Suffix Tree (Ukkonen's algorithm)

Wing

February 22, 2019

Table of contents

Trie

Trie

An ordered tree data structure used to store a dynamic set or map where the keys are usually strings

Suffix Trie

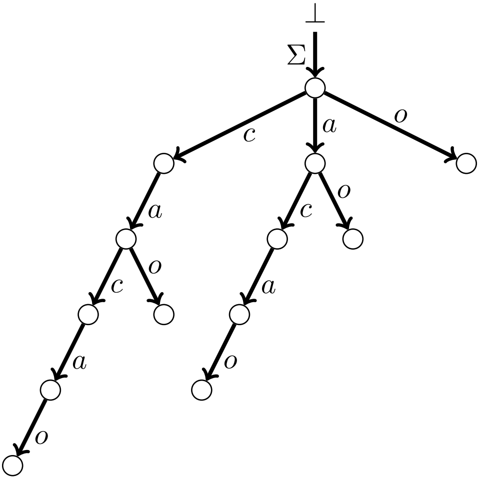

Figure: Suffix Trie for "*cacao*" (Suffix links omitted)

Suffix Trie

- Proposed by Esko Ukkonen (University of Helsinki, Finland)
- An algorithm easier to grasp than the those in the literature at that time
- On-line algorithm: Processes the string symbol by symbol from left to right, and always has the suffix tree for the scanned part of the string ready

Construction of Suffix Trie

String

Let $T = t_1 t_2 \cdots t_n$ be a string over alphabet Σ

Substring

Each string $x : T = uxv$ for some (possibly empty) string u and v is a substring of T

Suffix

$T_i = t_i \cdots t_n$ where $1 \leq i \leq n+1$

- $T_{n+1} = \epsilon$ is the *empty* suffix

Construction of Suffix Trie

Set of all suffixes of T

$\sigma(T)$

The suffix trie of T is a trie representing $\sigma(T)$

Construction of Suffix Trie

Suffix Trie

Denote suffix trie of T as $STrie(T) = (Q \cup \{\perp\}, root, F, g, f)$

Define such a trie as an augmented deterministic finite-state automaton which has a tree-shaped transition graph representing the trie for $\sigma(T)$

augmented with

- f : suffix function
- $\perp$: auxiliary state

Construction of Suffix Trie

Set Q of the states of $STrie(T)$

The set Q of the states of $STrie(T)$ can be put in a one-to-one correspondence with the substrings of T.

Denote $\bar{x}$ the state that corresponds to a substring x
Shorthand: $\bar{x} \sim x$

- $root \sim \epsilon$
- set F of final states $\sim \sigma(T)$

Construction of Suffix Trie

Transition function g

$$\begin{cases} g(\bar{x}, a) = \bar{y} & \forall \bar{x}, \bar{y} \in Q : y = xa, \text{ where } a \in \Sigma \\ g(\perp, a) = root & \forall a \in \Sigma \end{cases}$$

Suffix function f

$\forall \bar{x} \in Q$,

$$\begin{cases} f(\bar{x}) = \bar{y} & \text{if } \bar{x} \neq root, \text{then } x = ay, a \in \Sigma \\ f(root) = \perp \\ f(\perp) \text{ is undefined} \end{cases}$$

$\perp \sim a^{-1} \; \forall a \in \Sigma$

$a^{-1}a = \epsilon$

Construction of Suffix Trie

Suffix Link

$f(r)$ is the suffix link of state r

Prefix

$T^i = t_1 \cdots t_i$ of T for $0 \leq i \leq n$

Construction of Suffix Trie

Key observation

How is $STrie(T^i)$ obtained from $STrie(T^{i-1})$?

The suffixes of T^i can be obtained by catenating t_i to the end of each suffix of T^{i-1} and by adding an empty suffix, i.e.

$$\sigma(T^i) = \sigma(T^{i-1})t_i \cup \{\epsilon\}$$

$STrie(T^{i-1})$ accepts $\sigma(T^{i-1})$, to make it accept $\sigma(T^i)$, examine F_{i-1} of $STrie(T^{i-1})$

- $r \in F_{i-1}$ doesn't have a t_i-transition $\Rightarrow$ add transition $r \rightarrow$ new state
- $r \in F_{i-1}$ has a t_i-transition $\Rightarrow$ follow the transition to the old state
- All such states plus *root* will be F_i of $STrie(T^i)$

Construction of Suffix Trie

How to find states $r \in F_{i-1}$ that get new transitions?

From definition of the suffix function f,
$r \in F_{i-1} \Leftrightarrow r = f^j(\overline{t_1 \cdots t_{i-1}})$ for some $0 \leq j \leq i-1$

Boundary path

Boundary path of $STrie(T^{i-1})$:
Path starting from deepest state $\overline{t_1 \cdots t_{i-1}}$ of $STrie(T^{i-1})$,
following the suffix links and ending at $\perp$

$\therefore$ All states in F_{i-1} are on the boundary path of $STrie(T^{i-1})$

The boundary path is traversed.

If a state $\bar{z}$ on the boundary path does not have a transition on t_i yet, add a new state $\overline{zt_i}$ and a new transition $g(\bar{z}, t_i) = \overline{zt_i}$

To update f, new states $\overline{zt_i}$ are linked together with new suffix links starting from $\overline{t_1 \cdots t_i}$.
Obviously, this is the boundary path of $STrie(T^i)$

Construction of Suffix Trie

Observation

The traversal over F_{i-1} along the boundary path can be stopped immediately when the first state $\bar{z}$ is found s.t. state $\overline{zt_i}$ (and hence also transition $g(\bar{z}, t_i) = \overline{zt_i}$) already exists.

Let namely $\overline{zt_i}$ already be a state.

Then $STrie(T^{i-1})$ has to contain state $\overline{z't_i}$ and transition $g(z', t_i) = \overline{z't_i}\ \forall z' = f^j(\bar{z}), j \leq 1$.
In other words, if $\overline{zt_i}$ is a substring of T_{i-1} then every suffix of $\overline{zt_i}$ is a substring of T_{i-1}.

Such $\bar{z}$ must exist as $\perp$ is the last state on the boundary path that has the t_i-transition $\forall t_i$

Construction of Suffix Trie

top denotes the state $\overline{t_1 \cdots t_{i-1}}$

Algorithm 1:

$r \leftarrow top$
while $g(r, t_i)$ *is undefined* **do**
    create new state r' and new transition $g(r, t_i) = r'$;
    if $r \neq top$ **then** create new suffix link $f(oldr') = r'$;
    $oldr' \leftarrow r'$;
    $r \leftarrow f(r)$;
create new suffix link $f(oldr') = g(r, t_i)$
$top \leftarrow g(top, t_i)$.

Running Algorithm 1 for $t_i = t_1, t_2, \cdots, t_n$ visits each $\bar{x} \in Q$ once.

Theorem 1

Suffix trie $STrie(T)$ can be constructed in time proportional to the size of $STrie(T)$ which, in the worst case, is $\mathcal{O}(|T|^2)$.

Suffix Tree

Look at the figure.

Suffix tree

Suffix tree $STree(T)$ of T is a data structure that represents $STrie(T)$ in space linear in the length $|T|$ of T

Explicit states

$Q' \cup \{\perp\}$ is the explicit states of $STrie(T)$

$Q' \subseteq Q$ consists of all branching states and all leaves of $STrie(T)$
By definition, $root$ is included into the branching states

Implicit states

Other states of $STrie(T)$ is the implicit states

Construction of Suffix Tree

Generalized transition

$g'(s, w) = r$ in $STree(T)$ represents the string w spelled out by the transition path in $STrie(T)$ between two explicit states s and r

To save space, the string w is represented as a pair (k, p) of pointers: $t_k \cdots t_p = w$
Then $g'(s, (k, p)) = r$

Such pointers exist because there must be a suffix T_i s.t. the transition path for T_i in $STrie(T)$ goes through s and r

Select the smallest such i, and let k and p point to the substring of this T_i that is spelled out by the transition path from s to r

Construction of Suffix Tree

A transition $g'(s,(k,p)) = r$ is called an $\underline{a\text{-}transition}$ if $t_k = a$. Each s can have at most one $a\text{-}transition$ $\forall a \in \Sigma$.

Let $\Sigma = \{a_1, a_2, \ldots, a_m\}$.
$g(\perp, a_j) = root$ is represented as $g(\perp, (-j, -j)) = root$ for $j = 1, \ldots, m$.

Hence suffix tree $STree(T)$ has two components: The tree itself and the string T.

$STree(T)$ is of linear size in $|T|$.
$\because Q'$ has at most $|T|$ leaves (≤ 1 leaf for each nonempty suffix)
$\Rightarrow Q'$ has to contain at most $|T| - 1$ branching states (when $|T| > 1$).

$\therefore$ There can be at most $2|T| - 2$ transitions between the states in Q' , each taking a constant space because of using pointers instead of an explicit string.
$\Rightarrow$ In implementation, g' can take $\mathcal{O}(|T|)$ space

Construction of Suffix Tree

Suffix function f'

Let $B \subset Q$ be the set of branching states in $STrie(T)$

$$\forall \bar{x} \in B, \\ \begin{cases} f'(\bar{x}) = \bar{y} & \text{if } \bar{x} \neq root, \text{then } x = ay, a \in \Sigma, \bar{y} \in B \\ f'(root) = \perp \end{cases}$$

f' is well-defined $\because \bar{x} \in B \Rightarrow f'(\bar{x}) \in B$.

These suffix links are explicitly represented. Implicit suffix links are helpful but they are imaginary.

Construction of Suffix Tree

Suffix Tree

Denote suffix tree of T as $STree(T) = (Q' \cup \{\perp\}, root, g', f')$

Construction of Suffix Tree

Reference pair

$r = (s, w)$

Refer to an explicit or implicit state r of a suffix tree by a reference pair (s, w) where

s is some explicit state that is an ancestor of r and

w is the string spelled out by the transitions from s to r in the corresponding suffix trie

Canonical reference pair

A reference pair is canonical if s is the closest ancestor of r (and hence, w is shortest possible)

If r is explicit, canonical reference pair of $r = (r, \epsilon)$

Again, represent string w as a pair (k, p) of pointers s.t. $t_k \cdots t_p = w$.
Then, reference pair (s, w) gets form $(s, (k, p))$.
$(s, \epsilon) = (s, (p + 1, p))$

Caution

No contraints on k and p as long as w spells the string

It is technically convenient to omit the final states in the definition of a suffix tree.
When final states are necessary, either

- add a symbol $\#$ which doesn't occurs in T at the end of T or
- traverse from leaf $\bar{T}$ to $root$ and make all the states on the path explicit

In many applications of $STree(T)$, the start location of each suffix is stored with the corresponding state. Such an augmented tree can be used as an index for finding any substring of T.

The algorithm for constructing $STree(T)$ will be patterned after Algorithm 1.
Now, we make precise what Algorithm 1 does.

Let $s_1 = \overline{t_1 \cdots t_{i-1}}, s_2 = \overline{t_2 \cdots t_{i-1}}, s_3, \ldots, s_i = root, s_{i+1} = \perp$ be the states of $STrie(T^{t-1})$ on the boundary path.

Let j be the smallest index s.t. s_j is not a leaf, and
let j' be the smallest index s.t. $s_{j'}$ has a t_i-$transition$.

As s_1 is a leaf and $\perp$ is a non-leaf that has a t_i-$transition$, both j and j' are well-defined and $j \leq j'$

Lemma 1

Algorithm 1 adds to $STrie(T^{i-1})$ a t_i-$transition$ $\forall s_h, 1 \leq h < j'$, s.t.

- for $1 \leq h < j$,
  the new transition expands an old branch of the trie that ends at leaf s_h,
- and for $j \leq h < j'$, the new transition initiates a new branch from s_h.

Algorithm 1 does not create any other transitions

Construction of Suffix Tree

Active point

s_j is the active point of $STrie(T^{i-1})$

End point

$s_{j'}$ is the end point of $STrie(T^{i-1})$

These states are present, explicitly or implicitly, in $STree(T^{i-1})$

Lemma 1 says
1 Leaf states on the boundary path before the active point s_j get a transition that expands an existing branch of the trie.
2 Non-leaf states from the active point s_j to end point $s_{j'}$ ($s_{j'}$ excluded) get a transition that initiates a new branch

Interpret in terms of suffix tree $STree(T^{i-1})$.

Transitions from 1 that expand an existing branch is implemented by updating the right pointer of each transition that represents the branch.

Let $g'(s,(k,i-1)) = r$ be such a transition.
The right pointer has to point to the last position $i-1$ of T^{i-1}.
$\because r$ is a leaf $\Rightarrow$ a path leading to r has to spell out a suffix of T^{i-1} that does not occur elsewhere in T^{i-1}.
$\therefore$ Updated transition is $g'(s,(k,i)) = r$.

This only makes the string spelled out by the transition longer but does not change the states s and r. Making all such updates would take too much time. We use a trick for this.

Construction of Suffix Tree

Open transition

Any transition of $STree(T^{i-1})$ leading to a leaf

Open transitions are represented as $g'(s,(k,\infty)) = r$
Symbols ∞ can be replaced by $n = |T|$ after completing $STree(T)$

This way, transitions from $\textcircled{1}$ is automatically done.

For transitions from 2,
we need to find all $s_h,\ j \leq h < j'$, but s_h might not be explicit.

Let $h = j$ and let (s, w) be the canonical reference pair for s_h (the active point).
s_h is on the boundary path of $STrie(T^{i-1})$
$\Rightarrow w$ is a suffix of T^{i-1}
$\Rightarrow (s, w) = (s, (k, i-1))$ for some $k \leq i$

We need to create a new branch starting from the state $(s, (k, i-1))$.

First, if $(s, (k, i-1))$ is the end point, then done.
Otherwise, $s_h = (s, (k, i-1))$ has to be explicit in order to create a new branch from there.

If s_h is not explicit, create the explicit state s_h by splitting the transition that contains the corresponding implicit state.
After that, a t_i-$transition$ from s_h is created which is $g'(s_h, (i, \infty)) = s_{h'}$ where $s_{h'}$ is a new leaf.
Moreover, suffix link $f'(s_h)$ is added if s_h is created by splitting a transition.

Construction of Suffix Tree

Next the construction proceeds to s_{h+1}.

Reference pair for $s_h = (s, (k, i-1))$
$\Rightarrow$ canonical reference pair for $s_{h+1} = canonize(f'(s), (k, i-1))$
where $canonize$ makes the pair canonical by updating the state and the left pointer.

Repeat until $s_{j'}$ is found.

Construction of Suffix Tree

Procedure update(s, (k, i))

$(s, (k, i-1))$ the canonical reference pair for the active point;

$oldr \leftarrow root$; $(end\text{-}point, r) \leftarrow test\text{-}and\text{-}split(s, (k, i-1)), t_i)$;

while not $(end\text{-}point)$ **do**

    create new transition $g'(r, (i, \infty)) = r'$ where r' is a new state;

    if $oldr \neq root$ **then** create new suffix link $f'(oldr) = r'$;

    $oldr \leftarrow r$;

    $(s, k) \leftarrow canonize(f'(s), (k, i-1))$;

    $(end\text{-}point, r) \leftarrow test\text{-}and\text{-}split(s, (k, i-1), t_i)$;

if $oldr \neq root$ **then** create new suffix link $f'(oldr) = s'$;

return (s, k).

References

Original Paper:
https://www.cs.helsinki.fi/u/ukkonen/SuffixT1withFigs.pdf

Wikipedia for the definition of Trie