Suffix Trees

11 August 2024 23:58

Benefits: T(m), P(n)
→ O(m) preprocessing and O(n) query
→ Inexact matching supported.

Weiner → McCreight → Ukkonen

Radix tree: compressed trie.

Definition A suffix tree T for an m-character string S is a rooted directed tree with exactly m leaves numbered 1 to m. Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty substring of S. No two edges out of a node can have edge-labels beginning with the same character. The key feature of the suffix tree is that for any leaf i, the concatenation of the edge-labels on the path from the root to leaf i exactly spells out the suffix of S that starts at position i. That is, it spells out $S[i..m]$.

Suffix tree for an m character string S:

1. Rooted, directed.
2. Each internal node has atleast 2 children
3. Each edge is labeled with a non-empty substring of S.
4. No 2 edges can have edge labels with the same prefix.
5. For any leaf i, path to root spells out the suffix of S starting at i.

! If one suffix of S matches the prefix of another suffix of S then the suffix tree wouldn't be possible.

↓ Suffix won't end in a leaf

Soln. Terminator character

$O(m^2)$ construction

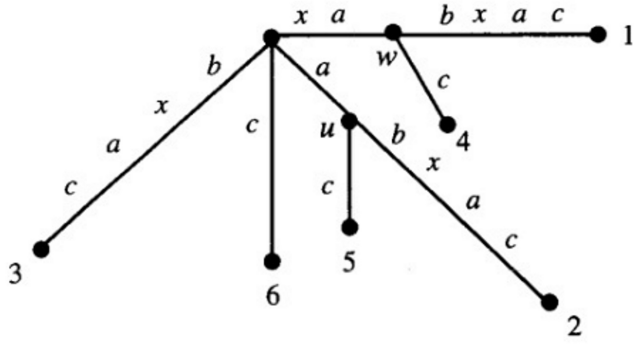
Let P = set of all suffixes of S

$|P| = m$

A trie constructed out of the m elements of P will be the suffix tree for S.

LABEL = Concatenation of strings on the path from the root to a node

STRING DEPTH = |label|



→ xabx labels a node that lies inside the edge (w, 1)

Exact Matching

T(m), P(n) → O(m) preproc. O(n) query.

→ Build suffix tree for text T.

→ Match characters along a unique path in T until

- P is exhausted — every leaf in the subtree corresponds to a match
- No possible match.

UKKONEN'S ALGORITHM

Gusfield — Algos. on Strings, Trees and Seqs

Implicit Suffix tree: A tree obtained from the suffix tree for S$ by removing

- every copy of the terminal symbol $ from the edge labels of the tree.
- removing edges without labels.
- removing any node that doesn't have two children.

I_i = implicit suffix tree for $S[1, i]$

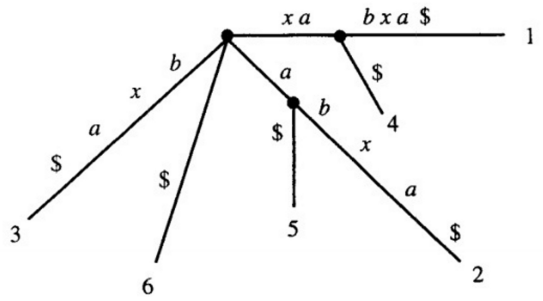


Figure 6.1: Suffix tree for string *xabxa* $.

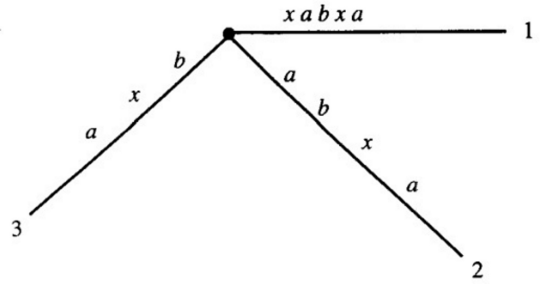


Figure 6.2: Implicit suffix tree for string *xabxa*.

→ The implicit suffix tree will have fewer nodes iff at least one of the suffixes is a prefix of another suffix.

HIGH-LEVEL UKKONEN'S ALGO

→ m phases
→ phase i+1 : I_{i+1} is constructed using I_i
→ Each phase has i+1 extensions
One for each suffix of $S[1 \ldots i]$

→ In the extension j of the $(i+1)^{th}$ phase, the algo first finds the end of the path from the root labeled with substr. $S[j \ldots i]$. It extends the substr. by adding $S(i+1)$ at the end if reqd.

High-level Ukkonen algorithm

Construct tree $\mathcal{I}_1$.
For i from 1 to $m-1$ do
begin {phase $i+1$}
  For j from 1 to $i+1$
    begin {extension j}
    Find the end of the path from the root labeled $S[j..i]$ in the
    current tree. If needed, extend that path by adding character $S(i+1)$,
    thus assuring that string $S[j..i+1]$ is in the tree.
    end;
end;

→ $O(m^3)$ time complexity

Optimizations

① Suffix Links

Let $x\alpha$ denote an arbitrary string
where x = single char
α = substring (possibly empty)

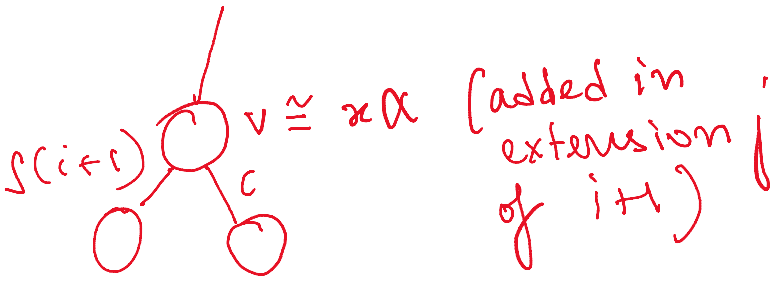
For an internal node v with path label $x\alpha$, if there is another node $s(v)$ with path label α, then a ptr. from v to $s(v)$ is a suffix link.

$v \longrightarrow s(v)$
$x\alpha \qquad \alpha$

- If α is empty, then the suffix label from an internal node with path label $x\alpha$ goes to the root node
- Root node is not considered internal

Lemma: If a new internal node v with path label $x\alpha$ is added to the

Lemma: If a new ... with path label $x\alpha$ is added to the current tree in extension j of phase $i+1$, then either the path labeled α already ends at an internal node of the current tree or an internal node at the end of string α will be created in extension $j+1$ in the same phase $i+1$.



Corollary: Any newly created internal node will have a suffix link from it by the end of the next extension.

Corollary In any implicit suffix tree I_i, if internal node v has path label $x\alpha$, then there is a node $s(v)$ of I_i with path label α.

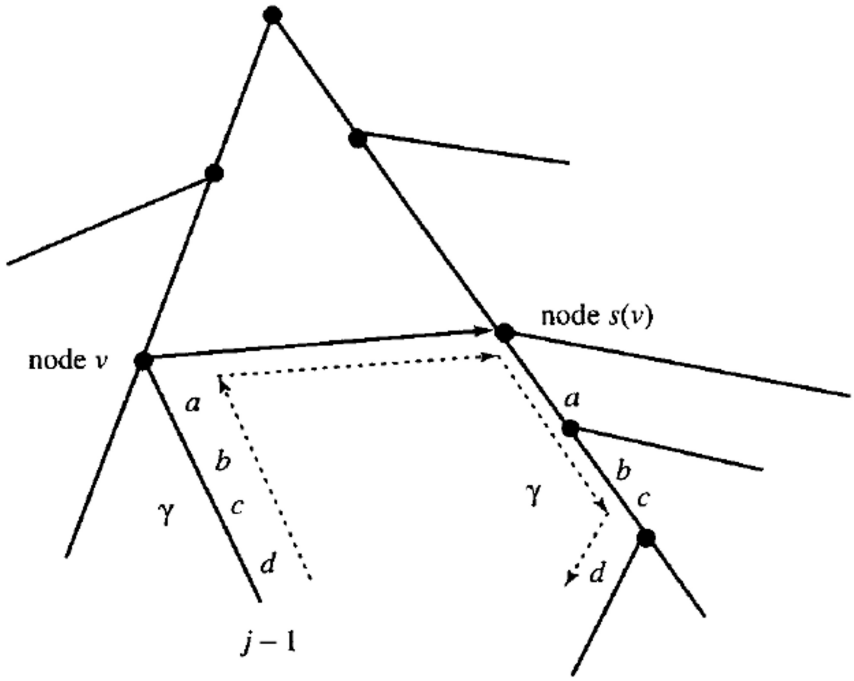


Figure 6.5: Extension $j > 1$ in phase $i+1$. Walk up atmost one edge (labeled γ) from the end of the path labeled $S[j-1..i]$ to node v; then follow the suffix link to $s(v)$; then walk down the path specifying substring γ; then apply the appropriate extension rule to insert suffix $S[j..i+1]$.

Suffix extension rules: $S[j \ldots i] = \beta$

① β ends in a leaf
Add $S(i+1)$ to the label

② No path from the end of string β starts with $S(i+1)$. but atleast one labeled path continues.
→ Split and add a new node

③ Some path from end of string β starts with $S(i+1)$.
→ Do nothing

FOLLOWING A TRAIL OF SUFFIX LINKS

• TO BUILD I_{i+1}.

For $j=1$

- The end of the full string $S[1 \ldots i]$ must end at a leaf of I_i since $S[1 \ldots i]$ is the longest string in the tree.
- Store a ptr. to this node at the end of each phase and use it. (Rule 1)

For $j=2$

Let $S[1,i] = x\alpha$

We need to find the node corresponding $S[2 \ldots i] = \alpha$.

- v is either an internal node or the root

Internal node

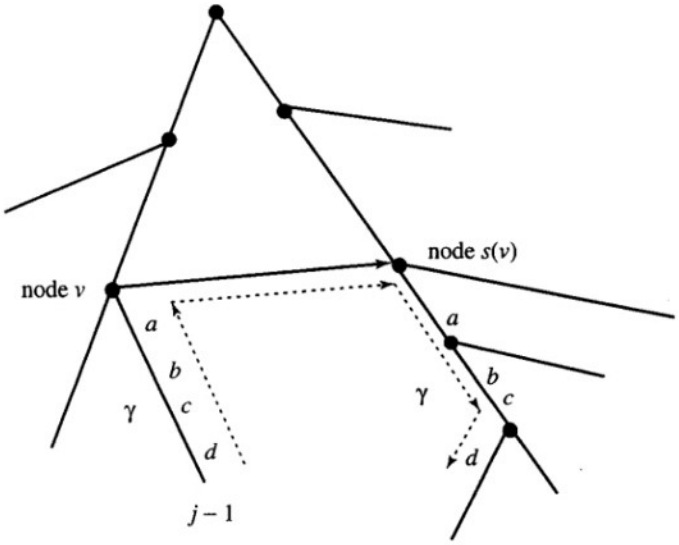
Consider the suffix link $v \rightarrow s(v)$.

$s(v)$ must be a prefix of α.

∴ The reqd. node with the label α can be found by walking down in this subtree.

Root

Walk down and get α.



$j > 2$

Starting at the end of string $S[j-1..i]$ repeat the same general idea.

→ If the end of $S[j-1..i]$ itself has a suffix link, take that.

Single extension algorithm

Begin

1. Find the first node v at or above the end of $S[j-1..i]$ that either has a suffix link from it or is the root. This requires walking up at most one edge from the end of $S[j-1..i]$ in the current tree. Let γ (possibly empty) denote the string between v and the end of $S[j-1..i]$.
2. If v is not the root, traverse the suffix link from v to node $s(v)$ and then walk down from $s(v)$ following the path for string γ. If v is the root, then follow the path for $S[j..i]$ from the root (as in the naive algorithm).
3. Using the extension rules, ensure that the string $S[j..i]S(i+1)$ is in the tree.
4. If a new internal node w was created in extension $j-1$ (by extension rule 2), then by Lemma 6.1.1, string α must end at node $s(w)$, the end node for the suffix link from w. Create the suffix link $(w, s(w))$ from w to $s(w)$.

End.

If a new internal node w is created during extension $j-1$, then α must end at node $s(w)$, the end node [illegible] $s(w)$).

during extension j, the end node ends at node $s(w)$, the end node for the suffix link $(w, s(w))$.

② Skip Count Trick

→ Note that the first character of γ must appear as the first character of one of the outgoing edges.

Let $|\beta| = g$ and $|\text{label of this edge}| = g'$.

Single phase of the Ukkonen's algo. takes $O(m)$ time

If $g \geq g'$: Skip to the end and update γ & g.

Otherwise

The node lies on this edge.

③ Edge Label Compression

$O(m^2)$ space → $O(m)$ space.

④ Rule 3 Bypass.

In any phase, if the suffix extension rule applies in extension j, it will

rule 0 applies in extension 0, also apply in all further extensions in the current phase.

→ When rule 3 applies, the path labelled S[j, i] in the current tree must continue with character S(i+1). Since S[j, i] . S(i+1) is in the tree, all its suffixes must be there too.

⇒ When rule 3 applies, move to the next phase.

(5) Once a leaf, always a leaf

- Once a leaf is created and labeled j (for the suffix starting at position j of S), then that leaf will remain a leaf during all successive phases.
- Each phase looks like:

rule 1, 2, 1, 1, 2, (3) X ——

∴ : last extension of

j_i : last extension of the sequence.

→ Since any applicⁿ of rule 2 creates a new leaf : $j_i \leq j_{i+1}$

leaf. In more detail, once there is a leaf labeled j, extension rule 1 will always apply to extension j in any successive phase. So once a leaf, always a leaf.

Now leaf 1 is created in phase 1, so in any phase i there is an initial sequence of consecutive extensions (starting with extension 1) where extension rule 1 or 2 applies. Let j_i denote the last extension in this sequence. Since any application of rule 2 creates a new leaf, it follows from Observation 2 that $j_i \leq j_{i+1}$. That is, the initial sequence of extensions where rule 1 or 2 applies cannot shrink in successive phases. This suggests an implementation trick that in phase $i+1$ avoids all explicit extensions 1 through j_i. Instead, only constant time will be required to do those extensions implicitly.

Primarily because,
once there is a leaf labeled j,
only extension rule 1 applies to it
in the extension j of any successive phase.

- Now, for any leaf edge in I_i, index q in $S[p..q]$ is equal to i and it just gets incremented to (i+1) in the next phase.

Trick 2 End any phase $i+1$ the first time that extension rule 3 applies. If this happens in extension j, then there is no need to explicitly find the end of any string $S[k..i]$ for $k > j$.

The extensions in phase $i+1$ that are "done" after the first execution of rule 3 are said to be done *implicitly*. This is in contrast to any extension j where the end of $S[j..i]$ is explicitly found. An extension of that kind is called an *explicit* extension.

Trick 2 is clearly a good heuristic to reduce work, but it's not clear if it leads to a better worst-case time bound. For that we need one more observation and trick.

Trick 3 In phase $i+1$, when a leaf edge is first created and would normally be labeled with substring $S[p..i+1]$, instead of writing indices $(p, i+1)$ on the edge, write (p, e), where e is a symbol denoting "the current end". Symbol e is a *global* index that is set to $i+1$ once in each phase. In phase $i+1$, since the algorithm knows that rule 1 will apply in extensions 1 through j_i at least, it need do no additional explicit work to implement

those j_i extensions. Instead, it only does constant work to increment variable e, and then does explicit work for (some) extensions starting with extension $j_i + 1$.

⇒ Skip all extensions from 1 to j_i in the phase $i+1$.

Single phase algorithm: SPA

Begin

1. Increment index e to $i+1$. (By Trick 3 this correctly implements all implicit extensions 1 through j_i.)
2. Explicitly compute successive extensions (using algorithm SEA) starting at $j_i + 1$ until reaching the first extension j^* where rule 3 applies or until all extensions are done in this phase. (By Trick 2, this correctly implements all the additional implicit extensions $j^* + 1$ through $i+1$.)
3. Set j_{i+1} to $j^* - 1$, to prepare for the next phase.

End

CREATING THE TRUE SUFFIX TREE

- Just run the algorithm with S$.
  → The implicit suffix tree obtained at the end will be the same as the true suffix tree.
  → Set e to m.