String searching: ① preprocess a pattern so that we can find it in any text
↳ search automata (KMP, AC, ...)
② preprocess a text so that we can find any patterns
↳ D.S. for strings — suffix trees
— suffix array,

Notation

- Σ, alphabet (finite)
- Σ^*, strings (words) over Σ
- $|\alpha|$ length
- $\alpha[i]$ i-th character (from 0)
- ε the empty string
- $\alpha\beta$ concatenation
- $\alpha[i:j]$ substring $\alpha[i]\,\alpha[i+1]\ldots\alpha[j-1]$
- $\alpha[:j]$ prefix of j first chars
- $\alpha[i:]$ suffix
- $\alpha[:] = \alpha$
- $\alpha[i:j)$ for $i \geq j = \varepsilon$
- $\alpha \leq \beta$ lexicographic order $\varepsilon < a < aa < ab$

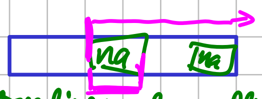
Example: $\overset{0\,1\,2\,3\,4\,5\,6\,\overset{\varepsilon}{7}}{\text{bananas}}$ 8 suffixes

sorted list of suffixes ... size $\Theta(n^3)$!

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

all suffixes starting with na

Find "na"



👁 substrings ⇔ prefixes of suffixes
$\alpha[i:j] = (\alpha[i:])[:j-i]$

Df: A suffix array for a string $\alpha \in \Sigma^*$ is a permutation S on $\{0 \ldots |\alpha|\}$ s.t. $\forall i$ $\alpha[S[i]:] < \alpha[S[i+1]:]$. } space $\Theta(n)$

Claim: Suffix array for a string of length n can be built in time $\Theta(n)$.

Corollary: Using a suffix array for a text α, we can count all occurrences of pattern β in time $O(\log|\alpha| \cdot |\beta|)$ and enumerate them in extra $O(1)$ per occurrence.
↑ can be improved to $O(\log|\alpha| + |\beta|)$

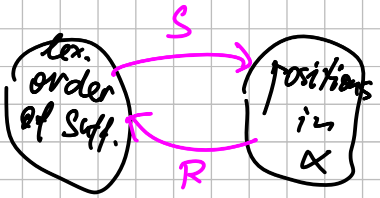
Df: A rank array R: inverse permutation to S.

👁 $R[i]$ tells us at which pos. in lex. order $\alpha[i:]$ occurs.

👁 $\alpha[i:] < \alpha[j:] \Leftrightarrow R[i] < R[j]$

👁 $R[i] = \#j : S[j:] < S[i:]$

👁 R can be built from S in $\Theta(n)$ time.



Df: $LCP(\beta, \gamma) = \max\{k \mid \beta[:k] = \gamma[:k]\}$
↑ longest common prefix

LCP array L: $L[i] := LCP(\alpha[S[i]:], \alpha[S[i+1]:])$

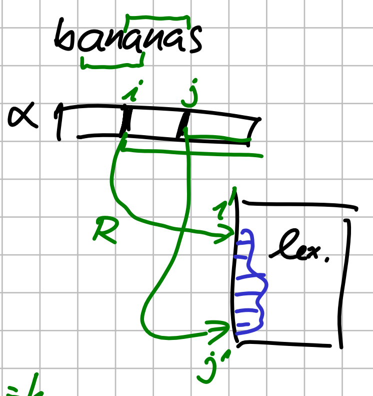
Claim: L can be built from S, R in $\Theta(n)$ time.

Using S + R + L for solving string problems:

① k-gram histogram # of occurrences of every k-char. substring

Split sorted list of suffixes after positions where $L[i] < k$
→ blocks — occurrences of a k-gram
— trivial with a single word shorter than k
} $O(n)$ time

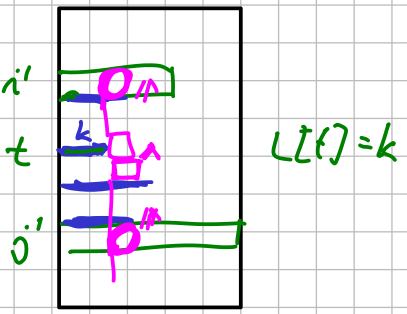
② the longest repeating substring



Lemma: $LCP(\alpha[i:], \alpha[j:]) = \min\{L[t] \mid i' \le t < j'\}$

$i' := R[i]$
$j' := R[j]$
WLOG $i' < j'$

Proof: Let $k := \min\{\dots\}$
① $LCP(\_) \ge k$
② $\le k$



↳ Range-Minimum Query D.S for L ⟶ use 1-D range queries from prev. lecture   Build $O(n)$ RangeMin $O(\log n)$

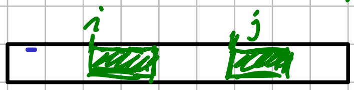
Build, RangeMin

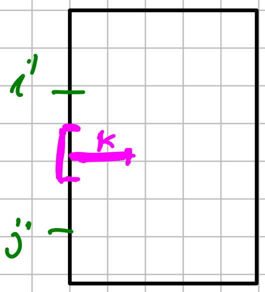
↘ better solutions exists: Build $O(n)$ RangeMin $O(1)$

↳ idea: try all starts i, j
compute $LCP(\alpha[i:], \alpha[j:])$
find max

↳ it suffices to consider lex. adjacent suffixes

$\max_{i'} L[i']$ $O(n)$ time



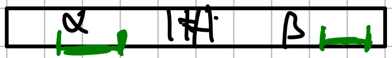


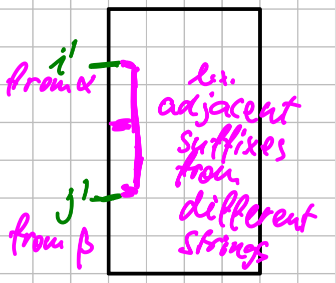
③ longest common substring of strings α, β

- Build S+R+L for a string $\alpha\#\beta$
  - # doesn't occur in α, β $\# < x$ for all $x \in \Sigma$

👁 suffixes of $\alpha\#\beta$ ⟺ suffixes of α ∪ suffixes of β

$\alpha = abx$ $abx\#yc$ → ε
$\beta = yc$ $\#yc$
$abx\#yc$
$bx\#yc$
c
$x\#yc$
yc



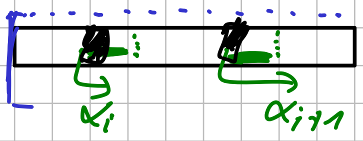


- Find max LCP$(\alpha[i:], \alpha[j:])$ s.t. $i < |\alpha|$, $j > |\alpha|$

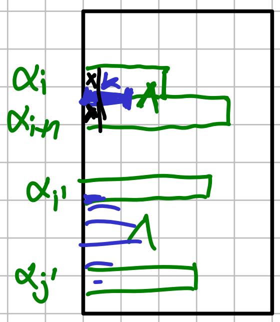
↳ reduce to $\max\{L[t] \mid t : S[t] < |\alpha|, S[t+1] > |\alpha|\}$

$O(n)$

Kasai's alg. : $S, R \rightarrow L$



Let $\alpha_i := \alpha[S[i]:]$



s.t. $LCP(\alpha_i, \alpha_{i+1}) = k > 0$
$L[i] = k$

find $\alpha_{i'} = \alpha_i[1:]$
$\alpha_{j'} = \alpha_{i+1}[1:]$

$i' < j'$

$LCP(\alpha_{i'}, \alpha_{j'}) = k-1$

so every $L[t]$ for $i' \leq t < j'$ is $\geq k-1$

so $L[i'] \geq k-1$

process suffixes in order of decreasing length

Amortization : potential $= k \in [0 \_ n]$

k is changed by $++$, $--$

$\Rightarrow$ # increases $\leq \underbrace{\text{\# decreases}}_{\leq n} + n \leq 2n$

total time is $\Theta(n)$.

Building suffix array

For $k = 1, 2, 4, 8, \ldots$ sort suffixes by the first k characters

Df: $\beta <_k \gamma \equiv \beta[:k] < \gamma[:k]$ not a linear order: $abc \leq_2 abd$

$abd \leq_2 abc$

although $abc \neq abd$

quasi-order

S_k … like S wrt. $\leq_k$

R_k …. like R : $R_k[i] := \#j : \alpha[S_k[i:]] <_k \alpha[S_k[j:]]$

$$\alpha[i:] \leq_{2k} \alpha[j:] \Leftrightarrow \alpha[i:] <_k \alpha[j:] \vee (\alpha[i:] =_k \alpha[j:] \,\&\, \alpha[i+k:] \leq_k \alpha[j+k:])$$

$$\Leftrightarrow R_k[i] < R_k[j] \vee (R_k[i] = R_k[j] \,\&\, R_k[i+k] < R_k[j+k])$$

$$\Leftrightarrow (R_k[i], R_k[i+k]) \leq_{lex} (R_k[j], R_k[j+k])$$

may be $\geq n$ → then replace by empty suffix

↳ using a general-purpose sorting alg. : $O(n \log n)$ time

Bucketsort with n+1 Buckets $O(n)$ time

↳ $O(\log n)$ doubling steps we have S → $O(n \log^2 n)$ total

$O(n \log n)$