UNIVERSITÄT DES SAARLANDES

ZBI ZENTRUM FÜR BIOINFORMATIK

Assignment 04

Algorithms for Sequence Analysis

Sven Rahmann, Jens Zentgraf and Johanna Schmitz

05.05.2025, due **12.05.2025**

04.1: Nodes and Edges in Suffix Trees (2 Theory)

Prove the following lemma:

Lemma

The suffix tree of string $s\$$ with $|s\$| = n \geq 2$ has exactly n leaves,
at most $n - 1$ inner nodes (without the root) and at most $2(n - 1)$ edges.

Hints

- Give names to the number of inner nodes and edges.
- Count the edges twice: as incoming edges and as outgoing edges.

04.2: Supermaximal repeats (6 Theory)

Task

Give a linear-time algorithm to find all **supermaximal repeats** of a string s,
using its enhanced suffix array and the definitions below.
To develop the algorithm, characterize supermaximal repeats
in terms of intervals in the enhanced suffix array.

04.2: Definitions

A triple (p, p', m) of two positions $p < p'$ and a length $m \geq 1$ is called

- a **repeated pair** if and only if $s[p : p + m] = s[p' : p' + m] =: w$ (using Python half-open indexing, i.e., the right boundary is not included),
- **right-maximal** if $s[p + m] \neq s[p' + m]$,
- **left-maximal** if $p = 0$ or $s[p - 1] \neq s[p' - 1]$,
- **maximal** if it is both left-maximal and right-maximal.

Then the repeated substring w is called a [left-/right-]maximal **repeat** of length m at p, p'.

A **supermaximal repeat** is a maximal repeat that is not a substring of any other repeat.

04.2: Example and Notes

Take $s = \texttt{mambamba\$}$.
Maximal repeats with corresponding maximal repeated pairs:
a: $(1, 7, 1)$, amba: $(1, 4, 4)$, m: $(0, 2, 1), (0, 5, 1)$.
Supermaximal repeats: amba only; the other maximal repeats are substrings of amba.

Note that there can be several maximal repeated pairs for the same string;
there can be also other (non-maximal) repeated pairs for the string.
It suffices that there is at least one maximal repeated pair for a string
to call the string a maximal repeat.
The repeated pair $(2, 5, 1)$ is not maximal for $w = \texttt{m}$, but $w = \texttt{m}$ is still a maximal repeat because other maximal repeated pairs exist for it, such as $(0, 2, 1)$, see above.

It may help to think about this problem in terms of the suffix tree first
(which inner nodes are supermaximal repeats?),
and then translate your insights to suffix array intervals.

04.3: From ESA to Suffix Tree (4 Theory + 4 Programming)

Theory Part (4 Theory)

[A] Given the `pos` and `lcp` arrays, how can you reconstruct the suffix tree?
Formally describe a left-to-right scanning algorithm to do this.
Analyze the running time of your algorithm.

[B] Run your algorithm by hand on the following input:

$$\texttt{pos} = [10, 9, 4, 7, 2, 5, 0, 8, 3, 6, 1], \quad \texttt{lcp} = [-1, 0, 1, 1, 3, 3, 5, 0, 2, 2, 4, -1]$$

Additionally, find a fitting text (over any alphabet).

Reminder

The `lcp` array gives the string depth of the deepest internal node of the tree between two lexicographically adjacent suffixes.

Programming Part (4 Programming)

Implement your algorithm to construct the suffix tree in DOT format
(explained **here**; see also the example on the **next slide**).
Use circles for inner nodes and boxes for leaves.
Annotate leaves with suffix starting positions, as usual.
Annotate inner nodes with `i`, a running ID number, an underscore,
and their string depth (the root is `i0` with a depth of 0, so `i0_0`).
Annotate edges with the substring that they represent.

There are many programs to visualize the corresponding graphs,
for instance the (py)graphviz package that contains the command `dot`.
It can be used like this, resulting in the figure on the next slide:

```
dot -Tpdf input.dot > output.pdf
```

Hint: Do it in 2 passes: First build the tree, then output it in DOT format.

04.3: Programming Part Example

The DOT file for the tree of `TAATA$` should look similar to the following:

```
digraph mytree {
  i0_0 -> 5 [label="$"]
  5 [shape=box]
  i0_0 -> i1_1 [label="A"]
  i1_1 -> 4 [label="$"]
  4 [shape=box]
  i1_1 -> 1 [label="ATA$"]
  1 [shape=box]
  i1_1 -> 2 [label="TA$"]
  2 [shape=box]
  i0_0 -> i2_2 [label="TA"]
  i2_2 -> 3 [label="$"]
  3 [shape=box]
  i2_2 -> 0 [label="ATA$"]
  0 [shape=box]
}
```