Name:- Mohammad Shaique Solanki ; 7062950

Q. 04-1

Lemma: Let $s\$$ be a string of length $|s\$| \geq n$
≥ 2

and let T be its compressed suffix tree.

Then the following holds about T:-

(i) T has exactly n leaves.

(ii) T has atleast $n-1$ inner nodes other than the root.

(iii) T has atmost $2(n-1)$ edges.

Proof:-

Let $l = |\text{No of leaves in } T|$

$I = |\text{No. of inner nodes of } T|$

$E = |\text{Edges of } T|$

W.K.T (we know that) : T is the suffix tree of s\$; There is exactly one leaf for each of the n suffixes of s\$.

$\therefore \boxed{L = n}$ — (1)

Now: Counting edges by incoming edges.

Every node except the root has exactly one incoming edges. Thus

$$E = |\text{no. of Non-root nodes}| = (I + L) - 1$$

(for root node)

$$\Rightarrow \boxed{E = I + L - 1} \quad \text{— (2)}$$

Now; Counting edges by outgoing edges.

Here, without loss of generality; we can assume that T is a compressed suffix tree; i.e No nodes have exactly one child; or Every inner node has atleast 2 children. (≥ 2). And leaves have 0 children.

$$\therefore \boxed{E = \sum_{i \in T} (\text{No. of children } (i)) \geq \sum_{i \in T} 2 = 2I} \quad -③$$

Combining ② & ③ we get

$$I + L - 1 = E \geq 2I$$

$$\Rightarrow \quad I + L - 1 \geq 2I$$

$$\Rightarrow \quad L - 1 \geq I$$

substituting ① $(L = n)$ we get

$$\boxed{I \leq n - 1} \quad -④$$

There are atmost $(n-1)$ inner nodes in T:

Now, Substituting (4) & (1) in (2) we get

$$E = I + 2 - 1 \leq (n-1) + n - 1) = 2n - 2 = 2(n-1)$$

$$\therefore \boxed{E = 2(n-1)}$$

Hence proved.

Q.4.2

Using the hint from the question;

We make relations b/w suffix trees and supermaximal repeats as follows.

1. Inner Nodes $\leftrightarrow$ repeats.

a) In suffix trees; an inner node v has atleast 2 children.

b) Each child is basically a distinct suffix that shares the label of v.

c) Therefore the label substring w occurs atleast twice in the Text.

Every inner node ~~La~~ labels a repeat of the text.

2. Maximal repeats $\leftrightarrow$ branching + left-maximality

a) Right maximality is taken care of ~~at~~ any innernode because we cannot extend substring w one more character to the right and still cover all of its occurences with a single node.

(b) To capture left maximality:-

We need to look at the character to the immidiate left of substring w.

In suffix tree, we can do this by looking at the edges that go to the leaves at inner node v.

(c) If there are at least two distinct left-contexts, we cannot extend substring w one more character to the left and still cover all occurrences.

$\boxed{v \text{ is a maximal-repeat node iff it has } \geq 2 \text{ children and } \geq 2 \text{ distinct left contexts}}$

3. Super-maximal repeats

(a) If we have any other maximal repeat w' and if $w \in w'$ then w is not supermaximal. Which means that a super maximal repeat should not appear inside any other maximal repeats.

(b) From Tree's perspective it means the node of any other Maximal repeat w' is parent or parent-of-parent of the node of substring w.

(c) So, for a supermaximal repeat substring w, the inner node v should be top-most in the tree. i.e. There should not be any other node above v in the hierarchy which is a node of another maximal substring.

(d) Vice-Versa is that there must be no inner node; which is heirarchically lower than v; which is maximal.

v is a supermaximal iff it has no descendant which is maximal.

Using Above conclusions, above, we will sketch out an algorithm.

Step 1 : Preprocessing

(i) Create a Suffix Array of String s and store starting positions in array. [Linear Time]

(ii) Compute lcp array (Linear Time)

(iii) Build child table $\to$ left child, $\to$ Right child ; such that each interval in lcp can list its children.

(IV) Build left context Array :- LC array

For each suffix in Suffix Array, look one character to the left in the original string. Correspond this Array with the Suffix Array.

Step 2

Check for Maximality.

Conditions:-

For any interval [i---j] in Suffix Array;

Check whether the LC table b/w [i--- j] have size ≥ 2. (i.e. those substrings have atleast two different characters immidiately before it)

And.

The interval [i---j] in Suffix Array should cover atleast two suffixes

i.e. $\boxed{j-i+1 \geq 2}$

If both conditions hold, then we can say that the interval [i---j] corresponds to maximal repeat.

Step 3 Check for Super maximality

This can be done by post-order DFS

(i) Process on each child.

For each child interval given by [i---j], we perform DFS. If any child DFS discovers any maximal repeats in its subtree; initiate a variable "max_present" and make it "True"

$\Rightarrow$ max_present == True.

(ii) Check current interval

Define a function "max_status" which if the current interval [i---j] is maximal or not.

The conditions of this are

$j - i + 1 < 2$ && $len(LC[i : j+1]) \geq 2$

As provided is step 2

If the condition is True, set "max status" == True.

If none of the descendants of the node given by this interval [i...j] has "max_present" == False; then this interval must by super maximal repeat.

(iii) Go up the heirarchy:-

Return whether "this subtree has any maximal repeat at or below [i...j]".

That way the parent if one of its descendants was already maximal or not.

Step-4

Start this Algorithm at root with interval [0 . . . n−1]

When initial call finishes, we will have exactly those repeats which are maximal but not contained in any larger repeats.

Conclusion:

1. We preprocess the string into Suffix Array, Lcp array, Child Array and Left Context Array.

   All of these can be done in O(n) time

2. We define a set of conditions to check whether an interval [i...j] is maximal repeat

3. We do one post-order DFS query, and when we hit a maximal repeat that no deeper node has already seen, we output it as supermaximal repeat.

— * End of Algorithm * —