IN4120

RAMI AL KAWADRI

SUFFIX ARRAYS: A NEW METHOD FOR ON-LINE STRING SEARCHES

Plan

- Presenting two algorithms from the paper: “Suffix arrays: A new method for on-line string searches”.
- The problem we are trying to solve is finding all occurrence of a string W within a string A using suffix arrays.
- The two algorithms presented here are:
  - Prefix doubling algorithm for sorting suffix array
  - Searching using LCP (longest common prefix)
- I assume that everyone knows what binary search is and what suffix arrays are since we had to implement them in assignment b.

Prefix doubling - notations

- Let $A = a_1a_2...a_N$ be a string.
- A_i is the suffix starting from the index i.
- a_i is the i^{th} character of A.
- Let A = *hello*, then A_2 = ***llo***, and a_2 = ***l***.
- We say $A <_p B$ if the first p-symbols of A are lexicographically less than the first p-symbols of B.
- We define $A =_p B$, $A >_p B$, etc… in a similar way.
- Let A be **flight** and B be **flower** then:

  $A <_3 B$ because **fli** < **flo**.

  $A =_2 B$ because **fl** = **fl**.

Prefix doubling

- An algorithm for sorting suffix array.
- The algorithm uses the fact that the suffix array contains suffixes.
- With some optimization the algorithm gives *O(N)* expected time for sorting the suffixes.
- I implemented the algorithm without the extra optimization, so the runtime in my code will be *O(n log n)*, but it represents the main idea.

Prefix doubling (cont.)

- The sorting is done in $\log_2(n + 1)$ stages.
- In the first stage we sort the suffixes into buckets according to their first symbol.
- In each stage the number of sorted symbols doubles.
- That is in the first stage, suffixes are sorted according to first character.
- In the 2nd stage according to the first two characters.
- In the third stage according to the first 4 characters etc...
- We number the stages 1, 2, 4, … to indicate the number of affected symbols.
- Thus, in the H^{th} stage the suffixes are sorted according to the $\leq_H$ order.

Prefix doubling (cont.)

- Now assume we were able to sort according to the $\leq_H$ order in the H^{th} stage, we need to find out how to sort according to the $\leq_{2H}$ order (that is the next stage).
- Observe that when A_i and A_j are in the same bucket in the H^{th} stage, then $A_i =_H A_j$.
- We need to sort the two suffixes according to the next H-symbols. That is the next H-symbols from A_{i+H} and A_{j+H}.
- But observe the fact that A_{i+H} and A_{j+H} are also suffixes and we know their relative order. Thus, we can use this fact for sorting!
- We start with the first bucket, which must contain the smallest suffixes according.
- We take the first suffix A_i and then move A_{i-H} to be the first in its 2H bucket and mark this fact. We do this for all elements.
- We do this as long as H < n, when H > n then we have sorted all suffixes. (code)

LCP search - intro

- Let $A = a_1 a_2 ... a_N$ be a string.
- We want to find all occurrence of $W = w_1 w_2 ... w_p$.
- Let **Pos** be the suffix array of A.
- If we find these two indexes:
  - $L_W = \min(k: W \leq_p A_{Pos[k]} \text{ or } k = N)$
  - $R_W = \max(k: A_{Pos[k]} \leq_p W \text{ or } k = -1)$
- Then all elements between those indexes contains W.

- We define **lcp**(v, w) to be the length of the **longest common prefix** of the two strings v and w. **lcp**(**fl**ight, **fl**ower) = 2.
- We will use lcp and binary search to find L_w with runtime $O(P + \log n)$ instead of $O(n \log n)$.
- Observe that (L_M, M, R_M) is always unique in binary search.
- Define: (those values can be precomputed while searching)
  - $Llcp[M] = lcp(A_{Pos[M]}, A_{Pos[LM]})$.
  - $Rlcp[M] = lcp(A_{Pos[M]}, A_{Pos[RM]})$.

LCP search (cont.)

- Before searching let:

  $l = lcp(A_{Pos[0]}, W)$ and **$r = lcp(A_{Pos[N-1]}, W)$**.

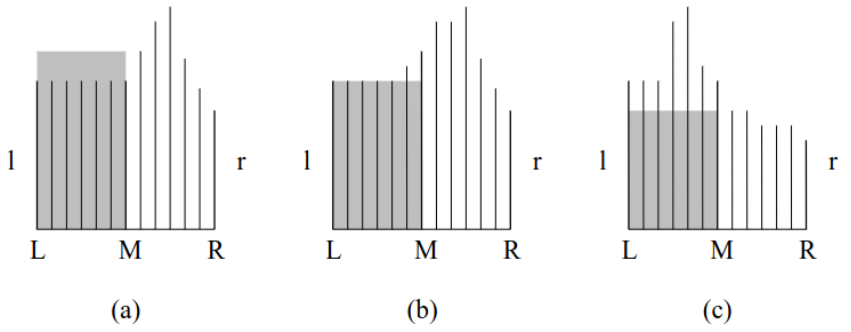
- Now we will use this to help us decide which half to search through using binary search to find L_w.

- Let $l \geq r$, and let M be the middle element in current binary search.

- Remember that: $Llcp[M] = lcp(A_{Pos[M]}, A_{Pos[LM]})$.

- We have three cases:

  1. Llcp[M] > l: W must be in the right part.
  2. Llcp[M] = l: we need to compare the next symbol after l to decide which half to continue.
  3. Llcp[M] < l: W must be in the left part.



Resources

- Suffix arrays: A new method for on-line string searches (Udi Manber, Gene Myers, 1989).
- Suffix Arrays (Freie Universität Berlin)

  http://www.mi.fu-berlin.de/wiki/pub/ABI/RnaSeqP4/suffix-array.pdf
- C++ implementation of prefix doubling

  https://gist.github.com/sumanth232/e1600b327922b6947f51