Compressed and Practical Data Structures for Strings

In this dissertation, I will cover a number of different topics related to strings in compressed and practical settings. I will first present some fundamental techniques from the area, and then cover 6 different topics within the area. A short introduction to each of these topics is given in the following. Finger Search in Grammar-Compressed Strings. Grammar-based compression, where one replaces a long string by a small context-free grammar that generates the string, is a simple and powerful paradigm that captures many popular compression schemes. Given a grammar, the random access problem is to compactly represent the grammar while supporting random access, that is, given a position in the uncompressed string report the character at that position. We study the random access problem with the finger search property, that is, the time for a random access query should depend on the distance between a specified index \( i \) called the finger, and the query index \( j \). We consider both a static variant, where we first place a finger and subsequently access indices near the finger efficiently, and a dynamic variant where also moving the finger such that the time depends on the distance moved is supported. Let \( n \) be the size of the grammar, and let \( N \) be the size of the string. For the static variant we give a linear space representation that supports placing the finger in \( O(\log N) \) time and subsequently accessing in \( O(\log D) \) time, where \( D \) is the distance between the finger and the accessed index. For the dynamic variant we give a linear space representation that supports placing the finger in \( O(\log N) \) time and accessing and moving the finger in \( O(\log D + \log \log N) \) time. Compared to the best linear space solution to random access, we improve a \( O(\log N) \) query bound to \( O(\log D) \) for the static variant and to \( O(\log D + \log \log N) \) for the dynamic variant, while maintaining linear space.

As an application of our results we obtain an improved solution to the longest common extension problem in grammar compressed strings. To obtain our results, we introduce several new techniques of independent interest, including a novel van Emde Boas style decomposition of grammars. Compressed Indexing with Signature Grammars. The compressed indexing problem is to preprocess a string \( S \) of length \( n \) into a compressed representation that supports pattern matching queries. That is, given a string \( P \) of length \( m \) report all occurrences of \( P \) in \( S \). We present a data structure that supports pattern matching queries in \( O(m + \text{occ}(lg \lg n + \lg z)) \) time and \( O(z \log(n/z)) \) space where \( z \) is the size of the LZ77 parse of \( S \) and \( > 0 \), when the alphabet is small or the compression ratio is at least polynomial. We also present two data structures for the general case; one where the space is increased by \( O(z \log z) \), and one where the query time changes from worst-case to expected. In all cases, the results improve the previously best known solutions. Notably, this is the first data structure that decides if \( P \) occurs in \( S \) in \( O(m) \) time using \( O(z \log(n/z)) \) space. Our results are mainly obtained by a novel combination of a randomized grammar construction algorithm with well known techniques relating pattern matching to 2D-range reporting. Dynamic Relative Compression, Dynamic Partial Sums, and Substring Concatenation. Given a static reference string \( R \) and a source string \( S \), a relative compression of \( S \) with respect to \( R \) is an encoding of \( S \) as a sequence of references to substrings of \( R \). Relative compression schemes are a classic model of compression and have recently proved very successful for compressing highly-repetitive massive data sets such as genomes and web-data. We initiate the study of relative compression in a dynamic setting where the compressed source string \( S \) is subject to edit operations. The goal is to maintain the compressed representation compactly, while supporting edits and allowing efficient random access to the (uncompressed) source string. We present new data structures that achieve optimal time for updates and queries while using space linear in the size of the optimal relative compression, for nearly all combinations of parameters. We also present solutions for restricted and extended sets of updates. To achieve these results, we revisit the dynamic partial sums problem and the substring concatenation problem. We present new optimal or near optimal bounds for these problems. Plugging in our new results we also immediately obtain new bounds for the string indexing for patterns with wildcards problem and the dynamic text and static pattern matching problem. Succinct Partial Sums and Fenwick Trees. We consider the well-studied partial sums problem in succinct space where one is to maintain an array of \( n \) k-bit integers subject to updates such that partial sums queries can be efficiently answered.

We present two succinct versions of the Fenwick Tree—which is known for its simplicity and practicality. Our results hold in the encoding model where one is allowed to reuse the space from the input data. Our main result is the first that only requires \( nk + o(n) \) bits of space while still supporting sum/update in \( O(\log b \ n) / O(\log \log \ n) \) time where \( 2 \leq b \leq \log O(1) \). The second result shows how optimal time for sum/update can be achieved while only slightly increasing the space usage to \( nk + o(nk) \) bits. Beyond Fenwick Trees, the results are primarily based on bit-packing and sampling—making them very practical—and they also allow for simple optimal parallelization. Fast Dynamic Arrays. We present a highly optimized implementation of tiered vectors, a data structure for maintaining a sequence of \( n \) elements supporting access in time \( O(1) \) and insertion and deletion in time \( O(n) \) for \( > 0 \) while using \( o(n) \) extra space. We consider several different implementation optimizations in C++ and compare their performance to that of vector and multi set from the standard library on sequences with up to 108 elements. Our fastest implementation uses much less space than multi set while providing speedups of 40× for access operations compared to multi set and speedups of 10,000× compared to vector for insertion and deletion operations while being competitive with both data structures for all other operations. Parallel Lookups in String Indexes. Here we consider the indexing problem on the parallel random access machine model. Recently, the first PRAM algorithms were presented for looking up a pattern in a suffix tree. We improve the bounds, achieving optimal results for all parameters but the preprocessing. Given a text \( T \) of length \( n \) we create a data structure of size \( O(n) \) that answers pattern matching queries for a pattern \( P \) of length \( m \) in \( O(\log m) \) time and \( O(m) \) work.

General information

State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic
Contributors: Christiansen, A. R., Bille, P., Gertz, I. L.
Number of pages: 149
Publication date: 2018