## Lowest common ancestor (LCA)

### Range minimum query (RMQ):

Alg.1 no preprocessing, $O(n)$ query

Alg.2 $O(n^2)$ preprocessing, $O(1)$ query

Alg.3 $O(n \log n)$ preprocessing, $O(1)$ query

**RMQ$\pm 1$:** $O(n)$ preprocessing, $O(1)$ time

split to blocks, use alg.2 within blocks, alg.3 between blocks

many blocks repeat in input – save time

**LCA:** $O(n)$ preprocessing, $O(1)$ time

use RMQ on array of depths in depth-first search

**RMQ:** $O(n)$ preprocessing, $O(1)$ time

use LCA on Cartesian tree

## Precomputing values over intervals

Operation $\circ$, compute $R_\circ(i,j) = A[i] \circ A[i+1] \circ \cdots \circ A[j]$

- Precompute all answers: $O(n^2)$ preprocessing, $O(1)$ query
- Precompute prefix "sums" $R_\circ(0,i)$

  good for groups (e.g. $\circ = +$ over Z,Q,R,etc.)

  $R_\circ(i,j) = R_\circ(0,j)) \circ R_\circ(0,i-1))^{-1}$

  **Optional HW:** what about multiplication?
- Precompute $\log_2 n$ intervals for each $i$

  combine 2 overlapping answers e.g. for min
- Precompute non-overlapping intervals of sizes $2^i$

  combine several intervals to cover each element exactly once

  good for any associative $\circ$, e.g. matrix multiplication

  **Optional HW:** running time/memory?

## Finding all small numbers

We have array $A$ precomputed for RMQ.

For given $i$, $j$, $x$ find all indices $k \in \{i, \ldots, j\}$ s.t. $A[k] \leq x$.

```
1  void small(i,j,x) {
2     if(j>i) return;
3     k = rmq(i,j);
4     if(a[k]<=x) {
5        print k;
6        small(i,k-1);
7        small(k+1,j);
8     }
9  }
```

Running time? (as a function of $p$, the number of printed indices)

## Printing documents

Preprocess texts $\{S_1, \ldots, S_z\}$

Query: which documents contain pattern $P$?

We can do $O(m + k)$ where $k =$number of occurrences of $P$

Want $O(m + p)$ where $p =$number of documents containing $P$

Array of leaves $L$ in DFS order

For leaf $L[i]$ let $A[i]$ be the index of previous leaf from the same $S_j$

Occurrences of $P$: subtree of corresponding to interval $[i, j]$ in $L$

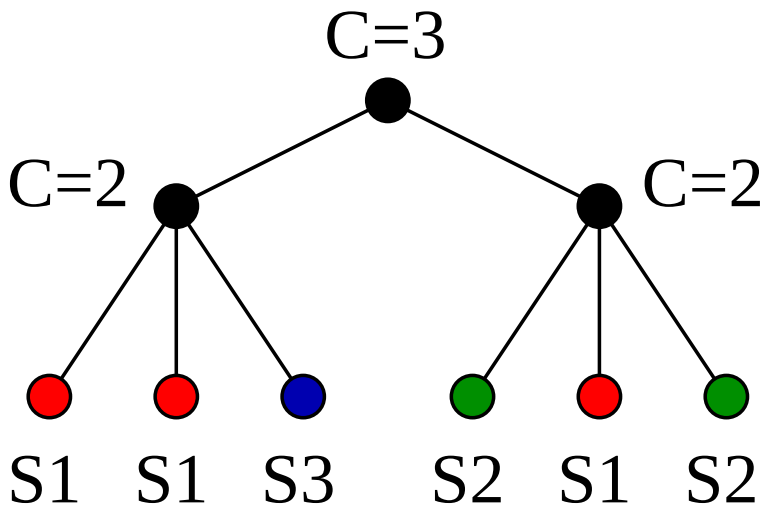
Find all $k \in [i, \ldots, j]$ that have $A[k] < i$

Running time? Preprocessing?

## Recall: counting documents

Generalized suffix tree of $\{S_1, \ldots, S_z\}$

For each node $C(v)$: how many different $S_i$ in its subtree




Use:

– find longest string which is a substring of each $S_i$

– how many $S_i$ contain pattern $P$?

Trivial: $O(nz)$; better: $O(n)$ using LCA

## Applications of suffix trees

– Index text for string matching

– Find longest substring with at least 2 occurrences

– Find longest words which occurs in at least 2 documents

– Find all maximal repeats

### With LCA

– Find maximal palindromes

– Find approximate matches under Hamming distance

– Find pattern with wildcards

– Count in how many documents pattern occurs

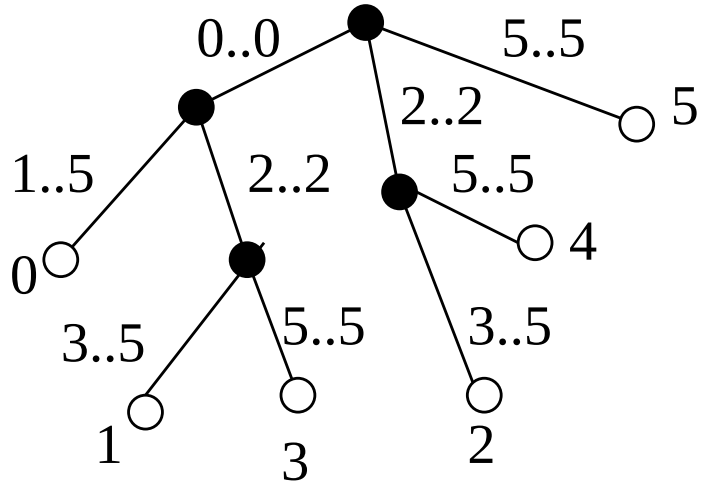
### With RMQ

– Print documents containing pattern

## Summary: suffix trees

- Compact representation of all suffixes of a string
- They can be built in $O(n)$ time (proof next week)
- They can answer interesting problems related to substring equality
- They need relatively large memory

  (several pointers/integers per character)



## Suffix array

Array of suffixes in lexicographic order

(assume $\$ < a \quad \forall a \in \Sigma$)

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $S[i]$ | b | a | n | a | n | a | $ |

| $i$ | $SA[i]$ | Suffix |
|---|---|---|
| 0 | **6** | $ |
| 1 | **5** | a$ |
| 2 | **3** | ana$ |
| 3 | **1** | anana$ |
| 4 | **0** | banana$ |
| 5 | **4** | na$ |
| 6 | **2** | nana$ |

| $i$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $S[i]$ | a | a | a | a | $ |

| $i$ | $SA[i]$ | Suffix |
|---|---|---|
| 0 | **4** | $ |
| 1 | **3** | a$ |
| 2 | **2** | aa$ |
| 3 | **1** | aaa$ |
| 4 | **0** | aaaa$ |

## Suffix array

- Array of suffixes in lexicographic order
- Simpler structure, continuous memory
- Less memory: one index per character ($4n$ bytes in total)
- Construction in $O(n)$ even for large alphabets (next week)

**Today:** Search for pattern $P$ in suffix array

## String matching with suffix arrays

Given suffix array for text $T$ (and possibly other structures), and pattern $P$, solve these three tasks:

- Task 1: Find out if $P$ occurs in $T$ (yes/no)
- Task 2: Count the number of occurrences of $P$ in $T$
- Task 3: List all occurrences of $P$ in $T$

Use binary search in $SA$

that for string $X$ finds highest $i$ such that $T[SA[i]..n] < X$.

## Binary search in suffix array: algorithm 1, $O(m \log n)$

```
1   //find max i such that T[SA[i]..n]< X
2   L = 0; R = n;
3   while(L < R){
4     k = (L + R + 1) / 2;
5     if(T[SA[k]..n]<X){
6       L = k;
7     }
8     else {
9       R = k — 1;
10    }
11  }
12  return L;
```

## Longest common prefix

- $\mathrm{lcp}(A, B) =$ the length of longest common prefix of strings $A$ and $B$
- $\mathrm{LCP}(i, j) = \mathrm{lcp}(T[SA[i]..n], T[SA[j]..n])$

  i.e. lcp of two suffixes in a suffix array

## Excercise

In one iteration we do $\mathrm{lcp}(X, T[SA[k]..n]) + 1$ comparisons

This can be any number between 1 and $\min(m, n + 1 - SA[k])$

Find a bad case (lower bound) for any values $m, n \geq 2$.

## Binary search in suffix array: algorithm 2, $O(m \log n)$

```
1   //find max i such that T[SA[i]..n—1]< X
2   L = 0; R = n;
3   XL = lcp(X, T[SA[L]..n]); XR = lcp(X, T[SA[R]..n]);
4   while(R — L > 1){
5     k = (L + R + 1) / 2;
6     h = min(XL, XR);
7     while(T[SA[k]+h]==X[h]) { h++; }
8     if(T[SA[k]+h]<X[h]){ L = k; XL = h; }
9     else { R = k; XR = h; }
10  }
11  sequential search in SA[L..R];
```

## Excercise

What is the number of comparisons for $T = ba^{n-1}\$$, $X = a^{n-1}\$$?

What is the number of comparisons for $T = a^{n}\$$, $X = a^{n-1}\$$?

## Binary search in suffix array: algorithm 3, $O(m + \log n)$

Recall: $LCP(i, j) = lcp(T[SA[i]..n - 1], T[SA[j]..n - 1])$

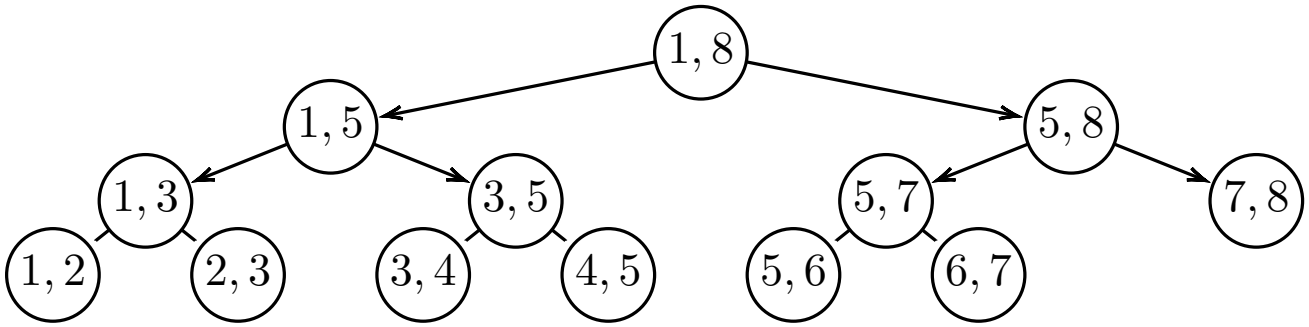
Comparing $T[SA[k]..n]$ and $X$, assume $XL \geq XR$

- If $LCP(L, k) > XL$: set $L \leftarrow k$
- If $LCP(L, k) < XL$: set $R \leftarrow k$; $XR \leftarrow LCP(L, k)$;
- If $LCP(L, k) = XL$: start comparing at $XL$

Case $XL < XR$ symmetrical to $XL \geq XR$

## LCP values for algorithm 3

Which values are needed? $\mathrm{LCP}(L, k)$ or $\mathrm{LCP}(R, k)$




$2n - 1$ LCP values needed

Let $L[i] = \mathrm{LCP}(i, i+1)$, precompute to an array in $O(n)$ (later)

For $j - i > 1$:

$$\begin{aligned} \mathrm{LCP}(i,j) &= \min\{\mathrm{LCP}(k, k+1) \mid k = i \ldots j-1\} \\ &= \min\{\mathrm{LCP}(i,x), \mathrm{LCP}(x,j)\} \text{ for any } x \in \{i+1, \ldots j-1\} \end{aligned}$$

## Summary

| Data structure | Search | # pointers/integers |
|---|---|---|
| Suffix tree | $O(m \log \sigma)$ | $7n$ or more |
| Suffix array | $O(m \log n)$ | $n$ |
| Suffix array + LCP | $O(m + \log n)$ | $3n$ |

More memory needed in preprocessing stage.