Text indexing

Preprocess a text string $T$ (length $n$) to answer pattern matching queries...

After preprocessing, a query is a pattern $P$ (length $m$), the output is a list of all matches in $T$.

- Last lecture we saw that the text indexing problem can be solved using a suffix tree which uses $O(n)$ space (when it's stored compacted)
- Queries take $O(m + occ)$ time when the alphabet size is constant
  - occ is the number of occurrences (matches)
- Suffix trees can be constructed in $O(n)$ time (but we only saw how to achieve $O(n^2)$ time)

The symbols must have an order throughout we will use alphabetical order

In lexicographical ordering we sort strings based on the first symbol that differs:

\[
\begin{align*}
\text{a} &< \text{b} \quad \text{a} &< \text{c} \\
\text{a} &< \text{b} \quad \text{a} &< \text{b} \\
\end{align*}
\]

If the symbols don’t have a natural order, we use their binary representation in memory

The suffix array is much smaller than the suffix tree (in terms of constants)
Searching in the Suffix Array

The suffix array

The DC3 method

Retrieval of the Pattern

The suffix array

The DC3 method

Retrieval of the Pattern
How do we merge these?

Introduce a new "filler symbol" $.

Number the blocks in lexicographical order

Concatenate $R_1$ and $R_2$ to obtain $R'$:

compute the suffix array of $R'$:

$0 \ 1 \ 6 \ 4 \ 2 \ 5 \ 3 \ 7$

How do we find the ordering of the suffixes from $R'$?

Recursion!

we then sort in $O(n)$ time using radix sort

the suffix array of $R'$:

after relabelling, $0 \ 1 \ 6 \ 4 \ 2 \ 5 \ 3 \ 7$

we have the suffix array of just the suffixes from $B_1 \cup B_2$

how do we find the ordering of the suffixes from $B_0$?

(where $i \mod 3 = 0$)
The DC3 method

**Theorem**
The DC3 algorithm constructs a suffix array in $O(n)$ time.

**Proof**
Suppose $T(n)$ is the running time. We have

$$T(n) = T(2n/3) + O(n)$$

Solving this recurrence gives $T(n) \in O(n)$.

Overall this merging phase takes $O(n)$ time

(because processing each suffix takes $O(1)$ time)

Finding an occurrence of a pattern (length $m$) takes $O(m \log n)$ time

Finding all occurrences takes $O(m \log n + \text{occ})$ time

where occ is the number of occurrences

This can be further improved to $O(m + \log n + \text{occ})$ time

(using LCP queries which we will see in a future lecture)

We can construct the suffix array in $O(n)$ time