Pattern matching problem - revisited

- KMP is a provable linear-time algorithm for the pattern-matching problem
- It works in a situation when the pattern is fixed and the text is streaming – the text is not known before the search starts

- Let’s consider a different scenario:
  - text $T$ is known first and it is kept fixed for some time
  - new search patterns are constantly arriving
  - search for each pattern should be as quick as possible
Suffix trees

- Suffix tree of $T$ exposes the internal structure of the input text
- Assuming that the text is re-written in a form of the suffix tree, the pattern matching problem can be performed in time $O(M+k)$, where $M$ is the length of a pattern, and $k$ is the number of occurrences. The search time does not depend on the length of $T$
- In addition, suffix trees provide optimal (linear-time) solutions to numerous complex problems other than pattern matching problem
Tree branch with suffixes

$T = \text{cacao}$
Tree branch with suffixes

$T = \textit{cacao}$
Tree branch with suffixes

$T = \text{cacao}$

While adding a new suffix, we follow the path of matches from the root, and create a new branch only when the next character of a suffix does not match.
Tree branch with suffixes

$T = \text{cacao}$
Tree branch with suffixes

$T = \text{cacao}$
Suffix tree: terminology

\( T = \textit{cacao} \)
Suffix tree - definition

- A *suffix tree* for string \( T \) (of length \( N \)) is a rooted tree with the following properties:
  - \( N \) leaves, numbered 1 to \( N \).
  - Each internal node has at least two children.
  - No two edges out of a node have edge-labels beginning with the same character.
  - For any leaf \( i \), the concatenation of the edge-labels on the path from the root to leaf \( i \) spells out the suffix \( T[i..N] \) of \( T \).
Suffix tree – number of nodes

- A *suffix tree* for string $T$ (of length $N$) is a rooted tree with the following properties:
  - $N$ leaves, numbered 1 to $N$.
  - Each internal node has at least two children.

- Because we go from $N$ leaves to 1 root node replacing at least 2 nodes with one, the entire process takes at most $\log N$ steps: the height of the suffix tree is at most $\log N$.

- **Corollary**: the total number of nodes in the tree is bounded by $2^{\log N} = O(N)$: $N$ leaves and $N$ internal nodes.
Full-text indexing

- Suffix tree is an example of a **full-text index** – the data structure designed for fast search of any substring of a given text.

- All different substrings of $T$ can be found in the suffix tree following the path from the root.
Search for pattern *ca*

*T=cacao*
Another suffix tree

seveves
1 2 3 4 5 6 7 8 9
Another suffix tree

$s e v e n e v e s$

1 2 3 4 5 6 7 8 9

What suffix is missing?
Another suffix tree

\[ s e v e n e v e s \]
\[ 1 2 3 4 5 6 7 8 9 \]

Where is the leaf for \( \tau[9\ldots9]=s \)?
What if we search for pattern \( P=s \)?
Proper suffix tree

We add a special character to the end of T – *sentinel*

The sentinel $ does not occur anywhere in T
Search for $P=\text{eve}$

```
sevences$
12345678910
```

Search in time $O(M+k)$
Search for $P=ne$

Seven evenves $s$

$1 2 3 4 5 6 7 8 9 1 0$

Search in time $O(M+k)$
Activity

- build a tree for $T=banana$
- explain how to search for a pattern $ana$
Space

$T = abcde$

This tree occupies quadratic space!

$1 + 2 + 3 + \ldots + N = O(N^2)$
Trick – re-label the edges

\[ s \rightarrow 1-1 \]
Trick – re-label the edges

\[ s \ e \ v \ e \ n \ e \ v \ e \ s \]$ 

1 2 3 4 5 6 7 8 9 1 0

\[ e \rightarrow 2-2 \]
Trick – re-label the edges

$seveneves$

1 2 3 4 5 6 7 8 9 10

neves$

R

2-2

neves$

3-4

neves$

$eveneves \rightarrow 2-10$

neves$

1-1

ve

4 8 2 6 5

9

1 3 7

s$
The total number of leaves is $O(N)$, the total number of internal nodes is $O(N)$

With a constant storage space per edge – the suffix tree can be stored in linear space
Search

s e v e n e v e s $
1 2 3 4 5 6 7 8 9 1
0

In order to find an outgoing edge which starts with e, we check which of $T[2]$, $T[5]$, $T[1]$ or $T[3]$ is e.

The search is as efficient as before, assuming constant time access to each character of $T$
Search with suffix trees: summary

- If we have preprocessed text $T$ into its suffix tree, we can answer a Boolean query about an occurrence of a pattern of length $M$ by performing only $M$ steps, independently of the length of the text $T$.

- In order to report all $k$ occurrences of a pattern, the traversal of a corresponding subtree is performed in $O(k)$ steps.
Readings

- Text book *Chapter 5*
- [http://www.allisons.org/ll/AlgDS/Tree/Suffix/](http://www.allisons.org/ll/AlgDS/Tree/Suffix/)