Compressed Pattern Matching

02282 Algorithms for Massive Data Sets

Patrick Hagge Cording

DTU Compute
Department of Applied Mathematics and Computer Science

Outline

Random access

- $O(h)$-time solution (solution to mandatory exercise)
- $O(\log N \log\log N)$-time solution

Compressed pattern matching

- Classic algorithm
- A more space efficient algorithm

SLP Notation

- n is the size of the SLP
- N is the size of the string compressed by the SLP
- h is the height of the SLP
- $S(X_i)$ is the substring produced by node X_i
- $|X_i| = |S(X_i)|$ is the length of the substring produced by node X_i

Problem definition

Random access:

Given an SLP compressing a string S. Build a data structure that supports $\textsc{Access}(i)$ queries, where $\textsc{Access}(i) = S[i]$.

Simple solution

Data structure

- Store $|X_i|$ for each node

$\textsc{Access}(i)$

- Let $X_k = X_l X_r$ be the current node
- $p = 0$
- While current node is not a leaf
  - If $i - p \leq |X_l|$ then continue from X_l
  - Else set $p = p + |X_l|$ and continue from X_r

Advanced algorithm – overview

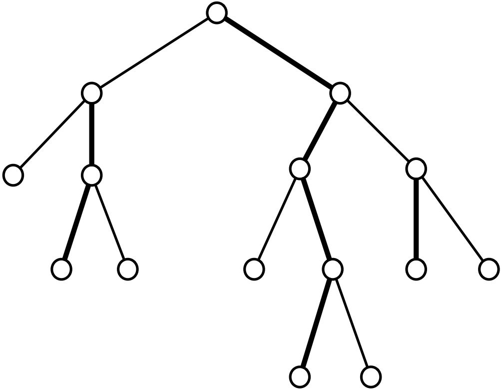
$O(\log N \log\log N)$ time and $O(n^2)$ space

- Heavy-path decomposition
- Predecessor data structure

$O(\log N \log\log N)$ time and $O(n)$ space

- Heavy-tree decomposition
- Weighted ancestor data structure

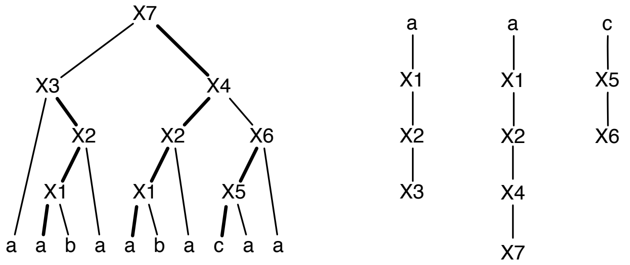
Advanced algorithm – heavy-path decomposition



- Lemma: on any root to leaf path there are at most $\log N$ light edges

Advanced algorithm – $O(n^2)$-space data structure (1/3)

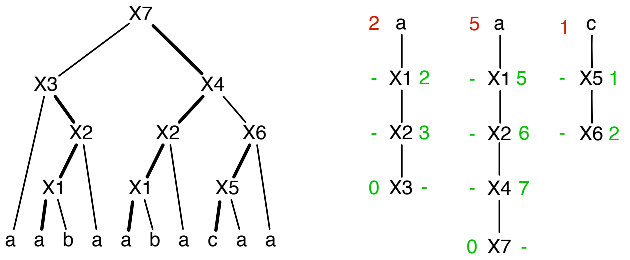
- Make heavy-path decomposition of parse tree



- Storing paths: $O(n^2)$ space
  - A path has length $\leq n$
  - At most one heavy path can start in each uniquely labelled node

Advanced algorithm – $O(n^2)$-space data structure (2/3)

- Store the relative index of the leaf at the end of the heavy path
- Store the accumulated sums of leaves in left and right hanging subtrees for each heavy path

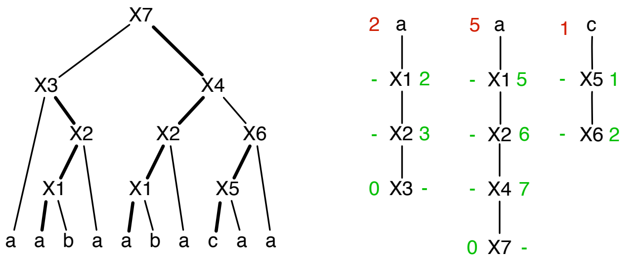


- $O(n^2)$ space

Advanced algorithm – $O(n^2)$-space data structure (3/3)

Query

- Check relative index of path
- Predecessor on left or right values



- Predecessor query on at most $\log N$ paths
- $\Rightarrow O(\log N \log\log N)$ time

Advanced algorithm – weighted ancestor problem

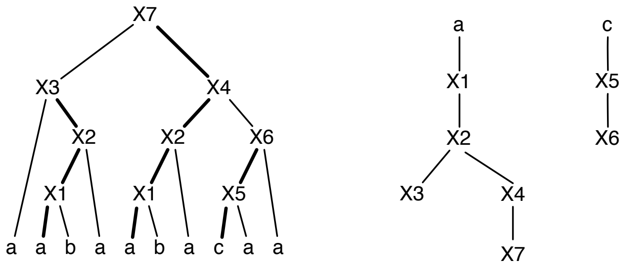
Input:
A tree of size t with integer weights (in the range 1 to N) on its edges.

Weighted ancestors query:
Given a node v and an integer d, return the highest node that is an ancestor and has depth at least d.

- $O(t)$ space and $O(\log \log N)$ query time

Advanced algorithm – $O(n)$-space data structure (1/2)

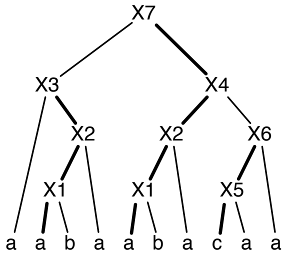
- Observation: Heavy paths share suffixes

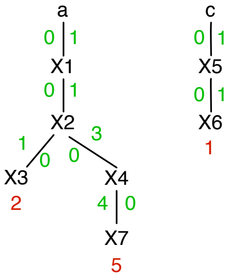


- Store paths as trees: heavy-tree decomposition
- A node occurs exactly once in the heavy forest $\Rightarrow O(n)$ space

Advanced algorithm – $O(n)$-space data structure (2/2)

- For each node $X_i = X_l X_r$
  - If $X_i \to X_l$ is a heavy edge then set $Left(X_i \to X_l) = 0$ and $Right(X_i \to X_l) = |X_r|$
  - If $X_i \to X_r$ is a heavy edge then set $Left(X_i \to X_r) = |X_l|$ and $Right(X_i \to X_r) = 0$
- Build weighted ancestor data structure over both set of values
- Store relative index of each heavy path





Space? Query?

Problem definition

- Fully-compressed pattern matching
- Semi-compressed pattern matching

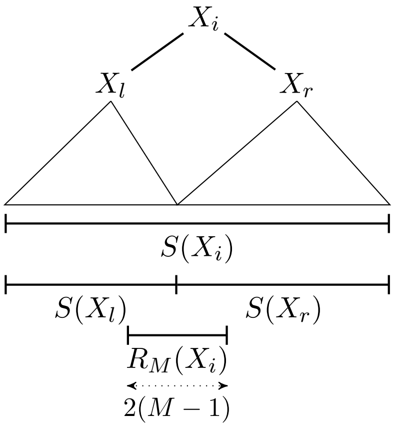
Semi-compressed pattern matching, decision variant:
Given an SLP of size n compressing a string S of length N and an uncompressed pattern P of size M, return YES if P occurs as a substring in S and NO otherwise.

Overview

- Reduction to uncompressed pattern matching
    - Relevant substrings
- $O(nM)$ time and space algorithm
- $O(nM)$ time and $O(n + M)$ space algorithm
    - Left-tree decomposition

Relevant substrings

- The relevant substring of X_i w.r.t. P is $R_M(X_i) = S(X_l)[|X_l| - M, |X_l|]S(X_r)[1, M-1]$



- Relevant substring Lemma: P occurs in S iff P occurs in $R_M(X_i)$ for some $1 \leq i \leq n$.

Reduction to uncompressed pattern matching

- Compute the relevant substrings
- Search for P in the relevant substrings using an algorithm for (uncompressed) string pattern matching, e.g., the Knuth-Morris-Prath algorithm

$O(nM)$ time and space algorithm (1/2)

- To compute $R_M(X_i)$ we need $S(X_l)[|X_l| - M + 1, |X_l|]$ and $S(X_r)[1, M - 1]$

$$Pref(X_i) = \begin{cases} a & \text{if } X_i = a \\ Pref(X_l) & \text{if } |X_l| \geq M - 1 \\ S(X_l)Pref(X_l)[1, M - |X_l| - 1] & \text{otherwise} \end{cases}$$

$$Suf(X_i) = \begin{cases} a & \text{if } X_i = a \\ Suf(X_r) & \text{if } |X_r| \geq M - 1 \\ Suf(X_l)[|X_r| + 1, M]S(X_r) & \text{otherwise} \end{cases}$$

- Tables require $O(nm)$ time and space

$O(nM)$ time and space algorithm (2/2)

- Compute $Pref$ and Suf tables for all X_i in the SLP
- Let $R_M(X_i) = Suf(X_l)Pref(X_r)$ for each $X_i = X_l X_r$ in the SLP
- Run uncompressed string pattern matching algorithm for each $R_M(X_i)$

- $Pref$ and Suf tables require $O(nM)$ time and space
- Since $|R_M(X_i)| \leq 2(M-1) = O(M)$ the sum of lengths of relevant substrings is $O(nM)$
- Using KMP $\Rightarrow O(nM)$ time and space for matching

Advanced algorithm – idea

- Throw away relevant substring after matching
- What is needed? Fast decompression of prefixes and suffixes of substrings

Advanced algorithm – levelled ancestor problem

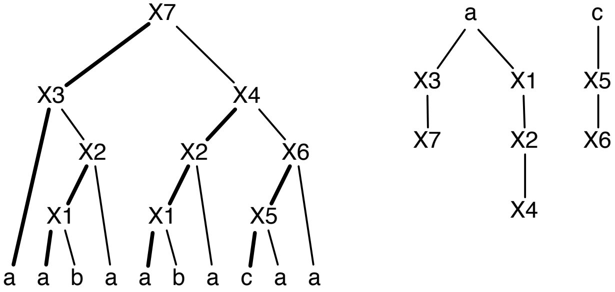
Input:
A rooted tree of size t.

Levelled ancestors query:
Given a node v and an integer d, return the ancestor v with depth d.

- $O(t)$ space and $O(1)$ query time

Advanced algorithm – left-path decomposition

- As with heavy-paths, we store the leftmost paths in trees



Advanced algorithm – data structure

- Make a left-tree decomposition of the SLP
- Store a pointer from each node to its corresponding node in the left-forest
- Build a levelled ancestor data structure for each left-tree
- Store a pointer from each node in the SLP to its leftmost leaf

- $O(n)$ space

Advanced algorithm – prefix decompression

$\textsc{PrefixDecompress}(X_i, k)$

- Jump to leftmost leaf X_j and output character
- $d = 1$
- While $X_j \neq X_i$ and $d < \min\{k, |X_i|\}$
  - Use levelled ancestor data structure to find parent X_p of X_j on left-path
  - Let X_r be the right child of X_p
  - $\textsc{PrefixDecompress}(X_r, k - d)$
  - $d = d + |X_r|$
  - Set X_j to be X_p

- $O(k)$ time
- Suffix decompression is symmetric

Advanced algorithm

- Let $R_M(X_i) = \textsc{SuffixDecompress}(X_l, M-1)\textsc{PrefixDecompress}(X_r, M-1)$ for each $X_i = X_l X_r$ in the SLP
- Run uncompressed string pattern matching algorithm for each $R_M(X_i)$

- $O(nM)$ time and $O(n+M)$ space