Suffix arrays

DS for string algorithms in text processing, pattern matching, data compression, bioinformatics, ... (any linear data, e.g. gps record)

substring search problem

find all occurrences of a substring β in a given string α, $n=|\alpha|$, $m=|\beta|$

algs: Robin-Karp (rolling hashing as a filter) ... $O(n \cdot m)$ worst case time

Knuth-Morris-Pratt, Aho-Corasick } build DS for a fixed β, use for any α ... $O(n+m)$ time
(partial match table, finite automaton)

LCSA: Build DS for a fixed α, use for any β.

suffix array, (suffix tree) Build (precomputation) $O(n)$ time $\Rightarrow$ querying in $O(\log n + m)$ (if only const reported)

(we show only weaker results)

Notation:

$\alpha = \alpha[0] \dots \alpha[n-1]$, $n=|\alpha|$ (symbols labeled from 0)

$\alpha[i:j] := \alpha[i] \dots \alpha[j-1]$ a substring of length $j-i$ if $i<j$,
ε (empty string) if $i \geq j$.

$\alpha[:j]$ prefix of length j in α, $0 \leq j \leq n$ (i.e. $\alpha[0:j]$)

$\alpha[i:]$ suffix starting in $\alpha[i]$, $0 \leq i \leq n$ (i.e. $\alpha[i:n]$)

$\alpha \leq \beta$ lexicographical order: α is a prefix of β or there is i s.t. $\alpha[i] < \beta[i]$ and $\alpha[:i] = \beta[:i]$.

↑ fixed total order on alphabet

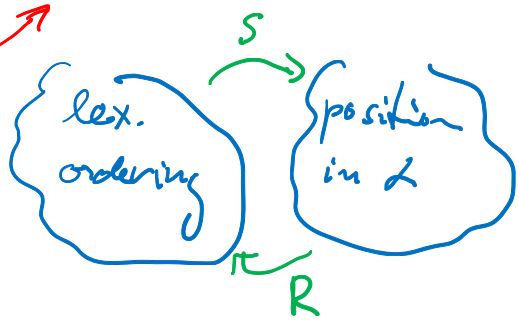
Note: To avoid this, we may use "padding" $\alpha\$$, $\beta\$$ where $\$$ (end of string) is a new symbol (smaller than other symbols)

- We have no assumption on the alphabet, is finite but can be arbitrarily large.

Ex: α = bananas (0 1 2 3 4 5 6 7), $n = |\alpha| = 7$

i	$\alpha[S[i]:]$	S[i]	R[i]	L[i]
0	ε	7	4	0
1	ananas	1	1	3
2	anas	3	5	1
3	as	5	2	0
4	bananas	0	6	0
5	nanas	2	3	2
6	nas	4	7	0
7	s	6	0	–

β = na, $m = |\beta| = 2$ occurrences of β in α



longest common prefix with the next row

in lex order

all suffices in lex. order

where i-th lex. suffix starts in α

lex. rank (which row) of $\alpha[i:]$

Def: The **suffix array** for a string α is a permutation $S[0 \dots n]$ s.t. $\alpha[S[i]:] < \alpha[S[i+1]:]$ for every $0 \le i < n$, i.e. $S[i]$ is the starting position of the i-th lex. smallest suffix in α.

All occurrences of β in α form a range in the suffix array. (as prefixes of suffices) ↑ interval

⇒ All occurrences of β in α can be enumerated from S by a binary search in time $O(m \log n + p)$, p = # occurrences.
(to be improved) ↑ not needed for const only

Def: The **rank array** $R[0 \dots n]$ is an inverse permutation to S, i.e. $R[i]$ is # lex. smaller suffices than $\alpha[i:]$.

Def: The **LCP array** (longest common prefix) $L[0 \dots n-1]$ is given by: $L[i] := LCP(\alpha[S[i]:], \alpha[S[i+1]:])$ for every $0 \le i < n$, where $LCP(\gamma, \delta) :=$ max. k s.t. $\gamma[:k] = \delta[:k]$.

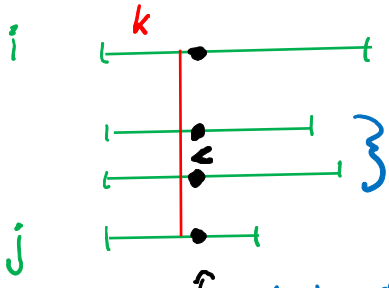
LCP of any two suffices

For any $0 \le i < j \le n$,

$$LCP(\alpha[SA[i]:], \alpha[SA[j]:]) = \min(L[i], \dots, L[j-1]).$$

(minimum of "adjacent steps"; say k)

Pf:



$LCP(\text{—}k\text{—}) \ge k$ trivially

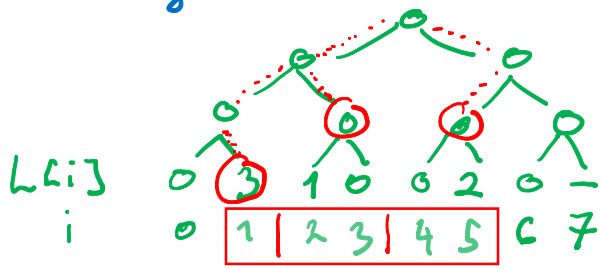
} $LCP = k$, here strictly increasing

} $LCP(\text{—}k\text{—}) \le k$

↳ non-decreasing sequence of symbols □

RMQ (range minimum queries) for L (to find min in range $[i:j]$)

ex: 1D-range trees (next lecture)



← minima in inner nodes (subintervals)

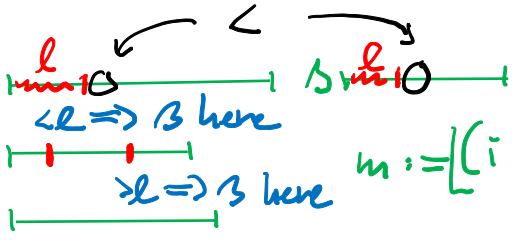
$\Rightarrow O(\log n)$ query time

$O(n)$ build time

Note: There is a DS with $O(1)$ query time, $O(n)$ build time. (without proof)

Given LCP for any two suffices, the lex. smallest (or largest) suffix of α with prefix β can be found in $O(m + \log n)$ time.

idea: current range $[i,j]$, assume $\ell = LCP(\beta, \alpha[SA[i]:])$ known.



$m := \lfloor (i+j)/2 \rfloor$

if $LCP(\alpha[SA[i]:], \alpha[SA[m]:]) < \ell$, continue in $[i, m]$

if $* > \ell$, continue in $[m, j]$

if $* = \ell$, compare $\beta, \alpha[SA[m]:]$ from $(\ell+1)$-th symbol.

$\Rightarrow$ uses $\le m$ comparisons in β

$\Rightarrow$ All occurrences of β in α can be enumerated in $O(m + \log n + p)$ time, p = # occurrences.

Applications of LCP

1) histogram of k-grams (substrings of length k) = # occurrences of each k-gram

ex:

i	$\alpha[S[i]:]$	$LL[i]$	$k=2$
0	ε	0	← too short
1	ananas	3	] an
2	anas	1	
3	as	0	] as
4	bananas	0	] ba
5	nanas	2	] na
6	nas	0	
7	s	—	← too short

⇒ ranges $[i,j]$ in LCP array s.t. $LL[i-1] < k$, $LL[l] \geq k$ $\forall\, i \leq l < j$, $LL[j] < k$

⇒ single pass through LL in $O(n)$ time

2) longest repeating substring in α = $\max_i LL[i]$ ⇒ $O(n)$ time ("redundance" in α)

3) longest common substring of α and β: from α / from β

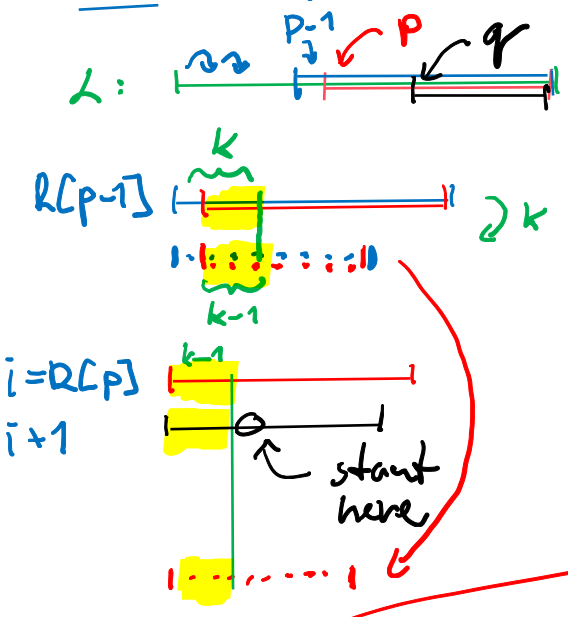
LCP for $\alpha \# \beta$, then $\max LL[i]$ s.t. $S[i] < |\alpha|$ and $S[i+1] > |\alpha|$ or vice versa

($\#$: separator)

Construction of LCP array (Kasai's alg.)

given S, R construct LL in $O(n)$ time.

idea: if $k = LL[R[p-1]]$ then $LL[R[p]] \geq k-1$.




```
BUILD LCP
k ← 0                                   // k = LL[R[p-1]]
for p = 0, ..., n-1
    k ← max(0, k-1)                     // safe dec.
    i ← R[p]
    q ← S[i+1]
    while (p+k < n) ∧ (q+k < n) ∧ (α[p+k] = α[q+k])   // stay in α
        k ← k+1                         // inc.
    LL[i] ← k
```

increments = # decrements + final value − initial $\leq 2n$ ⇒ runs in $O(n)$ time

($\leq n$ / $\leq n$ / ≥ 0)

Construction of the suffix array (by doubling)

idea: compare suffixes by prefixes of length 2^i, for $i = 0, \ldots, \lceil \log n \rceil$ (by prefixes of length k)

Def: $\gamma =_k \delta$ if $\gamma[:k] = \delta[:k]$

$\gamma \leq_k \delta$ if $\gamma[:k] \leq \delta[:k]$ (↑ lex.)

Note: $\leq_k$ is only quasi order (not necessarily antisymmetric),

we may have $\gamma =_k \delta$ although $\gamma \neq \delta$, e.g. $abc =_2 abd$.

Def: $R_k[i] = |\{ j \mid \alpha[j:] <_k \alpha[i:] \}|$, i.e. # strictly smaller (w.r.t. $\leq_k$) suffixes than $\alpha[i:]$.

("approximation of rank array", $R_n = R$)

☺ $\alpha[i:] \leq_{2k} \alpha[j:] \iff \alpha[i:] <_k \alpha[j:]$ or ($\alpha[i:] =_k \alpha[j:]$ and $\alpha[i+k:] \leq \alpha[j+k:]$)

(first half / second half)

$R_{2k}[i] \leq R_{2k}[j] \iff R_k[i] < R_k[j]$ or ($R_k[i] = R_k[j]$ and $R_k[i+k] \leq R_k[j+k]$)

$\iff (R_k[i], R_k[i+k]) \leq (R_k[j], R_k[j+k])$

↑ lexicographically as a pair

Alg: $k = 1$: compute R_1 by sorting of suffixes in the first symbol

$\Rightarrow O(n \log n)$ time (if no assumption on alphabet)

$k \to 2k$: sort suffixes by pairs $(R_k[i], R_k[i+k])$

+ S from R in $O(n)$ time

Bucket sort: two passes with $n+1$ buckets

$\Rightarrow O(n)$ time and space per step

$\lceil \log n \rceil$ steps $\Rightarrow O(n \log n)$ total time, $O(n)$ space

Note: It is possible to construct suffix array in $O(n)$ time and $O(1)$ additional space. (omitted)