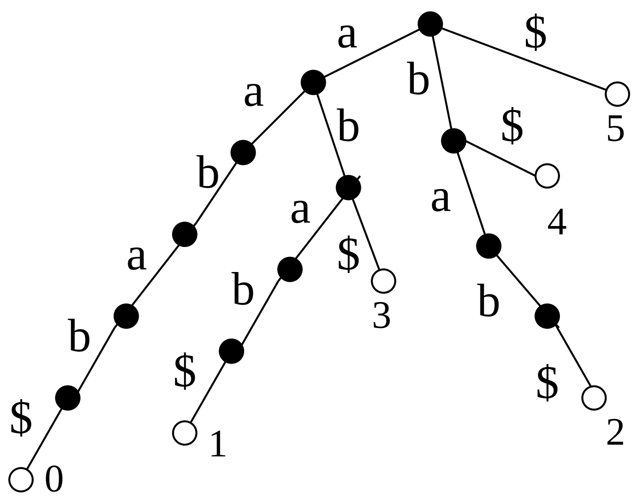
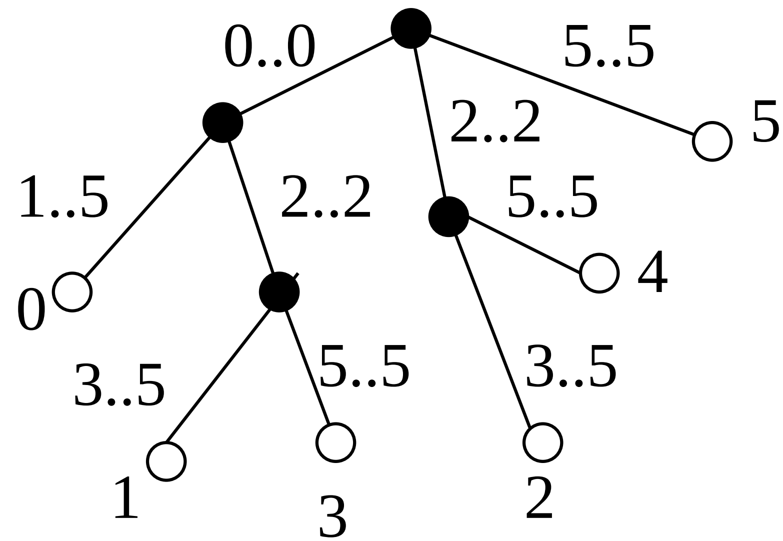
Suffix tree

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

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





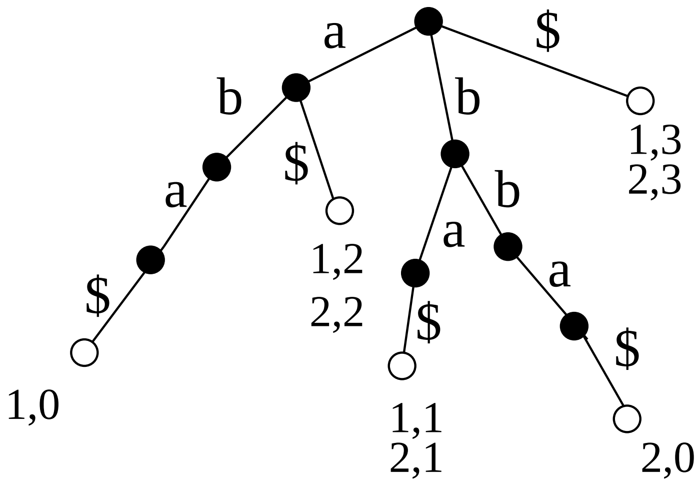
Generalized suffix tree

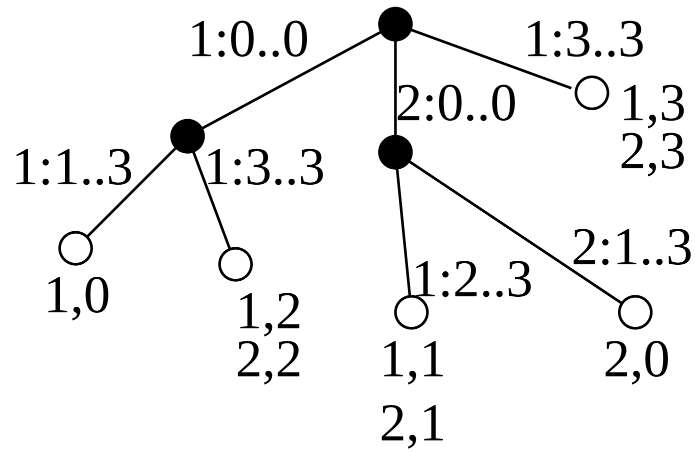
Store suffixes of several strings $\{S_1, \ldots, S_z\}$

Each leaf a list of suffixes

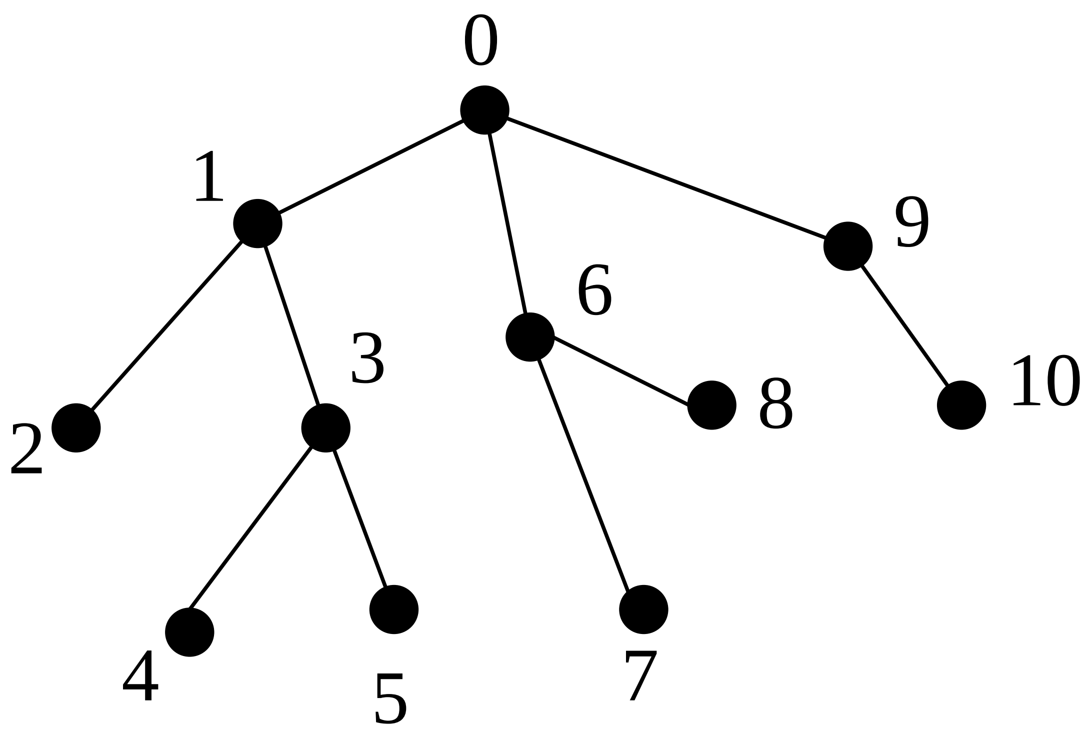
Each edge i and indices to some S_i

Example: $S_1 = aba\$$, $S_2 = bba\$$:





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 $O(1)$

Applications of suffix trees

– Index text for string matching

– Find longest substring with at least two occurrences

– Find longest words which occurs in at least 2 documents

– Find all maximal repeats

With LCA

– Find maximal palindromes

– Find approximate matches under Hamming distance

– Find pattern with wildcards

– Count in how many documents word occurs

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:

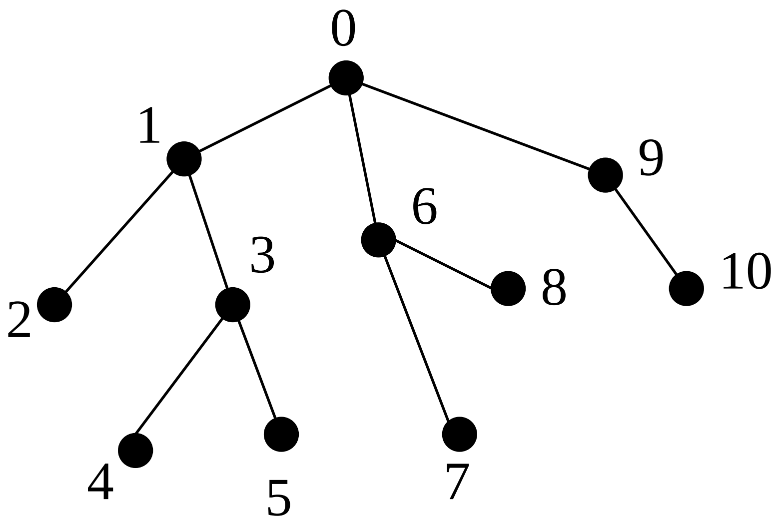
	a	a	b	a	b	b	a	a	b	a	a
k=2	4	3	2	2	1	3	4	3	3	2	1
k=3	2	2	2	2	1	2	2	2	2	2	1
k=5	1	1	0	1	0	0	1	1	0	1	1

Assume for each node we know its string depth $d(u)$ and the number of leaves in its subtree $l(u)$

Optional homework

```
search(root, 0, k);
void search(node v, int value, int k) {
    if(??) {
      value = ??;
    }
    if(v.is_leaf()) { a[v.id] = value; }
    else {
      foreach child w of v {
        search(w, value, k);
      }
    }
}
```

Lowest common ancestor (LCA)



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

Harel and Tarjan 1984, Schieber a Vishkin 1988 (Gusfield book, notes),

Bender and Farach-Colton 2000 (this lecture)

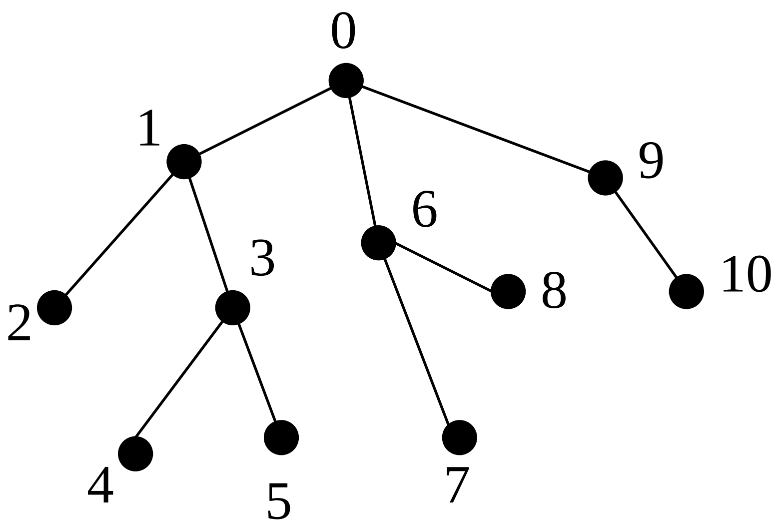
Trivial solutions:

– no preprocessing, $O(n)$ time per lca

– $O(n^3)$ preprocessing, $O(n^2)$ memory, $O(1)$ time per lca

Lowest common ancestor (LCA)

Preprocess tree to arrays V, D, R



V – visited nodes

D – their depths

R – first occurrence of node in V

i:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
V:	0	1	2	1	3	4	3	5	3	1	0	6	7	6	8	6	0	9	10	9	0
D:	0	1	2	1	2	3	2	3	2	1	0	1	2	1	2	1	0	1	2	1	0
R:	0	1	2	4	5	7	11	12	14	17	18										

Lowest common ancestor (LCA)

```
search(root, 0);
void search(node v, int depth) {
   R[v] = V.size;
   V.push_back(v);
   D.push_back(depth);
   foreach child u of v {
        search(u, depth+1);
        V.push_back(v);
        D.push_back(depth);
   }
}
```

LCA algorithm overview

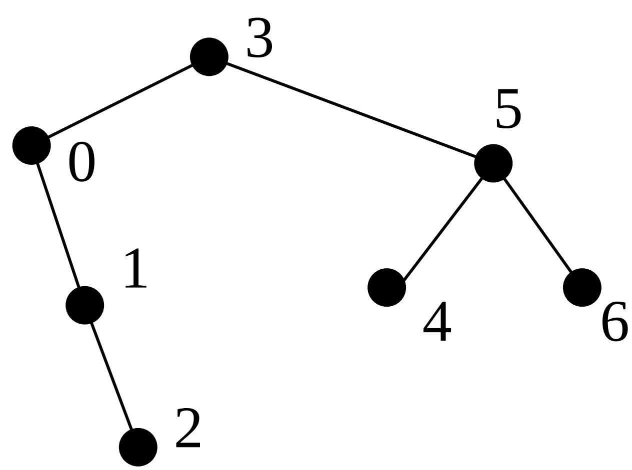
- Compute arrays V,D,R by depth-first search in the tree
- Enumerate all possible $+1, -1$ blocks of length $m-1$, precompute answers for all intervals in each type
- Split D into blocks of length $m = \log_2(n)/2$, precompute minimum and its index in each block (A', M'), find type of each block
- Precompute $O(n' \log n')$ data structure for RMQ in A'
- For lca(u, v) a query:
  $i = R[u], j = R[v]$, find position k of minimum in $D[i..j]$ as follows:
  – find block b_i containing i, block b_j containing j
  – compute minimum in $b_i \cap [i, j]$, $b_j \cap [i, j]$
  – compute minimum in $A'[b_{i+1} \dots b_{j-1}]$
  – find minimum of three numbers, let k be its index in D
  return $V[k]$

RMQ using LCA

Cartesian tree for A: root: minimum in A (at position k)

left subtree: recursively for $A[1..k-1]$

right subtree: recursively for $A[k+1..n]$



i	0	1	2	3	4	5	6
$A[i]$	1	2	4	0	5	3	6

$A \longrightarrow$ Cartesian tree in $O(n)$: add elements from left to right

min $A[i..j] =$ lca(i, j)

RMQ using LCA

Use auxiliary value $a[-1] = -\infty$

```
root = new node(-1, null);
r = root;
for(int i=0; i<n; i++) {
   while(a[r.id]>a[i]) {
      r = r.parent;
   }
   v = new node(i, r);
   v.left = r.right;
   r.right = v;
   r = v;
}
```