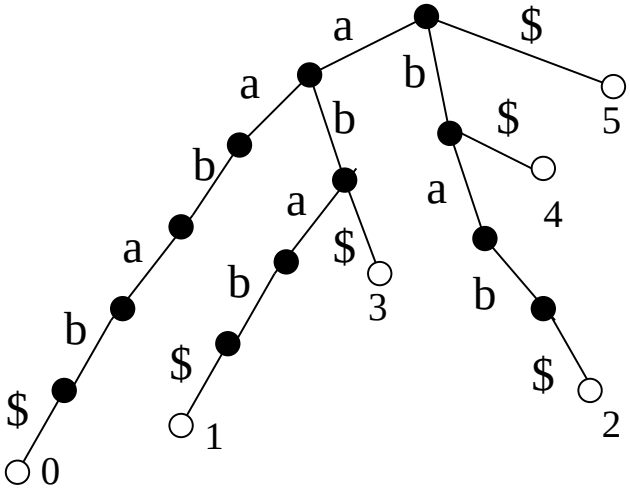
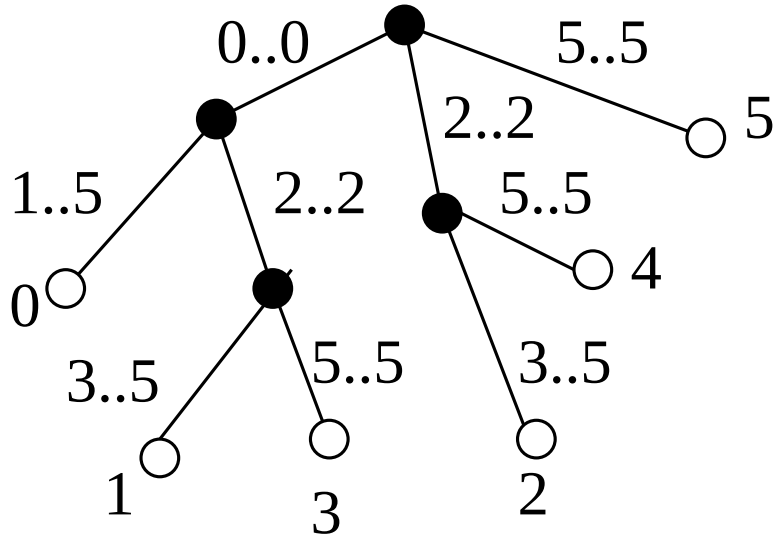
Suffix tree

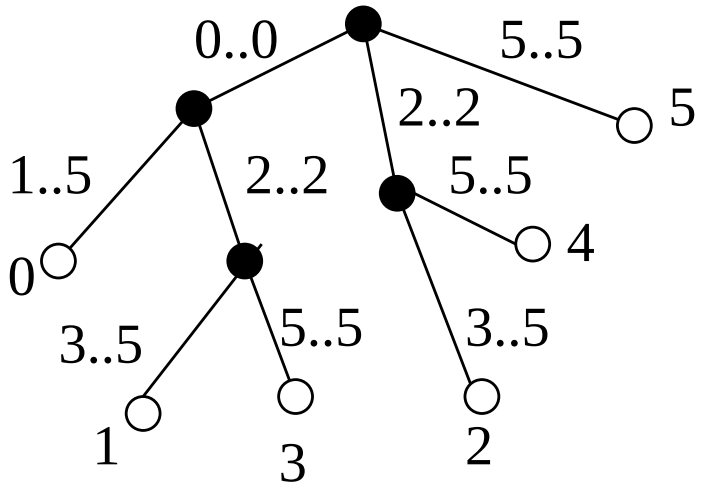
Trie of all suffixes of a string, e.g. T =aabab$

Compact all non-branching paths — get a tree with $O(n)$ nodes





Suffix tree



Each node:

– pointer to parent

– indices of substring for edge to parent

– suffix start (in a leaf)

– pointers to children (in an internal node)

– other data, e.g. string depth

$O(n)$ nodes, construct in $O(n)$ time for constant σ

Applications of suffix trees

- String matching: preprocess text,
  then process each pattern in $O(m + k)$
- Find longest substring with at least two occurrences in S in $O(n)$
  – internal node with highest “string depth”

Today more applications

Maximal repeats

Maximal pair in S is a pair of substrings $S[i..i+k]$ and $S[j..j+k]$
such that $S[i..i+k] = S[j..j+k]$,
but $S[i-1] \neq S[j-1]$ and $S[i+k+1] \neq S[j+k+1]$

Maximal repeat is a string which is in at least one maximal pair.

Goal: find all maximal repeats in S in $O(n)$ time

Use: poor man's approximate string matching

(e.g. for plagiarism detection)

Applications of suffix trees

- String matching: preprocess text,
  then process each pattern in $O(m + k)$
- Find longest substring with at least two occurrences in S in $O(n)$
  – internal node with highest “string depth”
- Find all maximal repeats in S

Still to do:

Find longest substring occurring in at least two different strings in $\{S_1, \ldots, S_z\}$ in $O(n)$

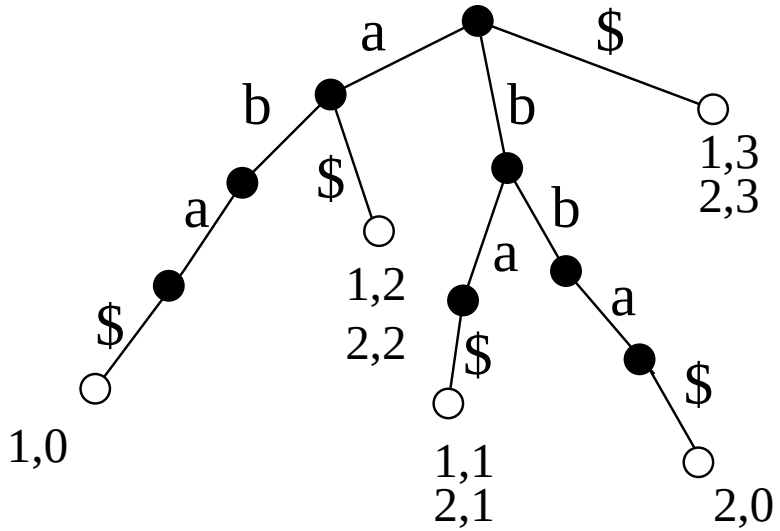
Generalized suffix tree

Store suffixes of several strings $\{S_1, \ldots, S_z\}$

Each leaf a list of suffixes

Each edge i and indices to some S_i

Trie of all suffixes for $S_1 = abba\$$, $S_2 = bbba\$$:



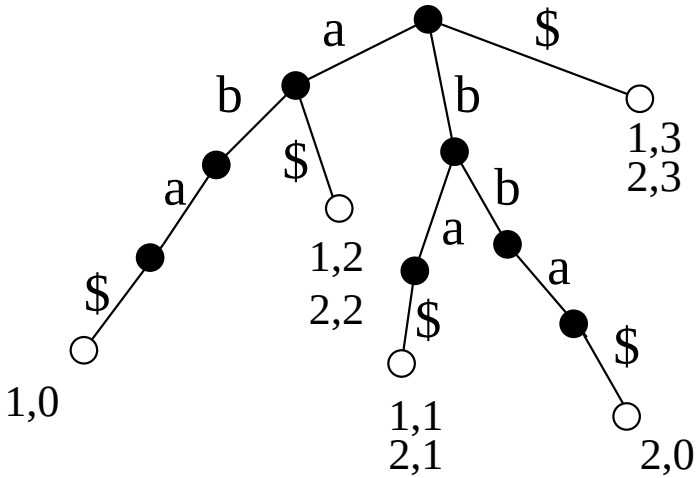
Generalized suffix tree

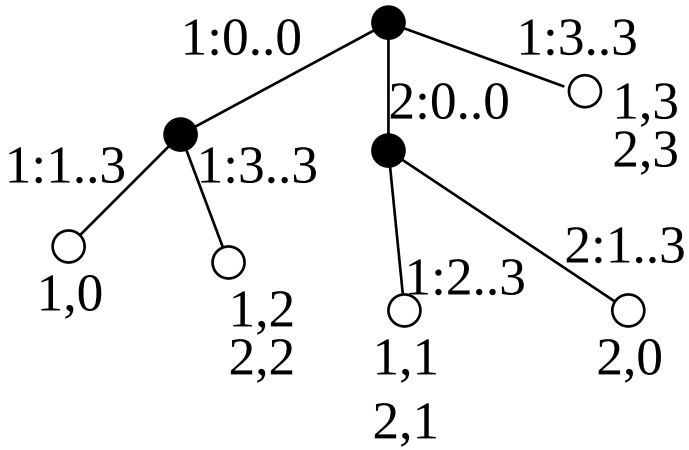
Store suffixes of several strings $\{S_1, \ldots, S_z\}$

Each leaf a list of suffixes

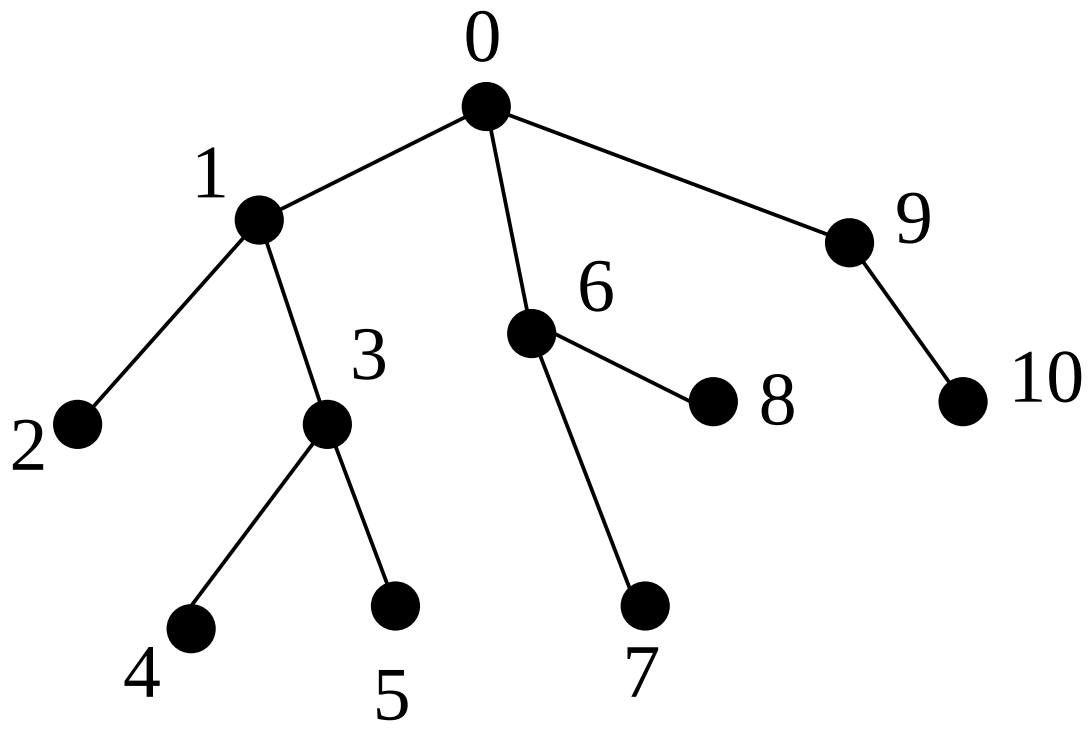
Each edge i and indices to some S_i

Example: $S_1 = abca\$$, $S_2 = bbca\$$:





Lowest common ancestor (LCA), najnižší spoločný predok

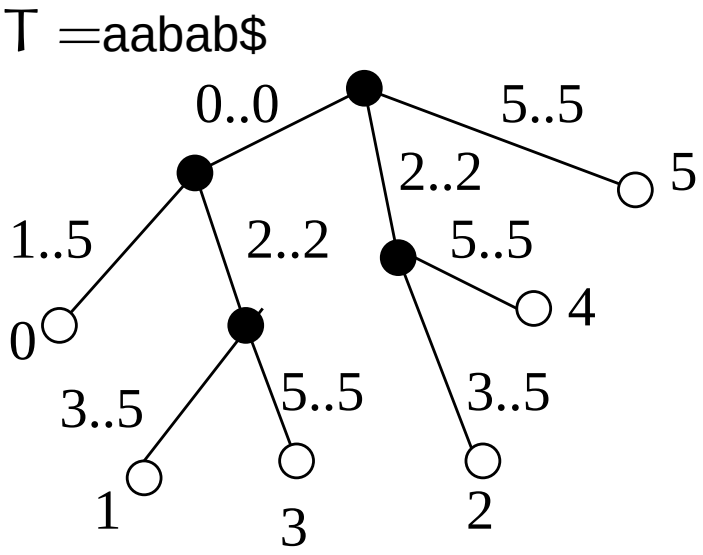


v is ancestor of u if it is on the path from u to the root

lca(u, v): node of greatest depth in ancestors$(u) \cap$ ancestors(v)

Next time: preprocess tree T in $O(n)$, answer lca(u, v) in $O(1)$

Lowest common ancestor in suffix trees



Consider leafs i, j

lca(i, j): longest common prefix of $T[i..n-1]$ and $T[j..n-1]$

String-depth of lca(i, j) gives the length of this prefix

Can be computed in $O(1)$ after $O(n)$ preprocessing for lca

Approximate string matching

Hamming distance $d_H(S_1, S_2)$ between two strings of equal length:

the number of positions where they differ

Task: find approximate occurrences of P in T with Hamming distance $\leq k$

$\{i \mid d_H(P, T[i..i+m-1]) \leq k\}$

Build generalized suffix tree for P and T in $O(n+m)$

Preprocess for LCA queries in $O(n+m)$

LCA for leaves $T[i..n-1]$ and $P[j..m-1]$ in suffix tree

gives longest common prefix of these two suffixes in $O(1)$

Approximate string matching

```
1   for(int i=0; i<=n—m; i++) {
2     j = 0; err = 0;
3     while(j < m) {
4       q = longest common prefix of T[i+j..n—1], P[j..m—1]
5       if(j+q < m) {   //P[j+q] != T[i+j+q]
6         err++;
7         if(err > k) { break; }
8       }
9       j += q+1;
10    }
11    if(err <= k) { print i; }
12  }
```

String matching with wildcards

Special character * matches any character from Σ

E.g. aa*b matches aaab, aabb, aacb,...

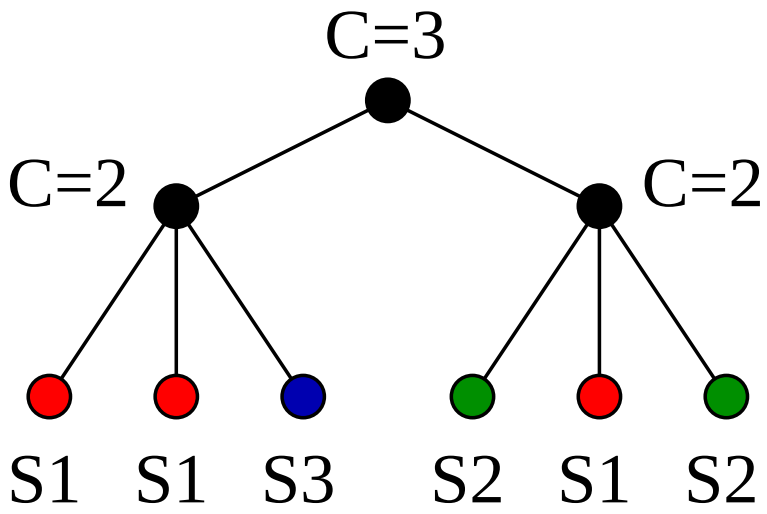
- Trivial algorithm $O(nm)$
- Shift-and $O(n + m + \sigma)$ from small m, BNDM good in average case
- Algorithm using FFT $O(n \log m)$
- Suffix trees $O(nk)$ where k is the number of wildcards

  How?

Counting documents

Generalized suffix tree of $\{S_1, \ldots, S_z\}$

For each node $C(v)$: how many different S_i in its subtree



Use:

– find longest string which is a substring of each S_i

– how many S_i contain pattern P?

Trivial: $O(nz)$; better: $O(n)$ using LCA

Optional homework

Given string S and $k > 1$. For each i find the longest prefix of $S[i..n-1]$ that occurs at least k times in S.

Goal: $O(n)$ time in total

Example:

```
    a a b a b b a a b a a
k=2 4 3 2 2 1 3 4 3 3 2 1
k=3 2 2 2 2 1 2 2 2 2 2 1
k=5 1 1 0 1 0 0 1 1 0 1 1
```