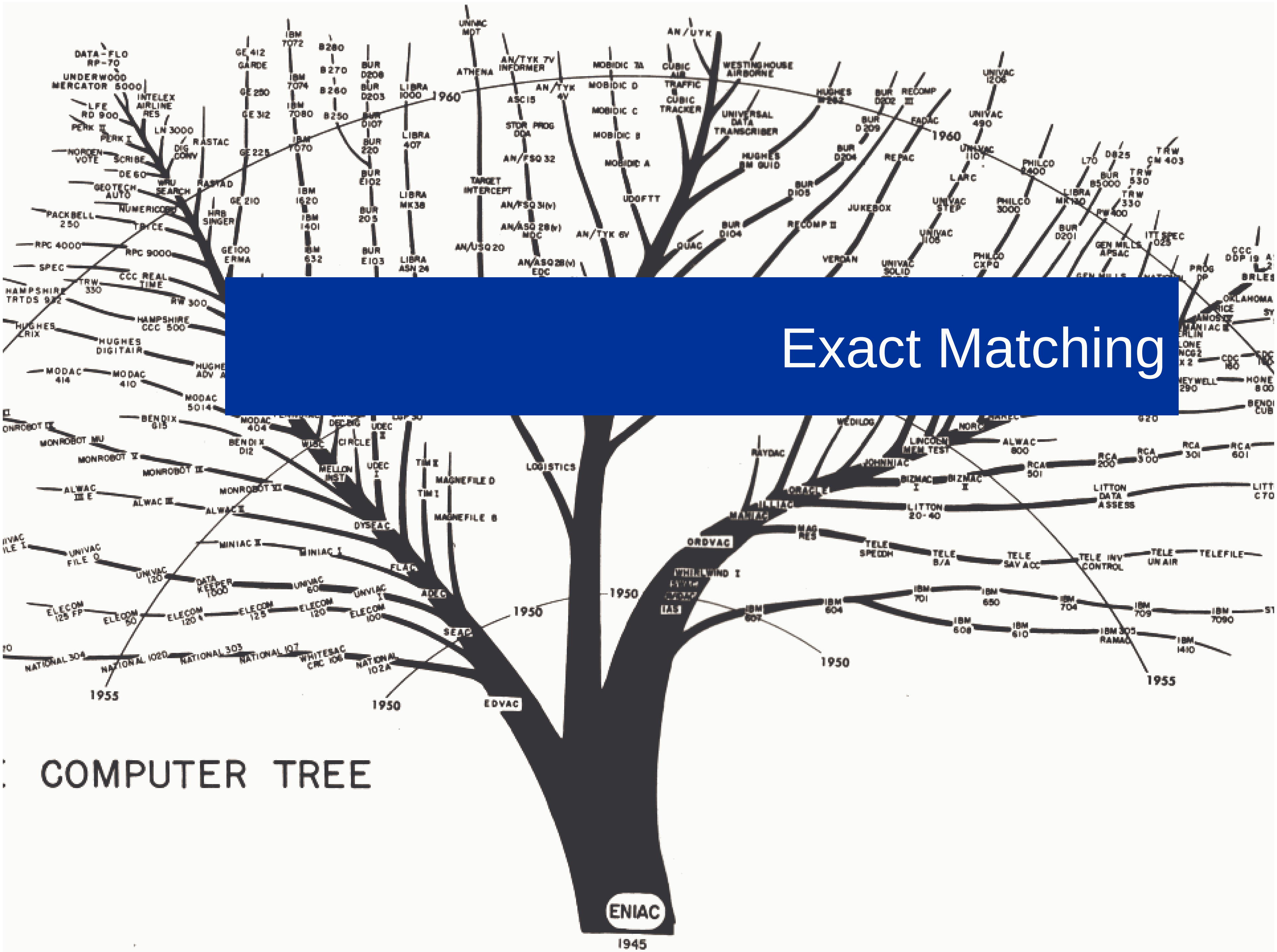
Exact Matching



exact matching: topics

exact matching

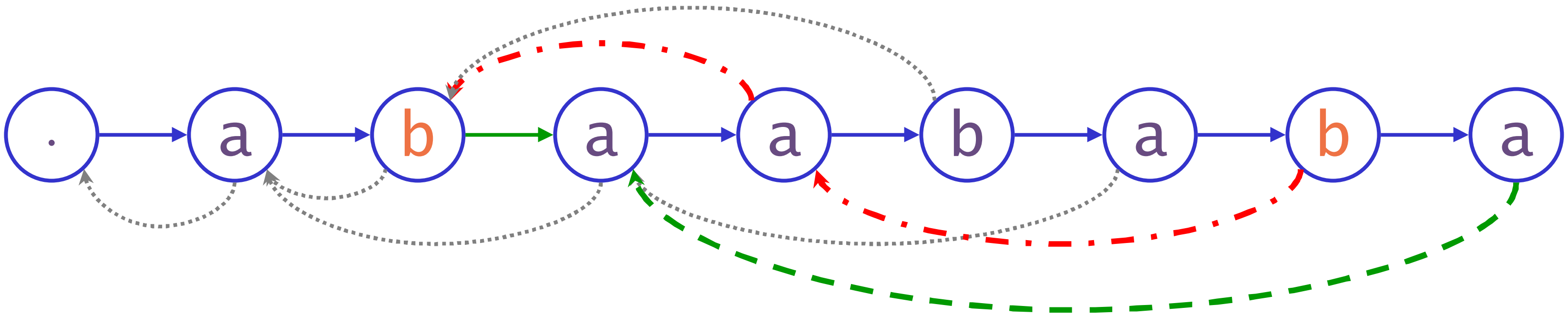
search pattern P in text T (P,T strings)

- **Knuth Morris Pratt**
  preprocessing pattern P
- **Aho Corasick**
  pattern of several strings
  $$P = \{ P_1, \ldots, P_r \}$$
- **Suffix Trees**
  preprocessing text T
  or several texts ‘database’

(A) preprocessing patterns

Knuth-Morris-Pratt
Aho-Corasick

KMP example



1	2	3	4	5	6	7	8
a	b	a	a	b	a	b	a
0	1	1	2	2	3	4	3

failure links

(suffix = prefix)

top: as ‘automaton’

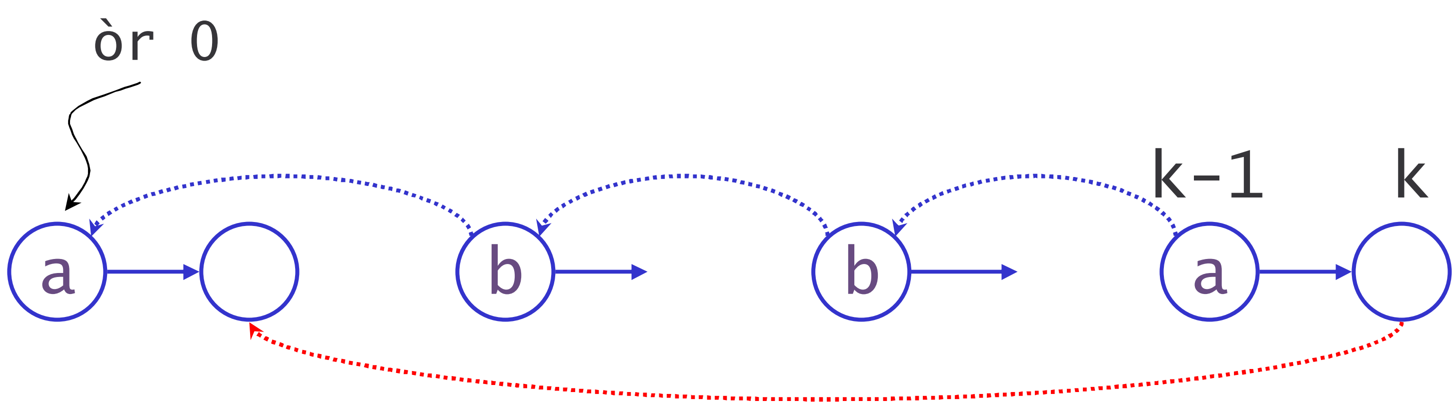
bottom: as table

- how to determine?
- how to use?

```
          6
a b a a b a b a
      a b a a b a b a
          3
            7
a b a a b a b a
      a b a a b a b a
            4
              8
a b a a b a b a
          a b a a b ...
              3
```

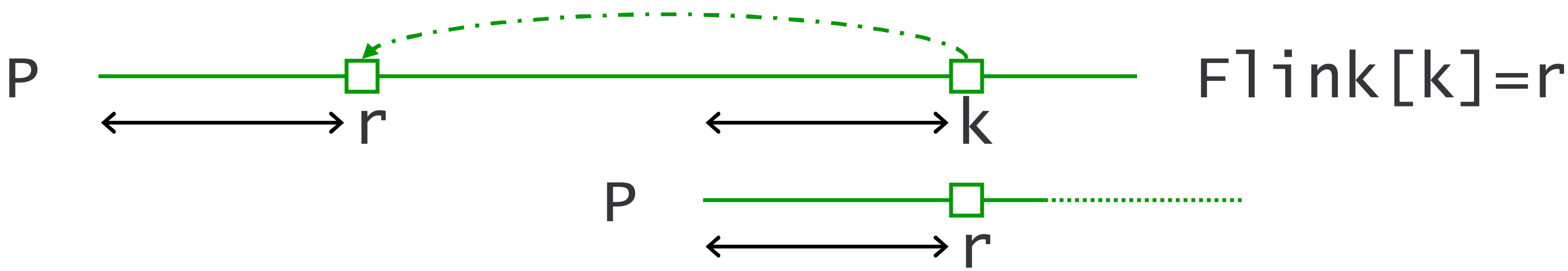
KMP computing failure links

failure link ~ new ‘best’ match (after mismatch)



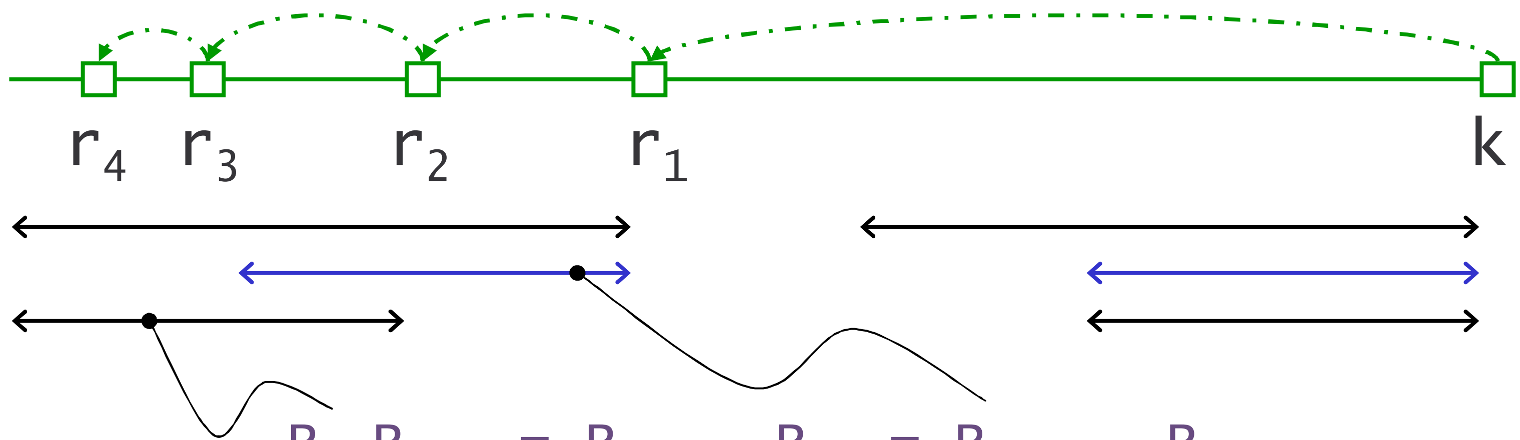

```
Flink[1] = 0;
for k from 2 to PatLen
do  fail = Flink[k-1]
    while ( fail>0 and P[fail]≠P[k-1] )
    do  fail = Flink[fail];
    od
    Flink[k] = fail+1;
od
```

prefixes via failure links



$P_1 \ldots P_{r-1} = P_{k-r+1} \ldots P_{k-1}$ maximal $r<k$

all such values r:



$P_1 \ldots P_{r2-1} = P_{k-r2+1} \ldots P_{k-1} = P_{r1-r2+1} \ldots P_{r1-1}$

$\Rightarrow$ Flink[r_1]=r_2

other methods

- Boyer-Moore

T = marktkoopman
P = schoenveter
schoe…

work backwards

- Karp-Rabin 'fingerprint'

$a_{i-1}\ a_i\ \ldots\ a_{i+n-1} a_{i+n}$

$p_1\ \ldots\ p_n$

hash-value

$a_i B^{n-1} + a_{i+1} B^{n-2} \qquad + a_{i+n-1} B^0$

$a_{i+1} B^{n-1} + \ \ldots\ + a_{i+n-1} B^1 + a_{i+n} B^0$

exact matching with a set of patterns

$P = \{ P_1, \ldots, P_r \}$ total length m
all occurrences in text T length n

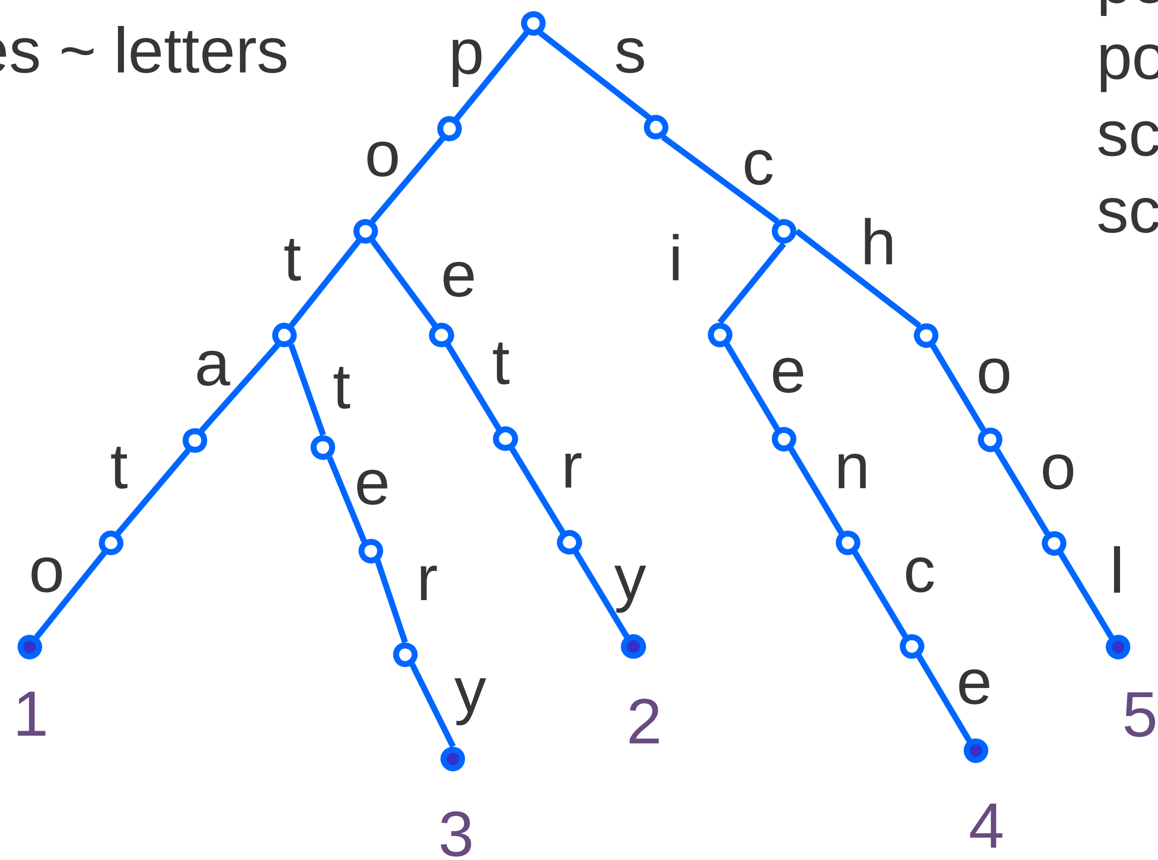
- **AHO CORASICK**
  generalizes KMP *failure links*
  longest suffix that is prefix
  (perhaps in another string)

> no subwords within P

keyword tree - *trie*

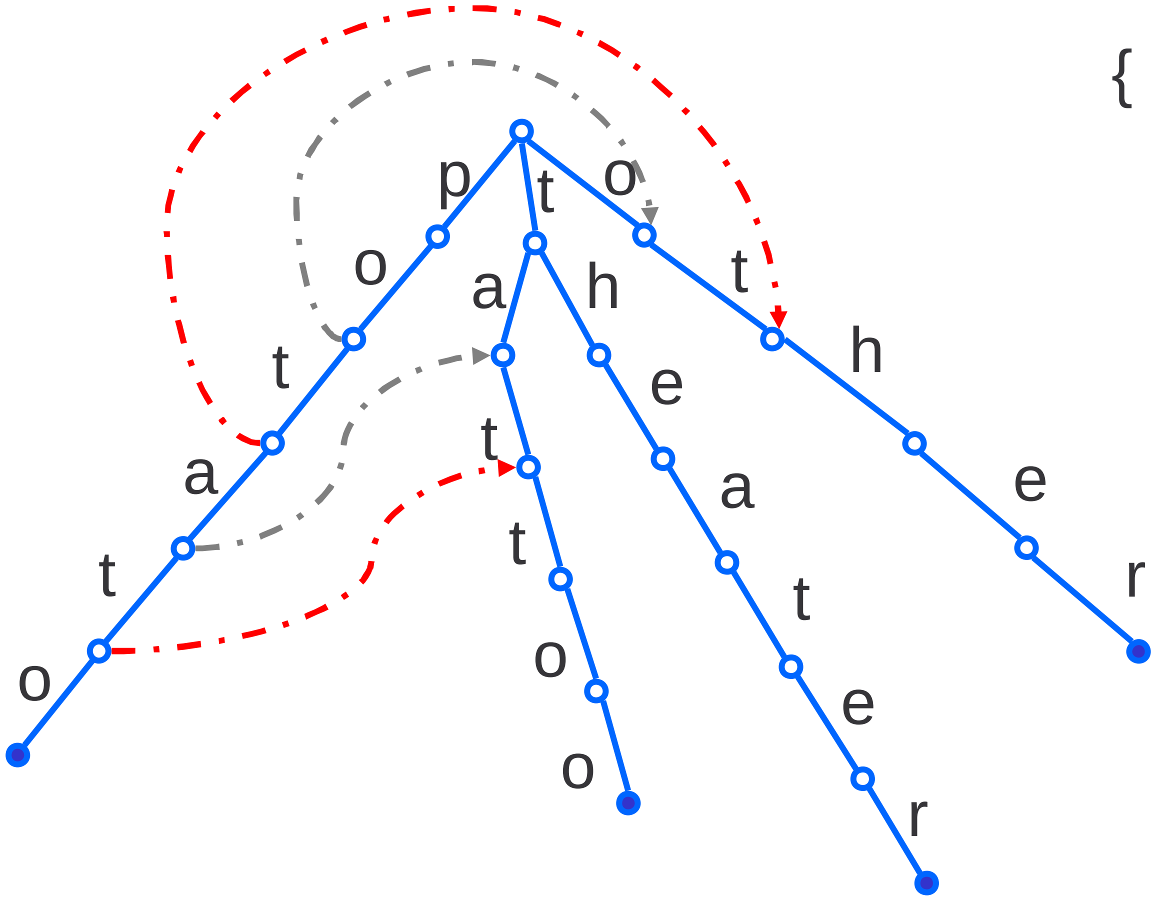
{ potato, 1
poetry, 2
pottery, 3
science, 4
school } 5

edges ~ letters



leaves ~ keywords

failure links



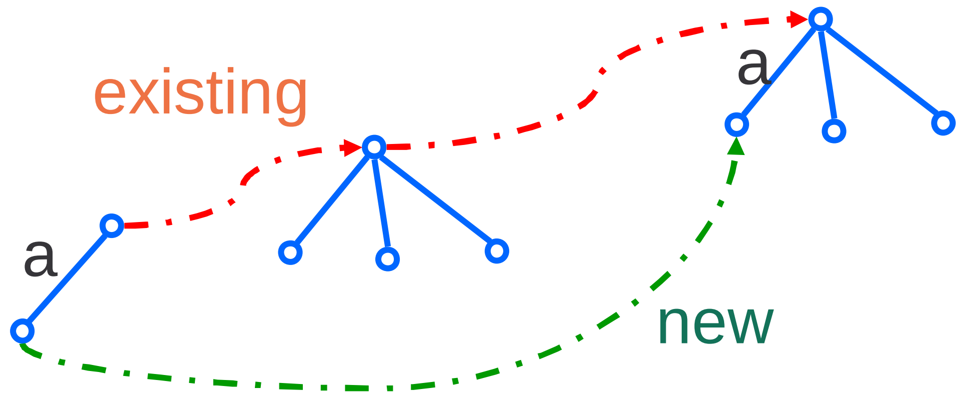
{ potato,
tattoo,
theater,
other }

```
potato
 other
```

```
potato
  tattoo
```

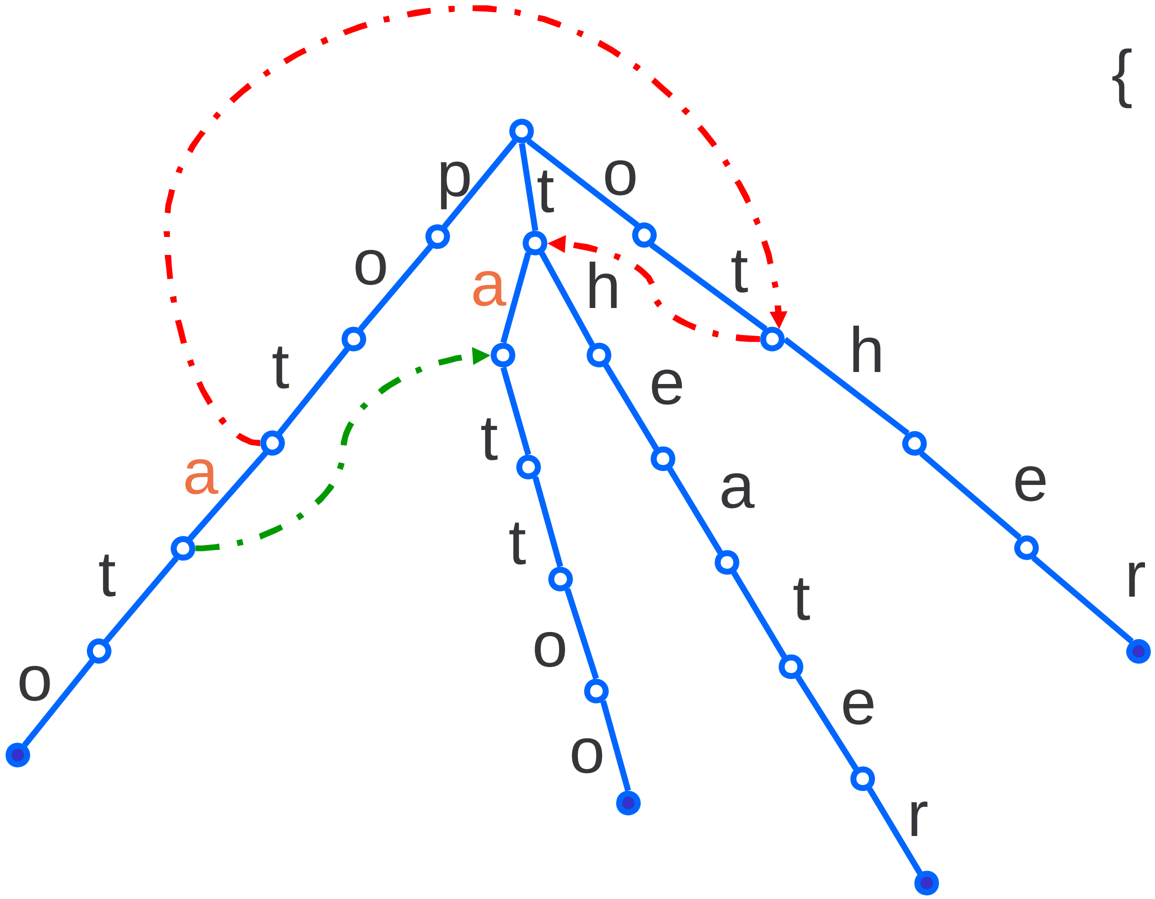
failure links into other branches!

algorithm: follow the links

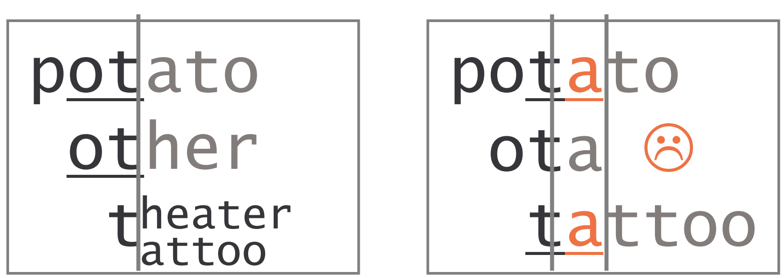


edge with incoming a
follow links starting at parent
until outgoing a is found

failure links

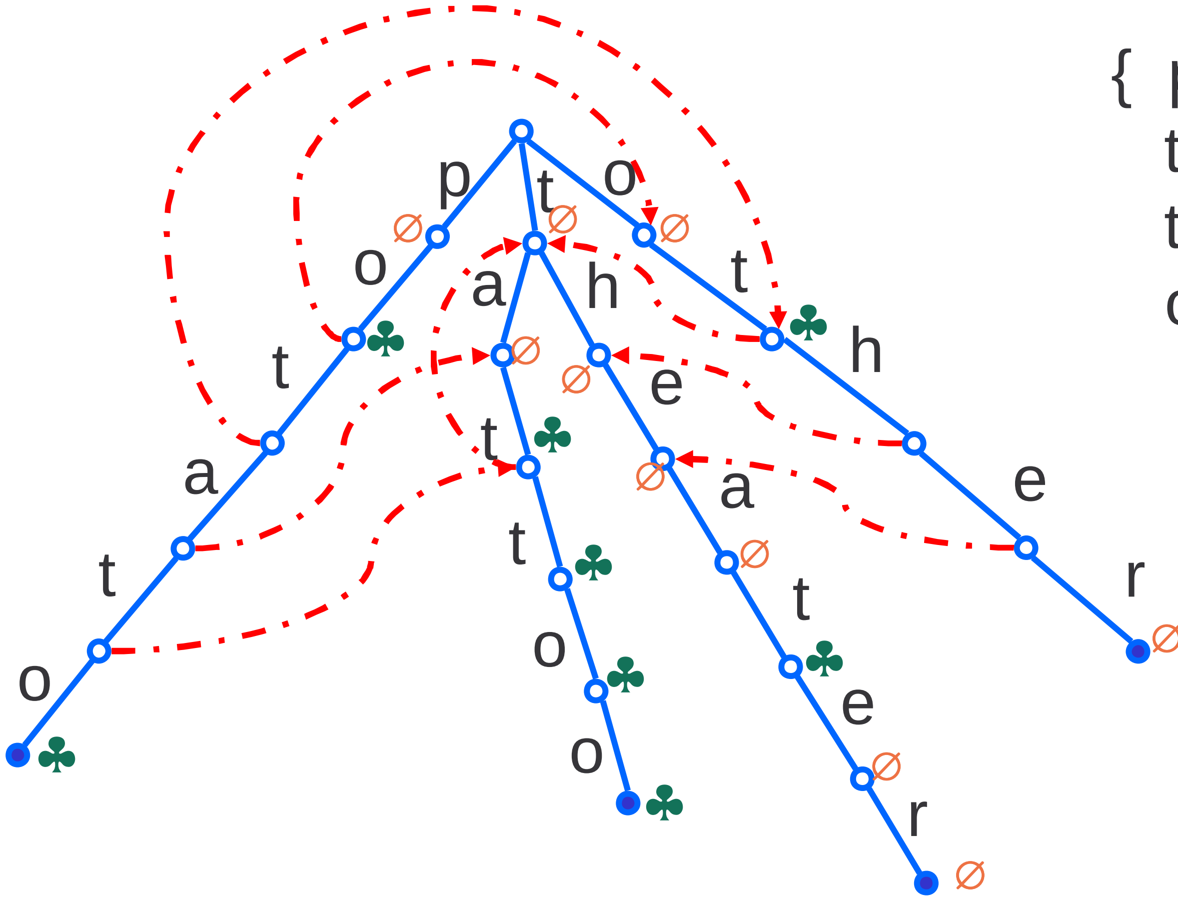


{ potato,
tattoo,
theater,
other }



breadth first
(level-by-level)

failure links



{ potato,
tattoo,
theater,
other }

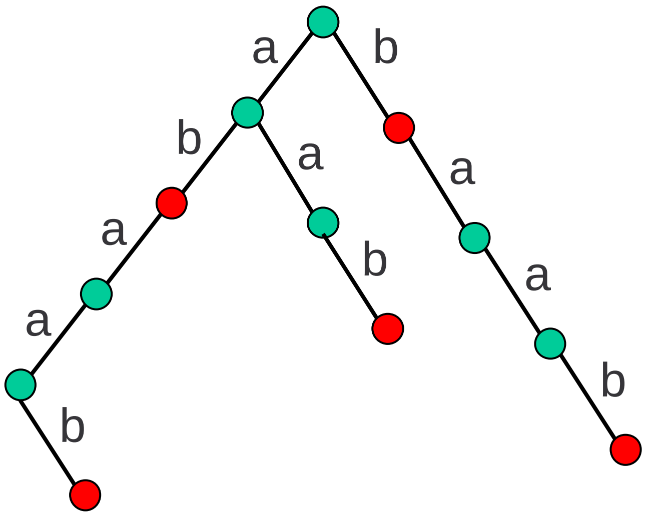
‘shortcuts’ ∅ root ♣ to child root [single letter]

(B) preprocessing text

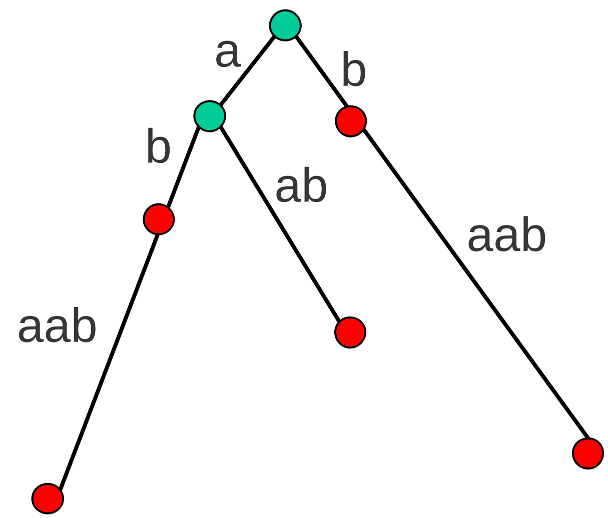
trie vs. suffix tree

string+suffixes

abaab
baab
aab
ab
b



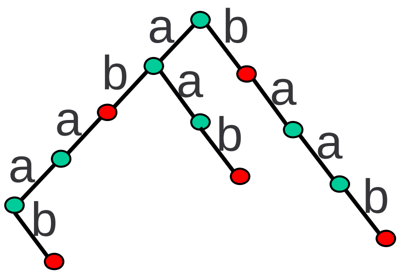
trie



suffix tree

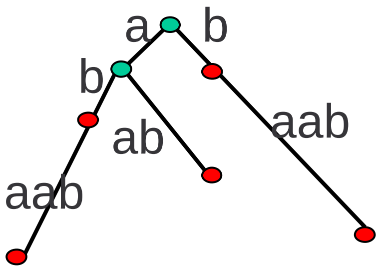
www.cs.helsinki.fi/u/ukkonen/Erice2005.ppt

trie vs. tree



- |Trie(T)| = O(|T|)^2
- bad example: T = $a^n b^n$
- Trie(T) like DFA for the suffixes of T
- minimize DFA → directed acyclic word graph

quadratic

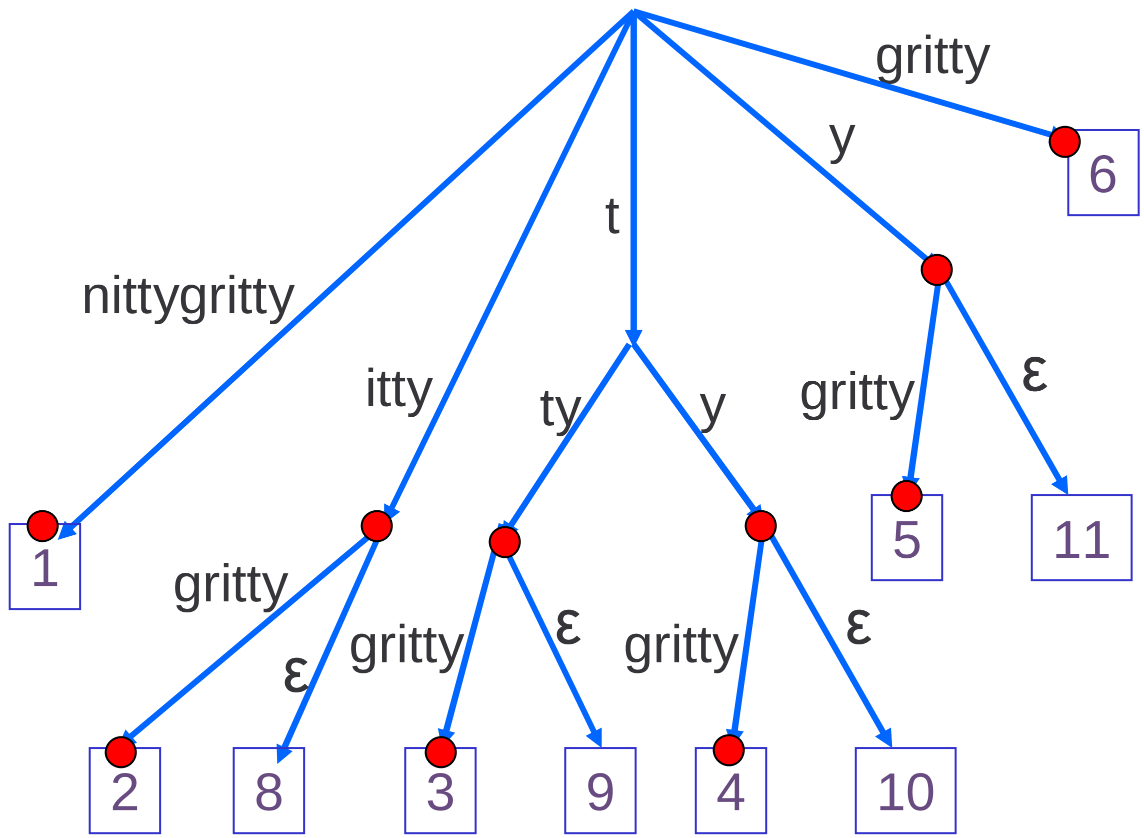


- only branching nodes and leaves represented
- edges labeled by substrings of T
- correspondence of leaves and suffixes
- |T| leaves, hence < |T| internal nodes
- |Tree(T)| = O(|T| + size(edge labels))

linear

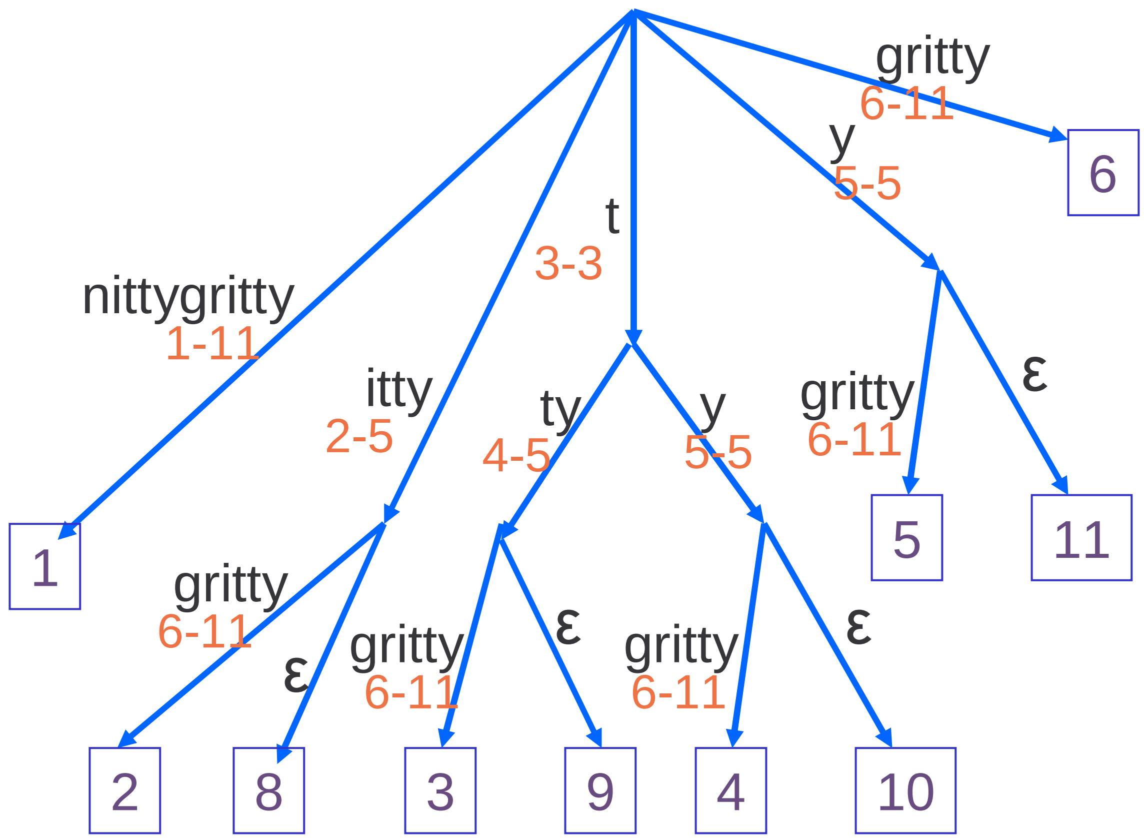
‘nittygritty’

nittygritty	1
ittygritty	2
ttygritty	3
tygritty	4
ygritty	5
gritty	6
ritty	7
itty	8
tty	9
ty	10
y	11



‘nittygritty’

nittygritty	1
ittygritty	2
ttygritty	3
tygritty	4
ygritty	5
gritty	6
ritty	7
itty	8
tty	9
ty	10
y	11



implementation: refer to positions

linear time construction

nittygritty
ittygritty
ttygritty
tygritty
ygritty
gritty
ritty
itty
tty
ty
y

Weiner (1973)

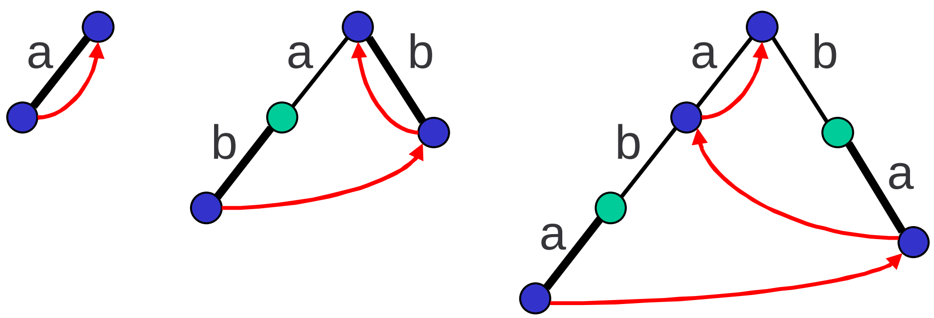
’algorithm of the year’

McCreight (1976)

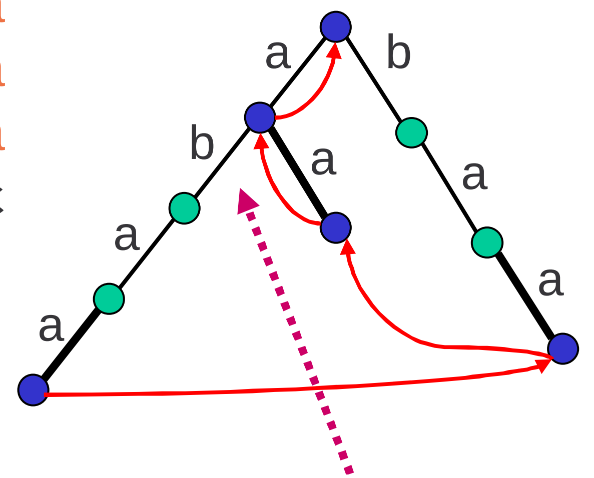
‘on-line’ algorithm (Ukkonen 1992)

suffix trie for abaab

suffix links



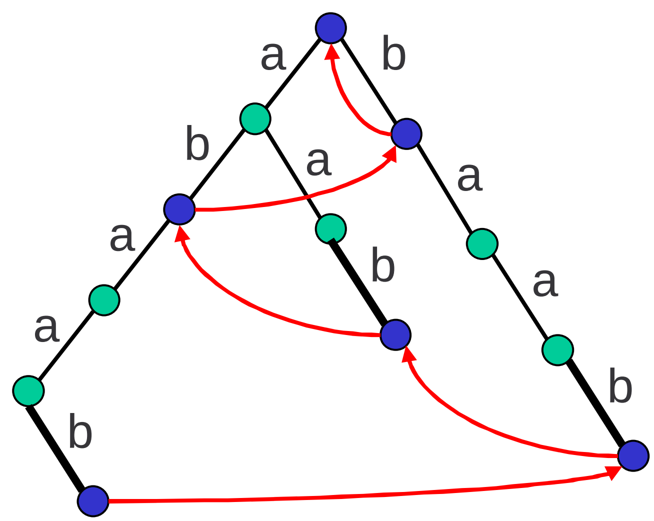
abaa
baa
aa
εa
ε



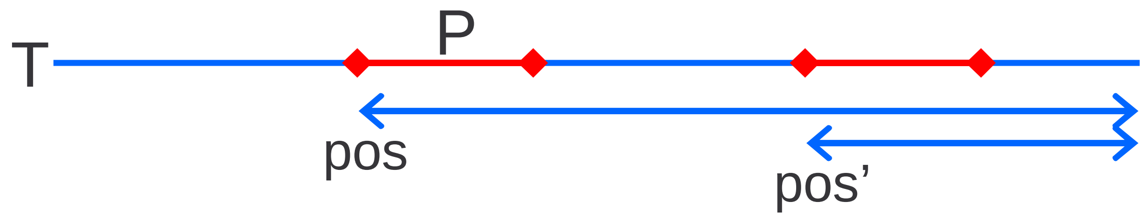
next symbol = b

from here
b already exists

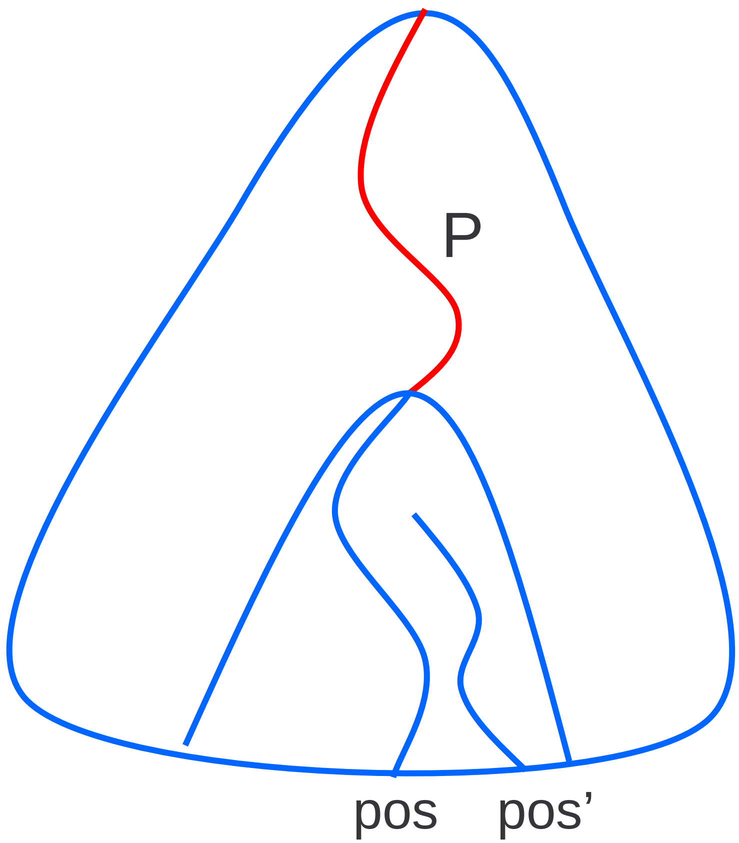
abaab
baab
aab
ab
εb
ε



application: full text index



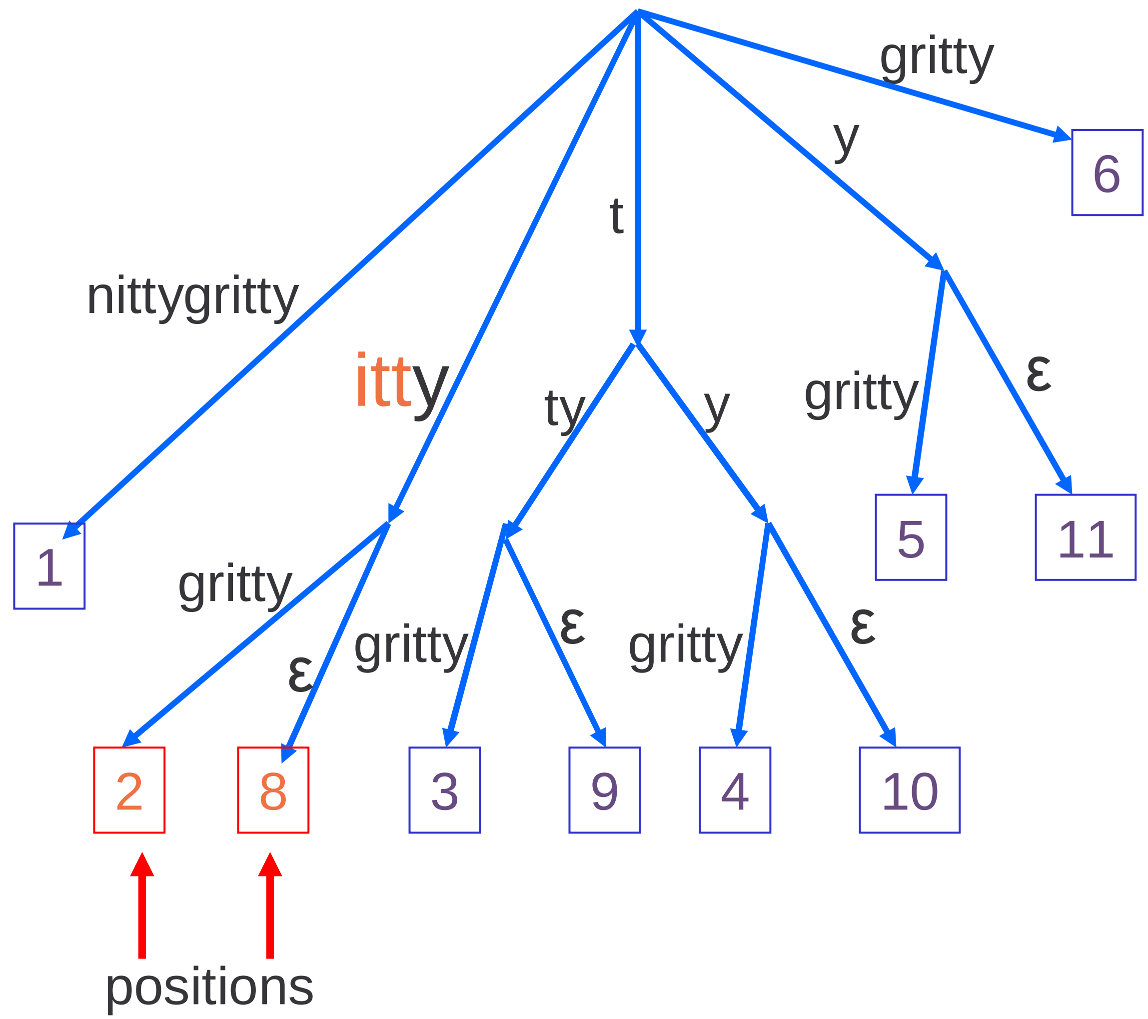
P in T ⇔ P is prefix of suffix of T



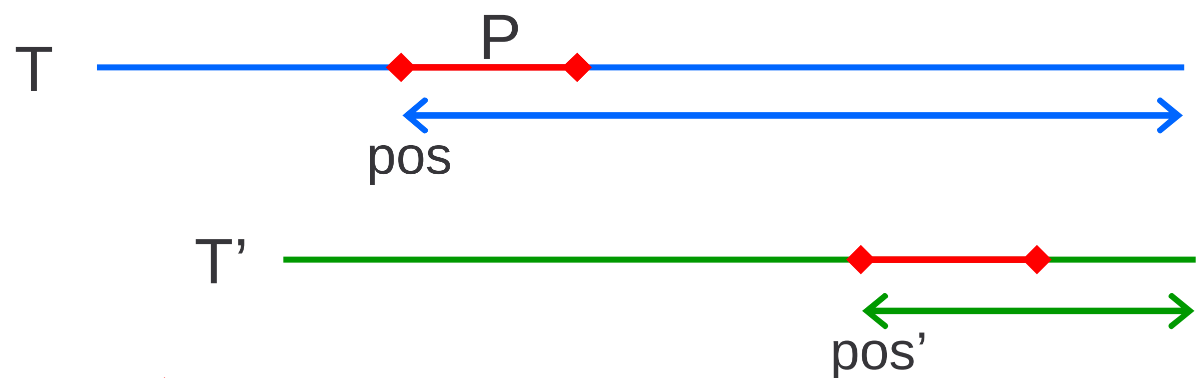
subtree under P
~ locations of P

example: find ‘itt’ in ‘nittygritty’

nittygritty	1
ittygritty	2
ttygritty	3
tygritty	4
ygritty	5
gritty	6
ritty	7
itty	8
tty	9
ty	10
y	11

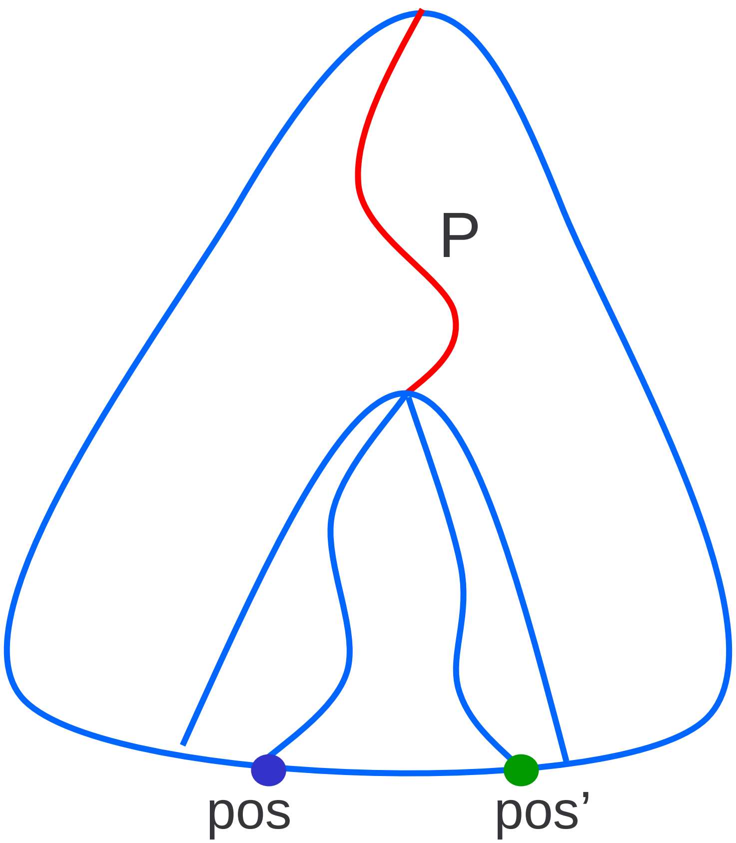


application: longest common substring



apples

plate

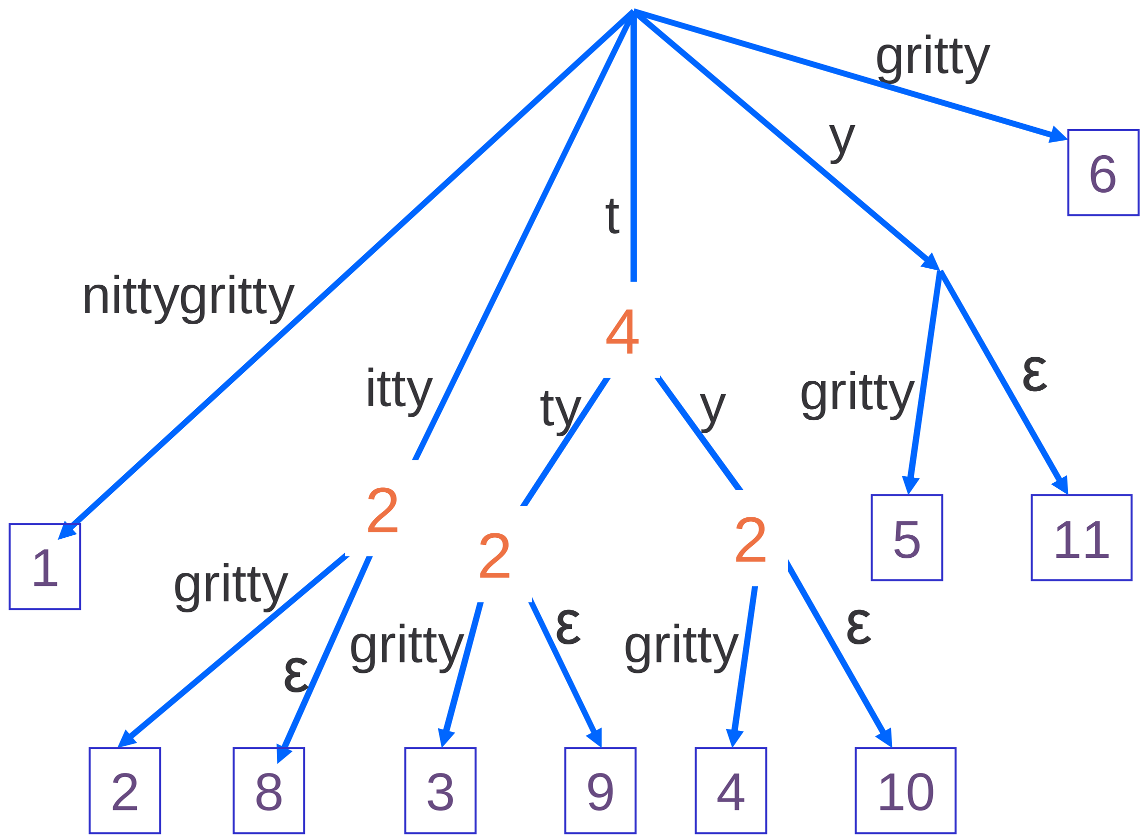


‘generalized’ suffix tree

(mark T and T’ suffixes)

application: counting 'motifs'

nittygritty	1
ittygritty	2
ttygritty	3
tygritty	4
ygritty	5
gritty	6
ritty	7
itty	8
tty	9
ty	10
y	11



‘motif’ : repeats in DNA

as reported by Ukkonen

- human chromosome 3
- the first 48 999 930 bases
- 31 min cpu time (8 processors, 4 GB)

- human genome: $3x10^9$ bases
- suffix tree for Human Genome feasible

longest repeat?

<u>Occurrences</u> at: 28395980, 28401554r **<u>Length</u>: 2559**

**ttagggtacatgtgcacaacgtgcaggtttgttacatatgtatacacgtgccatgatggtgtgctgcacccattaactcgtcatttagcgtta
ggtatatctccgaatgctatccctccccctcccccaccccacaacagtccccggtgtgtgatgttccccttcctgtgtccatgtgttctca
ttgttcaattcccacctatgagtgagaacatgcggtgtttggtttttgtccttgcgaaagtttgctgagaatgatggtttccagcttcatccata
tccctacaaaggacatgaactcatcatttttttatggctgcatagtattccatggtgtatatgtgccacattttcttaacccagtctacccttgttg
gacatctgggttggttccaagtctttgctattgtgaatagtgccgcaataaacatacgtgtgcatgtgtctttatagcagcatgatttataatcc
tttgggtatatacccagtaatgggatggctgggtcaaatggtatttctagttctagatccctgaggaatcaccacactgacttccacaatggt
tgaactagtttacagtcccagcaacagttcctatttctccacatcctctccagcacctgttgtttcctgactttttaatgatcgccattctaactg
gtgtgagatggtatctcattgtggttttgatttgcatttctctgatggccagtgatgatgagcatttttcatgtgtttttggctgcataaatgtctt
cttttgagaagtgtctgttcatatccttcgcccacttttgatggggttgtttgtttttttcttgtaaatttgttggagttcattgtagattctgggtatta
gccctttgtcagatgagtaggttgcaaaaattttctcccattctgtaggttgcctgttcactctgatggtggtttcttctgctgtgcagaagctct
ttagtttaattagatcccatttgtcaattttggcttttgttgccatagcttttggtgttttagacatgaagtccttgcccatgcctatgtcctgaatg
gtattgcctaggttttcttctagggtttttatggttttaggtctaacatgtaagtctttaatccatcttgaattaattataaggtgtatattataaggtg
taattataaggtgtataattatatattaattataaggtgtatattaattataaggtgtaaggaagggatccagtttcagctttctacatatggctag
ccagttttccctgcaccatttattaaatagggaatcctttccccattgcttgtttttgtcaggtttgtcaaagatcagatagttgtagatatgcgg
cattatttctgagggctctgttctgttccattggtctatatctctgttttggtaccagtaccatgctgttttggttactgtagccttgtagtatagttt
gaagtcaggtagcgtgatggttccagctttgttcttttggcttaggattgacttggcaatgtgggctctttttggttccatatgaactttaaagt
agtttttccaattctgtgaagaaattcattggtagcttgatggggatggcattgaatctataaattaccctgggcagtatggccattttcacaa
tattgaatcttcctacccatgagcgtgtactgttcttccatttgtttgtatcctcttttatttcattgagcagtggtttgtagttctccttgaagaggt
ccttcacatcccttgtaagttggattcctaggtattttattctctttgaagcaattgtgaatgggagttcactcatgatttgactctctgtttgtctg
ttattggtgtataagaatgcttgtgatttttgcacattgattttgtatcctgagactttgctgaagttgcttatcagcttaaggagattttgggctga
gacgatggggttttctagatatacaatcatgtcatctgcaaacagggacaatttgacttcctcttttcctaattgaatacccgttatttccctctc
ctgcctgattgccctggccagaacttccaacactatgttgaataggagtggtgagagagggcatccctgtcttgtgccagttttcaaaggg
aatgcttccagtttttgtccattcagtatgatattggctgtgggtttgtcatagatagctcttattattttgagatacatcccatcaatacctaattt
attgagagtttttagcatgaagagttcttgaattttgtcaaaggccttttctgcatcttttgagataatcatgtggtttctgtctttggttctgtttata
tgctggagtacgtttattgattttcgtatgttgaaccagccttgcatcccagggatgaagcccacttgatcatggtggataagctttttgatgt
gctgctggattcggtttgccagtattttattgaggatttctgcatcgatgttcatcaaggatattggtctaaaattctcttttttgttgtgtctctgt
caggctttggtatcaggatgatgctggcctcataaaatgagttagg**

ten occurrences?

tttttttttttttttgagacggagtctcgctctgtcgcccaggctggagtgcagtg gcgggatctcggctcactgcaagctccgcctcccgggttcacgccattct cctgcctcagcctcccaagtagctgggactacaggcgcccgccactacg cccggctaattttttgtatttttagtagagacggggtttcaccgttttagccgg gatggtctcgatctcctgacctcgtgatccgcccgcctcggcctcccaaag tgctgggattacaggcgt

Length: 277

Occurrences at: 10130003, 11421803, 18695837, 26652515, 42971130, 47398125
In the reversed complement at: 17858493, 41463059, 42431718, 42580925

finally

suffix tree
efficient (linear) storage, but constant ±40
large ‘overhead’

suffix *array* has constant ±5
hence more practical
but has its own complications

naïve n log(n) algorithm not too bad…

suffix array

nittygritty	1	gritty	6
ittygritty	2	itty	8
ttygritty	3	ittygritty	2
tygritty	4	nittygritty	1
ygritty	5	ritty	7
gritty	6	tty	9
ritty	7	ttygritty	3
itty	8	ty	10
tty	9	tygritty	4
ty	10	y	11
y	11	ygritty	5

lexicographic order of the suffixes

sources

Dan **Gusfield**
Algorithms on Strings, Trees, and Sequences
Computer Science and Computational Biology

lists *many* applications for suffix trees
(and extended implementation details)

slides on suffix-trees based on/copied from
Esko **Ukkonen**, Univ Helsinki
(Erice School, 30 Oct 2005)