Fingerprints in Compressed Strings

(In Proc. WADS 2013)

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 Bristol, Department of Computer Science, ben@cs.bris.ac.uk

October 10, 2013

WCTA 2013, Jerusalem

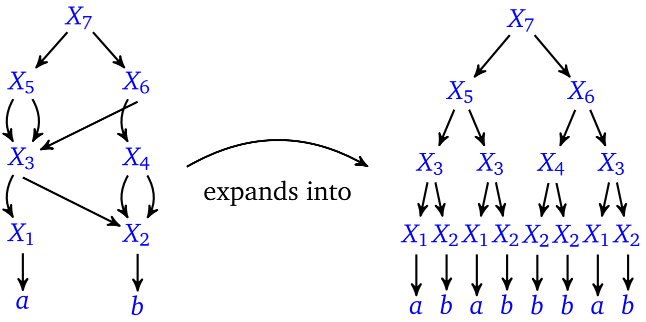
The Takeaway Message

"Karp-Rabin fingerprints can be computed efficiently on compressed strings."

Straight Line Programs

Compression model for strings

- Compression is modelled as a *Straight Line Program* (SLP).
- An SLP G is a grammar in Chomsky normal form.
- G consists of production rules $X_1, \ldots, X_n$ of the form $X_i = X_l X_r$ (nonterminal) or $X_i = \mathtt{a}$ (terminal) representable as a DAG.
- A node $v \in G$ produce a unique string $S(v)$ of length $|S(v)|$.



Karp-Rabin Fingerprints

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.

$$\begin{array}{rcl} & & \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ S & = & \begin{array}{cccccccc} \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{b} & \mathtt{b} & \mathtt{a} & \mathtt{b} \end{array} \\ & = & 0 \; \underbrace{1 \; 0 \; 1 \; 1} \; 1 \; 0 \; 1 \end{array}$$

$$\phi(S[2,5]) = 1c^1 + 0c^2 + 1c^3 + 1c^4 \bmod p$$

Karp-Rabin Fingerprints

Key properties

Composition

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

$$\begin{array}{cccccccccc} & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ S & = & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{b} & \mathtt{b} & \mathtt{a} & \mathtt{b} \\ & = & 0 & \underbrace{1\ 0\ 1\ 1}_{\phi(S[2,5])} & & & & \underbrace{1\ 0\ 1}_{\phi(S[6,8])} & & \end{array}$$

$$0\ \underbrace{1\ 0\ 1\ 1\ 1\ 0\ 1}_{\phi(S[2,8])}$$

Collisions are very unlikely

If $S[i,j] \neq S[i',j']$ then with high probability $\phi(S[i,j]) \neq \phi(S[i',j'])$.

The SLP Toolbox

Useful primitives on SLPs

- Decompress a prefix or suffix of a node in linear time.
(Gąsieniec, Kolpakov, Potapov and Sant. In Proc. 15th DCC, 2005)
- Access a random symbol $S[i]$ in $O(\log N)$ time.
(Bille, Landau, Raman, Sadakana, Satti, Weimann. In Proc. 22nd SODA, 2011)
- Decompress a substring incident to a bookmark in linear time.
(Gagie, Gawrychowski, Kärkkäinen, Nekrich, Puglisi. In Proc. LATA, 2012)

Our additions to the toolbox:

Fingerprints

- Compute $\phi(S[i,j])$ in $O(\log N)$ time
(or in $O(\log \log N)$ time if the SLP is "linear")

Longest Common Prefixes / Extensions

- Compute $\mathrm{LCP}(i,j)$ in $O(\log N \log \ell)$ time
(or in $O(\log \ell \log \log \ell + \log \log N)$ time if SLP is "linear")

Many applications: Approximate String Matching, Longest Common Substring, Palindromes, Tandem Repats, etc.

Main Ideas

We only need to look at prefixes

- Fingerprint composition means that it is sufficient to be able to compute fingerprints for prefixes of S, i.e., $\phi(S[1,i])$.
- Subtracting two prefix fingerprints, we can obtain any substring fingerprint $\phi(S[i,j])$ in $O(1)$ time.

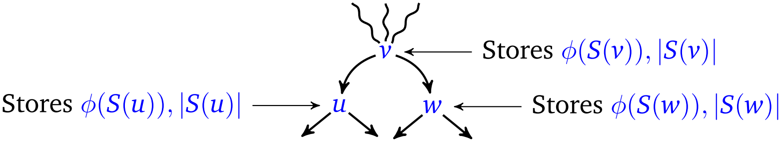
Compose prefix fingerprint during a random access traversal

- Augment the SLP with additional information, e.g., each node stores its fingerprint.
- Compose $\phi(S[1,i])$ from fingerprints of selected substrings of $S[1,i]$.
- Obtain these fingerprints from a random access traversal of the SLP and the resulting root-to-leaf path.

Fingerprints in $O(h)$ time

A simple solution

Data structure



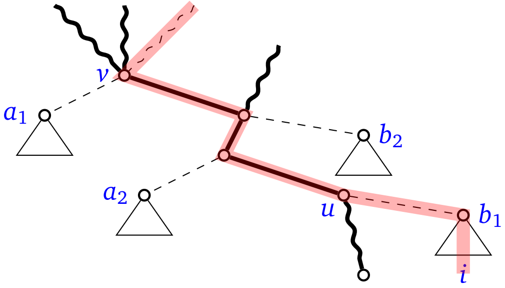
Composing $\phi(S[1,i])$ in $O(h)$ time

- Traverse the SLP for $S[i]$ from the root, comparing i to the substring length at each node to determine the path.
- If following a right edge, add the fingerprint for the string generated by the left child to the composed fingerprint.

Fingerprints in $O(\log N)$ time

Theorem (Bille et al., SODA 2011)

A random access query for $S[i]$ in an SLP can be performed in $O(\log N)$ time and $O(n)$ space, also retrieving the sequence of $O(\log N)$ heavy paths visited on the root-to-leaf path.



Composing $\phi(S[1,i])$ in $O(\log N)$ time

- Perform random access query for $S[i]$, and for each visited heavy path, add fingerprint for all left-hanging nodes in constant time.
- Store fingerprints for all left-hanging heavy path suffixes.

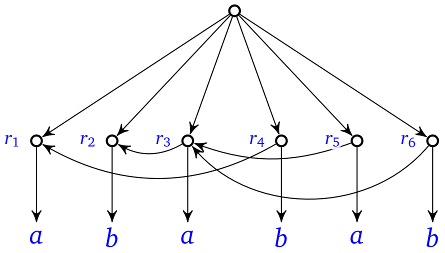
Linear Straight Line Programs

Almost a normal SLP, but with two differences:

- Allow the root to have k children, denoted $r_1, \ldots, r_k$.
- Restrict the right child of all other internal nodes to be a leaf.

Motivation:

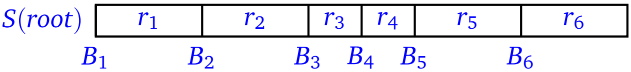
- Models LZ78 compression scheme with $O(1)$ overhead.
- Can be converted into a normal SLP of at most double size.



Fingerprints in $O(\log\log N)$ time

Root children in Linear SLP

- The start position of root child r_q is the sum of string lengths for children on the left, $B_q = \sum_{p=1}^{q-1} |S(r_p)|$.
- Data structure stores $\phi(S(r_i))$ and $\phi(S[1, B_i])$ ($i \in 1, \ldots, k$).

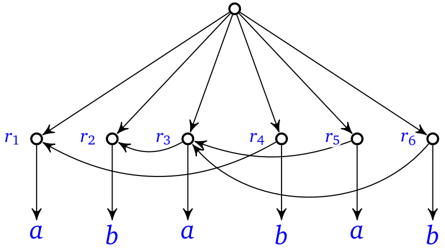


Composing $\phi(S[1,i])$ in $O(\log\log N)$ time

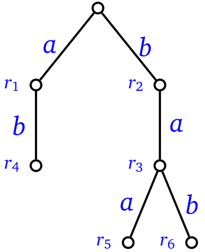
- Find the predecessor B_j of i in the set $\{B_1, \ldots, B_k\}$.
- Compose $\phi(S[1,i])$ from two fingerprints in constant time:
  - Fingerprint $\phi(S[1, B_j])$ for a string ending in r_{j-1} (which is stored).
  - Fingerprint $\phi(S[B_j + 1, i])$ for a prefix of a string generated by r_j.

Linear Straight Line Programs

All prefixes of $S(v)$ fully generated by other nodes (for non-root node v).



(a) Linear SLP.



(b) Dictionary tree.

- Store prefix relationships for non-root nodes in Linear SLP as parent relationship in a **dictionary tree** of size $O(n)$.
- Can find node generating m-length prefix of $S(r_j)$ in $O(1)$ time using level ancestor data structure.

Longest Common Prefixes / Extensions

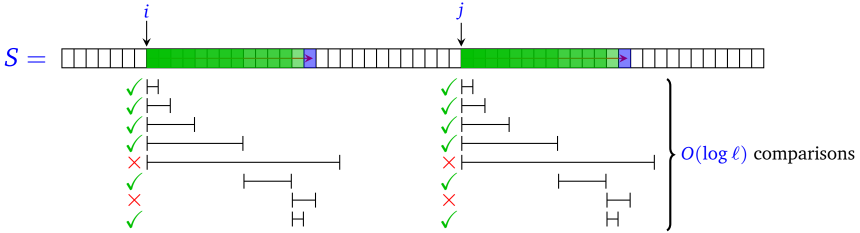
Preprocess a Straight Line Program (SLP) G of size n producing a string S of length N to support LCP queries:

- $\mathrm{LCP}(i,j) = \max \ell$ such that $S[i, i+\ell] = S[j, j+\ell]$.

Theorem

There are data structures solving the LCP problem on SLPs in

- $O(n)$ *space and query time* $O(\log \ell \log N)$
- $O(n)$ *space and query time* $O(\log \ell \log\log \ell + \log\log N)$ *if* G *is a Linear SLP*



The Takeaway Message

“Karp-Rabin fingerprints can be computed efficiently on compressed strings.”

Open Problems

- Other basic primitives on SLPs?
- Bookmarked fingerprints on unbalanced SLPs?
- LCP queries in same time as random access?

The Takeaway Message

"Karp-Rabin fingerprints can be computed efficiently on compressed strings."

Open Problems

- Other basic primitives on SLPs?
- Bookmarked fingerprints on unbalanced SLPs?
- LCP queries in same time as random access?

Thank you!