A suffix tree is a compressed trie of all suffixes of a string \(x\) \(x=abaab\) (\(x\) has length \(m-1\), \(\$\) is an unique sentinel character, thus \(x\$\) has a length \(m\)).

1: abaab$
2: baab$
3: aab$
4: ab$
5: b$
6: $

Each leaf node is marked with an integer \(j\) corresponding to suffix \(S[j…m]\).
A suffix tree is a compressed trie of all suffixes of a string $x$. The path labels are specified by two integers $(k, l)$, $k$=start index or position of the path and $l$= end position of the path.

Each leaf node is marked with an integer $j$ corresponding to suffix $S[j...m]$. 

1: abaab$
2: baab$
3: aab$
4: ab$
5: b$
6: $

$O(m)$ Storage Suffix Tree
Implicit Suffix Tree

- Remove all $$. Remove all edges without any label. Remove all nodes with a single child and the merge the path labels into one label.

1: abaab$
2: baab$
3: aab$
4: ab$
5: b$
6: $

Suffixes 4,5 and 6 appear implicitly.
Ukkonen’s Algorithm

- This is an on-line algorithm. Given the sequence $S[1,2,\ldots,m]$, it constructs implicit suffix trees $I_i$ for the prefix $S[1,2,\ldots,i]$ starting from $I_1$ incrementing $i$ by 1 until $I_m$ is built. There are $m$ phases. In phase $i+1$, $I_{i+1}$ is constructed from $I_i$. The $(i+1)$ phase has $(i+1)$ extensions. Extension $j$ ($1\leq j \leq i+1$) deals with the sequence $S[j,\ldots(i+1)]$.
High-Level Description

- Construct $I_1$
- For $i$ from 1 to $m-1$
  (begin phase $i+1$)
    For $j$ from 1 to $i+1$
    (begin extension $j$)
      Walk down the tree from root along the path $S[j...i]$ in the current implicit tree. If needed, add $S[i+1]$ at the end of the path to insure $S[j...i+1]$ is in the tree.
    end
  end
end
Types of Nodes

- **Leaf Node**: corresponds to a suffix
- **Explicit Node**: Internal node that has at least two branches
- **Implicit Nodes**: corresponds to a suffix but due to path compression resulting from elimination of $, it only has one child and hence has been reduced to an implicit node.

We will use Greek symbols $\alpha$, $\beta$, $\gamma$ to denote strings and symbols $x, y, z$ to denote single characters. Let $\beta=S[j \ldots i]$ be a suffix of $S[1 \ldots i]$
**Suffix Extension Rules**

- **Rule 1**: Once a leaf, always a leaf.
- **Rule 2**: No path from end of string $\beta$ in $I_i$ starts with $S[i+1]$ but the path continues. Then attach an edge with label $S[i+1]$ at the end of $\beta$ creating an explicit node if necessary.

![Diagram](image)

(The ‘white’ tree is in $I_i$. White plus **purple** is in $I_{i+1}$)
Rule 3: Some path from the end of $\beta$ starts with $S[i+1]$ so $\beta S[i+1]$ is already in the tree. Do nothing.
An example: $S=axabxb$

Phase $i=1$ (Note, in phase $i=0$, the tree simply a single node, the root corresponding to the empty string $\varepsilon$)

$\beta=\varepsilon$; Only one extension. $\beta S[i+1]=a$.

Rule 2 is applicable and $I_1$ is $a/1$

Phase $i=2$ Two extensions for suffixes $ax$ and $x$. Suffix $ax$ is inserted using Rule 1 and $x$ in inserted by using Rule 2  
(Rule Sequence:12)
Example (continued)

Phase i=3

Three extensions for suffixes axa, xa and a. Note, the first two extensions are simply copying l₂ and adding last character S[i+1]=a to it at the leaf nodes by applying Rule 1. Then the last suffix is a single character a which is handled, in this case, by Rule 3 (do nothing).

(Rule Sequence: 113)
Phase 4

Phase $i=4$

The four suffixes are $axab, xab, ab$ and $b$

The Rule sequence (1122). Note due to Rule 2, two explicit nodes have been added.
Phase 5

- Phase i=5
- The sequences are axabx, xabx, abx, bx and x.
- Rule Sequence (11113)
  (Node 5 is implicit)
Phase 6

- Phase $i=6$
- The suffixes are $axabxb,xabxb,abxb, bxb,xb$ and $b$

The Rule

Sequence: (111123)
Properties of Rule Sequence

- A very important property of the Rule sequence is that it consists of an initial sequence of 1’s (except for phase 1 which has one Rule 2) followed by possibly a sequence of 2’s and if a 3 occurs at the end, further application of Rule 3 can be abandoned.

- Another property is that the length of the Rule 1 sequence in phase i+1 is equal or strictly one more than the length of Rule 1 sequence in phase i. (Justify the above statements)
The Naïve Algorithm

- In the naïve algorithm that follows from the ‘High Level Description’, once we locate the end of the string β in the current tree Iᵢ, inserting S[i+1] after it, takes constant amount of work. The crux of the problem is to find out where β ends in the current tree. For this, we engaged in walking down the tree matching characters taking \( O(|β|) \) time for \(|β| = i, i-1, \ldots, 2, 1 \) for the (i+1) phase.
- So, Iᵢ₊₁ is created from Iᵢ making \( O(i^2) \) character comparisons. So, total number of comparisons for all phases is

\[
\sum_{i=1}^{m} o(i^2) = O(m^3)
\]
Suffix Link- how to be lazy but smart (or walking is good for your health but not for your algorithm)

- To avoid walking down the tree for each $\beta$, suffix links are introduced.
- Definition: Let an internal node $v$ has a path label $x \alpha$ where $x$ is a character and $\alpha$ is a string (possibly empty). If there is another node $s(v)$ in the tree with path label $\alpha$, then a pointer from $v$ to $s(v)$ is called a suffix link, denoted $(v,s(v))$. If $\alpha$ is empty string $\epsilon$, the suffix with path label $x$ goes to the root. The root is not considered internal and has no suffix link.
Suffix Link Examples

S = abababc

S = aaaaa$
Lemma: Let an internal node \( v \) with path label \( x \alpha \) be added to the current tree \( I_i \) in extension \( j \) of the \( (i+1) \) phase. Then

- Either an internal node with edgelabel \( \alpha \) already exits in the current tree \( I_i \)
- Or an internal node with path label \( \alpha \) will be created in extension \( j+1 \) in the same phase \( i+1 \).
Suffix Link Creation

- **Corollary 1**: In Ukkonen’s algorithm, any newly created internal node will have a suffix link from it by the end of the next extension.

- **Corollary 2**: In any suffix tree $I_i$, if an internal node $v$ has a path label $x \alpha$, then there is a node $s(v)$ of $I_i$ with path label $\alpha$. 
First Extension

- Using Rule 1 (once a leaf, always a leaf), the first extension can be done in constant time. Keep a pointer to the leaf node 1 of current tree \( I_i \) corresponding to \( S[1...i] \). Just add \( s[i+1] \) at the end, node label still remains 1 and the pointer is adjusted to point to new node 1.
Second Extension

- Let $S[1..i] = x \alpha$ ( $X$ a character, $\alpha$ could be $\varepsilon$). To do the extension, we need to find the end of $S[2..i] = \alpha$ in the current tree $I_i$. Let $(v, 1)$ be the edge that enters leaf 1. (Node $v$ could be the root [viz. for $S=aaaaa\$] or an internal node.

- If $v$ is the root, walk down the tree following the path label $\alpha$. 
Second Extension

- If $v$ is an internal node, walk up from leaf 1 via edge $(v,1)$ to node $v$.
- Follow the suffix link $(v,s(v))$.
- Walk from $s(v)$ down the path checking for all the characters in the string $γ$ which is the path label of $(v,1)$. This journey may use more than one edge.
- Update the tree following the applicable extension rule at the end of the path (it could be Rule 1, 2 or 3).

$γ = abcd$
General Extension j>2

- The procedure is essentially the same as for j=2 except we start from string S[j-1..i] in the current tree I_i and walk up at most one node to either root or node v, follow path γ to the end of S[j..i] and then extend the suffix to s[j..i+1] using the applicable extension rule.
- There is one difference: the end of S[j-1..i] may itself have a suffix link; then do not walk up any node, just follow the suffix link.
- If a new internal node was created in extension j-1 (by extension Rule 2) then a string α must end at node s(w) [by Lemma]. Then create the suffix link (w, s(w)).
Single Extension Algorithm

- Find the first node $v$ at or above the end of $S[j-i..i]$ that either has a suffix link from it or is the root. This requires walking up at most one edge with label $\gamma$.
- If $v$ is not the root, traverse suffix link $(v,s(v))$ and then walk down $\gamma$ from $s(v)$. If $v$ is the root follow the path $S[j..i]$ from root, as in the naïve algorithm.
- Using the extension rules, ensure that $S[j..i]S[i+1]$ is in the tree.
- If a new internal node was created in extension $j-1$ (by extension Rule 2) then a string $\alpha$ must end at node $s(w)$ [by Lemma]. Then create the suffix link $(w, s(w))$.

The suffix link improves the performance in practice but so far the worst case time complexity is still $O(m^3)$. 
Trick 1: Skip and Count

- The complexity of walking down $\gamma$ from $s(v)$ is $O(|\gamma|)$. $g = |\gamma|$=number of character in $\gamma$. No two edges out of $s(v)$ have the same character; so the first character of $\gamma$ appears in a unique path from $s(v)$. Let $g'$ be the number of characters in the edge with this unique character. If $g' < g$ the algorithm can simply skip to the node at the end of the edge and set $g <- g - g'$ $h <- g' + 1$ and look for $h$th character of $\gamma$ for the next match to follow the downward path.
Skip and Count

- This process can be iterated for the succeeding edges as long as g’<g. When an edge is reached such that g<=g’, the algorithm skips to character number g in the path.
Complexity of Skip and Count

- It should be obvious that moving from node to node using (k, l) labels and the g and h values on the γ path takes constant time. The total time to traverse the path is thus depends on the number of nodes traversed rather than the number of characters.

- **Node Depth**: Node depth of a node u is the number of nodes on the path from the root to u.

- **Lemma**: Let (v, s(v)) be the suffix link traversed at any time in the algorithm. At this time, the node depth of v is at most 1 greater than node depth of s(v).

- **Theorem**: Using skip and count technique, any phase of Ukkonen’s algorithm takes O(m) time.

- **Corollary**: Ukkonen’s algorithm can be implemented with suffix link to run in O(m^2) time.
Trick 2: Rule 3 is a show stopper

- Rule 3 means do nothing because the path labeled $s[j..i]$ continues with character $S[i+1]$. So, do the paths labeled $S[j+1..i]$, $S[j+2..i]$, $\ldots$, $S[i]$. Thus if extension Rule 3 applies in extension $j$, the same Rule 3 must apply to all succeeding extensions in $(i+1)$ phase. This leads to:

- Trick 2: End any phase $i+1$ the first time 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 $k>j$. 
Extensions 1 in bulk

- First, to conserve storage, we are not going to write the character sequences in any edge. Instead we use a pair of indices \((k,l)\) to limit storage to \(O(m)\).
- Second, due to “once a leaf, always a leaf” rule, once there is a leaf labeled \(j\), extension Rule 1 will apply always to extension \(j\) in any successive phase.
- 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 phase \(i\). It follows from “Once a leaf, always a leaf” that \(j_i \leq j_{i+1}\). That is, the initial sequence of extension rules 1 or 2 cannot shrink in successive phases.
- Let us take an example.
An example: $S = axabxb$

Phase $i=1$ (Note, in phase $i=0$, the tree simply a single node, the root corresponding to the empty string $\varepsilon$)

$\beta = \varepsilon$; Only one extension. $\beta S[i+1] = a$.

$I_1$ is with value of $e=1$ $(1,e)$

Phase $i=2$ Two extensions for suffixes $ax$ and $x$. Suffix $ax$ is inserted using Rule 1 and $x$ in inserted by using also Rule 1. Root node is not considered an Internal node. (Rule Sequence: 11) $e=2$
Example (continued)

Phase i=3

Three extensions for suffixes axa, xa and a. Note, the first two extensions are simply copying $l_2$ and adding last character $S[i+1]=a$ to it at the leaf nodes by applying Rule 1. Then the last suffix is a single character a which is handled, in this case, by Rule 3 (do nothing).

(Rule Sequence: 113)

$e=3$
Phase 4

Phase i=4

The four suffixes are axab, xab, ab and b. The Rule sequence is (1122). Note due to Rule 2, two explicit nodes have been added. The k values in (k,e) for the edges are determined at the time Rule 2 is applied. Note e=4 for all leaf nodes.
Phase 5

- Phase i=5
- The sequences are axabx, xabx, abx bx and x.
- Rule Sequence
  (11113) e=5
(Node 5 is implicit)
Phase 6

- Phase i=6
  - (Wait, wait!! You have been misleading us saying all these e-values are changed in O(m) times. If we change the value of e at every phase, it is going to take O(m^2) time.)
  - Here’s the punch line! Don’t change the value of e until the last phase and make e equal to the value of maximum phase for the i+1 extension, which is i+1. Since there are total of m extensions, the work involved in all the Rule 1 applications is O(m)

The Rule Sequence: (111123)

The suffixes are axabxb, xabxb, abxb, bxb, xb and b
Extension Sequences

The extension sequences for different phases are (1),(11),(113),(1122),(11113) and (111123). Note \( j_1 < j_2 = j_3 < j_4 = j_5 < j_6 \). This suggests an implementation trick that avoids in phase \( i+1 \) all explicit extensions 1 through \( j_i \). Only constant time will be needed to do all these extensions.
Trick Number 3

- In phase $i+1$, when a leaf is first created and would normally be labeled as $S[p...i+1]$ written as $(p,i+1)$ on the edge, write this as $(p,e)$, where $e$ (denoting “the current end”) is a global variable that is set to value $i+1$ once in each phase.

- In phase $i+1$, the algorithm knows that Rule 1 will apply in extensions $1$ through $j_i$ at least, only constant amount of work is needed up to extensions $j_i$ to increment the value of $e$. The algorithm can then proceed to extension $j_i+1$ and perform explicit work if needed.
The Punch Line

- With tricks 2 and 3, explicit extensions in phase i+1 (using SEA – Single Extension algorithm) are only required from extension $j_i+1$ until the first Rule 3 applies or until i+1 is done. All other extensions before or after those explicit extensions, are done implicitly.
The Single phase Algorithm SPA

begin

1. Increment index e by i+1. By Trick 3, this correctly implements all implicit extensions 1 through $j_i$.

2. Explicitly compute successive extensions (using SEA) starting at $j_i+1$ until reaching first extension $j^*$ where Rule 3 applies or until all extensions are done in this phase. By Trick 2 (show stopper), this correctly implements all the additional implicit extensions $j^* +1$ through $j+1$.

3. Set $j_{i+1}$ to $j^*$ to prepare for the next phase.

end