Bioinformatics Algorithms

(Fundamental Algorithms, module 2)

Zsuzsanna Lipták

Masters in Medical Bioinformatics
academic year 2018/19, II. semester

Suffix Trees 2

Pattern matching with the suffix tree

Recall: Pattern matching

Pattern matching

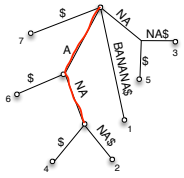
Given a string T of length n (the text), and a string P of length m (the pattern), find all occurrences of P as substring of T.

Variants:

- all-occurrences version: output all occurrences of P in T
- decision version: decide whether P occurs in T (yes - no)
- counting version: output occ_P, the number of occurrences of P in T

Pattern matching with suffix tree

Let text T = BANANA and pattern P = ANA. We try to match the pattern starting from the root and following the labels on the edges; when we encounter a node, we have at most one possible edge which to follow[1]:

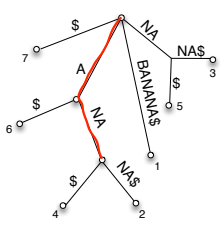


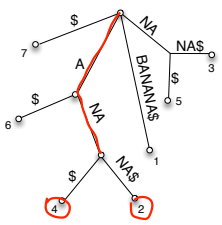
Since we have matched all of the pattern, we now know that P = ANA occurs in T (decision).

[1] recall that every outgoing edge from an inner node starts with a different character

Pattern matching with suffix tree

Moreover, the occurrences of P are exactly the numbers of the leaves in the subtree below $locus(P)$ (the position where we finished matching P).

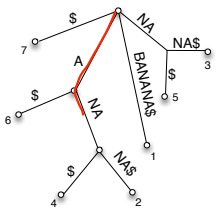


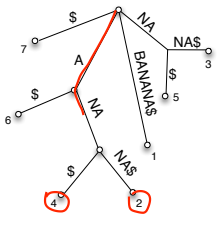


Why is this? Because P occurs in position i iff P is a prefix of Suf_i. As we have seen, the path from the root to leaf number i spells exactly Suf_i.

Pattern matching with suffix tree

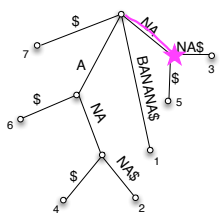
We may end in the middle of an edge, as for P = AN. Still the occurrences of P are the leaves in the subtree rooted in u, where $locus(P) = (u, d)$.

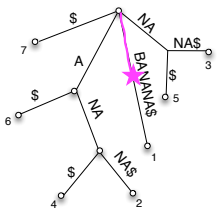




Pattern matching with suffix tree

The matching could also be unsuccessful, as for P = NAB or P = BAD:





Pattern matching with suffix tree: Analysis

- Time for decision is O(m) (at most one comparison per position of P).
- Time for finding all occurrences: $O(m + occ_P)$.
  Let $locus(P) = (u, d)$: traverse the subtree rooted in u, this takes time linear in the size of the subtree, which is $O(occ_P)$, thus altogether $O(m + occ_P)$.
  (Proof for size of subtree: Number of leaves of subtree $= occ_P \Rightarrow$ number of inner nodes $< occ_P$ (since all inner nodes branching) $\Rightarrow$ total number of nodes $< 2occ_P \Rightarrow$ number of edges $< 2occ_P - 1 \Rightarrow$ size of subtree $< 4occ_P$.)
- Time for counting: with same algorithm: $O(m + occ_P)$.
  Can be improved to $O(m)$ with linear-time preprocessing of ST (store in each node u the number of leaves in subtree rooted in u).

Note that all these times are independent of the size n of the text.

Suffix tree construction

Construction of suffix trees

Theorem:
$ST(T)$ can be constructed in $O(n)$ time.

Several linear time algorithms exist (beyond the scope of this course). We will see two simple quadratic-time construction algorithms.

Simple ST construction algorithm 1

Simple suffix insertion algorithm

1. start with tree $\mathcal{T}$ with one node (the root)
2. for $i = 1, \ldots, n+1$: insert Suf_i into $\mathcal{T}$

Insert string S into $\mathcal{T}$

1. $\ell \leftarrow |S|$
2. start matching S (as for pattern matching) in $\mathcal{T}$, starting from root
3. at first mismatch j in S:
   - if currently in node u, add new child v to u
   - otherwise, create new node u at current locus with new child v
4. add edge label $L(u, v) = S_j \ldots S_\ell$

Note that there is always a mismatch, because no suffix is the prefix of another suffix (that's why we chose \$ as a new character!)

Simple ST construction algorithm 2

Another simple algorithm is the following recursive algorithm (Giegerich & Kurtz, 1995):

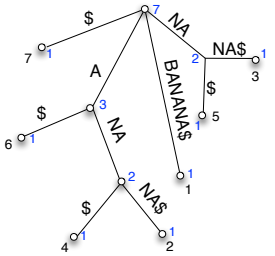
WOTD algorithm (write-only, top-down)

1. Let X be the set of all suffixes of $T\$$.
2. Sort the suffixes in T according to their first character;
   for $c \in \Sigma \cup \{\$\}$: X_c = suffixes starting with character c.
3. For each group X_c:
   (i) if X_c is a singleton, create a leaf;
   (ii) otherwise, find the longest common prefix of the suffixes in X_c, create an internal node, and recursively continue with Step 2, X being the set of remaining suffixes from X_c after splitting off the longest common prefix.

N.B.: Both of these algorithms have worst-case running time $O(n^2)$ (without proof).

Storing addition information in the suffix tree

Recall the pattern matching problem, counting variant: Return the number of occurrences of pattern P. Let $g(u)$ = number of leaves in subtree rooted in u.



If we store $g(u)$ in u, then we can solve the counting problem in $O(m)$ time: match P in ST, if found in $locus(P) = (u, d)$, then return $g(u)$. E.g. the number of occurrences of $P =$ AN is 2, as can be seen immediately in ST.

Postorder traversal of ST

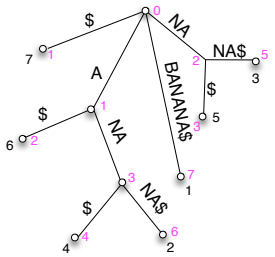
Note that the number of leaves in subtree rooted in u, where u has children $v_1, \ldots, v_k$, equals the sum of the leaves in the subtrees of the v_i.

Compute the number of leaves in subtree, $g(u)$, via post-order traversal of the ST (bottom-up):

1. if u leaf: $g(u) \leftarrow 1$
2. if u inner node: $g(u) = \sum_{v \text{ child of } u} g(v)$

This takes linear time in the size of the tree, i.e. $O(n)$ time. Moreover, the information stored is constant per node, so the space needed for the ST is still $O(n)$.

Another piece of information we often need is the stringdepth $sd(u)$ of a node u (the length of its label).



Preorder traversal of ST

Note that the stringdepth of a node u with parent v equals the stringdepth of v plus the length of the label of the edge connecting v and u.

Compute the stringdepth of a node, $sd(u)$, via pre-order traversal of the ST (top-down):

1. for the root: $sd(root) = 0$
2. for all other nodes u: Let $v = parent(u)$.
   Then $sd(u) = sd(v) + |L(v, u)|$.

Again, this takes linear time $O(n)$ and total space $O(n)$ (since we store constant amount per node).

Summary

- The suffix tree is an extremely versatile data structure for solving problems on strings/sequences.
- It takes linear storage space in the size of the text $O(n)$. (Remember: edge labels are stored as two pointers into T.)
- It can be constructed in linear time $O(n)$ (not studied in this course).
- Leaves of ST correspond to suffixes of T.
- Loci (inner nodes or "positions on edges") corr. to substrings of T.
- Leaves in subtree rooted in u correspond to occurrences of substrings whose locus is on edge leading to u.
- The ST can be used to solve pattern matching queries in time independent of the text size: $O(m)$ for decision, $O(m + occ_P)$ for all-occurrences, $O(m)$ for counting (after linear time preproc.)
- The ST can be used to solve many many other types of queries on strings efficiently.