Window Subsequence Problems for Compressed Texts

Patrick Cégielski[1], Irène Guessarian[2], **Yury Lifshits**[3] and Yuri Matiyasevich[3]

[1]Université Paris 12, cegielski@univ-paris12.fr

[2]LIAFA and Université Paris 6, ig@liafa.jassieu.fr

[3]Steklov Institute of Mathematics at St.Petersburg,
yura@logic.pdmi.ras.ru, yumat@pdmi.ras.ru

St.Petersburg
11/06/2006

Window Subsequence Matching

INPUT:

Text: | C | O | M | P | U | T | E | R | | S | C | I | E | N | C | E | | I | N | | R | U | S | S | I | A |

Pattern: **CES**

Window size: 10

TASK: to find substrings of the length at most 10 in the text that contains **CES** as a subsequence

Window Subsequence Matching

INPUT:

Text: C O M P U T E R ␣ S C I E N C E ␣ I N ␣ R U S S I A

Pattern: **CES**

Window size: 10

TASK: to find substrings of the length at most 10 in the text that contains **CES** as a subsequence

OUTPUT:

C O M P U T **E** R ␣ **S** C I E N **C** **E** ␣ I N ␣ R U **S** S I A

Window Subsequence Matching

INPUT:

Text: COMPUTER SCIENCE IN RUSSIA

Pattern: **CES**

Window size: 10

TASK: to find substrings of the length at most 10 in the text that contains **CES** as a subsequence

OUTPUT:

COMPUT**E**R **S**CIEN**CE** IN RU**S**SIA

Problem for this talk:

How given a COMPRESSED text to solve window subsequence matching faster than just "unpack-and-search"?

Outline of the Talk

1. New topic in computer science: **algorithms for compressed texts**
2. Our problem and our result
3. Sketch of the algorithm

Part I

What are **compressed** texts?

Can we do something interesting **without unpacking**?

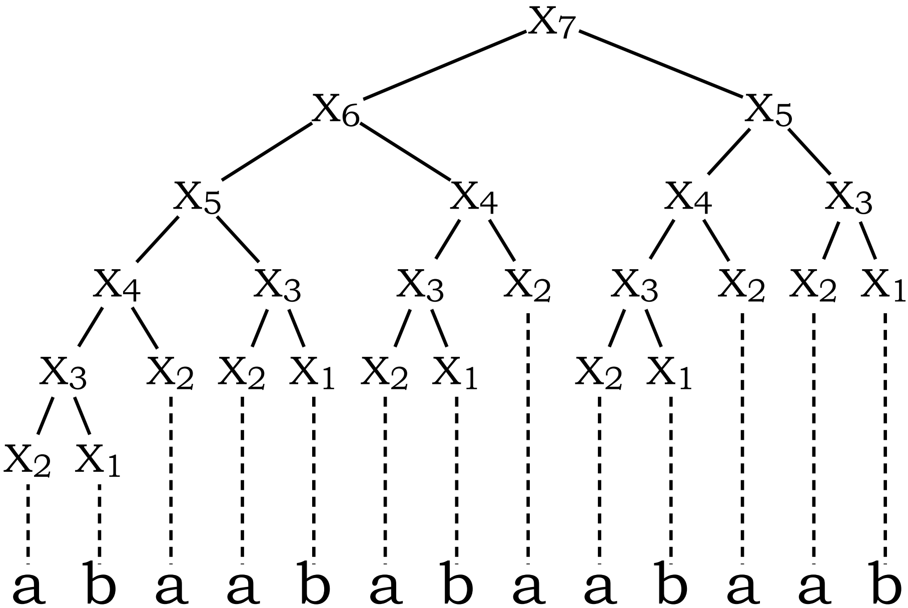
Straight-line Programs: Definition

Straight-line program (SLP) is a
Context-free grammar generating **exactly one** string
Two types of productions:
$X_i \longrightarrow a$ and $X_i \longrightarrow X_p X_q$

Example

abaababaabaab

$X_1 \longrightarrow b$
$X_2 \longrightarrow a$
$X_3 \longrightarrow X_2 X_1$
$X_4 \longrightarrow X_3 X_2$
$X_5 \longrightarrow X_4 X_3$
$X_6 \longrightarrow X_5 X_4$
$X_7 \longrightarrow X_6 X_5$


X_7
X_6 X_5
X_5 X_4 X_4 X_3
X_4 X_3 X_3 X_2 X_3 X_2 X_2 X_1
X_3 X_2 X_2 X_1 X_2 X_1 X_2 X_1
X_2 X_1
a b a a b a b a a b a a b

SLP = Compressed Text

Fact [Rytter, 2003]: given the archive of the text T compressed by LZ78,LZW or some dictionary-based method of original length n and the size of archive z we can in time $O(z)$ convert it to SLP of size $O(z)$ generating the same text.

SLP = Compressed Text

Fact [Rytter, 2003]: given the archive of the text T compressed by LZ78,LZW or some dictionary-based method of original length n and the size of archive z we can in time $O(z)$ convert it to SLP of size $O(z)$ generating the same text.

Fact [Rytter, 2003]: given the LZ77-compressed or RLE-compressed text T of original length n and the size of archive z we can in time $O(z \log n)$ convert it to SLP of the size $O(z \log n)$ generating the same text.

SLP = Compressed Text

Fact [Rytter, 2003]: given the archive of the text T compressed by LZ78,LZW or some dictionary-based method of original length n and the size of archive z we can in time $O(z)$ convert it to SLP of size $O(z)$ generating the same text.

Fact [Rytter, 2003]: given the LZ77-compressed or RLE-compressed text T of original length n and the size of archive z we can in time $O(z \log n)$ convert it to SLP of the size $O(z \log n)$ generating the same text.

Further by compressed text we mean an SLP generating it

Why algorithms on compressed texts?

Answer for algorithms people:

- Might be faster than “unpack-and-search”
- Saving storing space and transmitting costs
- Many fields with highly compressible data: statistics (internet log files), automatically generated texts, massage sequence charts for parallel programs

Answer for complexity people:

- Some problems are hard in worst case. But they might be easy for **compressible** inputs
- New complexity relations. Similar problems becomes different

Problems on SLP-generated texts

∃ poly algorithms:

GKPR'96 Equivalence
GKPR'96 Fully Compressed Pattern Matching
GKPR'96 Regular Language Membership
GKPR'96 Shortest Period
L'06 Shortest Cover
L'06 Fingerprint Table

At least NP-hard:

L'06 Hamming distance
LL'06 Fully Compressed Subsequence Problem
Lohrey'04 Context-Free Language Membership
LL'06 Longest Common Subsequence
BKLPR'06 Two-dimensional Compressed Pattern Matching

Part II

Our Problem and Our Result

Window Subsequence Problems

Definition: w-window = substring of the length w
Definition: minimal window = substring containing the pattern, but any substring of which does not contain the pattern

Window Subsequence Problems

Definition: w-window = substring of the length w
Definition: minimal window = substring containing the pattern, but any substring of which does not contain the pattern

INPUT: SLP generating text T, pattern P, window size w

Computational tasks:

1. To decide whether pattern P is a subsequence of text T
2. To compute the number of minimal windows of T containing P
3. To compute the number of w-windows of T containing P

Window Subsequences: Motivation

Why do we do window subsequence matching (in compressed texts)?

- Variation of approximate pattern matching
- Useful for finding access patterns in databases
- Virus search in archives
- Pattern discovery in bioinformatics
- New step in the framework "what problems could be solved without unpacking?"

Our Algorithm

Main result:

Given a straight-line program of size m,
a pattern of length k and an integer k
we can solve all window subsequence problems on SLP-generated text in time $O(mk^2 \log k)$

Part III

Algorithm for Window Problems on Compressed Texts

Our Small Plan

1. Define auxiliary data structures
2. Compute them
3. Derive answers for our tasks from these structures

Auxiliary Arrays

Let $X_1, \ldots, X_m$ be the nonterminals of SLP generating T, while $P_1, \ldots, P_l$ be all different substrings of pattern P

Left inclusions

For every X_i and every P_j let us define $L(i, j)$ as the length of the minimal **prefix** of X_i that contains P_j, in case of no such prefix exists let $L(i, j) := \infty$

Auxiliary Arrays

Let $X_1, \ldots, X_m$ be the nonterminals of SLP generating T, while $P_1, \ldots, P_l$ be all different substrings of pattern P

Left inclusions

For every X_i and every P_j let us define $L(i,j)$ as the length of the minimal **prefix** of X_i that contains P_j, in case of no such prefix exists let $L(i,j) := \infty$

Right inclusions

For every X_i and every P_j let us define $R(i,j)$ as the length of the minimal **suffix** of X_i that contains P_j, in case of no such prefix exists let $R(i,j) := \infty$

Auxiliary Arrays II

Minimal windows

$M(i) =$ number of minimal windows containing P in X_i

Fixed windows

$F(i) =$ number of w-windows containing P in X_i

Computing Left Inclusions

We compute $L(i, j)$ by induction on i:

Computing Left Inclusions

We compute $L(i,j)$ by induction on i:

Base: if $X_i \to a$, then $L(i,j) = \infty$ for all $P_j \neq a$, and $L(i,j) = 1$ in case $P_j = a$

Computing Left Inclusions

We compute $L(i, j)$ by induction on i:

Base: if $X_i \to a$, then $L(i, j) = \infty$ for all $P_j \neq a$,
and $L(i, j) = 1$ in case $P_j = a$

Induction step: let $X_i \to X_p X_q$.
If $L(p, j) \neq \infty$, then $L(i, j) = L(p, j)$. Assume $L(p, j) = \infty$.

Computing Left Inclusions

We compute $L(i, j)$ by induction on i:

Base: if $X_i \rightarrow a$, then $L(i, j) = \infty$ for all $P_j \neq a$, and $L(i, j) = 1$ in case $P_j = a$

Induction step: let $X_i \rightarrow X_p X_q$.
If $L(p, j) \neq \infty$, then $L(i, j) = L(p, j)$. Assume $L(p, j) = \infty$.

If we find a decomposition $P_j = P_u P_v$ with minimal $|P_v|$ where $L(p, u) \neq \infty$ and $L(q, v) \neq \infty$, then we immediately get $L(i, j) = |X_p| + L(q, v)$

Computing Left Inclusions

We compute $L(i,j)$ by induction on i:

Base: if $X_i \to a$, then $L(i,j) = \infty$ for all $P_j \neq a$, and $L(i,j) = 1$ in case $P_j = a$

Induction step: let $X_i \to X_p X_q$.
If $L(p,j) \neq \infty$, then $L(i,j) = L(p,j)$. Assume $L(p,j) = \infty$.

If we find a decomposition $P_j = P_u P_v$ with minimal $|P_v|$ where $L(p,u) \neq \infty$ and $L(q,v) \neq \infty$, then we immediately get $L(i,j) = |X_p| + L(q,v)$

Such a decomposition can be found by a binary search
Total complexity $O(mk^2 \log k)$
where m is the size of SLP and k is the length of the pattern

Computing Left Inclusions

We compute $L(i,j)$ by induction on i:

Base: if $X_i \to a$, then $L(i,j) = \infty$ for all $P_j \neq a$, and $L(i,j) = 1$ in case $P_j = a$

Induction step: let $X_i \to X_p X_q$.
If $L(p,j) \neq \infty$, then $L(i,j) = L(p,j)$. Assume $L(p,j) = \infty$.

If we find a decomposition $P_j = P_u P_v$ with minimal $|P_v|$ where $L(p,u) \neq \infty$ and $L(q,v) \neq \infty$, then we immediately get $L(i,j) = |X_p| + L(q,v)$

Such a decomposition can be found by a binary search
Total complexity $O(mk^2 \log k)$
where m is the size of SLP and k is the length of the pattern
Mikhail Dvorkin: $O(mk^2)$

Computing Minimal Windows

We compute $M(i)$ by induction on i and using already computed right/left inclusions:

Base: if $X_i \to a$, then $M(i) = 0$ only except $P = a$, in the latter case $M(i) = 1$

Inductive step: $X_i \to X_p X_q$.
$M(i) = M(p) + M(q) + ???$

Computing boundary minimal windows

- ◇ Consequently consider decompositions $P = P_u P_v$
- ◇ For every decomposition with the help of L/R inclusions info
- ◇ find the unique minimal window such that
- ◇ P_u is falling in X_p and P_v is falling X_q
- ◇ If this window **is shifted**, then we increment the counter

Complexity: $O(mk)$

Deriving the Answer

Computational tasks:

- To decide whether P is a subsequence of T
  - Answer: "yes" iff $M(m) \neq 0$
- To compute the number of w-windows of T containing P
  - Answer: $F(m)$
- To compute the number of minimal windows of T containing P
  - Answer: $M(m)$

Complexity: $O(mk^2 \log k)$.

Summary

Main points:

- Compressed text = text generated by SLP
- Given SLP we can solve window subsequence matching in time $O(mk^2 \log k)$
- Method: dynamic programming over SLP

Open Problems:

- Decrease the k-depended factor in complexity
- 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

Last Slide

Yury Lifshits http://logic.pdmi.ras.ru/~yura/

Relevant papers:

Yu. Lifshits
Solving Classical String Problems on Compressed Texts

Yu. Lifshits and M. Lohrey
Querying and Embedding Compressed Texts

P. Cégielski, I. Guessarian, Yu. Lifshits and Yu. Matiyasevich
Window Subsequence Problems for Compressed Texts

L.Boasson, P. Cégielski, I. Guessarian, and Yu. Matiyasevich
Window-Accumulated Subsequence Matching Problem is Linear

P. Cégielski, I. Guessarian, and Yu. Matiyasevich
Multiple Serial Episode Matching

Last Slide

Yury Lifshits http://logic.pdmi.ras.ru/~yura/

Relevant papers:

- Yu. Lifshits
  Solving Classical String Problems on Compressed Texts
- Yu. Lifshits and M. Lohrey
  Querying and Embedding Compressed Texts
- P. Cégielski, I. Guessarian, Yu. Lifshits and Yu. Matiyasevich
  Window Subsequence Problems for Compressed Texts
- L.Boasson, P. Cégielski, I. Guessarian, and Yu. Matiyasevich
  Window-Accumulated Subsequence Matching Problem is Linear
- P. Cégielski, I. Guessarian, and Yu. Matiyasevich
  Multiple Serial Episode Matching

Thanks for attention!