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
failure links
(suffix = prefix)
top: as ‘automaton’
bottom: as table
• how to determine?
• how to use?
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

KMP computing failure links

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

\[ P \quad \text{Flink}[k] = r \]

\[ P \quad r \quad k \]

\[ P_1 \cdots P_{r-1} = P_{k-r+1} \cdots P_{k-1} \quad \text{maximal } r < k \]

all such values \( r \):

\[ P_1 \cdots P_{r-2} = P_{k-r+2} \cdots P_{k-1} = P_{r_1-r_2+1} \cdots P_{r_1-1} \]

\[ \Rightarrow \text{Flink}[r_1] = r_2 \]
Other methods

- **Boyer-Moore**

  \[ T = \text{marktkoopman} \]
  \[ P = \text{schoenveter} \]

  Work backwards

- **Karp-Rabin ‘fingerprint’**

  hash-value

  \[ a_{i-1} \ a_i \ \ldots \ a_{i+n-1}a_{i+n} \]
  \[ p_1 \ \ldots \ p_n \]

  \[ a_iB^{n-1} + a_{i+1}B^{n-2} + a_{i+n-1}B^0 \]
  \[ a_{i+1}B^{n-1} + \ldots + a_{i+n-1}B^1 + a_{i+n}B^0 \]
exact matching with a set of patterns

\[ P = \{ P_1, \ldots, P_r \} \]
all occurrences in text \( T \)

**Aho Corasick**

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

\[ > \text{no subwords within } P \]
keyword tree - trie

edges ~ letters

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

leaves ~ keywords
failure links

{ potato, tattoo, theater, other }
algorithm: follow the links

edge with incoming \texttt{a}
follow links starting at parent
until outgoing \texttt{a} is found
failure links

\{ potato, tattoo, theater, other \}

breadth first
(level-by-level)
failure links

{ potato, tattoo, theater, other }
(B) preprocessing text
trie vs. suffix tree

abaab  
baab  
aab  
ab  
b

trie

string+suffixes

aab  
ab  
b

suffix tree

• \(|\text{Trie}(T)| = O(|T|)^2\)
• bad example: \(T = a^n b^n\)
• \(\text{Trie}(T)\) like DFA for the suffixes of \(T\)
• minimize DFA \(\rightarrow\) directed acyclic word graph

• only branching nodes and leaves represented
• edges labeled by substrings of \(T\)
• correspondence of leaves and suffixes
• \(|T|\) leaves, hence \(<|T|\) internal nodes
• \(|\text{Tree}(T)| = O(|T| + \text{size(edge labels)})\)
implementation: refer to positions
linear time construction

Weiner (1973) 'algorithm of the year'

McCreight (1976) 

'nittygritty'

ittygritty

ttygritty

tygritty

ygritty

gritty

ritty

itty

tty

ty

y

'online' algorithm
(Ukkonen 1992)
suffix trie for abaab

suffix links

abaab
baab
aab
ab
εb
ε

from here b already exists

next symbol = b
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
ittygritty
ttygritty
tygritty
ygritty
gritty
itty
tty
y
1 2 3 4 5 6 7 8 9 10 11

positions
```
application: longest common substring

T

T'

P

pos

pos'

T and T' suffixes

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
itty 7
ritty 8
tty 9
y 10

Diagram:

```
  gritty
   / \
  y   6
 /   /
itty t  ty
 /   /
itty  y gritty
 /   /
itty gritty gritty
 /   /   
itty gritty gritty gritty
 /   /   /   
itty gritty gritty gritty gritty
 /   /   /   /   
itty gritty gritty gritty gritty gritty
 /   /   /   /   /   
itty gritty gritty gritty gritty gritty gritty
 /   /   /   /   /   /   
itty gritty gritty gritty gritty gritty gritty gritty
 /   /   /   /   /   /   /   
itty gritty gritty gritty gritty gritty gritty gritty gritty
 /   /   /   /   /   /   /   /   
itty gritty gritty gritty gritty gritty gritty gritty gritty gritty
 /   /   /   /   /   /   /   /   /   
itty gritty gritty gritty gritty gritty gritty gritty gritty gritty gritty
```

Numbers:

1 2 3 4 5 6 7 8 9 10 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: $3 \times 10^9$ bases
- suffix tree for Human Genome feasible
**Occurrences at:** 28395980, 28401554r  
**Length:** 2559

ttaggtacatgtgcacacagctcgaggttttgatcatagttatcatacagtgccatagatggtgtgcgcacaccattaactgtcatattagcgtta

ggtatatcctccgagctatctcctcctcctcccctcccccacccccacaacagtcccgggttgtgtgtgttctctgccatagttctca
tttgtcaatctccccacactgtagtgagaacatcgccggttgttgtgtttgtgtgctcgtggaagttgctgagaalactagttttcagcgtcatcctcatcacaatc
tcccctacaagggcatagacactcatcattttttatatgcgtcataagtccatcagttgtatatcgtgacacactttagatgtccctgaatcacaagctgtt
tgaactagttttacgtacgagcaacagtccattttctccattctcctcctccggacaccttgcgtgttgtttttcattttttccatctctctactctct
tatctatgtccagaatggtttgggttataagttctttttaatcctggtgaatctctagatgagaagtcttttttactctctctcttttttacagtttttttttaaactctc
tttgtcagatatatgtgtttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttt
tttttttttttttggagacggagtctcgctctgtcgccgccaggctggagtgctcgagtgcccaggctggagtgcagtg
gcggtatctcggctcactgcaagctccgcctcccgggttcacgccattctcctgcctcgctccttcaagtagctgggactacaggcgcccgccactacgcctccggctaatgggttttttagacggggtttcaccgttttagccgggatggtctcgatctccttgacctctcggtatccgcgccgttcctccaaagtgctggggattacaggcggt

Length: 277

Occurrences at: 10130003, 11421803, 18695837, 26652515, 42971130, 47398125 In the reversed complement at: 17858493, 41463059, 42431718, 42580925
suffix tree
efficient (linear) storage, but constant ±40
\[ \text{large} \; \text{‘overhead’} \]

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

naïve n \log(n) algorithm not too bad…
### Suffix Array

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>gritty</td>
<td>1</td>
</tr>
<tr>
<td>itty</td>
<td>2</td>
</tr>
<tr>
<td>ttygritty</td>
<td>3</td>
</tr>
<tr>
<td>tygritty</td>
<td>4</td>
</tr>
<tr>
<td>ygritty</td>
<td>5</td>
</tr>
<tr>
<td>gritty</td>
<td>6</td>
</tr>
<tr>
<td>itty</td>
<td>7</td>
</tr>
<tr>
<td>nittygritty</td>
<td>8</td>
</tr>
<tr>
<td>ritty</td>
<td>9</td>
</tr>
<tr>
<td>ttygritty</td>
<td>10</td>
</tr>
<tr>
<td>tygritty</td>
<td>11</td>
</tr>
</tbody>
</table>

Lexicographic order of the suffixes:
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)