Chapter 4. Sets & Sequences

In this chapter we discuss 3 topics:

- Optimal binary search trees
- Pattern matching
- The Union. Find probl.

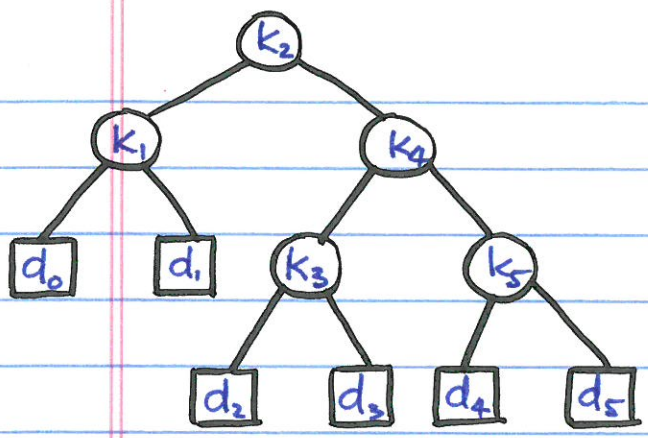
4.1. Optimal Binary Search Trees

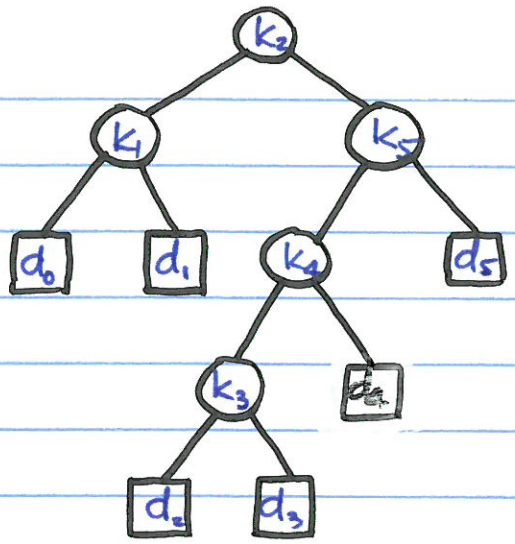
A **binary search tree** for a set of distinct keys $S = \{k_1 < k_2 < \dots < k_n\}$ is a bin. tree with n internal nodes $v_1, \dots, v_n$ labeled by $k_1, \dots, k_n$ and $n+1$ leaves labeled by $d_1, \dots, d_{n+1}$ representing unsuccessful searches s.t. if v_i (v_j) is a node in left (right) subtree of v_k, then

$$\text{label}(v_i) < \text{label}(v_k) < \text{label}(v_j)$$

Ex. $S = \{k_1 < k_2 < \dots < k_5\}$

i		1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10





3 operations on bin. search trees:

SEARCH, INSERT, DELETE.

Complexity of SEARCH, INSERT, DELETE is bounded by depth of tree.

Thm. The expected number of comparisons required to insert n randomly chosen elements into an initially empty bin. search tree is $O(n \lg n)$. The worst-case complexity is $\Theta(n^2)$

Pf. Let $a_1, a_2, \ldots, a_n$ be input sequence and $b_1, b_2 \ldots, b_n$ be sequence sorted in ascending order. Then

$$P(a_i = b_j) = 1/n \quad \text{for } j = 1, \ldots, n$$

When building a search tree for $a_1, \ldots, a_n$:

- a_1 becomes root
- left subtree contains $b_1, \ldots, b_{j-1}$
- right subtree contains $b_{j+1}, \ldots, b_n$

Thus,

$$E[T(n)] = \frac{1}{n} \sum_{j=1}^{n} \{ E[T(j-1)] + (j-1) + E[T(n-j)] + (n-j) \}$$

(# of comparisons w/ a_1)

$$= \frac{1}{n} \sum_{j=1}^{n} (n-1 + E[T(j-1)] + E[T(n-j)])$$

$$= (n-1) + \frac{2}{n} \sum_{j=0}^{n-1} E[T(j)]$$

(similar to QUICKSORT 's average running time)

$$\leq cn \lg n \quad \text{for some } c.$$

If $a_1, a_2, \ldots, a_n$ is initially sorted, then worst-case complexity is $\Omega(n^2)$.

The $O(n^2)$ upper bound is trivial as each INSERT operation is $O(n)$. □

Weighted Trees.

$$S = \{k_1 < k_2 < \dots < k_n\}$$

Dummy keys $d_0, d_1, \dots, d_n$ repr. unsuccess. ful searches. For $i = 1, \dots, n$, $j = 0, \dots, n$:

p_i = Probability of search for k_i

q_j = Probability of search for d_j

It holds: $\sum_{i=1}^{n} p_i + \sum_{j=0}^{n} q_j = 1$

Let T be a bin. search tree (BST) for S.

b_i^T = depth of k_i in T

a_j^T = depth of d_j in T

Consider a search for x in universe:

- $x = k_i$: $b_i^T + 1$ comparisons
- $x = d_j$: a_j^T comparisons

Thus, expected cost of a search is:

$$P^T = \sum_{i=1}^{n} p_i \cdot (b_i^T + 1) + \sum_{j=0}^{n} q_j a_j^T$$

which is called **weighted path length** of BST T.

T is an optimal BST for S if its weighted path length is minimal

Notation. T_{opt} denote an optimal BST for S
P_{opt} denote its weighted path length

Goal. Given $S = \{k_1 < \cdots < k_n\}$,
dummy keys $d_0, d_1, \ldots, d_n$, and
probability dist. $(q_0, p_1, q_1, \ldots, p_n, q_n)$,
construct T_{opt} for S in $O(n^2)$ time/space

$\Rightarrow$ We apply dynamic programming

Given a BST T for $S = \{k_1 < \cdots < k_n\}$
and $d_0, d_1, \ldots, d_n$, a subtree of T has
- nodes $k_i, \ldots, k_j$ and leaves $d_{i-1}, \ldots, d_j$

Such subtree is a search tree for subset
$\{k_i < \cdots < k_j\}$, $d_{i-1}, \ldots, d_j$ with prob. def. by:

Let $w_{ij} = q_{i-1} + p_i + q_i + \cdots + p_j + q_j$, and
for $\ell = i, \ldots, j$ and $m = i-1, \ldots, j$:

$$\bar{p}_\ell := p_\ell / w_{ij}, \quad \bar{q}_m := q_m / w_{ij}$$

Then: $$\sum_{\ell=i}^{j} \bar{p}_\ell + \sum_{m=i-1}^{j} \bar{q}_m = 1$$

Note: $w_{i,i-1} = q_{i-1} \quad i = 1, \ldots, n$

Let T_{ij} denote an opt. BST for $\{k_i < \cdots < k_j\}$, $d_{i-1}, \ldots, d_j$ with prob. distr. $(\bar{q}_{i-1}, \bar{p}_i, \bar{q}_i, \ldots, \bar{p}_j, \bar{q}_j)$.

Let P_{ij} denote weighted path length of T_{ij} and r_{ij} denote the index of root of T_{ij}, i.e., $r_{ij} = m$ s.t. k_m is root of T_{ij}.

We have the recursion:

(a) $P_{i,i-1} = 0$, $P_{ii} = 1$, $i = 1, \ldots, n$

(b) for $1 \le i < j \le n$:

$$P_{ij} = 1 + \min_{i \le m \le j} \left\{ \frac{w_{i,m-1}}{w_{ij}} P_{i,m-1} + \frac{w_{m+1,j}}{w_{ij}} P_{m+1,j} \right\}$$

For convenience, letting $\bar{P}_{ij} = w_{ij} P_{ij}$:

(a') $\bar{P}_{i,i-1} = 0$, $\bar{P}_{ii} = w_{ii}$

(b') $$\bar{P}_{ij} = w_{ij} + \min_{i \le m \le j} \{ \bar{P}_{i,m-1} + \bar{P}_{m+1,j} \}$$

($w_{i,i-1} = q_{i-1}$ $\quad i = 1, \ldots, n$)

Opt-BST (p, q, n) $\quad \bar{P}_{n+1,n} = 0$, $w_{n+1,n} = q_n$

```
for i = 1 to n do   w[i,i-1] = q[i-1]
    w[i,i] = q[i-1] + p[i] + q[i]
    P̄[i,i] = w[i,i] ; P̄[i,i-1] = 0;
for l = 1 to n-1 do
    for i = 1 to n-l do
        j = i + l ; P̄[i,j] = ∞
        w[i,j] = w[i,j-1] + p[j] + q[j]
(§)     for m = i to j do
            s = w[i,j] + P̄[i,m-1] + P̄[m+1,j]
            if s < P̄[i,j] then
                P̄[i,j] = s
                r[i,j] = m
```

Prop. Opt-BST constr. for S an opt. BST T_{opt} in $O(n^3)$ time and $O(n^2)$ space

Next Goal. Improve Opt-BST's running time to $O(n^2)$.

Ex. $S = \{k_1 < k_2 < k_3 < k_4\}$

$(q_0, p_1, q_1, \dots, p_4, q_4) = (1/6, 1/24, 0, 1/8, 0, 1/8, 1/8, 0, 5/12)$

Then the arrays $\overline{P}$, w, and r are:

$$24\overline{P} = \begin{bmatrix} 0 & 5 & 11 & 24 & 48 & 1 \\ & 0 & 3 & 12 & 31 & 2 \\ & & 0 & 6 & 22 & 3 \\ & & & 0 & 13 & 4 \\ & & & & 0 & 5 \\ 0 & 1 & 2 & 3 & 4 & \end{bmatrix}$$

$$24w = \begin{bmatrix} 4 & 5 & 8 & 14 & 24 \\ & 0 & 3 & 9 & 19 \\ & \downarrow & 0 & 6 & 16 \\ & w_{i,i-1} & & 3 & 13 \\ & & & & 10 \end{bmatrix}$$

$1 \le i \le 5, \quad 0 \le j \le 4$

$$r = \begin{bmatrix} 1 & 1 & 2 \to 3 & \\ & & \downarrow & \\ & 2 & 3 & 4 \\ & & 3 & 4 \\ & & & 4 \end{bmatrix}$$

$1 \le i \le 4, \quad 1 \le j \le 4$

$\overline{P}_{14} = ?$ $\quad m = 1, \dots, 4$

$m=1$: $24\overline{P}_{14} = 24\left(w_{14} + \overline{P}_{1,0} + \overline{P}_{24}\right) = 24 + 31 = 55$

$m=2$: $24\overline{P}_{14} = 24\left(w_{14} + \overline{P}_{1,1} + \overline{P}_{34}\right) = 24 + 5 + 22 = 51$

$m=3$: $24\overline{P}_{14} = 24\left(w_{14} + \overline{P}_{12} + \overline{P}_{44}\right) = 24 + 11 + 13 = 48$

$m=4$: $24\overline{P}_{14} = 24\left(w_{14} + \overline{P}_{13} + \overline{P}_{54}\right) = 24 + 24 + 0 = 48$

$\Rightarrow r_{14} = 3, \quad \overline{P}_{14} = \frac{48}{24}$ $\square$

Observation. $r_{i,j-1} \le r_{i,j} \le r_{i+1,j}$

i.e., matrix r is monotone in each row and column.

Ex. $S = \{k_1 < k_2 < k_3 < k_4\}$

$(q_0, p_1, q_1, \dots, p_4, q_4) = (1/6, 1/24, 0, 1/8, 0, 1/8, 1/8, 0, 5/12)$

Then the arrays $\overline{P}$, w, and r are:

$$\overline{P} = \begin{bmatrix} 0 & 5 & 11 & 24 & 38 \\ & 0 & 3 & 12 & 31 \\ & & 0 & 6 & 22 \\ & & & 0 & 13 \\ & & & & 0 \end{bmatrix} \qquad 24w = \begin{bmatrix} 4 & 5 & 8 & 14 & 24 \\ & 0 & 3 & 9 & 19 \\ & & 0 & 6 & 16 \\ & & & 3 & 13 \\ & & & & 10 \end{bmatrix}$$

$$r = \begin{bmatrix} 1 & 1 & 2 \to 3 & \downarrow \\ & 2 & 3 & 4 \\ & & 3 & 4 \\ & & & 4 \end{bmatrix} \qquad \square$$

Thus, the **for** loop (§) can be mod.:

for $m = r[i, j-1]$ **to** $r[i+1, j]$ **do**

Then the running time of Opt-BST is:

$$O\left(\sum_{\ell=1}^{n-1} \sum_{i=1}^{n-\ell} \left(1 + r_{i+1,i+\ell} - r_{i,i+\ell-1}\right)\right)$$

$$= O\left(\sum_{\ell=1}^{n-1} \left(n - \ell + r_{n-\ell+1,n} - r_{1,\ell}\right)\right)$$

$$= O\left(\sum_{l=1}^{n-1} n\right) = O(n^2)$$

Dyn. Progr. with Quadrangle Inequalities

Let $w(i,j) \in \mathbb{R}$ for $1 \le i \le j \le n$ and

$$c(i,i) = 0$$

$$c(i,j) = w(i,j) + \min_{i<k\le j} \{c(i,k-1) + c(k,j)\}$$

Rem. Opt. BSTs are a special case:

If $w(i,j) := w_{i+1,j} = q_i + p_{i+1} + q_{i+1} + \cdots + p_j + q_j$

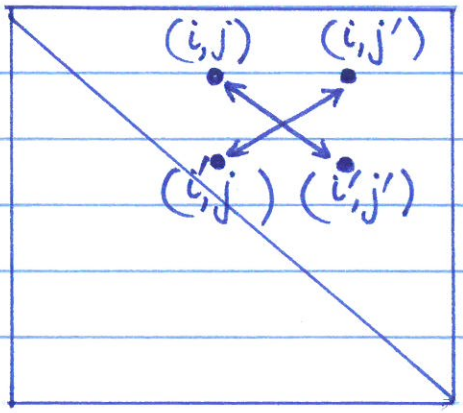
then $c(i,j) = \overline{p}_{i+1,j}$ □

Suppose that $w(i,j)$ sat. the quadrangle inequality, i.e., for $i \le i' \le j \le j'$:

$$\boxed{(QI) \quad w(i,j) + w(i',j') \le w(i',j) + w(i,j')}$$

Illustration.

$w =$



w is monotone w.r.t. interval inclusion if

$$w(i',j) \leq w(i,j') \quad \text{for } i \leq i' < j \leq j'.$$

Lem 1. If w sat. (QI) and is monotone, then c(i,j) also sat. the QI:

(QIc) $c(i,j) + c(i',j') \leq c(i',j) + c(i,j')$

for $i \leq i' \leq j \leq j'$.

Notation. Let $c_k(i,j) := w(i,j) + c(i,k-1) + c(k,j)$

and $K(i,j) := \text{Max}\{k \mid c_k(i,j) = c(i,j)\}$

Lem 2. If c sat. the QI, then K is monotone in each row and column, i.e.,

for $i \leq j$: $K(i,j) \leq K(i,j+1) \leq K(i+1,j+1)$.

From Lem 1 and Lem 2 we obtain

Theorem. If w sat. the quadrangle inequality and is monotone, then c(i,j) can be computed in $O(n^2)$ time.

Pf of Lem 1. By induction on $\ell = j' - i$

Basis. $\ell \leq 1$. Then either $i = i'$ or $j = j'$.

If $i = i'$, then (QIc) becomes

$$c(i,j) + c(i,j') \leq c(i,j') + c(i,j')$$

which is obviously true. Case $j = j'$ is similar.

Inductive step. Consider 2 cases: $i' = j$ or $i' < j$.

Case 1. $i < i' = j < j'$. Then (QIc) is:

$$c(i,j) + c(j,j') \leq c(i,j')$$

which is an inverse triangle inequality.

Letting $k = K(i,j')$ we have 2 subcases:

Case 1.1. $k \leq j$.

Since $c(i,j') = w(i,j') + c(i,k-1) + c(k,j')$,

$$c(i,j) + c(j,j') \leq \underbrace{w(i,j) + c(i,k-1) + c(k,j)}_{\geq c(i,j)} + c(j,j')$$

$$\leq w(i,j') + c(i,k-1) + c(k,j) + c(j,j')$$

(since w is monotone)

By ind. hyp. $c(k,j) + c(j,j') \leq c(k,j')$.

Hence

$$\leq w(i,j') + c(i,k-1) + c(k,j')$$

$$\leq c(i,j').$$

Case 1.2. $k \geq j$: similar to case 1.1.

Case 2. $i < i' < j < j'$.

Letting $k_1 = K(i,j')$ and $k_2 = K(i',j)$ we distinguish 2 cases: $k_1 \leq k_2$ or $k_1 \geq k_2$

Case 2.1. $k_1 \leq k_2$

Note that $k_1 \leq k_2 \leq j$

and $k_1 > i$ (from def. of c)

We have: $c(i,j) + c(i',j') \leq c_{k_1}(i,j) + c_{k_2}(i',j')$

$= [w(i,j) + c(i,k_1-1) + c(k_1,j)] + [w(i',j') + c(i',k_2-1) + c(k_2,j')]$

(QIw) $\Rightarrow \leq w(i,j') + w(i',j) + c(i,k_1-1) + c(k_1,j) + c(i',k_2-1) + c(k_2,j')$

$k_1 \leq k_2 \leq j < j'$

(IH on / QIc) $\Rightarrow \leq w(i,j') + w(i',j) + c(i,k_1-1) + c(i',k_2-1) + c(k_2,j) + c(k_1,j')$

$= w(i,j') + c(i,k_1-1) + c(k_1,j') + w(i',j) + c(i',k_2-1) + c(k_2,j)$

$= c(i,j') + c(i',j)$

Case 2.2. Similar to Case 2.1. ⊠

Pf of Lemma 2. Since the case $i = j$ is trivial, we consider $i < j$ and prove

Claim. $K(i,j) \leq K(i,j+1)$

We prove that for all $i < k \leq k' \leq j$:

$$c_{k'}(i,j) < c_k(i,j) \Rightarrow c_{k'}(i,j+1) \leq c_k(i,j+1)$$

which means that if $K(i,j)$ "prefers" k' over k, then so does $K(i,j+1)$.

A stronger statement is

$$c_k(i,j) - c_{k'}(i,j) \leq c_k(i,j+1) - c_{k'}(i,j+1)$$

which is equiv. to

$$c_k(i,j) + c_{k'}(i,j+1) \leq c_{k'}(i,j) + c_k(i,j+1)$$

which in turn is equiv. to (by exp. using def)

$$c(k,j) + c(k',j+1) \leq c(k',j) + c(k,j+1).$$

This is exactly QI for c at $k \leq k' \leq j \leq j+1$.

The ineq. $K(i,j+1) \leq K(i+1,j+1)$ can be shown in a similar fashion. ⊠

4.2. String Matching

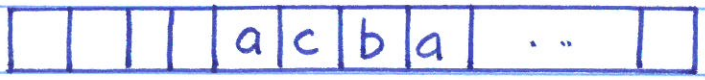
Given text $T[1..n]$

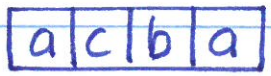
pattern $P[1..m]$, $T[i], P[j] \in \Sigma$

P occurs with shift s, $0 \leq s \leq n-m$

if $P[1..m] = T[s+1..s+m]$

s is then a valid shift

Ex: $T =$ 

$P =$ 

$s = 4$

Problem. Given T and P, find all valid shifts!

Some terminologies.

Σ^* = set of all strings over alphabet Σ

ε denotes empty string

The concatenation of 2 strings x, y is: xy. $|xy| = |x| + |y|$.

w is prefix of x if $x = wy$: $w \sqsubset x$

w is suffix of x if $x = yw$: $w \sqsupset x$

Lemma. Let $x, y, z \in \Sigma^*$ be s.t. $x \sqsupset z$ and $y \sqsupset z$. If $|x| \leq |y|$, then $x \sqsupset y$.
If $|x| \geq |y|$, then $y \sqsupset x$.
If $|x| = |y|$, then $x = y$.

The naive string matching algorithm

Naive-String-Matcher (T, n, P, m)

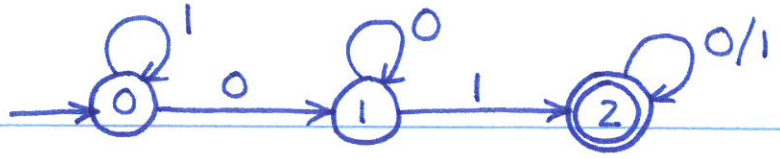
```
for s = 0 to n-m do
    if P[1..m] = T[s+1,..,s+m]
        then print s.
```

Complexity: $\Theta((n-m+1)m) = \Theta(n^2)$

String Matching with Finite Automata

Def. A finite automaton (FA) M is a 5-tuple $M = (Q, \Sigma, \delta, q_0, A)$, where

(1) Q = set of states
(2) Σ = input alphabet
(3) $q_0 \in Q$ is start state
(4) $A \subseteq Q$ is set of final states
(5) $\delta : Q \times \Sigma \to Q$ is transition fct.

Ex. 

$Q = \{0, 1, 2\}$ $\Sigma = \{0, 1\}$

0 is initial state, $\{2\}$ = set of final states

This FA accepts bin. strings containing 01 as a substring. □

M induces a final state fctn

$$\varphi : \Sigma^* \longrightarrow Q$$
$$w \longmapsto \delta(q_0, w)$$

i.e., $\varphi(w)$ is the last state of M after processing w.

(Thus, if w is accepted by M, then $\varphi(w) \in A$.)

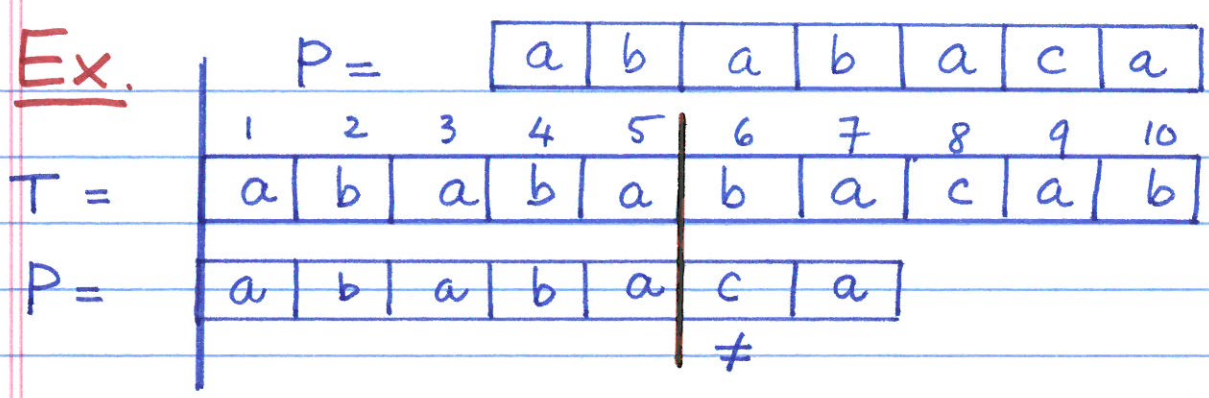
String-Matching Automata

Def. (Suffix Function) Given $P[1..m]$.

The suffix function of pattern P is a function $\sigma : \Sigma^* \longrightarrow \{0, 1, \ldots, m\}$ s.t. $\sigma(x) =$ length of longest prefix of P that is suffix of x

$$= \text{Max}\{k \mid P[1..k] \sqsupset x\}$$

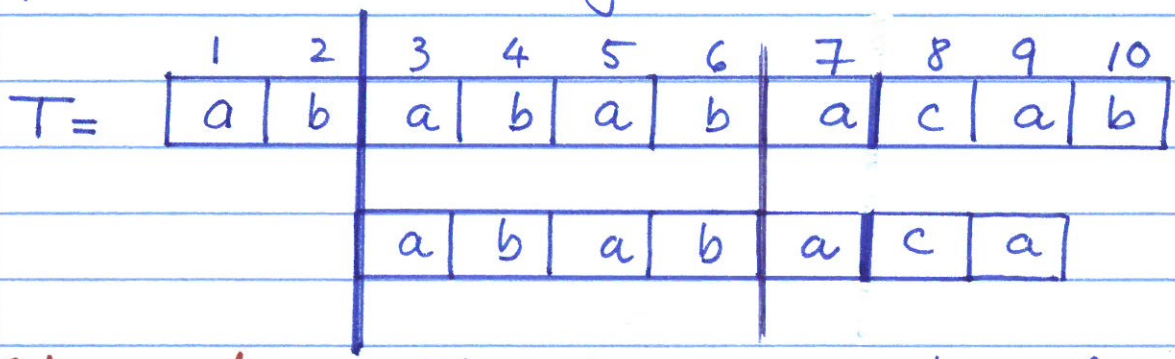
Ex.



Q. How far to the right should we slide P?
What is the longest prefix of P that is suffix of T[1..6]?

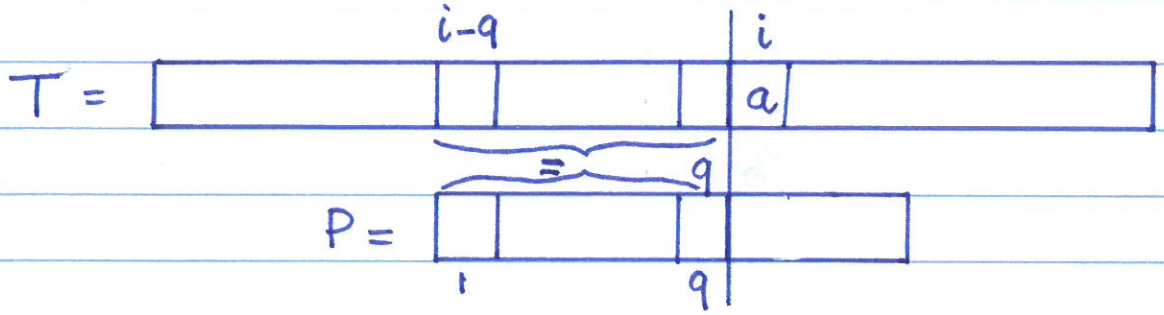
Answer: abab and $\sigma(ababab) = 4$

Thus, σ tells us to slide the pattern to the right s.t.



Observation. The decision on how far to the right we need to slide P depends only on the portion of text currently being compared with pattern.

$\Rightarrow$ Thus it depends on just P

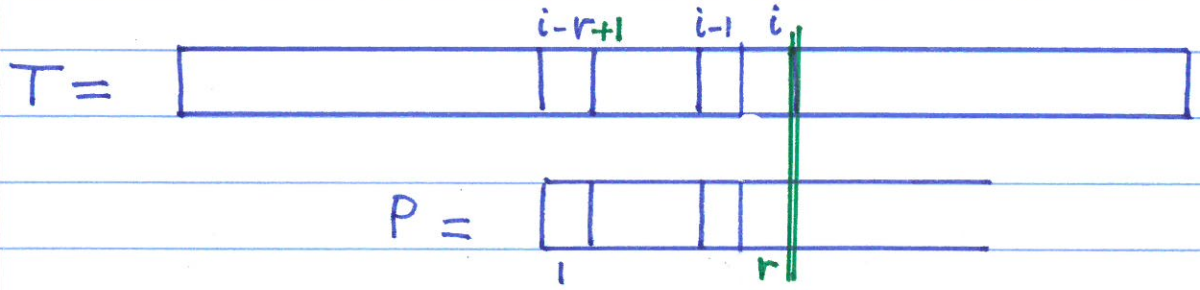


$\sigma(P[1..q]a)$ (or $\sigma(P_q a)$)

tells us:

- if $a = P[q+1]$ then move to next cell
- Otherwise, if $r = \sigma(P[1..q]a)$ then slide P to the right s.t.

$$P[1..r] = T[i-r+1..i]$$



$x =$ | c | c | a | b |
| a | b |

Ex. $P = ab$. $\sigma(\varepsilon) = 0$

$\sigma(cca) = 1$, $\sigma(ccab) = 2$. □

Fact. $x \supset y \Rightarrow \sigma(x) \leq \sigma(y)$

Def. For pattern $P[1..m]$ def. a **String-matching automaton** M as follows:

(1) $Q = \{0, 1, \ldots, m\}$

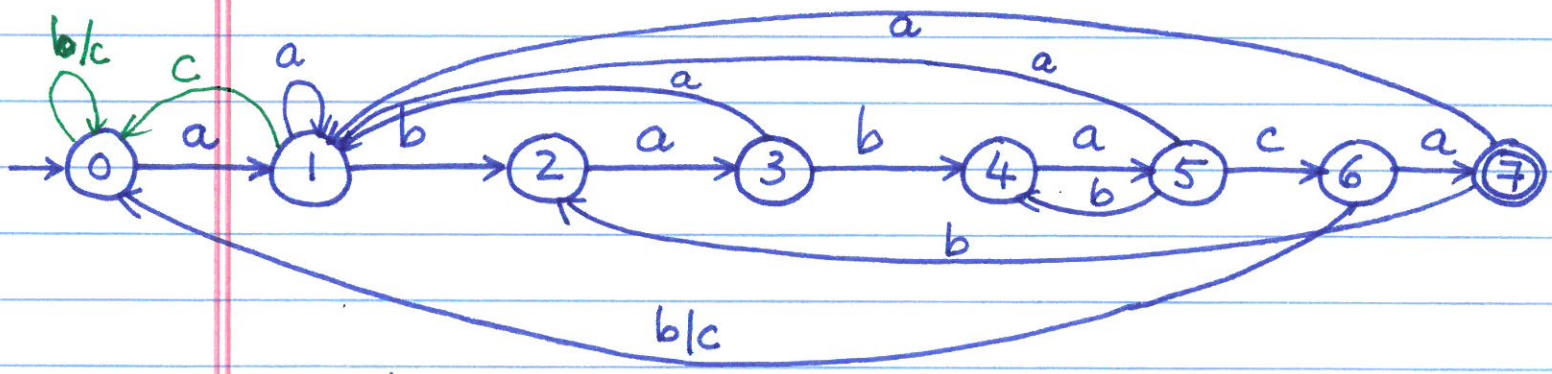
(2) 0 is initial state

(3) m is the only accept state

(4) δ is def. by $\delta(q, a) := \sigma(P_q a)$

where $P_q := P[1..q]$

Ex. $P[1..7] = abababaca$



	a	b	c
0	1	0	0
1	1	2	0
2	3	0	0
3	1	4	0
4	5	0	0
5	1	4	6
6	7	0	0
7	1	2	0

This is an FA accepting strings ending in P.

$\delta(5, b) = \sigma(P_5 b)$

$= \varphi(P_5 b)$

i	1	2	3	4	5	6	7	8	9	10
$T[i]$	a	b	a	b	a	b	a	c	a	b
$\varphi(T[1..i])$	0	1	2	3	4	5	4	5	6	7	2

□

Lem. (1) $\sigma(xa) \leq \sigma(x)+1$

(2) $\sigma(x)=q \Rightarrow \sigma(xa)=\sigma(P_q a)$

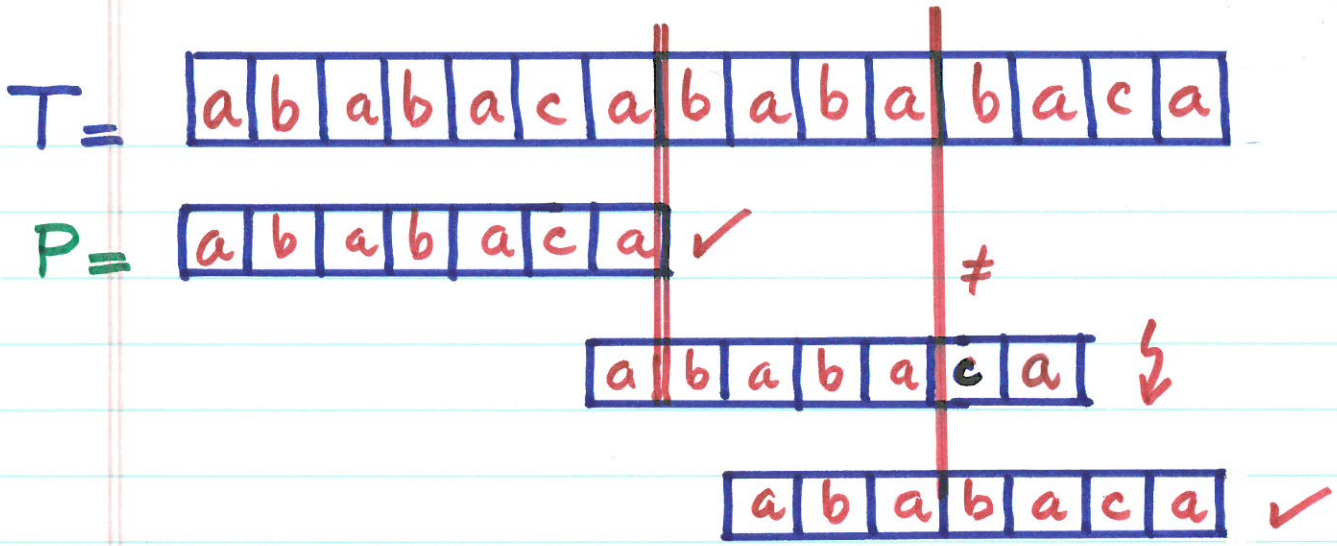
FA

```
FA-Matcher (T, n, M, m)
    q = 0
    for i = 1 to n
        q = δ(q, T[i])
        if q = m
            then  s = i - m
                  print "valid shift s"
```

Theorem. Given $T[1..n]$, $P[1..m]$, string-matching FA $M=(Q,\Sigma,\delta,0,\{m\})$ for P. Then for $i=1,\ldots,n$

$$\varphi(T[1..i]) = \sigma(T[1..i])$$

Running time of FA-Matcher is $\Theta(n)$ if pattern P is fixed.


T=
a b a b a c a b a b a b a c a
P=
a b a b a c a
✓
≠
a b a b a c a
a b a b a c a
✓

Question. How to compute δ for the string-matching automaton?

COMPUTING-TRANSITION-FUNCTION(P, m, Σ)

```
for q = 0 to m do                          O(m)
    for each a ∈ Σ do                      O(|Σ|)
        k = min{m+1, q+2}
        repeat                             O(m)
            k = k-1
        until P_k ⊐ P_q a                  O(m)
        δ(q,a) = k
return δ
```

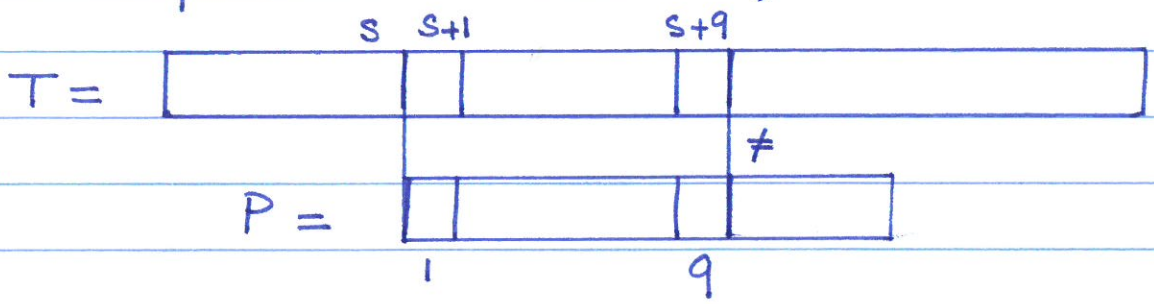
Running time is $O(m^3 \cdot |\Sigma|)$

Question. Can we do better?

Observation. When trying to match pattern against text, the question about how far we should slide P to the next shift doesn't depend on symbol being compared $\Rightarrow$ ~~FA~~

Knuth-Morris-Pratt Algor. (KMP)

In previous discussion, we've seen:



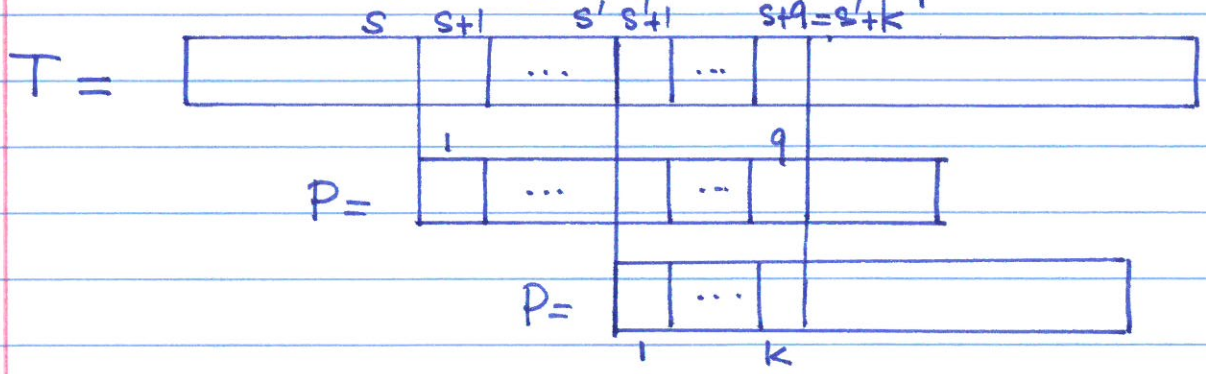
If $T[s+q+1] \neq T[q+1]$, we slide P to the next shift s' $(>s)$ s.t. for some $k < q$:

$$P[1..k] \supset T[s'+1..s'+k]$$

Note: $T[s'+1..s'+k] \supset P[1..q]$

Thus, k is largest s.t.

$$P[1..k] \sqsupset P[1..q]$$

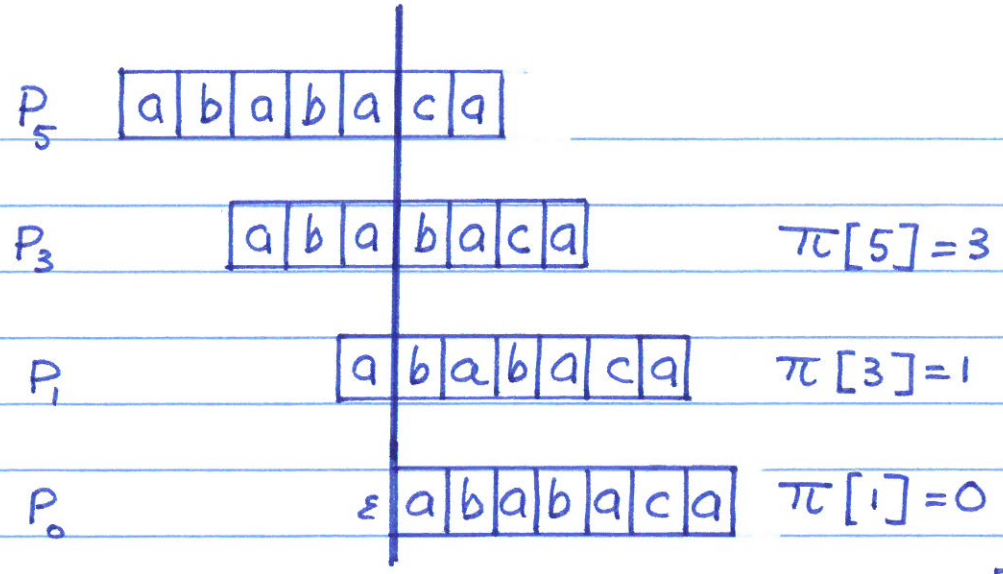


Def. (Prefix Function) Given $P[1..m]$, the prefix function for P is

$$\pi : \{1, 2, .., m\} \longrightarrow \{0, .., m-1\}$$

s.t. $\pi[q] = \max\{k < q \mid P_k \supset P_q\}$

Ex.



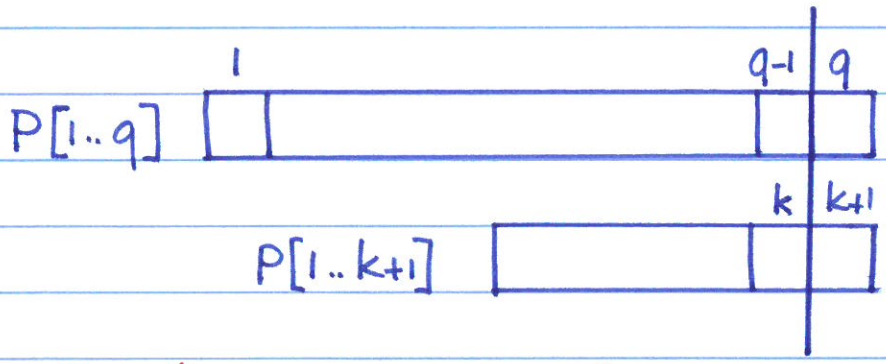
□

KMP-Matcher(T, n, P, m)

```
π = COMPUTE-PREFIX-Fct(P, m)
q = 0
for i = 1 to n do
    while q > 0 and P[q+1] ≠ T[i]
        do q = π(q)
    if P[q+1] = T[i] then
        q = q+1
    if q = m then
        print "i-m is valid shift"
        q = π[q]
```

Compute_Prefix_Fctn (P, m)

```
π[1] = 0
k = 0
for q = 2 to m do
(*)     while k > 0 and P[k+1] ≠ P[q] do
            k = π[k] ;
(**)    if P[k+1] = P[q] then
            k = k+1 ;
        π[q] = k
return π
```



Complexity of KMP algor.

(**) increases k at most m-1 times.

When exec. $k = \pi(k)$, k is decreased by (*) at **most** m-1 times.

Thus, total running time of Compute_Prefix_Fctn is $\Theta(m)$.

Example:

P	a	b	a	b	a	c	a
k	1	2	3	4	5	6	7
π	0	0	1	2	3	0	

$\pi(4) = 2$

$\pi(5) = 3 \Rightarrow k = 3$

$\pi(6) = ?$ $\quad k = 3, \; q = 6$

$P[k+1] = b \neq c = P[q]$

$k = \pi(k) \Rightarrow k = \pi(3)$

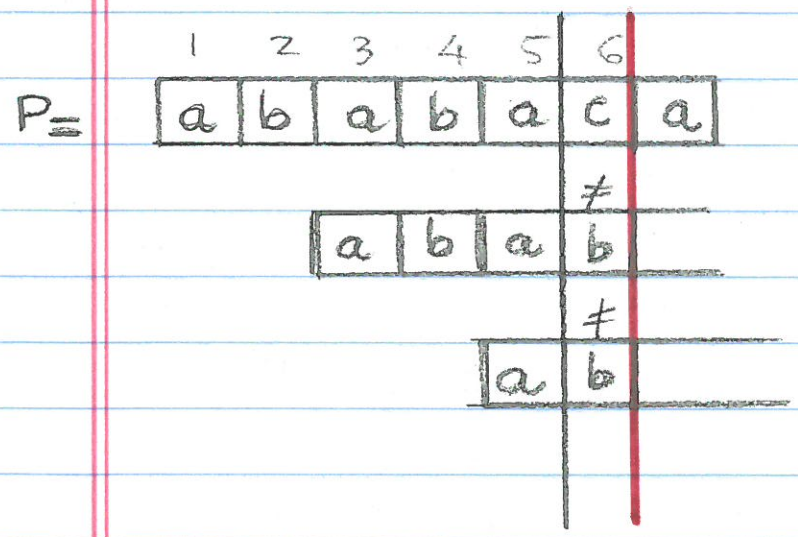
$k = 1$

$P[k+1] = b \neq c = P[q]$

$k = \pi(k) \Rightarrow k = \pi(1)$

$k = 0$

Thus, $\pi(6) = 0$



$\pi(6) = ?$

$\pi(5) = 3$

$\pi(3) = 1$

$\pi(1) = 0$

$\Rightarrow \pi(6) = 0$

4.3. Data Structures for Disjoint Sets

Goal. Maintain a collection $\mathcal{S}$ of disjoint dynamic sets $\{S_1, S_2, \ldots, S_k\}$, $S = \bigcup_{i=1}^{k} S_i$.
Each S_i is represented by an elem. in S_i

There are 3 operations:

- MAKE-SET(x) = Create a set contain. x with x as representative.
- UNION(x,y) = $S_i \cup S_j$, $x \in S_i$, $y \in S_j$
- FIND(x) = return y s.t. $x \in S_y$.

Running time analysis is done in terms of:

- n = # of MAKE-SET operations = $|S|$
- m = # of MAKE-SET, UNION & FIND

Note: $m \geq n$ and n MAKE-SET oper. are performed first.

An Application (Finding connected comp.)

```
CONNECTED-COMPONENT(G=(V,E))
    for each v∈V do MAKE-SET(v)
    for each e=(u,v)∈E do
        if FIND(u) ≠ FIND(v) then
            UNION(u,v)
```

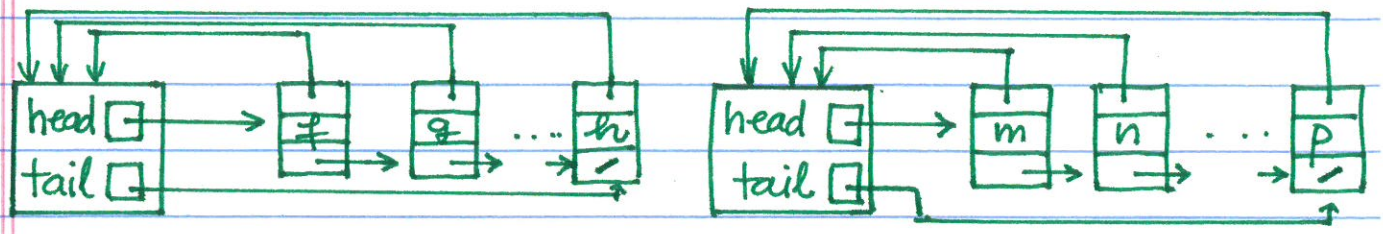
SAME-COMPONENT (u,v)

if FIND (u) = FIND(v)

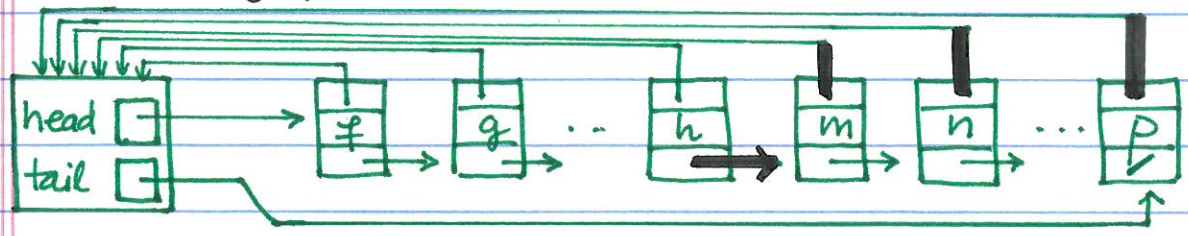
then return true

else return false

Linked_List Representation



UNION (g,p):



- Append list cont. p to list cont. g
- Update (tail) pointer in h to point to m
- Update tail pointer to point to p
- Update pointers in p's list to point to head of g's list

Claim. There is a sequence of m operations on n objects that requires $\Theta(n^2)$ time.

Pf. Let $S = \{x_1, \ldots, x_n\}$.

- First perform MAKE-SET(x_i), $i = 1, \ldots, n$
- Then perform $n-1$ UNION operations UNION(x_2, x_1), UNION(x_3, x_2), ..., UNION(x_n, x_{n-1})

Since each UNION(x_i, x_{i-1}) requires $i-1$ updates of pointers, the $n-1$ UNION oper. requires $\sum_{i=2}^{n} (i-1) = \Theta(n^2)$ steps.

Thus, total of $2n-1$ operations costs $\Theta(n^2)$ time. Each costs $\Theta(n)$ time on average □

Observation. Above implementation performs badly since pointers on longer lists have to be updated.

Weighted-Union Heuristic

Idea. Append shorter lists to longer ones!

Theorem. Using weighted_union heuristic, a sequence of m MAKE_SET, UNION and FIND operations takes $O(m + n\log n)$ time.

Pf. Each time x's pointer is updated, x is on shorter list.

After an update, list containing x is at least twice longer.

Thus, x's pointer is updated at most $\lceil \log n \rceil$ times $\Rightarrow$ n UNIONs take $O(n\lg n)$ time. Thus total time is $O(m+n\lg n)$. □

Tree Representation

In linked_list implementation:

- FIND costs $O(1)$ time
- UNION costs $O(\lg n)$ time on average

A more efficient solution for UNION:

- Each set S_i is repr. by a tree:
  - root is labeled by representative of S_i

- Other nodes labeled by members of S_i

- UNION takes $O(1)$ time:
  - To unite S_x, S_y make root of smaller tree child of root of larger tree
- FIND (x) takes time proportional to depth of node x in tree containing x:

Theorem. Using union-by-rank heuristic, a sequence of m MAKE-SET, UNION and FIND operations takes $O(m \lg n)$ time.

To prove Theorem, suppose we perform n MAKE-SET operations first, followed by $\leq n-1$ UNION operations.

Let T be resulting tree. For $x \in S$,

rank(x) = height of x is T

weight(x) = # of x's descendants in T

The above Theor. follows from

Lem. $\forall x \in U$: $\text{weight}(x) \geq 2^{\text{rank}(x)}$ and $\text{rank}(x) \leq \lg n$.

Pf. (By ind. on $\text{rank}(x)$)

Basis. $\text{rank}(x) = 0$.

Obvious since $\text{weight}(x) \geq 1$

Ind. Step. Assume $\text{rank}(x) = k \geq 1$

Then, x has a child z s.t. $\text{rank}(z) = k-1$.

Immediately before z becomes child of x, $\text{weight}(x) \geq \text{weight}(z)$

After UNION,

$$\text{weight}(x) \geq 2 \cdot \text{weight}(z)$$

Later UNIONs do not modify subtree rooted at z. Thus,

$$\begin{aligned} \text{weight}(x) &\geq 2 \cdot \text{weight}(z) \\ &\geq 2 \cdot 2^{\text{rank}(z)} \quad \text{(IH)} \\ &= 2^{\text{rank}(z)+1} \\ &= 2^{\text{rank}(x)} \end{aligned}$$

Since $\text{weight}(x) \leq n$, $\forall x$, $\text{rank}(x) \leq \lg n$.

□

Path Compression : A further Improvement

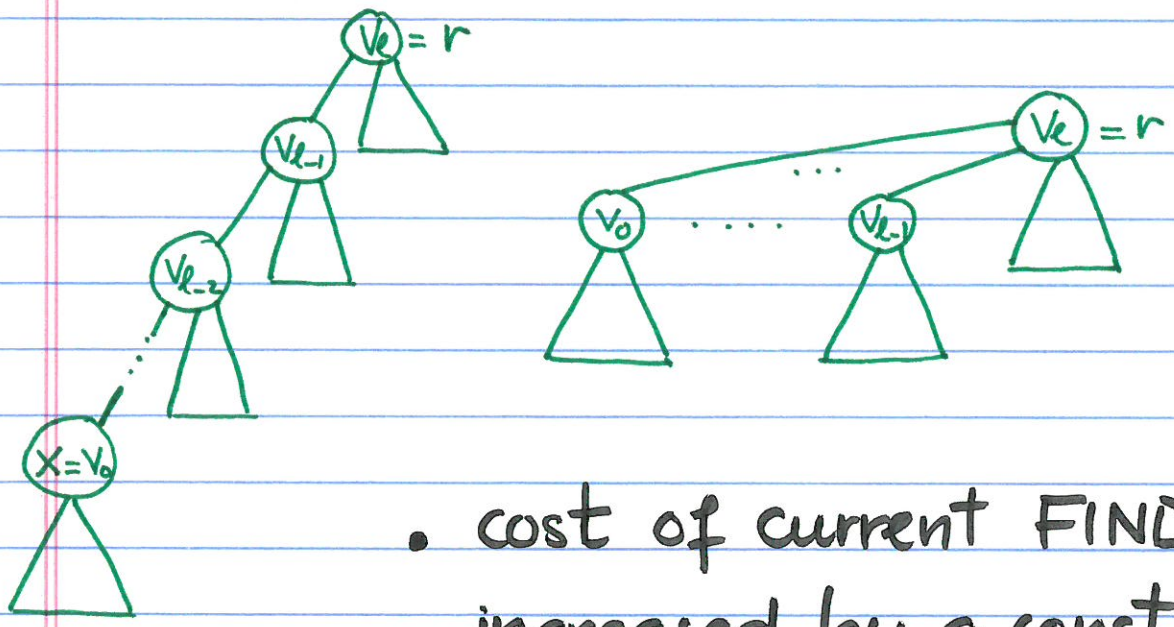
Goal. To reduce the amortized cost of FIND

Suppose the execution of FIND(x) requires to traverse the path

$(x = v_0, v_1, \ldots, v_\ell = \text{root})$

in the tree containig x

$\Rightarrow$ compress this path by making node $v_0, v_1, \ldots, v_{\ell-1}$ children of root $(=r)$



- cost of current FIND is increased by a const. factor

$\Rightarrow$ cost of future FIND (v_i), $i = 0, \ldots, \ell-1$ will be reduced.

Complexity of union_by_rank + path compr.

Using union_by_rank combined with path compression, a sequence of m MAKE-SET, UNION and FIND operations can be done practically in linear time.

Def. (Ackermann's function) For $k \geq 0$, $j \geq 1$:

$$A_k(j) := \begin{cases} j+1 & \text{if } k=0 \\ A_{k-1}^{(j+1)}(j) & \text{if } k \geq 1. \end{cases}$$

Remark. A is an extremely fast-growing fct:

(1) $A_0(j) = j+1$ (successor fctn)

(2) $A_1(j) = 2 \cdot j + 1$

(3) $A_2(j) = 2^{j+1}(j+1) - 1$

(4) $A_3(1) = 2047$

(5) $A_4(1) \geq 10^{80}$

Def (Inverse of Ackermann's fctn)

For $n \geq 0$:

$$\alpha(n) = \min\{k \mid A_k(1) \geq n\}$$

Remark. For $2048 \leq n \leq A_4(1)$, $\alpha(n) = 4$

Theorem. Using union_by_rank combined with path compression, a sequence of m MAKE-SET, UNION and FIND operations can be performed in $O(m \cdot \alpha(n))$ time.