FINGERPRINTS IN COMPRESSED STRINGS

Philip Bille[1], Patrick Hagge Cording[1], Inge Li Gørtz[1], Benjamin Sach[2], Hjalte Wedel Vildhøj[1] and Søren Vind[1]

[1]Technical University of Denmark, DTU Compute, {phbi,phaco,inge,hwvi,sovi}@dtu.dk
[2]University of Warwick, Department of Computer Science, sach@dcs.warwick.ac.uk

INTRODUCTION

Massive amounts of data are stored in all kinds of string formats, requiring vast quantities of storage space. These strings are often highly compressible.

Compression can reduce storage space significantly. However, compression complicates things when we want to answer queries on the decompressed string.

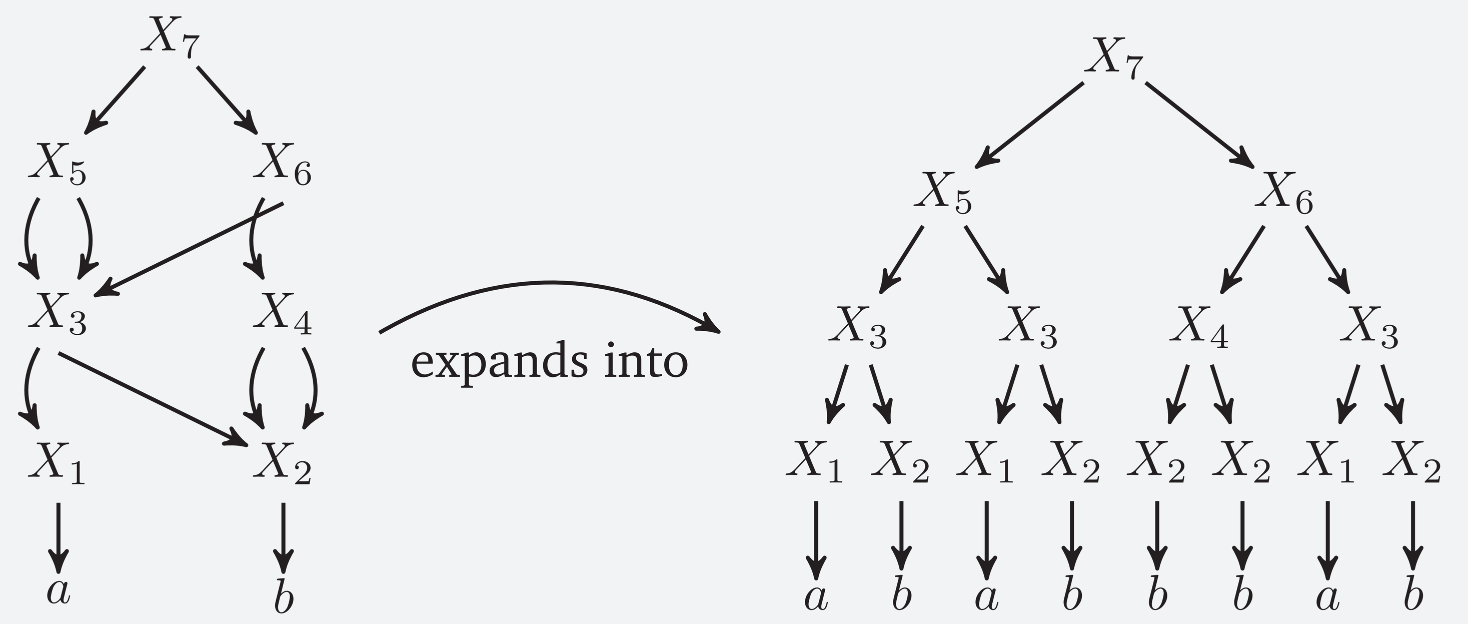
We aim for a **method for answering generic queries** on decompressed strings with little overhead, while only using compressed space. An additional bonus is that if the data is highly compressible, *answering a query on the compressed data can be faster than on the uncompressed data*!

We present a data structure for compressed strings that supports extraction of the fingerprint of any substring in $O(\log N)$ time for a string of length N. These fingerprints allow us to solve many other queries on strings in compressed space.

STRAIGHT LINE PROGRAMS

Compression is modelled as a *Straight Line Program* (SLP), which models the LZ77 and LZ78 compression schemes with little overhead:

- An SLP G is a grammar with n production rules $X_1, \ldots, X_n$, which we consider as a DAG. The rules are in Chomsky normal form:
  - $X_i = X_l X_r$ (nonterminal)
  - $X_i = \mathtt{a}$ (terminal)
- A node $v \in G$ produce a unique string $S(v)$ of length $|S(v)|$.



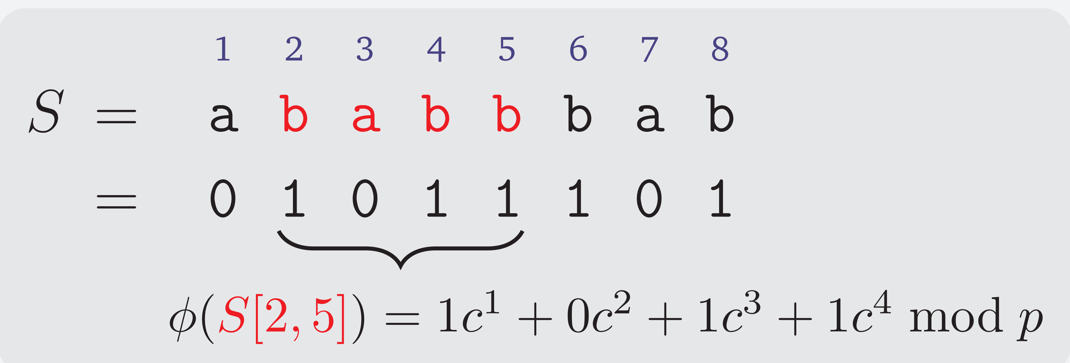
KARP-RABIN FINGERPRINTS

A Karp-Rabin fingerprint function is a rolling hash over the string, allowing for constant time equality checking and composition of hashes for substrings.

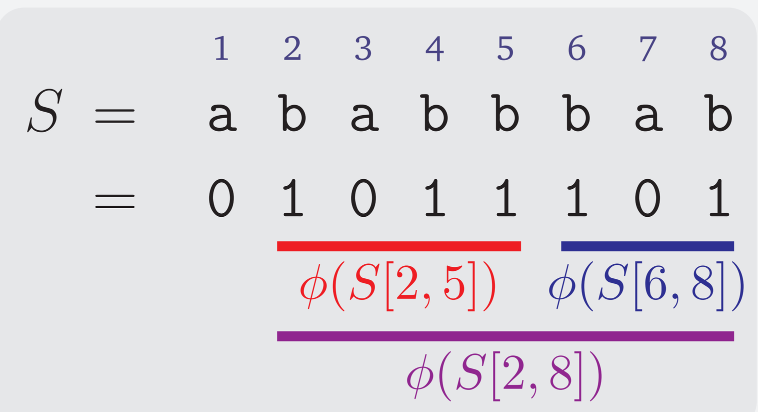
Definition The Karp-Rabin Fingerprint of a string S is defined as

$$\phi(S) = \sum_{k=1}^{|S|} S[k]c^k \bmod p\,,$$

where $p = O(2^w)$ is a sufficiently large prime and $c \in \mathbb{Z}_p$ is chosen uniformly at random. Storing a fingerprint requires constant space.



Property: Composition
Given any two of $\phi(S[i,j])$, $\phi(S[j+1,k])$ and $\phi(S[i,k])$, the remaining substring fingerprint can be computed in $O(1)$ time.



Property: Equality
By choosing the fingerprint function (i.e. c and p) randomly, we get the property that with high probability $S[i,j] = S[k,l]$ if $\phi(S[i,j]) = \phi(S[k,l])$.

PROBLEM DEFINITION

Preprocess a Straight Line Program G of size n producing a string S of length N to support $\textsc{Fingerprint}(i,j) = \phi(S[i,j])$ queries.

Is there a solution using $O(n)$ space?

Why do we care about fingerprints?
Fingerprints yield solutions to many string problems *in compressed space* due to their equality property, including the following:

- Longest common extension
- Longest common substring
- Approximate string matching
- Finding palindromes
- Finding tandem repeats

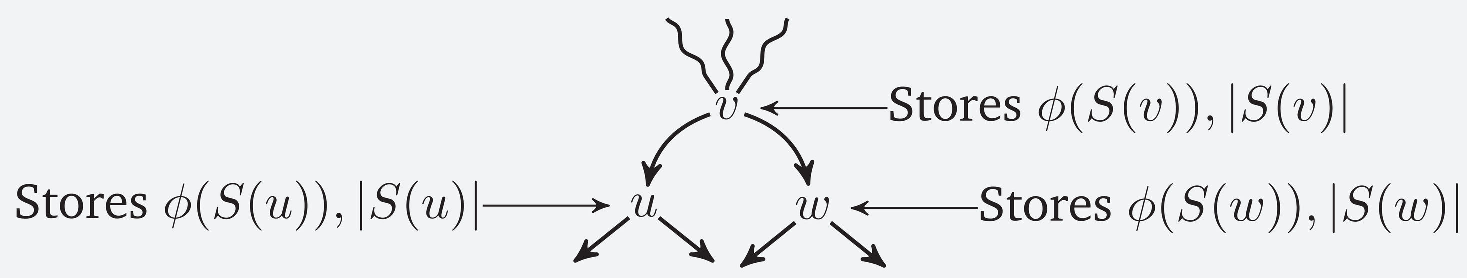
RESULTS

We show how to solve the fingerprint problem in $O(\log N)$ time (and $O(n)$ space) for a string of length N compressed into an SLP of length n.

Idea Subtract fingerprints for two prefixes to answer. Stitch the fingerprint for a prefix together from several smaller ones, using fingerprint composition.

- We store fingerprints for selected substrings of S in the SLP G.
- To answer a $\textsc{Fingerprint}(1,i)$ query, we find a path in G from the root to $S[i]$ and compose the answer from several fingerprints for substrings.

Traverse G: Fingerprints in $O(h)$ time, $O(n)$ space

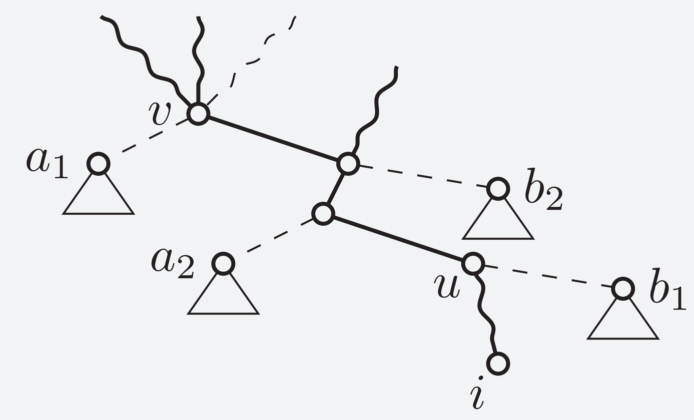


Traverse G by comparing i to lengths of substrings generated by left children:

- Add fingerprint of non-visited left children to answer in constant time.
- The height of G is h, and we proceed to child after each comparison.

Random access query on G: Fingerprints in $O(\log N)$ time, $O(n)$ space

Bille et al. (SODA 2011): It is possible to answer a random access query for $S[i]$ in an SLP $O(\log N)$ time and $O(n)$ space, also retrieving the sequence of $O(\log N)$ *heavy paths* visited on the root-to-leaf path.



Perform random access for $S[i]$ and for each visited heavy path:

- Add fingerprint for *all left-hanging nodes* in constant time.
- These heavy path fingerprints can be stored in $O(n)$ space.

OTHER RESULTS

- We introduce *Linear SLPs*, which is a restricted form of SLP modelling LZ78 compression with $O(1)$ overhead, and show how to answer fingerprint (and random access) queries in $O(\log \log N)$ time, $O(n)$ space.
- A general theorem for *Finger Predecessor* data structures, used to obtain a linear space data structure with query time $O(\log \log |f - i|)$ for finger f and query point i.
- Using fingerprints, we can solve the *Longest Common Extension* problem in:
  - $O(n)$ space and query time $O(\log \ell \log N)$
  - $O(n)$ space and query time $O(\log \ell \log \log \ell + \log \log N)$ for Linear SLPs