** of **size n**
- You are required to count all **palindromic substrings** in this **string** (starting from **size 2**)
- **Example Input: abaab**
- **Output: 3**
- **Explanation: (aba, aa, baab)**

REGULAR APPROACH

- For each two indices $i \neq j$ check if **substring** between $i \; and \; j$ is a palindrome
- The complexity for checking if **substring** is a palindrome is $O(n)$
- Then, overall complexity is $O(n^3)$

CAN WE DO BETTER?

DYNAMIC PROGRAMMING APPROACH

- If we know that **substring** between i and j is a palindrome, does it help us for the **substring** between $i - 1 \; and \; j + 1$?
- Yes, it does, we just need to check if $S[i-1] == S[j+1]$
- This can be translated to a **recursive approach**
- This **recursive approach** can be optimized using **DP**

THE RECURSIVE APPROACH

- $pal(i, j)$

  $if\ (i == j - 1)\ return\ P[i][j] = (S[i] == S[j]);$

  $pal(i, j\ - 1);$

  $pal(i + 1, j);$

  $if\ \big(pal(i + 1, j\ - 1)\big)\ return\ P[i][j] = (S[i] == S[j]);$

  $return\ 0;$

THE RECURSIVE APPROACH

- $count(i, j)$

  $if\ (i == j\ - 1)\ return\ P[i][j];$

  $C = count(i, j\ - 1);$

  $C\ += count(i + 1, j);$

  $C\ -= count(i\ + 1, j\ - 1);$

  $C\ += P[i][j];$

  $return\ C;$

IMPLEMENTATION (WITH DP)

EDIT DISTANCE

- You are given **2 strings** $\boldsymbol{S1}$ $\boldsymbol{and}$ $\boldsymbol{S2}$ you need to print the minimum number of moves to convert $\boldsymbol{S1}$ into $\boldsymbol{S2}$
- You can:
  - Insert
  - Remove
  - Replace

EXAMPLES

- $S1 = \text{“}cat\text{”}\ and\ S2 = \text{“}cut\text{”}$

Solution: 1

Explanation: Replace a with u cat – cut

- S1 = “sunday” and S2 = “saturday”

Solution: 3

Explanation: Replace **n** with **r** sunday – surday

Insert **t** surday – sturday

Insert **a** sturday – saturday

RECURSIVE APPROACH

- $edit(n, m)$

$$
\begin{aligned}
&if\ (n == 0)\ return\ m;\ //Insert\ elements\\
&if\ (m == 0)\ return\ n;//Delete\ elements\\
&if\ (S1[n] == S2[m])\ return\ edit(n\ -1, m\ -1);\\
&return\ 1 + \min(edit(n\ -1, m), \qquad //Delete\\
&\qquad\qquad edit(n, m\ -1), \qquad //Insert\\
&\qquad\qquad edit(n\ -1, m\ -1));\ //Replace
\end{aligned}
$$

- And we just add **DP** on the recursive approach.

IMPLEMENTATION (WITH DP)

PROBLEMS TO SOLVE

- Mashup contest ready on the group

THANK YOU! GOOD LUCK!