Organizačné poznámky

- DÚ na stránke, termín do stredy pred Veľkou nocou (27.3.)
- Témy nepovinného projektu do konca marca

Regular expressions, summary

- Thompson's algorithm 1968
  - parse R to a tree with m nodes
  - create NFA for $L(R)$ with $\leq 2m$ states and $\leq 4m$ transitions
  - simulate NFA on T in $O(nm)$ time
- DFA: $O(m^2\sigma 2^m + n)$
- Other approaches: hybrid of NFA and DFA, prefiltering, bit parallelism, ...

Patterns 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
- Suffix trees $O(nk)$ where k is the number of wildcards
- Algorithm using FFT $O(n \log m)$
  - represent characters as numbers $\{1, 2, \ldots, \sigma\}$, wildcard as 0
  - for each i compute $a_i = \sum_{j=0}^{m-1} P[j]T[i+j](P[j]-T[i+j])^2$
  - expression similar to multiplying polynomials P and T^R
  - occurrences of P have $a_i = 0$
  - trick with cutting T to overlapping windows of length $2m$

Polynomial multiplication by Fast Fourier Transform (FFT)

Input: Two polynomials

$A(x) = \sum_{k=0}^{n-1} a_k x^k$

$B(x) = \sum_{k=0}^{n-1} b_k x^k$

Goal: Compute product $C(x) = A(x)B(x)$

$C(x) = \sum_{k=0}^{2n-2} c_k x^k$

$c_k = \sum_{j=0}^{n-1} a_j b_{k-j}$

(define $b_j = 0$ if $j < 0$ or $j \geq n$)

Trivial algorithm: $O(n^2)$

FFT: $O(n \log n)$

Polynomial multiplication by FFT

– find $n = 2^k$ so that $A(x)B(x)$ has degree $< n$

– pad coefficients of A and B to length n

– compute $A(\omega_n^j)$ for $j = 0 \dots n-1$ by FFT

– compute $B(\omega_n^j)$ for $j = 0 \dots n-1$ by FFT

– compute $C(\omega_n^j) = A(\omega_n^j)B(\omega_n^j)$ for all j in $O(n)$

– convert $C(x)$ back to coefficient form by FFT

Use nth complex root of unity:

$$\omega_n = e^{2\pi i/n} = \cos(2\pi/n) + i\sin(2\pi/n)$$

Useful facts:

$$\omega_n^i \omega_n^j = \omega_n^{(i+j) \bmod n}$$

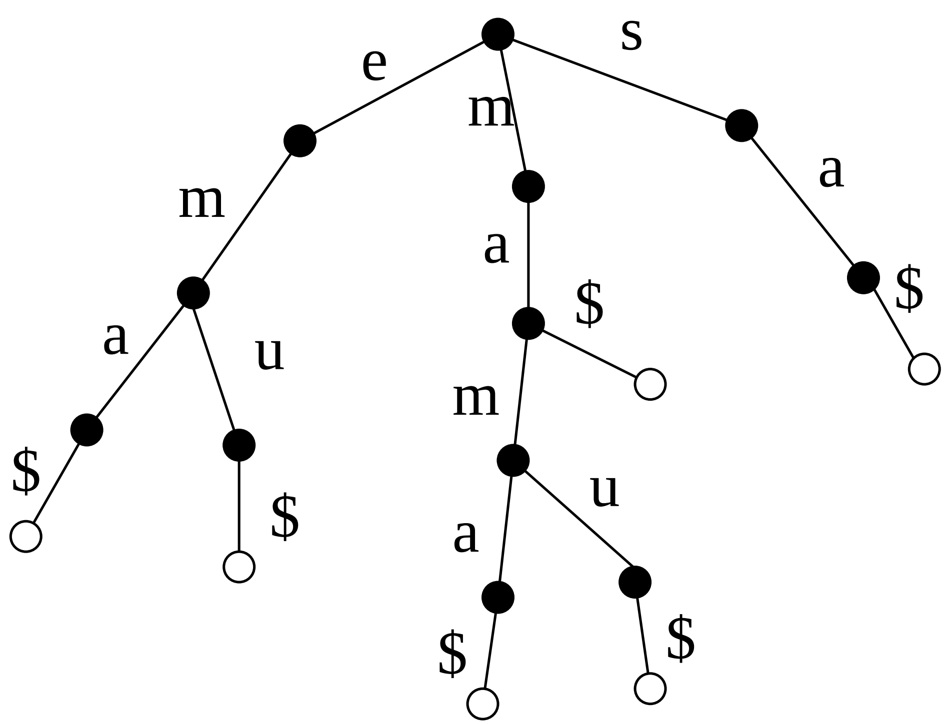
$$\omega_n^n = 1, \quad \omega_n^{n/2} = -1$$

$$\omega_{n/2} = \omega_n^2, \quad \omega_n^{2j} = \omega_{n/2}^j$$

Fast Fourier transform

```
complex FFT(A,n) {
   if n=1, return (A[0]);
   A1 = A[0,2,4,...,n−2];    Y1 = FFT(A1);
   A2 = A[1,3,5,...,n−1];    Y2 = FFT(A2);
   omega = cos(2∗pi/n) + i∗ sin(2∗pi/n);
   x = 1;
   for(int j=0; j<n/2; j++) {
     Y[j] = Y1[j] + x∗Y2[j];
     Y[j+n/2] = Y1[j] − x∗Y2[j];
     x = omega∗x;
   }
   return Y;
}
```

Recall: trie (lexikografický strom)



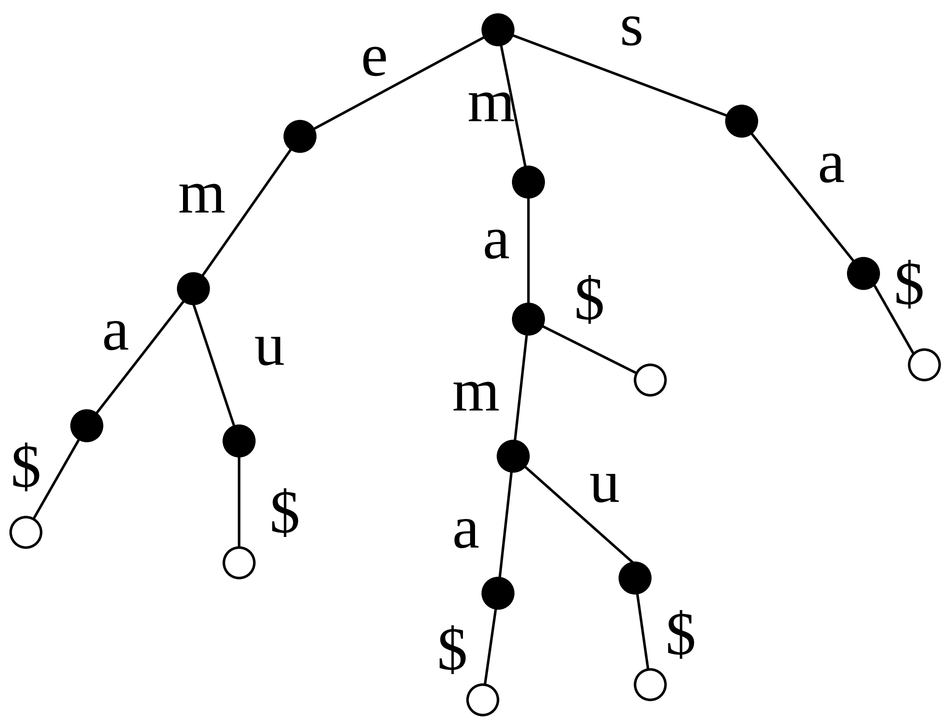
Represents a set of words, e.g. {ema, emu, ma, mama, mamu, sa}

Modification: add special symbol $ to the end of each word

Leafs: words in the set (may store additional info)

Internal nodes: prefixes of words in the set

Recall: trie (lexikografický strom)



Insert, delete, search $O(m)$ where is the length of the word

For large alphabets $O(m \log \sigma)$

Applications of tries

Work with individual words:

- Keyword search
- Spell-checking
- Counting word frequencies (homework)

Also used in multiple pattern search (Aho-Corasick)

and LZW compression

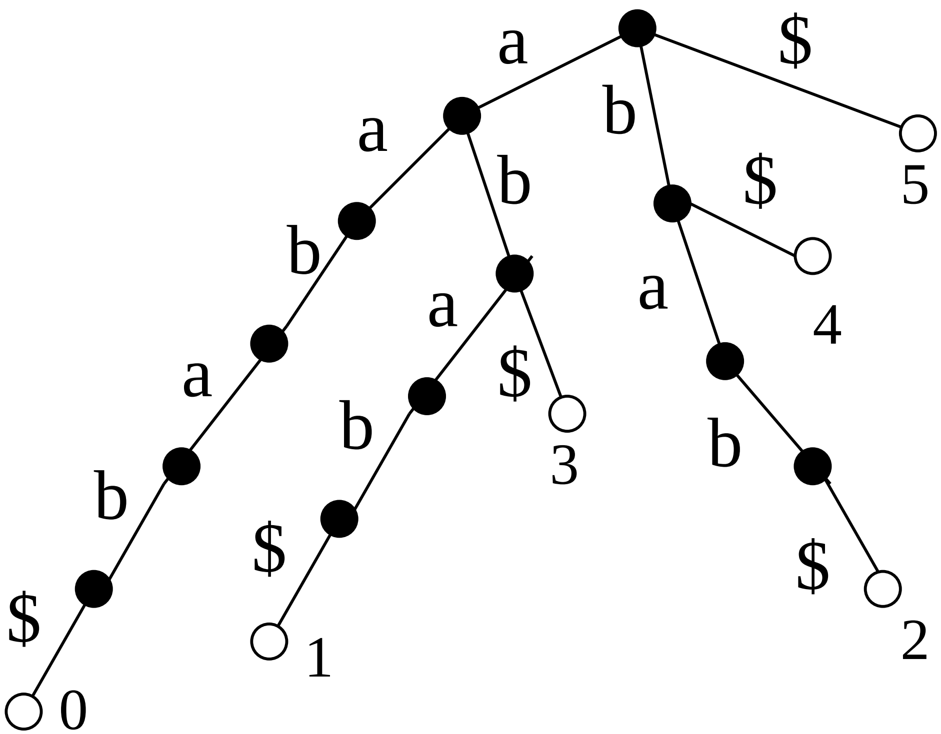
What about the following problems?

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

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

Suffix tree (sufixový strom)

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

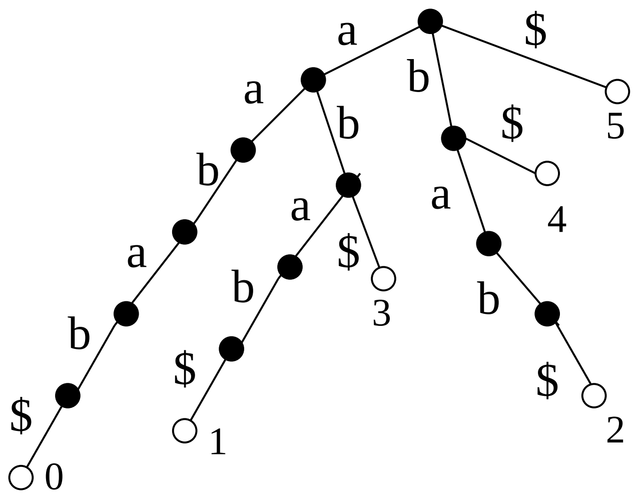


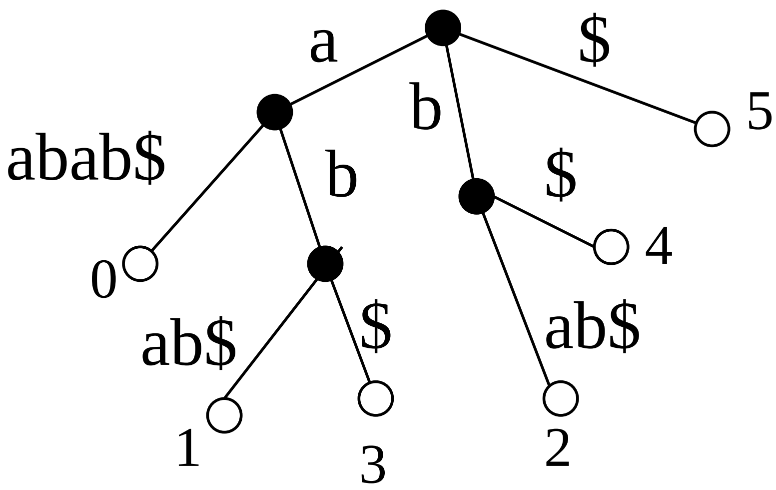
How many nodes in the tree?

Suffix tree

Compact all non-branching paths

T =aabab$



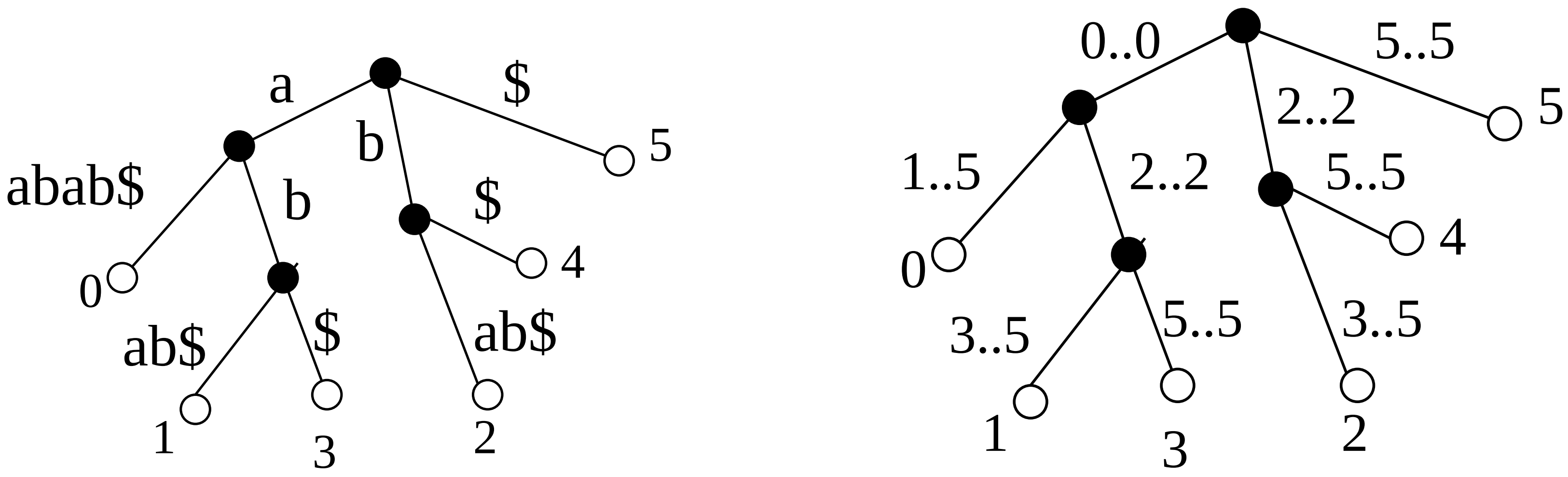


How many nodes in the new tree?

Suffix tree

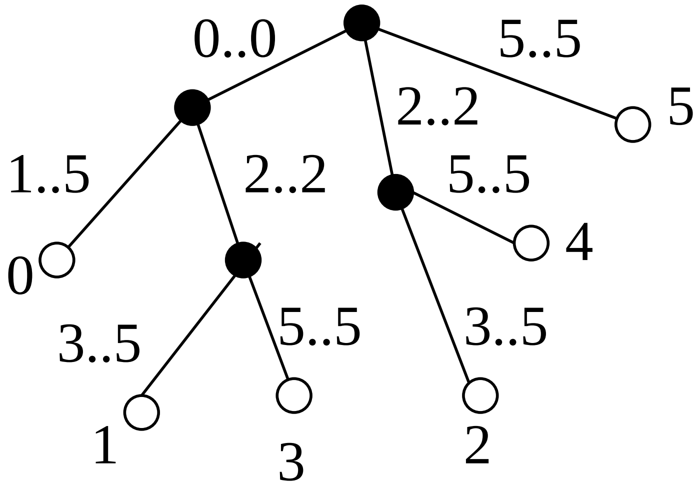
Store indices to T instead of substrings

T =aabab$



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