2-INF-237 Vybrané partie z dátových štruktúr

2-INF-237 Selected Topics in Data Structures

- Instructor: Broňa Brejová
- E-mail: brejova@fmph.uniba.sk
- Office: M163

- Course webpage: http://compbio.fmph.uniba.sk/vyuka/vpds/

String matching (vyhľadávanie vzorky v texte)

Given: pattern (vzorka) P of length m, text T of length n.

Goal: Find all positions $\{i_1, i_2, \ldots\}$ such that $T[i_j..i_j + m - 1] = P$.

Example:

Input: $P =$ma, $T =$Ema ma mamu

Output: 1, 4, 7

Input: $P =$"a ma", $T =$Ema ma mamu

Output: 2, 5

Trivial algorithm

```
for(i=0; i<=n-m; i++) {
   j=0;
   while(j<m && P[j]==T[i+j]) {  // (*)
      j++;
   }
   if(j==m) {
      print(i);
   }
}
```

String matching (vyhľadávanie vzorky v texte)

- Trivial algorithm $O(nm)$
- Knuth-Morris-Pratt algorithm $O(n+m)$ (later)
- What if we want to preprocess T, then search in $O(m+k)$?
  $k =$ the number of occurrences of P in T

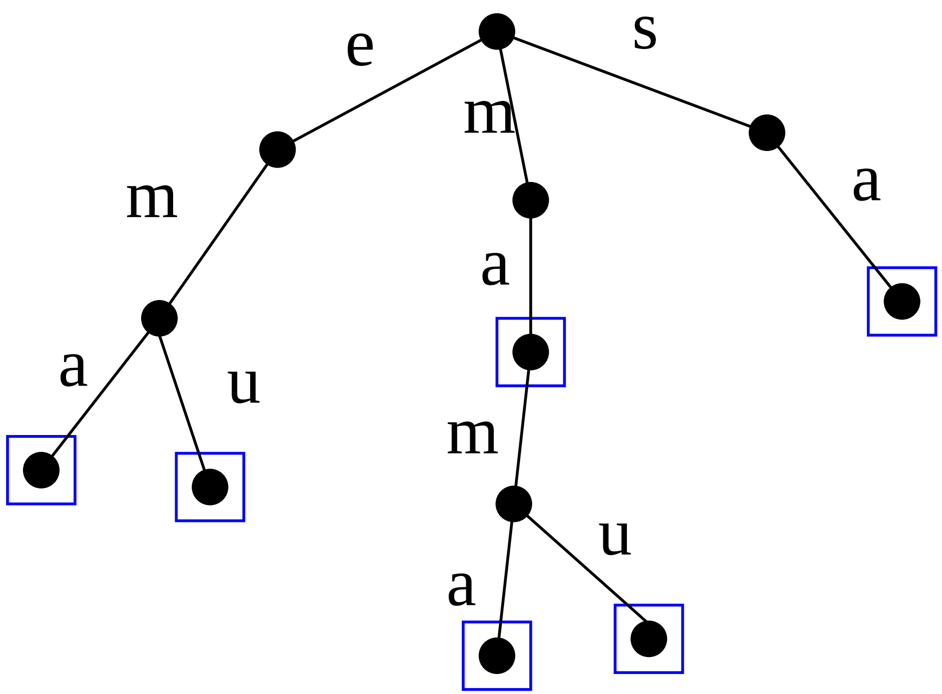
What about the following problems?

Given a set of strings $\mathcal{S} = \{S_1, \ldots, S_z\}$:

- Find the longest string w which is a prefix of at least two strings in $\mathcal{S}$
- Find the longest string w which is a substring of at least two strings in $\mathcal{S}$
- Simpler: Find the longest string w which occurs at least twice in a string T

The longest string which is a prefix of at least two strings in $\mathcal{S}$

Example: $\mathcal{S} = \{$ema, ma, mamu, mama, emu$\}$

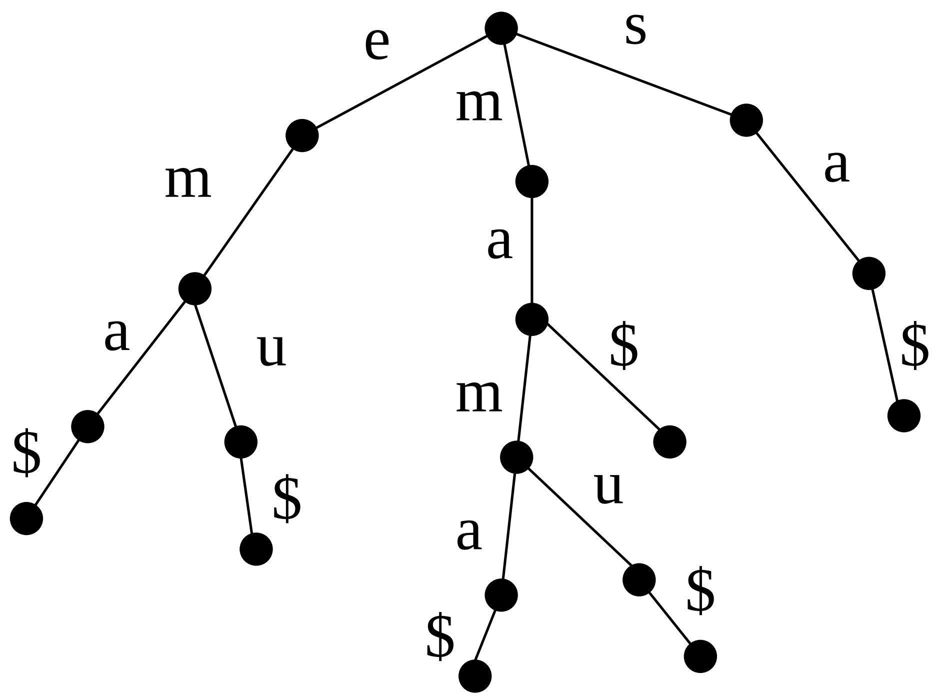


The longest string which is a prefix of at least two strings in $\mathcal{S}$

Simplification: add \$ at the end of each string

Strings in S correspond exactly to leaves

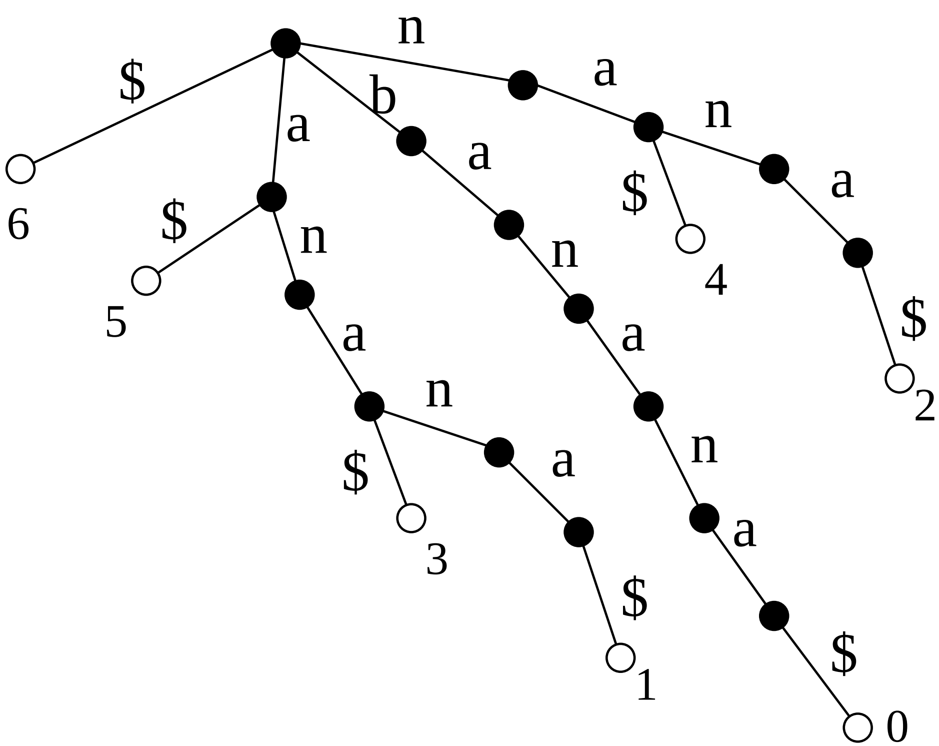
Example: $\mathcal{S} = \{$ema\$, ma\$, mamu\$, mama\$, emu\$$\}$



Trie of all suffixes of a string

Example:

i	0	1	2	3	4	5	6
$T[i]$	b	a	n	a	n	a	$

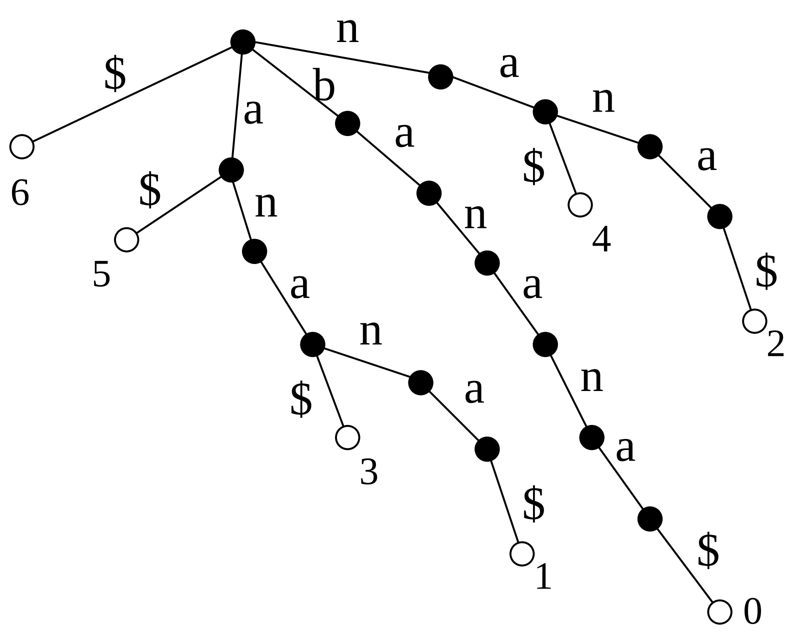


Nodes correspond to substrings of T

Problem: the number of nodes is $O(n^2)$

Optional homework

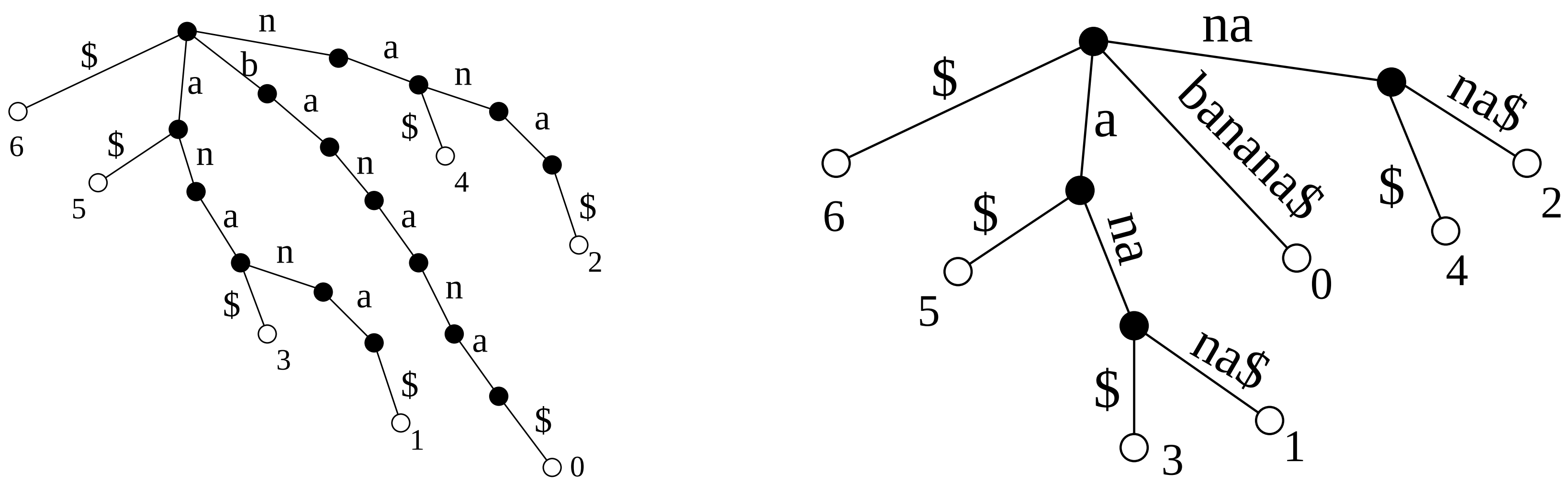
Find a string over binary alphabet which results in a big trie of suffixes



Suffix tree (sufixový strom)

Compact all non-branching paths

T =banana$



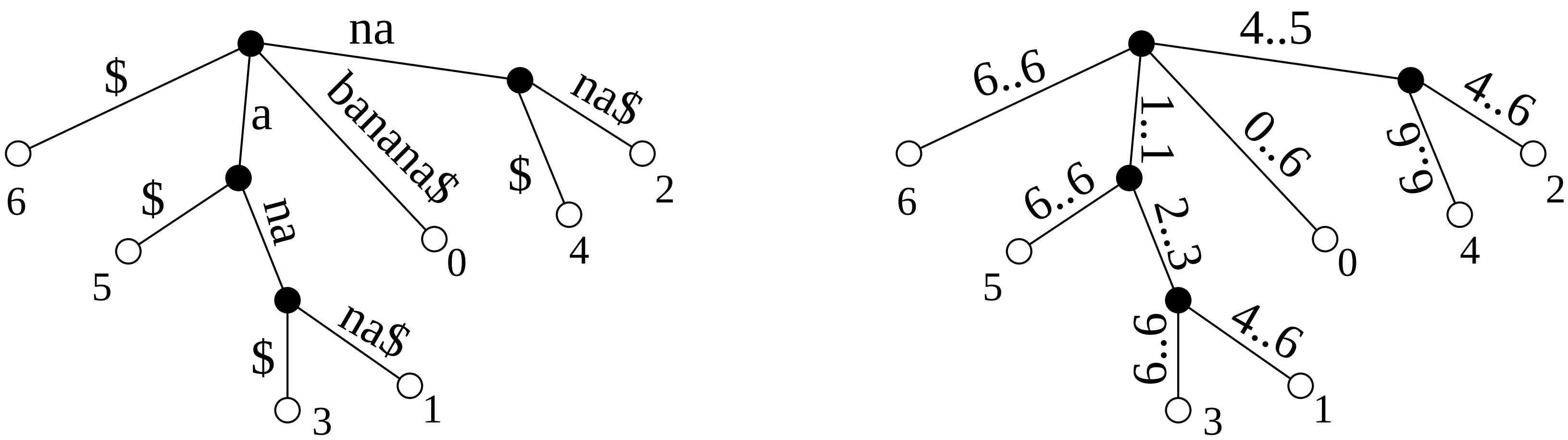
The number of leaves is n

The number of internal nodes is at most $n-1$

Suffix tree

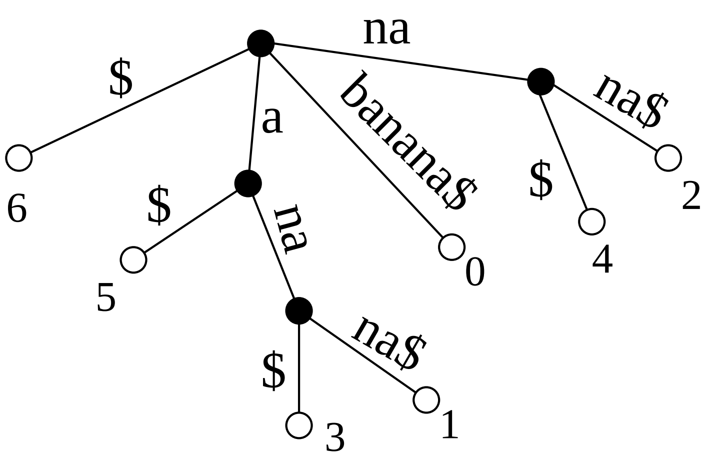
Store indices to T instead of substrings

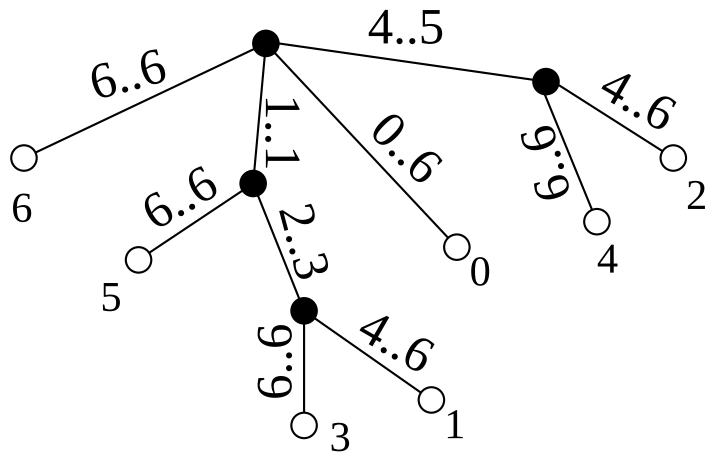
i	0	1	2	3	4	5	6
$T[i]$	b	a	n	a	n	a	$



Edges from one node start with different characters.

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

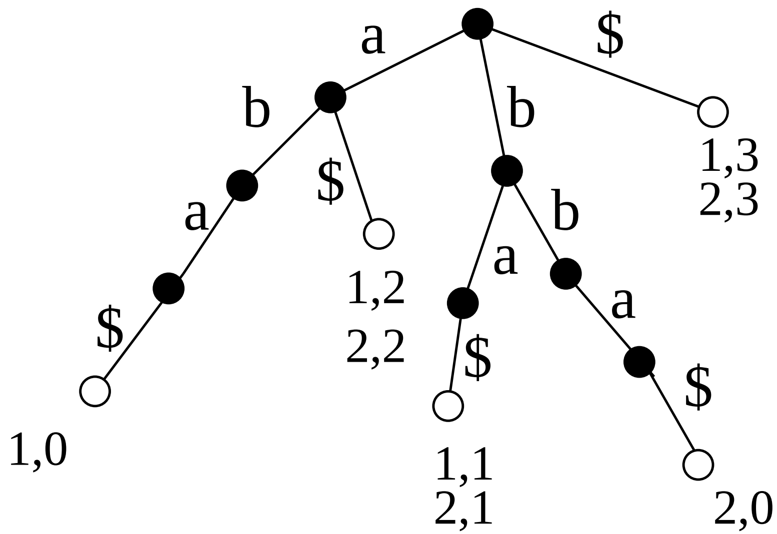
Generalized suffix tree

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



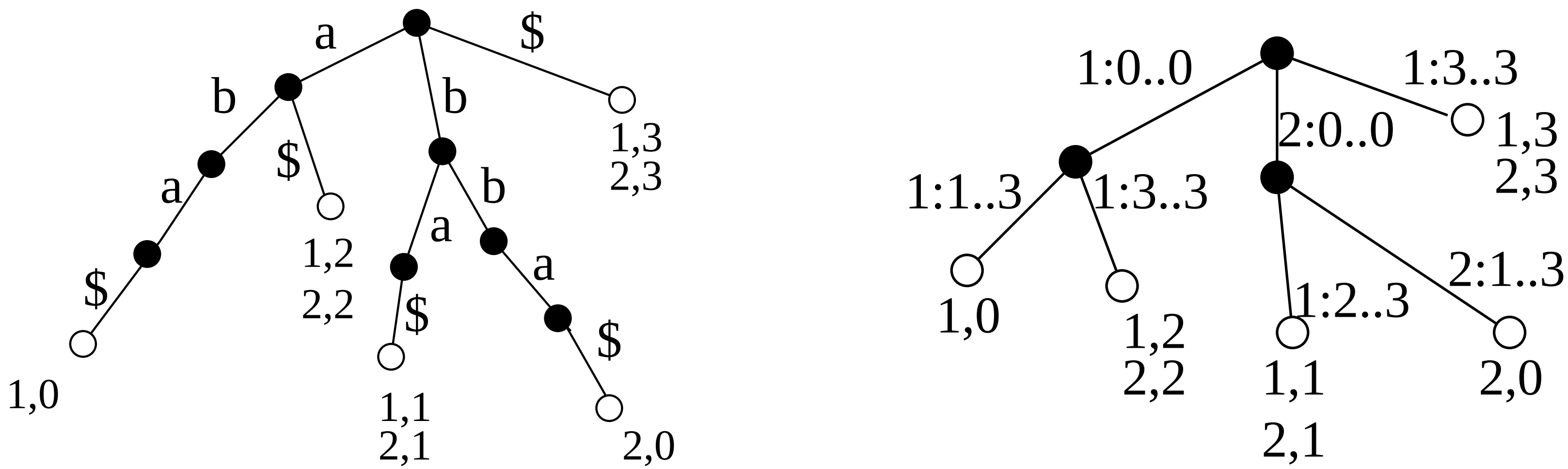
Generalized suffix tree

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



Alternatives:

ordinary suffix tree for $S_1\$_1S_2\$_2\ldots\$_{z-1}S_z\$_z$ or $S_1\$S_2\$\ldots\$S_z\$\#$

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"

Generalized suffix trees

For a set of documents $\{S_1, \ldots, S_z\}$

- Find longest substring occuring in at least two input strings in $O(n)$
  – node with highest "string depth" that has at least two different document labels in its subtree

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)

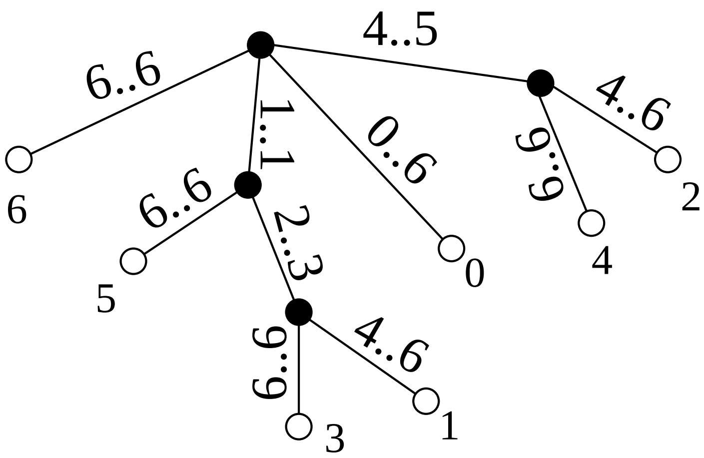
Maximal repeats

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 $S[i+k+1] \neq S[j+k+1]$

- Each maximal repeat corresponds to an internal node. Why?
- Def. If v is a leaf for suffix $S[i..n-1]$, its left character is $l(v) = S[i-1]$.
- Def. We will call a node v diverse (rôznorodý) if its subtree contains leaves labeled by at least 2 different values of $l(v)$.
- Thm. Node v corresponds to a maximal repeat iff v is diverse.

Lowest common ancestor in suffix trees

T =banana$



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 $\leq k$

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

Approximate string matching: Trivial algorithm

```
for(i=0; i<=n-m; i++) {
    j=0; err = 0;
    while(j<m) {
        if(P[j]!=T[i+j]) err++;
        if(err>k) { break; }
        j++;
    }
    if(err<=k) {
        print(i);
    }
}
```

Approximate string matching

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

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

Trivial algorithm $O(nm)$

Algorithm with suffix trees and LCA:

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

```
for(int i=0; i<=n-m; i++) {
  j = 0; err = 0;
  while(j < m) {
    q = longest common prefix of T[i+j..n-1], P[j..m-1]
    if(j+q < m) {   //P[j+q] != T[i+j+q]
      err++;
      if(err > k) { break; }
    }
    j += q+1;
  }
  if(err <= k) { print i; }
}
```

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 $\leq k$

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

Trivial algorithm $O(nm)$

Algorithm with suffix trees and LCA: $O(nk)$ [Landau, Vishkin 1986]

More complex version: $O(n\sqrt{k \log k})$ [Amir, Lewenstein, Porat 2000]

Using fast Fourier transform: $O(n\sigma \log m)$ [Fischer and Paterson 1974]

String matching with wildcards

Special character * matches any character from Σ

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

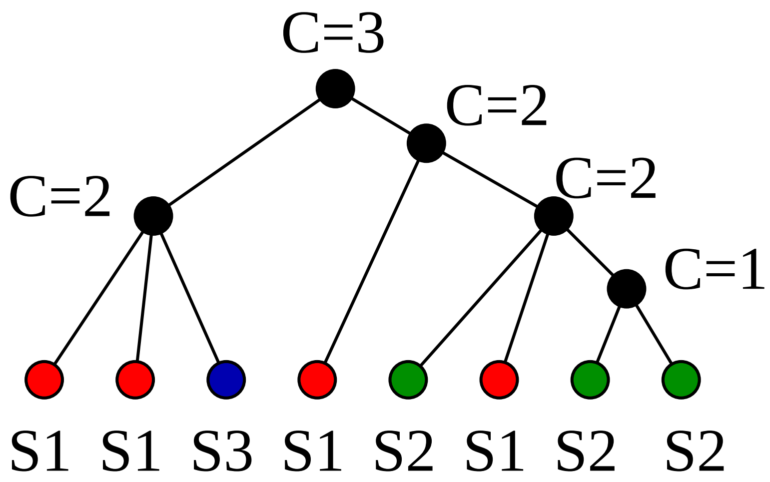
- Trivial algorithm $O(nm)$
- Bit-parallel algorithms for small m: $O(n + m + \sigma)$
- Fast Fourier transform $O(n \log m)$
- Suffix trees $O(nk)$ where k is the number of wildcards

  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

Counting documents

- List of leaves in DFS order
- Separate to sublists: L_i = list of suffixes of S_i in DFS order
- Compute lca for each two consecutive members of each L_i

  In each node counter $h(v)$: how many times found as lca
- Compute in each node

  $\ell(v)$: the number of leaves in subtree

  $s(v)$: sum of $h(v)$ in subtree

  $C(v) = \ell(v) - s(v)$

Counting documents

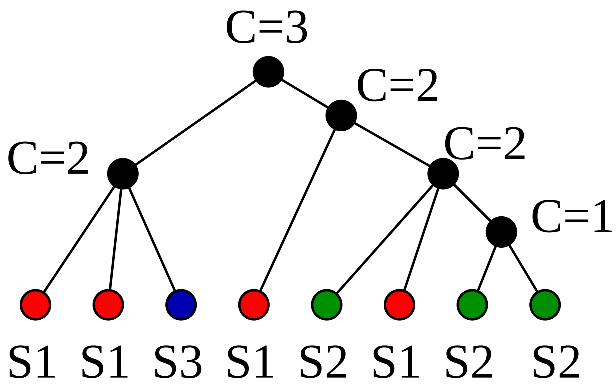
$C(v)$: how many different S_i in its subtree (goal)

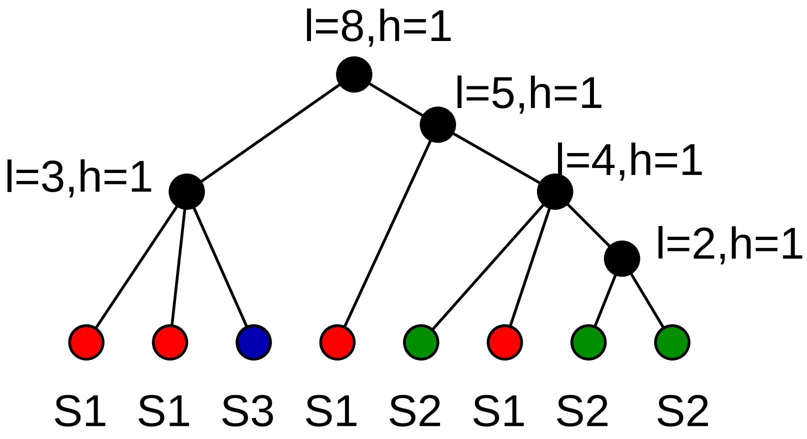
$\ell(v)$: how many leaves in the subtree

$h(v)$: how many times v found as lca

$s(v)$: sum of $h(v)$ in subtree

$$C(v) = \ell(v) - s(v)$$





Finding all small numbers

We have array A precomputed for RMQ.

For given i, j, x find all indices $k \in \{i, \ldots, j\}$ s.t. $A[k] \leq x$.

```
void small(i, j, x) {
   if(j > i) return;
   k = rmq(i, j);
   if(a[k] <= x) {
      print k;
      small(i, k-1, x);
      small(k+1, j, x);
   }
}
```

Running time $O(p)$, where p is the number of printed indices

Printing documents

Preprocess texts $\{S_1, \ldots, S_z\}$

Query: which documents contain pattern P?

We can do $O(m+k)$ where k =number of occurrences of P

Want $O(m+p)$ where p =number of documents containing P

Array of leaves L in DFS order

For leaf $L[i]$ let $A[i]$ be the index of previous leaf from the same S_j

Occurrences of P: subtree of corresponding to interval $[i, j]$ in L

Find all $k \in [i, \ldots, j]$ that have $A[k] < i$

Running time? Preprocessing?

Applications of suffix trees

– Index text for string matching

– Find the longest substring with at least 2 occurrences

– Find the longest string which occurs in at least 2 documents

– Find all maximal repeats

With LCA

– Find approximate matches under Hamming distance

– Find pattern with wildcards

– Count in how many documents a pattern occurs

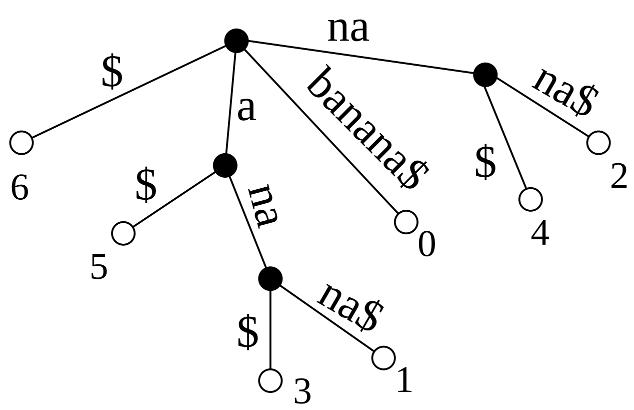
With RMQ

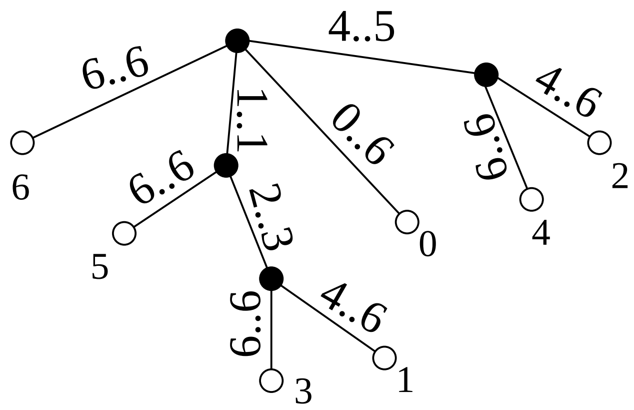
– Print documents containing a pattern

Summary: suffix trees

- Compact representation of all suffixes of a string
- They can be built in $O(n)$ time (proof later)
- They can answer interesting problems related to substring equality
- They need relatively large memory

  (several pointers/integers per character)





Suffix array

Array of suffixes in lexicographic order

(assume $\$ < a \quad \forall a \in \Sigma$)

i	0	1	2	3	4	5	6
$S[i]$	b	a	n	a	n	a	$

i	0	1	2	3	4
$S[i]$	a	a	a	a	$

i	$SA[i]$	Suffix
0	**6**	$
1	**5**	a$
2	**3**	ana$
3	**1**	anana$
4	**0**	banana$
5	**4**	na$
6	**2**	nana$

i	$SA[i]$	Suffix
0	**4**	$
1	**3**	a$
2	**2**	aa$
3	**1**	aaa$
4	**0**	aaaa$

Suffix array

- Array of suffixes in lexicographic order
- Simpler structure, continuous memory
- Less memory: one index per character ($4n$ bytes in total)
- Search for a pattern P by binary search in $O(m \log n)$ time, can be improved to $O(m + \log n)$ with additional memory
- Construction in $O(n)$ even for large alphabets

String matching with suffix arrays

Given suffix array for text T (and possibly other structures), and pattern P, solve these three tasks:

- Task 1: Find out if P occurs in T (yes/no)
- Task 2: Count the number of occurrences of P in T
- Task 3: List all occurrences of P in T

Use binary search in SA:
for string X find maximum i such that $T[SA[i]..n] < X$.

Binary search in suffix array: algorithm 1, $O(m \log n)$

```
// find max i such that T[SA[i]..n] < X
L = 0; R = n;
while(L < R){
    k = (L + R + 1) / 2;
    h = 0;
    while(T[SA[k] + h] == X[h]) h++;
    if(T[SA[k]+h] < X[h]) L = k;
    else R = k - 1;
}
return L;
```

Longest common prefix

- $\mathrm{lcp}(A, B) =$ the length of longest common prefix of strings A and B
- $\mathrm{LCP}(i, j) = \mathrm{lcp}(T[SA[i]..n], T[SA[j]..n])$

  i.e. lcp of two suffixes in a suffix array

i	$SA[i]$	Suffix
0	**6**	$
1	**5**	a$
2	**3**	ana$
3	**1**	anana$
4	**0**	banana$
5	**4**	na$
6	**2**	nana$

LCP(2,3) = lcp(ana$,anana$) = 3

LCP(2,5) = lcp(ana$, na$) = 0

Longest common prefix

$\text{lcp}(A, B)$: the length of longest common prefix of strings A and B

lcp(ana$,anana$) = 3

lcp(ana$, na$) = 0

Exercise

In one iteration we do $\text{lcp}(X, T[SA[k]..n]) + 1$ comparisons

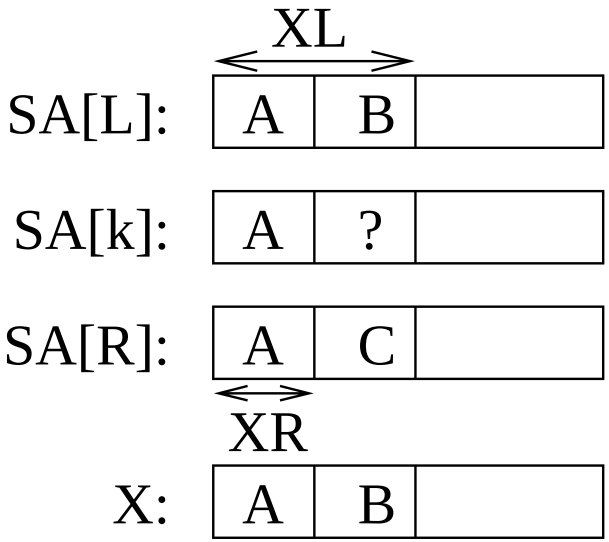
This can be any number between 1 and min$(m + 1, n + 1 - SA[k])$

Find a bad case (lower bound) for any values $m, n \geq 2$.

Binary search in suffix array: algorithm 2, $O(m \log n)$

```
// find max i such that T[SA[i]..n-1] < X
L = 0; R = n;
XL = lcp(X, T[SA[L]..n]); XR = lcp(X, T[SA[R]..n]);
while(R - L > 1){
   k = (L + R + 1) / 2;
   h = min(XL, XR);
   while(T[SA[k] + h] == X[h]) h++;
   if(T[SA[k] + h] < X[h]){ L = k; XL = h; }
   else {  R = k; XR = h; }
}
sequential search in SA[L..R];
```

Binary search in suffix array: algorithm 2, $O(m \log n)$



Exercise

What is the number of comparisons for $T = ba^{n-1}\$$, $X = a^{n-1}\#$?

What is the number of comparisons for $T = a^{n}\$$, $X = a^{n-1}\#$?

Binary search in suffix array: algorithm 3, $O(m + \log n)$

Recall: $LCP(i, j) = lcp(T[SA[i]..n-1], T[SA[j]..n-1])$

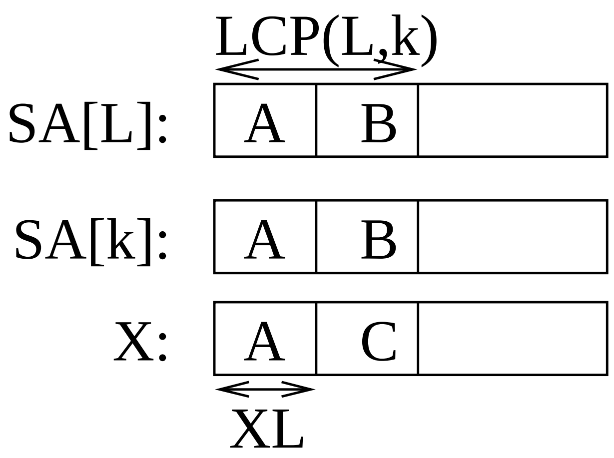
assume we know $LCP(i, j)$ for any i, j (more later)

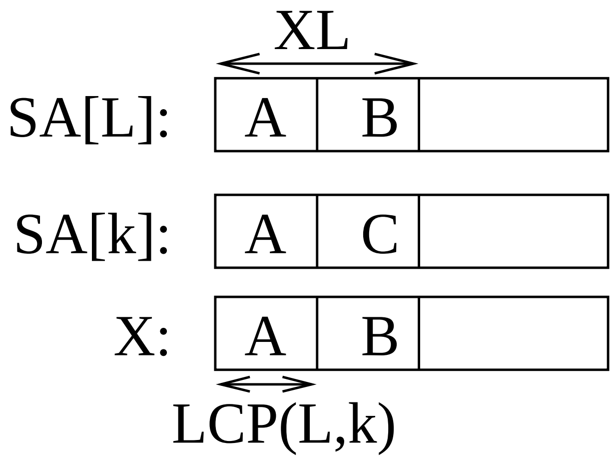
Comparing $T[SA[k]..n]$ and X, assume $XL \geq XR$

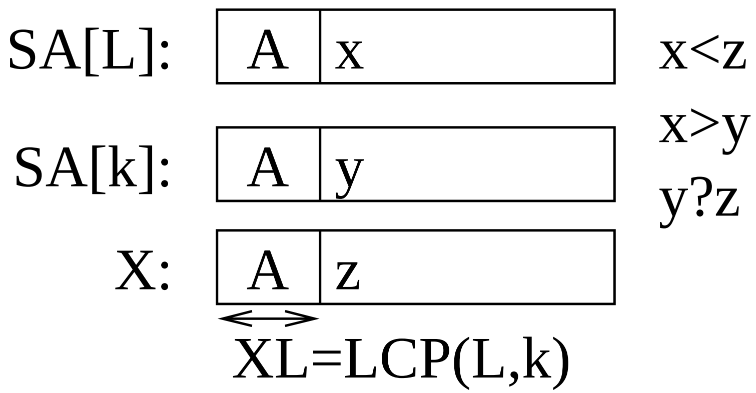
- If $LCP(L, k) > XL$: set $L \leftarrow k$
- If $LCP(L, k) < XL$: set $R \leftarrow k$; $XR \leftarrow LCP(L, k)$;
- If $LCP(L, k) = XL$: start comparing at XL

Case $XL < XR$ symmetrical to $XL \geq XR$

Binary search in suffix array: algorithm 2, $O(m \log n)$





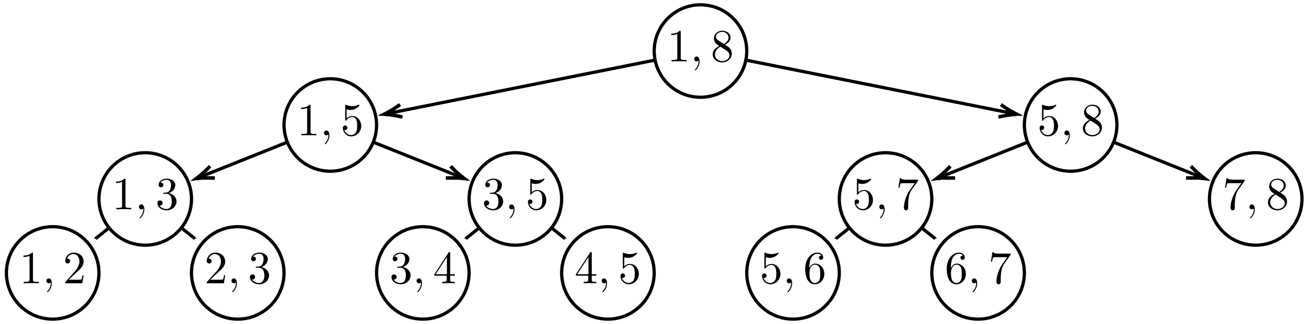


Binary search in suffix array: algorithm 3, $O(m + \log n)$

```
L = 0; R = n; XL = lcp(X, T[SA[L]..n]); XR = lcp(X, T[SA[R]..n]);
while(R − L > 1){
  k = (L + R + 1) / 2;
  if(XL>=XR && LCP(L,k)>XL) { L = k; }
  else if(XL>=XR && LCP(L,k)<XL) { R = k; XR = LCP(L,k); }
  else if(XL<XR && LCP(R,k)>XR) { R = k; }
  else if(XL<XR && LCP(R,k)<XR) { L = k; XL = LCP(R,k); }
  else {
    h = max(XL, XR); while(T[SA[k]+h]==X[h]) { h++; }
    if(T[SA[k]+h] < X[h]){ L = k; XL = h; } else { R = k; XR = h; }
  }
}
sequential search in SA[L..R];
```

LCP values for algorithm 3

Which values are needed? $\mathrm{LCP}(L, k)$ or $\mathrm{LCP}(R, k)$



$2n - 1$ LCP values needed

Let $L[i] = \mathrm{LCP}(i, i+1)$, precompute to an array in $O(n)$ (later)

For $j - i > 1$:

$$\begin{aligned} \mathrm{LCP}(i, j) &= \min\{\mathrm{LCP}(k, k+1) \mid k = i \ldots j-1\} \\ &= \min\{\mathrm{LCP}(i, x), \mathrm{LCP}(x, j)\} \text{ for any } x \in \{i+1, \ldots j-1\} \end{aligned}$$

Suffix trees and arrays - summary

Data structure	Search	# pointers/integers
Suffix tree	$O(m \log \sigma)$	$7n$ or more
Suffix array	$O(m \log n)$	n
Suffix array + LCP	$O(m + \log n)$	$3n$

More memory needed in preprocessing stage.

Next:

- Construction of suffix arrays
- Computation of lcp values
- Construction of suffix trees from suffix arrays

Inverse of a suffix array

Array rank such that $rank[i] = x \iff SA[x] = i$

Can be computed in $O(n)$ from SA:

```
for(i = 0; i <= n; i++) {
    rank[SA[i]] = i;
}
```

Go from suffix to its position in the suffix array, its neighbors, etc.

Longest common prefix

- $\mathrm{lcp}(A, B) =$ the length of longest common prefix of strings A and B
- $\mathrm{LCP}(i, j) = \mathrm{lcp}(T[SA[i]..n-1], T[SA[j]..n-1])$
- $L[i] = \mathrm{LCP}(i, i+1)$

  i.e. lcp of two consecutive suffixes in a suffix array

Lemma: If $SA[x+1] + 1 = SA[y+1]$, then $L[y] \geq L[x] - 1$.

Suffix	i	$SA[i]$	$L[i]$
$	0	5	0
aabab$	1	0	1
ab$	2	3	2
abab$	3	1	0
b$	4	4	1
bab$	5	2	-

Computation of the LCP array

```
h = 0;
for(i=0; i<=n; i++) {
   if(rank[i]>0) {
      k = SA[rank[i]-1];
      // compare suffixes S[i..n-1] a S[k..n-1]
      // assuming they have at least h characters in common
      while (S[i+h]==S[k+h]) { h++; }
      // we have found first mismatch
      L[rank[i]-1] = h;
      if(h>0) { h--; }
   }
}
```

Similar but quadratic-time algorithm

```
for(i=0; i<=n; i++){
   if(rank[i]>0){
      k = SA[rank[i]-1];
      // compare suffixes starting at i a k
      h = 0;
      while (T[i+h]==T[k+h]) { h++; }
      // we have found the first difference
      L[rank[i]-1] = h;
   }
}
```

Recall: RadixSort

Bucket sort:

Sort n numbers from $\{0, \ldots, d-1\}$

Time $O(n+d)$

Stable (does not change order of equal keys)

Radix sort:

Sort n k-digit numbers with digits from $\{0, \ldots, d-1\}$

Time $O(k(n+d))$

Use Bucket sort on each digit, starting from the least significant

How to create a suffix array

Goal: sort all suffixes lexicographically

Not so good options:

- Use MergeSort $O(n^2 \log n)$
- Use RadixSort $O(n^2)$
- Create a suffix tree, then convert to array by DFS. $O(n)$, but linear construction of suffix trees complicated.

Instead use $O(n)$ algorithm, e.g. Karkkainen and Sanders 2003

$O(n)$ algorithm for suffix array construction

Assume alphabet $\{1, \ldots, n\}$

Replace $ by several zeroes

i	0	1	2	3	4	5	6	7	8			
$S_p[i]$	f	a	b	b	c	a	b	b	d			
$S[i]$	5	1	2	2	3	1	2	2	4	0	...	0

Step 1: sort suffixes $S[3i + k..n]$ for $k = 1, 2$ to $SA_{1,2}$

$S_p = \text{fabbcabbd}$

$S = 5, 1, 2, 2, 3, 1, 2, 2, 4, 0$

$S' = [\text{abb}][\text{cab}][\text{bd0}][\text{bbc}][\text{abb}][\text{d00}]$

$= [1, 2, 2][3, 1, 2][2, 4, 0][2, 2, 3][1, 2, 2][4, 0, 0]$

$= 1, 4, 3, 2, 1, 5, 0$

S'	1	4	3	2	1	5	0
index in S'	0	1	2	3	4	5	6
index in S	1	4	7	2	5	8	

SA'	6	0	4	3	2	1	5
$SA_{1,2}$	–	1	5	2	7	4	8

Step 1: sort suffixes $S[3i+k..n]$ for $k = 1, 2$ to $SA_{1,2}$

$$\text{rank}[i] = \begin{cases} 0 & i \geq n \\ - & i \bmod 3 = 0 \\ j & SA_{1,2}[j] = i \end{cases}$$

j	0	1	2	3	4	5	6
$SA_{1,2}$	–	1	5	2	7	4	8

i	0	1	2	3	4	5	6	7	8	9
$S_p[i]$	f	a	b	b	c	a	b	b	d	
$S[i]$	5	1	2	2	3	1	2	2	4	0
$\text{rank}[i]$	–	1	3	–	5	2	–	4	6	0

Step 2: sort suffixes $S[3i..n]$ to SA_0

$S[3i..n]$ represent as $(S[3i], rank[3i+1])$

$S[3i..n] < S[3j..n]$

$\iff (S[3i], rank[3i+1]) < (S[3j], rank[3j+1])$

i	0	1	2	3	4	5	6	7	8	9
$S_p[i]$	f	a	b	b	c	a	b	b	d	
$rank[i]$	–	1	3	–	5	2	–	4	6	0
	$f, 1$	–	–	$b, 5$	–	–	$b, 4$	–	–	

$SA_0 = (6, 3, 0)$.

Step 3: merge SA_0 and $SA_{1,2}$ to SA.

To compare $S[i..n]$ from $SA_{1,2}$ and $S[j..n]$ from SA_0:

$$\begin{aligned}
\text{if } i \bmod 3 = 1\text{:} \quad & S[i..n] \leq S[j..n] \\
\iff \quad & (S[i], rank[i+1]) \leq (S[j], rank[j+1]) \\
\text{if } i \bmod 3 = 2\text{:} \quad & S[i..n] \leq S[j..n] \\
\iff \quad & (S[i], S[i+1], rank[i+2]) \leq (S[j], S[j+1], rank[j+2])
\end{aligned}$$

i	0	1	2	3	4	5	6	7	8	9
$S_p[i]$	f	a	b	b	c	a	b	b	d	
$rank[i]$	–	1	3	–	5	2	–	4	6	0

$SA_{1,2} = (-, 1, 5, 2, 7, 4, 8)$

$SA_0 = (6, 3, 0)$

Algorithm overview

- Step 1: sort suffixes $S[3i+k..n]$ for $k = 1, 2$ to $SA_{1,2}$

  (create triples, radixsort and rename, call recursion, renumber)

  $T(2n/3) + O(n)$

- Step 2: sort suffixes $S[3i..n]$ to SA_0

  (create pairs, use radixsort)

  $O(n)$

- Step 3: merge SA_0 and $SA_{1,2}$ to SA.

  (compare as triples or pairs)

  $O(n)$

Overall running time $T(n) = T(\frac{2}{3}n) + O(n)$

Master theorem

One of the three cases from the theorem:

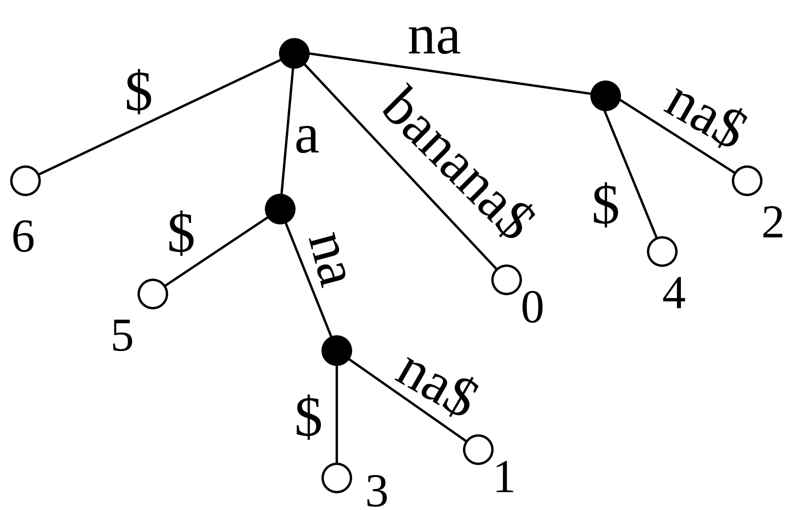
If $T(n) = aT(\frac{n}{b}) + f(n)$
where $f(n) = \Omega(n^{\log_b(a)+\varepsilon})$
and $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ for some $c < 1$,
then $T(n) = \Theta(f(n))$.

In our algorithm: $T(n) = T(\frac{2}{3}n) + O(n)$
$a = 1$, $b = \frac{3}{2}$
$\log_{\frac{3}{2}}(1) = 0$
$a \cdot f(\frac{n}{b}) = \frac{2}{3}n$
Therefore $T(n) = \Theta(n)$.

From suffix array to suffix tree

$T = $banana$

i	$SA[i]$	$L[i]$	suffix
0	**6**	0	$
1	**5**	1	a$
2	**3**	3	ana$
3	**1**	0	anana$
4	**0**	0	banana$
5	**4**	2	na$
6	**2**	-	nana$



From suffix array to suffix tree

```
create roof and leaf w corresponding to SA[0]
v = w;
for(int i=1; i<=n; i++) {
    while(v.parent.string_depth>L[i-1]) {
      v = v.parent;
    }
    if(v.parent.string_depth<L[i-1]) {
      split edge from v.parent to v with a new vertex
      at string depth L[i-1]
    }
    attach new leaf w for SA[i] from v.parent;
    v = w;
}
```

Suffix trees and arrays - summary

Data structure	Search	# pointers/integers
Suffix tree	$O(m \log \sigma)$	$7n$ or more
Suffix array	$O(m \log n)$	n
Suffix array + LCP	$O(m + \log n)$	$3n$

More memory needed in preprocessing stage.

- Construction of suffix arrays in $O(n)$
- Computation of lcp values in $O(n)$
- Construction of suffix trees from suffix arrays in $O(n)$