Assignment 05

Algorithms for Sequence Analysis

Sven Rahmann and Jens Zentgraf

12.05.2025, due **19.05.2025**

05.1: Distinct substrings of a string (4 Theory)

Task

Give a **linear-time** algorithm
to compute the total number of distinct substrings of a string s (without sentinel).
(Do not count the empty string.)

Hint

Think in terms of $s\$$ and the enhanced suffix array,
but do not count substrings with the sentinel \$.

Example

`baaa` has 7 distinct non-empty substrings: `a`, `b`, `aa`, `ba`, `aaa`, `baa`, `baaa`.

05.2: SAIS Example (4 Theory)

For text $T = \texttt{ACATACATACATACCATACATACAT\$}$, do the following:

1. Compute the type array, LMS-substrings, and the string's representation based on the reduced alphabet.
2. Is the order of suffixes at LMS-positions determined by the LMS-substrings computed in the previous subtask?
   - If yes, show the suffix array of the reduced text.
   - If not, illustrate how the induced sorting algorithm recurses to find the the suffix array of the reduced text,
3. Compute the suffix array by induced sorting from the sorted LMS suffixes.

It is recommended to do this task by pen and paper.

05.3: Amortized Analysis (4 Theory)

We have seen a few examples of amortized analysis of algorithms:

- Knuth-Morris-Pratt, Aho-Corasick: number of times we follow an lps link
- Ukkonen's suffix tree construction: number of downward hops
- Kasai's algorithm: number of character comparisons

The common situation is:

- An algorithm proceeds in many (say, n) iterations.
- Each single iteration is potentially expensive (time $t_i \in O(m)$).
- A naive analysis would yield a total time of $O(mn)$.
- However, one can show that in total, $\sum_i t_i = O(n)$, because most iterations are actually quick.
- One has to find reasons why the sum is small.

Another Example: Dynamic Arrays (Python Lists)

Python's lists grow (and shrink) dynamically.
Each list has a certain capacity C and a certain number of objects $k \leq C$.
When we create a new (empty) list, $k = 0$ but $C = 1$.

When appending a new object and $k < C$, the object is put into the available space in constant time. If $k = C$, however, then new space must be allocated, and the existing data must be copied in time proportional to the current capacity $O(C)$.

Analyze three strategies for expanding the list when this happens:

1. $C \leftarrow C + 1$ (expand capacity by one slot just for the new element)
2. $C \leftarrow 2 \cdot C$ (double the capacity)
3. $C \leftarrow \lceil 1.01 \cdot C \rceil$ (increase capacity by a small factor, but at least by 1 slot)

In O-notation, what is the total amount of time spent after n append operations?

05.4 Ukkonen's Algorithm (8 Programming, due 26.05.)

- Given a string, build its suffix tree with Ukkonen's linear time algorithm.
- Check that the string consists only of ASCII characters with ASCII code ≥ 37, except for the last character, and ends with '$' (ASCII code 36).
- Use a simple Python dict for accessing the children of a node.
- You may use classes or just tuples to represent nodes and edges.
- Start early, as this is a long-lasting task.
- Use many small independent functions,
  like for splitting an edge, inserting a node, inserting a new edge, etc.
- For automated tests, track the number of nodes and leaves after each phase.