New Algorithms on Compressed Texts

Yury Lifshits
Steklov Institute of Mathematics, St. Petersburg, Russia
yurlif@pdmi.ras.ru

Tallinn
20/03/2006

Outline of the Talk

1. Processing Compressed Texts: Bird’s Eye View
2. Fully Compressed Pattern Matching: Idea of a New Algorithm
3. More Algorithms and Some Negative Results
4. Conclusions and Open Problems

Motivation

Reasons for algorithms on compressed texts:
- Potentially faster than “unpack-and-solve”
- Lower memory requirements
- Theoretical applications: word equations in PSPACE, pattern matching in message sequence charts

Real data with high level of repetitions:
- Genomes
- Internet logs, any statistical data
- Automatically generated texts

Important Related Results

Algorithms on compressed texts:
- Amir et al.’94: Compressed Pattern Matching
- Gasieniec et al.’96: Regular Language Membership

The following problems are hard for compressed texts:
- Lohrey’04: Context-Free Language Membership
- Berman et al.’02: Two-dimensional Compressed Pattern Matching

FCPM: Problem Description

Fully Compressed Pattern Matching (FCPM)

INPUT: Compressed strings P and T
OUTPUT: Yes/No (whether P is a substring in T?)

Example

Text: abababaabab
Pattern: baba
We know only compressed representation of P and T

Straight-Line Programs

Straight-line program (SLP) is a
Context-free grammar generating exactly one string
Two types of productions: $X_i \rightarrow a$ and $X_i \rightarrow X_j X_k$
Most of practically used compression algorithms (Lempel-Ziv family, run-length encoding...) can be efficiently translated to SLP

Example

abaababaabab

Central idea

If some text is highly compressible, then it contains long identical segments and therefore it is likely that we can solve some problems more efficiently than in general case

FCPM: Problem Description

Fully Compressed Pattern Matching (FCPM)

INPUT: SLP-compression of P and of T
OUTPUT: Yes/No (whether P is a substring in T?)

Let m and n be the sizes of straight-line programs generating correspondingly P and T

Gasieniec et al.’96: $O((n + m)^3 \log^3 |T|)$ algorithm
Miyazaki et al.’97: $O(n^2 m^2)$ algorithm
Lifshits’06: $O(n^2 m)$ algorithm
**Basic Lemma**

Notation:
- **Position** = place between neighbor characters.
- **Occurrence** = starting position of a substring.

Lemma
All occurrences of $P$ in $T$ touching any given position form a single arithmetical progression.

![Common position](image)

**Two Claims**

Claim 1: We can solve all variants of FCPM from AP-table in linear time:
- Find the first occurrence
- Count the number of all occurrences
- Check whether there is an occurrence from the given position
- Compute a "compressed" representation of all occurrences

Claim 2: We can compute the whole AP-table by dynamic programming method using $O(n)$ time for every element.

**AP-table**
Let $P_1, \ldots, P_m$ and $T_1, \ldots, T_n$ be the compression symbols.

A cut is a merging position for $X_i = X_{i-1}X_i$.

**AP-table:**
For every $1 \leq i \leq m, 1 \leq j \leq n$ let $AP[i,j]$ be a code of ar.pr. of occurrences of $P_i$ in $T_j$ that touches the cut of $T_j$.

How to check whether $P$ occurs in $T$ from AP-table?

Answer:
$P$ occurs in $T$ iff there is $j$ such that $AP[m,j]$ is nonempty.

**Getting the answer**

**Auxiliary Procedure: Local PM**

LocalPM($i,j,[\alpha,\beta]$) returns occurrences of $P_i$ in $T_j$ inside the interval $[\alpha,\beta]$.

Important properties:
- Local PM uses values $AP[i,k]$ for $1 \leq k \leq j$.
- It is defined only when $|\beta - \alpha| \leq 3|P_i|$
- It works in time $O(n)$.
- The output of Local PM is a pair of ar.pr.

Proposition: answer of Local PM indeed could be always represented by pair of ar.pr.

**Computing the next element**

Let $P_i = P, P_s$, and let $|P_s| \geq |P_i|$. 

Naive approach:
- Compute all occurrences of $P_i$ around cut of $T_j$.
- Compute all occurrences of $P_s$ around cut of $T_j$.
- Shift the latter by $|P_s|$ and intersect.

Remark: we can do only step 1 by Local PM.

Idea: not all occurrences of $P_s$ but only those that are starting at the ends of $P_s$ ones.

**Computing the next element II**

Some blackboard explanation...
- Take the first ar.pr of $P_i$ occurrences.
- Divide all ends to "continental" and "seaside".
- Check one continental.
- Check all seaside (by Local PM).
- The same for the second ar.pr.

Total complexity:
- Local PM for $P_i$ plus 2 Local PM for $P_s$ plus 2 point checks for $P_i$ time:

$$O(n)$$

We are done! (Modulo basic computation of AP-table and realization of Local PM.)
Covers and Periods

A period of a string $T$ is a string $W$ such that $T$ is a prefix of $W^k$ for some integer $k$.

![Diagram showing a string with periods]

A cover of a string $T$ is a string $C$ such that any character in $T$ is covered by some occurrence of $C$ in $T$.

Compressed Periods/Covers: given a compressed string $T$, to find the shortest period/cover and compute a “compressed” representation of all periods/covers.

Hamming Distance and LCS

Compressed Hamming Distance: given compressed strings $T_1$ and $T_2$, to compute Hamming distance (the number of positions at which the corresponding characters differ) between them.

Example

$T_1$: abaababaabab
$T_2$: babababaabab

$HD(T_1, T_2) = 7$

Compressed LCS: given compressed strings $T_1$ and $T_2$, to compute the length of the longest common subsequence.

Example

$T_1$: abaababaabab
$T_2$: babababaabab

$LCS(T_1, T_2) = 12$

Fingerprint Table

A fingerprint is a set of used characters of any substring of $T$. A fingerprint table is the set of all fingerprints.

Example

Text: abacaba
Fingerprint Table: $∅, \{a\}, \{b\}, \{c\}, \{a,c\}, \{a,b,c\}$

Compressed Fingerprint Table: given a compressed string $T$, to compute a fingerprint table.

Summary

Main points:

- New field: algorithms working on compressed objects (including strings) without unpacking them.
- New algorithm: fully compressed pattern matching in cubic time.

Open Problems

- To construct a $O(nm + k)$ algorithm for Fully Compressed Pattern Matching.
- To construct $O(nm)$ algorithms for edit distance, where $n$ is the length of $T_1$, and $m$ is the compressed size of $T_2$.

Questions?

Contact: Yury Lifshits
Email: yura@logic.pdmi.ras.ru
Home page: http://logic.pdmi.ras.ru/~yura/

Thanks for attention.