Weekplan: Suffix Trees

Philip Bille

References and Reading

[1] Linear work suffix array construction, J. Kärkkäinen, P. Sanders, S. Burkhardt, J. ACM, 2006.

[2] Scribe notes from MIT.

[3] Algorithms on Strings, Trees, and Sequences, Chap. 5-9, D. Gusfield

[4] On the sorting-complexity of suffix tree construction, M. Farach-Colton, P. Ferragina, S. Muthukrishnan, J. ACM, 2000

We recommend reading [1] and [2] in detail. [3] provide an extensive list of applications of suffix trees and [4] is the first suffix-tree construction algorithm matching the sorting time bound.

Exercises

1 Suffix Trees for Multiple Strings The suffix tree for a *set* of strings $S_1, \ldots, S_k$ of total length n over alphabet Σ is the compact trie of all suffixes of the strings $S_1\$_1, S_2\$_2, \ldots, S_k\$_k$. Each $\$_i$ is a special character not in Σ. The label of each leaf is a set of pairs. Each pair consists of a string index i and position j indicating that the path corresponds to suffix j in string S_i. Show how to construct the suffix tree for a set of strings. *Hint:* Concatenate and use the suffix tree construction algorithm for a single string.

2 Common Substrings and Repeats Solve the following exercises.

2.1 A *repeat* in a string S is a substring R that occurs at least twice in S. Show how to efficiently compute the length of a longest substring of S that is a repeat.

2.2 Given strings S_1 and S_2 a *longest common substring* is a substring of both S_1 and S_2 of maximal length. Show how to efficiently compute the length of a longest common substring of S_1 and S_2.

3 Suffix Tree Construction Bounds Solve the following exercises.

3.1 Show that any algorithm for suffix tree construction of a string of length n over an alphabet Σ must use $\Omega(\text{sort}(n, |\Sigma|))$ worst-case time. *Hint:* Show that an algorithm using $o(\text{sort}(n, |\Sigma|)$ time would lead to a contradiction.

3.2 Suppose that we drop the requirement that sibling edges are sorted from left-to-right. Show how construct such a suffix tree in $O(n)$ expected time. *Hint:* hash.

4 Restricted Suffix Search Let S be a string of length n over alphabet Σ. Give an efficient data structure for S that supports the following query:

- rsearch(P, i, j): report the starting positions of occurrences of string P in $S[i, j]$.

5 LSD and MSD Radix Sort Radix sort that process digits in right-to-left order is called *LSD radix sort*. If we instead process digits in left-to-right order we call the algorithm *MSD radix sort*. Solve the following exercises.

5.1 Show that LSD radix sort correctly sorts any input.

5.2 Explain why each step in LSD radix sort must use a stable sorting algorithm.

5.3 Show that MSD radix sort does not correctly sort any input.

5.4 Explain how to modify MSD radix sort to sort correctly.

6 Odd-Even Sampling Suppose we modify the sampling of suffixes in the DC3 algorithm such that the sampled and non-sampled suffixes are those starting at even and odd positions, respectively. Determine if the algorithm still works, i.e., show that it still works or explain where it fails.

7 Suffix Arrays Let S be a string of length n. The *suffix array* is the array SA of length $n+1$ containing the left-to-right sequence of labels of leaves in the suffix tree. Given the SA and S show how to support search(P) for a string P of length m in time $O(m \log n + \mathrm{occ})$.

8 Approximate String Matching with Hamming Distance The *Hamming distance* between two equal length strings S_1 and S_2 is the number of positions i such that $S_1[i] \neq S_2[i]$. Let P and S be strings over alphabet Σ of lengths m and n, respectively. Given a parameter k, show how to compute all ending positions of substrings in S whose Hamming distance to P is at most k. *Hint:* Longest common extensions.

9 String Predecessors Let $\mathcal{S} = S_1, \ldots, S_k$ be a set of strings of total length n over alphabet Σ. We are interested in a data structure for $\mathcal{S}$ that supports the following query.

- stringpred(P): report the lexicographical predecessor of string P.

Solve the following exercises.

9.1 Give a compact data structure that supports stringpred(P) efficiently.

9.2 Show that any data structure must use $\Omega(|P| + \mathrm{pred}(k, |\Sigma|))$ worst-case time for the stringpred(P) query, where $\mathrm{pred}(k, |\Sigma|)$ denotes the time for a predecessor query on a set of size k from a universe Σ. How does this bound match your upper bound from above?