Weekplan: Compression

Philip Bille

References and Reading

[1] Introduction to Data Compression, Guy E. Blelloch.

[2] Improved Approximate String Matching and Regular Expression Matching on Ziv-Lempel Compressed Texts, P. Bille, R. Fagerberg, and I. L. Gørtz. TALG, 2009.

[3] Random Access to Grammar-Compressed Strings and Trees, P. Bille, G. M. Landau, R. Raman, K. Sadakane, S. Rao Satti, O. Weimann, SICOMP, 2015.

We recommend reading [1] for an overview of data compression. [2] and [3] provide concise definitions of LZ78 and grammar compression and demonstrate core techniques for compressed computation.

Exercises

1 Lempel-Ziv Encoding and Decoding Solve the following exercises.

1.1 [w] Encode the string `bcabccabccab$` using LZ77 and LZ78.

1.2 [w] Decode the following LZ77 encoded string

$$(\mathtt{a}), (\mathtt{b}), (2, 1, \mathtt{a}), (3, 1, \mathtt{b}), (4, 8, \mathtt{b})(\$).$$

1.3 Draw the LZ78 trie and decode the following LZ78 parse.

$$(\mathtt{a}), (\mathtt{b}), (1, \mathtt{a}), (2, \mathtt{b}), (3, \mathtt{b}), (2, \mathtt{a}), (1, \mathtt{b}), (4, \$).$$

2 Compression Ratios for Lempel-Ziv Let S be a string of length n. We are interested in the best compression ratios for Lempel-Ziv compression schemes. Solve the following exercises.

2.1 What is the smallest number of phrases in the LZ77 parse of S?.

2.2 What is the smallest number of phrases in the LZ78 parse of S?

3 Greedy Optimality LZ77 *greedily* selects the longest substring starting before the current position as a phrase in each step. A natural question is to ask whether this is the best strategy, i.e., can there be another choice of phrases that results in fewer phrases in total. Show that the greedy choice is always optimal. *Hint:* assume that there is a better parsing into phrases and compare it with the greedy.

4 Highly-Repetitive String Collections Let S be a string of length n and D be a collection of $d-1$ copies of S. On the copies, a total of k *edits* have been applied. An edit is either a deletion of a character, an insert of a new character, or a change of a character. Thus the total size of the collection is nD. Show how to compress the collection and give an upper bound on the compressed size.

5 Random Access in Lempel-Ziv Let S be a string of length n and let z_{77} and z_{78} be the number of phrases in the LZ77 and LZ78 parse of S, respectively. Consider the following operation.

- access(i): return the character $S[i]$.

We are interested in compressed data structures that support the access operation efficiently. Solve the following exercises.

5.1 Give a data structure that uses $O(z_{77})$ space and supports access in $O(z_{77})$ time.

The *level ancestor problem* is to preprocess a tree into a data structure supporting *level ancestor queries*, that is, given a node a node v and an integer k, return the kth ancestor of v. Assume in the following that you have a linear space solution that support queries in constant time.

5.2 Give a data structure that uses $O(z_{78})$ space and supports random access in $O(\log \log n)$ time.

5.3 Extend the solution from the previous exercise to access *any substring* of the string in time $O(\log \log n + \ell)$, where ℓ is the length of the substring.

6 Re-Pair Compression

6.1 [w] Run the Re-Pair compression algorithm on the string `ababababcbccccddcabababaaa`.

6.2 [w] Consider the following grammar G.

$$\begin{aligned} X_1 &= \texttt{ab} \\ X_2 &= X_1\texttt{a} \\ X_3 &= \texttt{a}X_2 \\ X_4 &= X_2X_6 \\ X_5 &= \texttt{ca} \\ X_6 &= X_5\texttt{a} \\ X_7 &= X_3X_4 \end{aligned}$$

Draw the parse tree and decode the string for G.

7 Grammar Compression Rate Suppose you are given a string S of size n. What is the best possible grammar compression size?

8 Random Access in Grammars Let S be a string of length n, and let G be a grammar of size g representing S. The *size* of a rule X is the length of the string generated by X. Suppose that G is binary and is balanced, that is, the righthand side of each rule has two symbols unless it's a terminal (leaf in the parse tree), and the size the rules on the righthand side differ by at most a factor 2. Show how to solve the random access problem for G in $O(g)$ space and $O(\log n)$ time.

9 Re-Pair Speed Let S be a string of length n. Solve the following exercises.

9.1 [w] Give a straightforward algorithm that implements the Re-Pair compression in $O(n^2)$ time.

9.2 Give a data structure for a dynamic string S that allows efficient replacement of an arbitrary pair. Specifically, we want a compact representation of S that supports the following operation.

- replace(`ab`, X): replace all (non-overlapping) occurrences of the pair `ab` with symbol X.

The data structure should use time $O(f_{\texttt{ab}})$, where $f_{\texttt{ab}}$ is the frequency of `ab`. *Hint:* maintain string as linked list. Store a dictionary of pairs and for each pair store pointer to occurrences in linked list.

9.3 Show how to implement Re-Pair in $O(n \log n)$ time. *Hint:* extend the solution from the previous exercise to maintain the frequencies of pairs in sorted order using a priority queue.

9.4 [$*$] Show how to implement Re-Pair in $O(n)$ time. *Hint:* Exploit properties of the frequencies to efficiently implement the priority queue.

10 Re-Pair Cut-Off The original Re-Pair algorithm stops as soon as all pairs in the current string are unique. The output is the final string and the sets of rules generated so far. How does the compression rate of this version compare to the one presented?

11 LZ78 and Grammar Compression Let T be a LZ78 trie of size t representing a string S of length n. Show how to convert T into a grammar of size $O(t)$ that encodes the string S.