Weekplan: Compressed Pattern Matching

Patrick Hagge Cording

References and Reading

[1] *"Random Access to Grammar-Compressed Strings and Trees"*, Philip Bille, Gad M. Landau, Rajeev Raman, Kinihiko Sadakane, Srinivasa Rao Satti, Oren Weimann.

[2] *"Real-time Traversal in Grammar-based Compressed Files"*, Leszek Gasieniec, Roman Kolpakov, Igor Potapov, Paul Sant.

[3] Exercise 7, *Weighted Level Ancestor*, from week 5.

[4] *"Perfect hashing for strings: Formalization and algorithms"*, Martin Farach, S. Muthukrishnan.

Read the introduction and section 2 of [1] carefully. We will use a solution to the weighted ancestor problem; see [3] and [4].

Exercises

1 Compressed k-mismatch problem. Explain how to solve the (decision version) of the k-mismatch problem on a string compressed by an SLP in $O(nMk)$ time.

2 Relevant substring Lemma. Prove the relevant substring lemma.

3 Reporting occurrences. The compressed pattern matching algorithms shown in the lecture returns YES if there is a match and NO otherwise. Extend one of the algorithms to report the number of occurrences within the same time and space bounds.

4 Compressed rank and select. You are given an SLP of size n compressing a string S of size N over an alphabet of size σ. The rank and select operations are defined as follows.

$\textsc{Rank}(i, c)$**:** the number of characters c in $S[1, i]$.

$\textsc{Select}(i, c)$**:** the index of the i-th occurrence of c in S.

Describe a data structure that supports $\textsc{Rank}(i, c)$ and $\textsc{Select}(i, c)$; both in $O(h)$ time. Your data structure should use $O(n\sigma)$ space.

5 Compressed subsequence recognition. A string S' is a subsequence of S if S' can be obtained by deleting characters from S.

You are given a SLP of size n compressing a string S of size N and a pattern string P of length M. S and P contain characters from an alphabet of size σ. The compressed subsequence recognition problem is to decide if P is a subsequence of S.

5.1 The query $\textsc{FindNext}(i, c)$ returns the index of the first occurrence of the character c after position i in S. Describe a data structure that uses $O(n\sigma)$ space that can answer a $\textsc{FindNext}(i, c)$ query in $O(h)$ time.
Hint: Your data structure should support random access and a $\textsc{FindNext}(i, c)$ query should start by using an $\textsc{Access}(i)$ query as a subroutine.

5.2 [$**$] Describe a data structure that uses $O(n\sigma)$ space that can answer a FINDNEXT(i,c) query in $O(\log N \log\log N)$ time.

5.3 Give an algorithm for the compressed subsequence recognition problem that uses FINDNEXT(i,c). What is its running time and space usage?

6 Random access to an LZ78 compressed string. We have seen how to support ACCESS(i) in an SLP in $O(\log N \log\log N)$ time and linear space. In fact, this can be reduced to $O(\log N)$ time in the same space with a much more elaborate data structure [1]. Recently, it has been shown that ACCESS(i) can be performed in $O(\log N / \log\log N)$ time if we allow the data structure to use more space. This also happens to be optimal for a subset of SLPs, i.e., we can not hope to find a faster solution for ACCESS(i) in SLPs.

Now we return to the LZ78 compression. You are given a LZ78 factorization with z factors of a string S of size N. Describe a data structure that supports ACCESS(i) in $O(\log\log N)$ time and $O(z)$ space.

7 Semi-dynamic compression. In semi-dynamic compression we would like to be able to add characters to the compressed string without having to decompress the entire string, add the character, and compress the string again.

Given an SLP compressing a string S, we would like to support the following operations.

APPEND(c)**:** appends the character c to S.

INSERT(i,c)**:** inserts the character c to position i of S. The character in position i not deleted. After the operation $S = S[1,i-1]cS[i,N]$.

7.1 Give a fast implementation of APPEND(c). How many new nodes are added to the SLP?

7.2 Show how to implement INSERT(i,c) in $O(h)$ time. How many new nodes are added to the SLP?