Karlsruhe Institute of Technology

Text Indexing

Lecture 04: Longest Common Prefix Array

Florian Kurpicz

PINGO

https://pingo.scc.kit.edu/964642

Recap: Suffix Array and LCP-Array

Definition: Suffix Array [GBS92; MM93]

Given a text T of length n, the **suffix array** (SA) is a permutation of $[1..n]$, such that for $i \leq j \in [1..n]$

$$T[SA[i]..n] \leq T[SA[j]..n]$$

Definition: Longest Common Prefix Array

Given a text T of length n and its SA, the **LCP-array** is defined as

$$LCP[i] = \begin{cases} 0 & i = 1 \\ \max\{\ell \colon T[SA[i]..SA[i]+\ell) = \\ \qquad T[SA[i-1]..SA[i-1]+\ell)\} & i \neq 1 \end{cases}$$

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
LCP	0	0	1	2	2	5	0	2	1	1	4	0	3
	\$	a\$	ababcabcabba\$	abba\$	abcabba\$	abcabcabba\$	ba\$	babcabcabba\$	bba\$	bcabba\$	bcabcabba\$	cabba\$	cabcabba\$

Naive Computation of the LCP-Array

Task

- given: text T of length n and its suffix array
- wanted: longest common prefix array

Naive Construction

- for each pair $(SA[i-1], SA[i])$
- compare $T[SA[i-1]+\ell]$ and $T[SA[i]+\ell]$
- until missmatch

Running Time

- naive construction requires $O(n^2)$ time
- all-a texts are worst case
- here $LCP[1]=0$, $LCP[1]=0$, and $LCP[i]=i-2$
- only distinguishable character is $

Properties of the LCP-Array

- do not compare all suffixes naively
- compare only unknown parts

Lemma: Values in LCP-array

Given a text T of length n, its suffix array SA and LCP-array LCP, then

$$\exists i \in [1, n) \colon LCP[i] = \ell > 0 \Rightarrow \exists j \in [1, n) \colon LCP[j] = \ell - 1$$

Proof (Sketch)

- let $LCP[i] = k > 0$
- $T[SA[i]..SA[i] + k) = T[SA[i-1]..SA[i-1] + k)$
- $T[SA[i] + 1..SA[i] + k) = T[SA[i-1] + 1..SA[i-1] + k)$
- not necessarily next to each other in SA

The Inverse Suffix Array

Definition: Inverse Suffix Array

Given a suffix array SA of length n, the **inverse suffix array** ($ISA = SA^{-1}$) is

$$ISA[SA[i]] = i$$

for $n \in [1..n]$

- inverse permutation as hinted by the name
- where is a suffix in the suffix array

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1

Linear Time Construction [Kas+01]

Function LinearTimeLCP(T, $SA[1..n]$):

```
for i = 1, ..., n do ISA[SA[i]] = i
ℓ = 0, LCP[1] = 0
for i = 1, ..., n do
    if ISA[i] ≠ 1 then
        j = SA[ISA[i] − 1]
        while T[i + ℓ] = T[j + ℓ] do
            ℓ = ℓ + 1
        LCP[ISA[i]] = ℓ
        ℓ = max{0, ℓ − 1}
return LCP
```

- compute suffixes in text order
- use *ISA* to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

- correctness and running time

The Φ-Array

Definition: Φ-Array

Given a text T of length n and its suffix array SA, the Φ-array is defined (for $i > 1$) as

$$\Phi[SA[i]] = SA[i-1]$$

- $\Phi[i]$ gives suffix that is needed for comparison
- not a permutation of SA

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
Φ	12	11	6	7	8	9	10	4	1	2	3	13	-

Better Linear Time Construction [KMP09]

Function Φ-Algorithm(T, $SA[1..n]$):

- $\Phi[n] = SA[n]$ ⓘ $SA[1] = n$; T has sentinel
- **for** $i = 2, \ldots, n$ **do** $\Phi[SA[i]] = SA[i-1]$
- $\ell = 0$
- **for** $i = 1, \ldots, n$ **do**
  - $j = \Phi[i]$
  - **while** $T[i+\ell] = T[j+\ell]$ **do**
    - $\ell = \ell + 1$
  - $\Phi[i] = \ell$
  - $\ell = \max\{0, \ell - 1\}$
- **for** $i = 1, \ldots, n$ **do** $LCP[i] = \Phi[SA[i]]$
- **return** LCP

- **PINGO** why better?

- compute LCP-array in text order
- reorder LCP-array as final step

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
Φ	12	11	6	7	8	9	10	4	1	2	3	13	-
LCP	0	0	1	2	2	5	0	2	1	1	4	0	3

- example:
- correctness and running time similar

Brief Remainder: Cache & Cache Misses

- cache is small but fast memory
- located on CPU
- cache miss is failure to retrieve data from cache
- instead data has to be loaded from main memory

Cache Sizes (AMD Ryzen 7 PRO 4750U)

- L1: 256 KiB (8 instances)
- L2: 4 MiB (8 instances)
- L3: 8 MiB (2 instances)

- **PINGO** how much slower is a main memory compared to L1 cache?

Latency Numbers

- L1 cache reference $\approx$ 1 ns
- L2 cache reference $\approx$ 4 ns
- main memory reference $\approx$ 100 ns

Better Due to Less Cache Misses

Function LinearTimeLCP($T, SA[1..n]$):

- **for** $i = 1, \ldots, n$ **do** $ISA[SA[i]] = i$
- $\ell = 0, LCP[1] = 0$
- **for** $i = 1, \ldots, n$ **do**
  - **if** $ISA[i] \neq 1$ **then**
    - $j = SA[ISA[i] - 1]$
    - **while** $T[i + \ell] = T[j + \ell]$ **do**
      - $\ell = \ell + 1$
    - $LCP[ISA[i]] = \ell$
    - $\ell = \max\{0, \ell - 1\}$
- **return** LCP

- **PINGO** number of cache misses?

Function Φ-Algorithm($T, SA[1..n]$):

- $\Phi[n] = SA[n]$ ℹ $SA[1] = n$; T has sentinel
- **for** $i = 2, \ldots, n$ **do** $\Phi[SA[i]] = SA[i - 1]$
- $\ell = 0$
- **for** $i = 1, \ldots, n$ **do**
  - $j = \Phi[i]$
  - **while** $T[i + \ell] = T[j + \ell]$ **do**
    - $\ell = \ell + 1$
  - $\Phi[i] = \ell$
  - $\ell = \max\{0, \ell - 1\}$
- **for** $i = 1, \ldots, n$ **do** $LCP[i] = \Phi[SA[i]]$
- **return** LCP

- **PINGO** number of cache misses?

Practical Comparison of Both Algorithms (1/2)

Pizza & Chili Corpus

- `http://pizzachili.dcc.uchile.cl/`
- de facto standard text corpus

Used in Experiment (50 MB)

- **dblp** XML-Data providing bibliographic information
- **DNA** DNA reads from the Gutenberg Project
- **english** English texts of the Gutenberg Project
- **sources** Source code from the Linux kernel

Experimental Setup

- used text described above
- on T14s with AMD Ryzen 7 PRO 4750U
- times are average of five runs

Practical Comparison of Both Algorithms (2/2)

Text	Naive (ms)	[Kas+01] (ms)	[KMP09] (ms)
dblp	9121.6	3479.0	2567.2
DNA	6763.0	6152.2	4174.6
english	99811.4	4899.8	3316.2
sources	12687.6	3486.4	2536.6

Permuted LCP-Array [KMP09]

Definition: PLCP-Array

- $PLCP[SA[i]] = LCP[i]$
- $PLCP[i] = lcp(i, SA[i-1]) = lcp(i, \Phi[i])$

- $PLCP[i] \geq PLCP[i-1] - 1$
- $T[i-1] = T[\Phi[i] - 1] \Rightarrow PLCP[i]$ is reducible
- $PLCP[i]$ is reducible
  $\Rightarrow PLCP[i] = PLCP[i-1] - 1$

- only compute irreducible PLCP-values
- sum of all irreducible PLCP-values is $\leq n \lg n$

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	a	c	a	c	a	c	b	a	a	c	b	$
SA	13	1	9	2	4	10	6	12	8	3	5	11	7
LCP	0	0	3	1	4	2	3	0	1	0	3	1	2
$PLCP$	0	1	0	4	3	3	2	1	3	2	1	0	0
Φ	12	8	7	1	2	9	10	11	0	3	4	5	-

Recap: Pattern Matching with the Suffix Array

Karlsruhe Institute of Technology

Function SeachSA(T, $SA[1..n]$, $P[1..m]$)**:**

$\ell = 1, r = n + 1$
while $\ell < r$ **do**
    $i = \lfloor (\ell + r)/2 \rfloor$
    if $P > T[SA[i]..SA[i] + m)$ **then**
      $\ell = i + 1$
    else $r = i$
$s = \ell, \ell = \ell - 1, r = n$
while $\ell < r$ **do**
    $i = \lceil \ell + r/2 \rceil$
    if $P = T[SA[i]..SA[i] + m)$ **then** $\ell = i$
    else $r = i - 1$
return $[s, r]$

Lemma: Running Time SeachSA

The SeachSA answers counting queries in $O(m \lg n)$ time and reporting queries in $O(m \lg n + occ)$ time

Proof (Sketch)

- two binary searches on the SA in $O(lgn)$ time
- each comparison requires $O(m)$ time
- counting in $O(1)$ additional time
- reporting in $O(occ)$ additional time

- comparison of pattern is expensive

Speeding Up Pattern Matching with the LCP-Array (1/4)

- remember how many characters of the pattern and suffix match
- identify how long the prefix of the old and next suffix is
- do so using the LCP-array and
- range minimum queries detailed introduction in **Advanced Data Structures**

Definition: Range Minimum Queries

Given an array $A[1..m)$, a **range minimum query** for a range $\ell \leq r \in [1, n)$ returns

$$RMQ_A(\ell, r) = \arg\min\{A[k] \colon k \in [\ell, r]\}$$

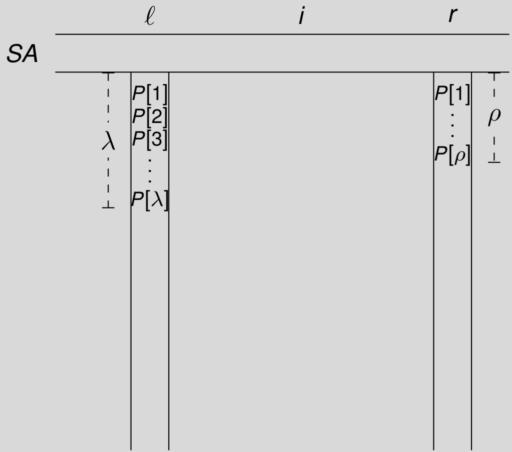
- $lcp(i, j) = \max\{k \colon T[i..i + k)$
- $lcp(i, j) = T[j..j+k)\} = LCP[RMQ_{LCP}(i+1, j)]$
- RMQs can be answered in $O(1)$ time and
- require $O(n)$ space

Speeding Up Pattern Matching with the LCP-Array (2/4)

- during binary search matched
- λ characters with left border ℓ and
- ρ characters with right border r
- w.l.o.g. let $\lambda \geq \rho$

- middle position i
- decide if continue in $[\ell, i]$ or $[i, r]$

- let $\xi = lcp(SA[\ell], SA[i])$ $O(1)$ time with RMQs



Speeding Up Pattern Matching with the LCP-Array (3/4)

- let $\xi = lcp(SA[\ell], SA[i])$

$\xi > \lambda$

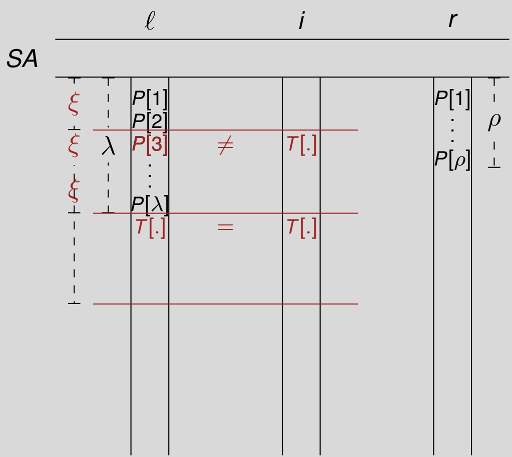
- $P[\lambda + 1] > T[SA[\ell] + \lambda] = T[SA[i] + \lambda]$
- $\ell = i$ without character comparison

$\xi = \lambda$

- compare as before

$\xi < \lambda$

- $\xi \geq \rho$ and $P[\xi + 1] < T[SA[i] + \xi]$
- $r = i$ and $\rho = \xi$ without character comparison



Speeding Up Pattern Matching with the LCP-Array (4/4)

Lemma:

Using RMQs, `SeachSA` answers counting queries in $O(m + \lg n)$ time and reporting queries in $O(m + \lg n + occ)$ time

Proof (Sketch)

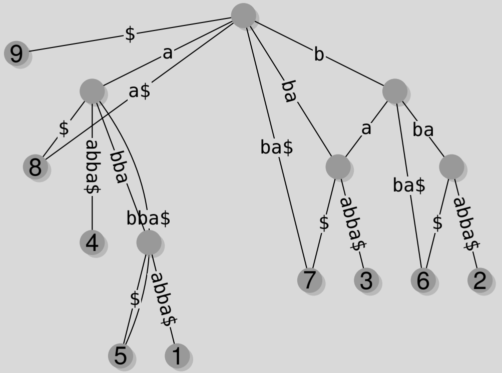
- either halve the range in the suffix array ($\xi \neq \lambda$) or
- compare characters of the pattern (at most m)

Back to the Roots: Suffix Tree Construction

- naive in $O(n^2)$ time

- use *SA* and *LCP*
- only look at rightmost path in tree
- find deepest node with string-depth $\leq$ $LCP[i]$
- total $O(n)$ time

	1	2	3	4	5	6	7	8	9
T	a	b	b	a	a	b	b	a	$
SA	9	8	4	5	1	7	3	6	2
LCP	0	0	1	1	4	0	2	1	3



Conclusion and Outlook

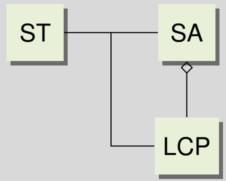
Karlsruhe Institute of Technology

This Lecture

- linear time LCP-array construction
- suffix tree construction based on *SA* and *LCP*

- engineered LCP-Array construction algorithms
- cache misses are costly
- interesting properties of the PLCP-array

Linear Time Construction



Next Lecture

- text compression using *SA* and *LCP*

Bibliography I

[GBS92] Gaston H. Gonnet, Ricardo A. Baeza-Yates, and Tim Snider. “New Indices for Text: Pat Trees and Pat Arrays”. In: *Information Retrieval: Data Structures & Algorithms*. Prentice-Hall, 1992, pages 66–82.

[Kas+01] Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. “Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications”. In: *CPM*. Volume 2089. Lecture Notes in Computer Science. Springer, 2001, pages 181–192. DOI: 10.1007/3-540-48194-X_17.

[KMP09] Juha Kärkkäinen, Giovanni Manzini, and Simon J. Puglisi. “Permuted Longest-Common-Prefix Array”. In: *CPM*. Volume 5577. Lecture Notes in Computer Science. Springer, 2009, pages 181–192. DOI: 10.1007/978-3-642-02441-2_17.

[MM93] Udi Manber and Eugene W. Myers. “Suffix Arrays: A New Method for On-Line String Searches”. In: *SIAM J. Comput.* 22.5 (1993), pages 935–948. DOI: 10.1137/0222058.