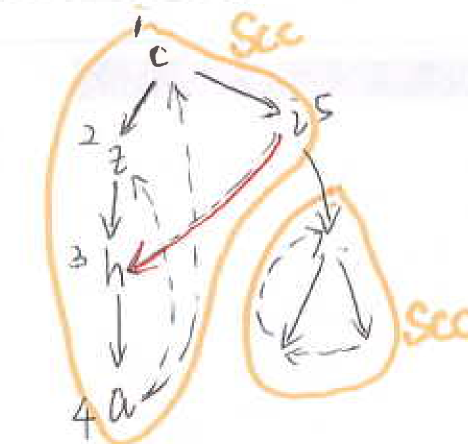
$\boxed{\text{Directed DFS}}$



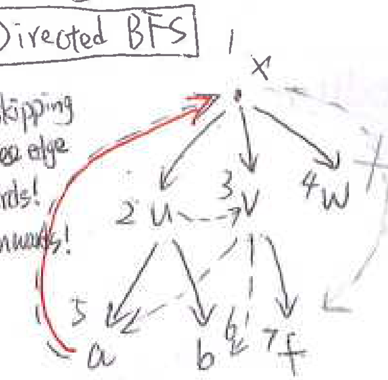
Search #1: From vertex x in G (DFS/BFS)

Search #2: From vertex x in reverse G. (DFS/BFS)

Strong connected component

$\boxed{\text{Directed BFS}}$

level skipping
non tree edge
upwards!
no downwards!



x

~~u~~ ~~v~~ ~~w~~ a b f

BFS: solve shortest path for (all edge has cost of 1).

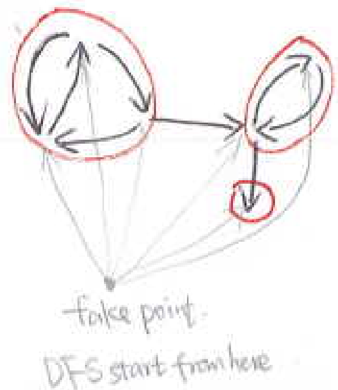
BFS: solve shortest path : use fake points.

$G \rightarrow G' \rightarrow \boxed{BFS} \rightarrow$ shortest path

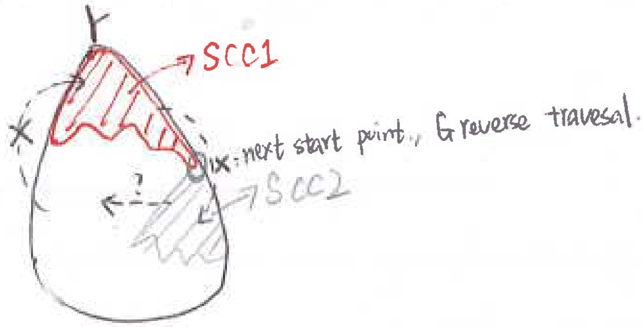
$\underset{G}{u \xrightarrow{2} v} \Rightarrow \underset{G'}{u \xrightarrow{1} \cdot \xrightarrow{1} v}$

Oct 18 CS381

SCC



DFS from root once.
then find next X and do G reverse traversal. then peel off ~~G~~ G reverse subtree of x. find next x...

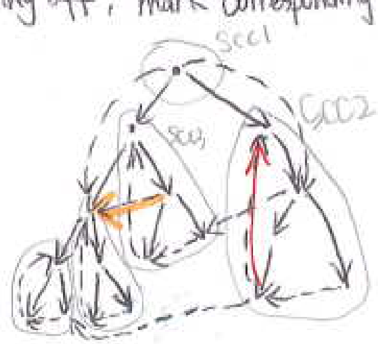


no holes in SCC: if a point is in SCC then it's ~~parents~~ ancestors should be reached too.

Peel off SCC. one by one
- To find next x:

post[x] > post[not peeled off]

right to left.
ask whether a point has been peeled off.
if no, we find next x. do G reverse traversal and peel off scc
when peeling off, mark corresponding points...



identify cross edge and backward edge:
both: DFS#[from] > DFS#[to].
cross edge: onStack[to] = false / markPassed[to] = true
backward edge: onStack[to] = true / markPassed[to] = false

Proof: If a graph has cycles. it must have backward edge.

Assume only tree edges and cross edges. forward edges.

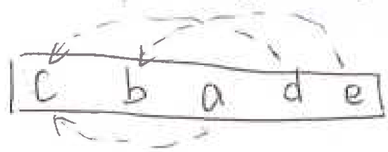
all edges: post order[from] > postorder[to].

$a \to b \to c \to a$. $post[a] > post[a]$

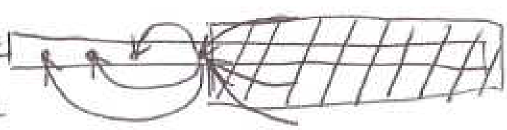
Contradict.

Topological sorting

1. DFS. from fake point.
2. if no backward edges. output according to post order number



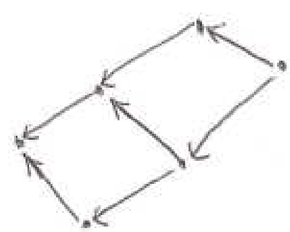
find ~~shortest~~ (longest) path in acyclic graph



right to left. update left points' label that current point could reach

reverse post order

Critical path. (longest path) for points.



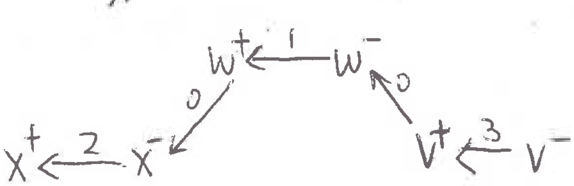
use the longest path for edges.

transform a vertex to two vertices and an edge. assign it a weight.

original edges have 0 weight.

E

V.



$E' = E+V$

$V' = 2V$

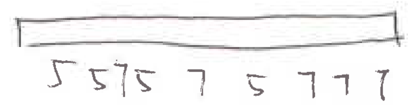
Minimum Spanning Tree

Kruskal: Heap of edges.

pop out edges in increasing order

determine form a cycle or not. (union $\overset{m\log n}{\text{find}}$).

use array, change label to smaller label...



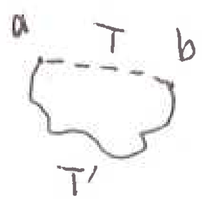
use tree structure...

$\boxed{O(E \log V)}$

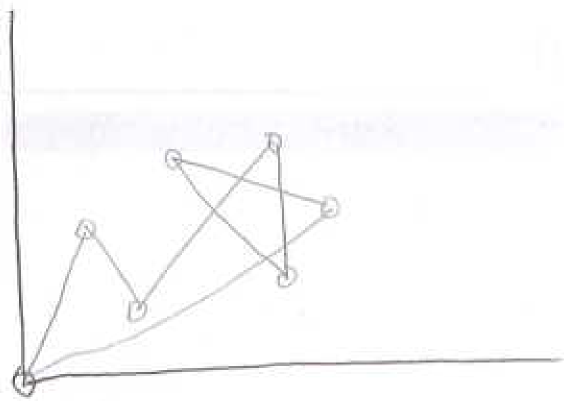
Proof: Kruskal find the MST.

T: Kruskal ~~tree~~ MST

T': other algorithm which find better solution (assumingly)



Compare edges: cost (x, y), min(,), max(,)



Input: points (holes)

Output: Schedule to drill the holes.

Optimal: Expotential.

Approximation algo: MST.

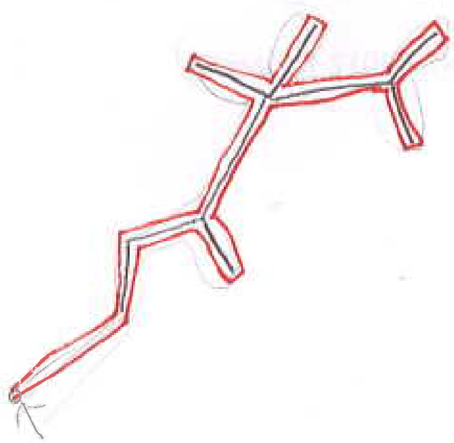
LHS RHS

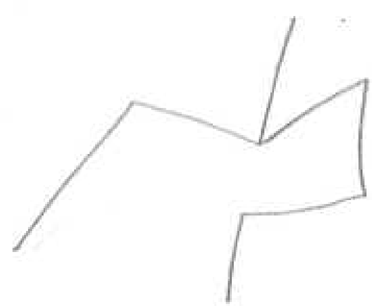
cost of MST based (red path) ≤ 2×optimal

Proof:

LHS = cost of MST × 2

2 cost of MST ≤ 2 cost of OPT

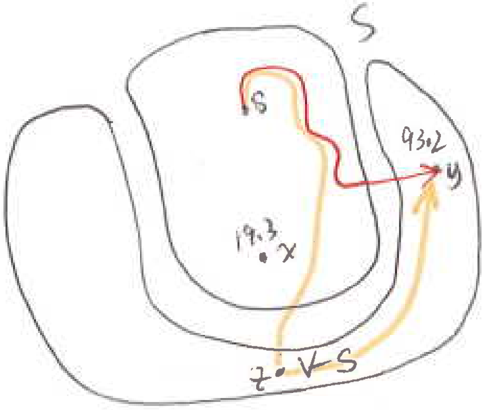




MST

Di

Single source shortest path



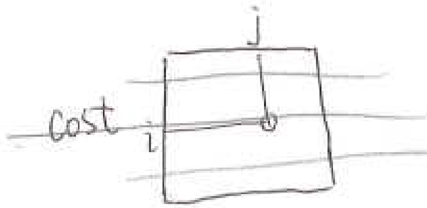
if y has the min label in V−S, y has the final lab

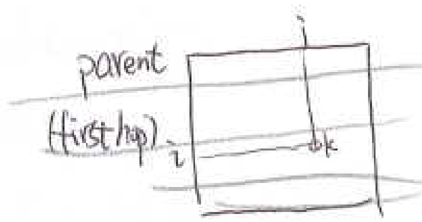
then y could join S, update y's parent.

proof: if there exist another shorter path s→z→y

s→z > s→y (label y min)

contradict with the fact that its shorter.

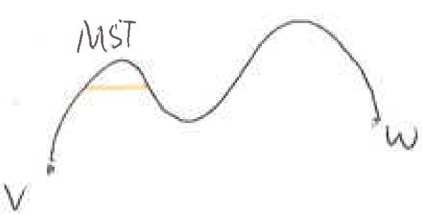




if undirected, cost of a path is the longest edge on that path.

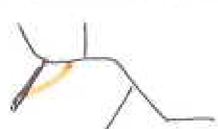
max matrix.

then shortest path from v to w is the edges that connect v and w in MST



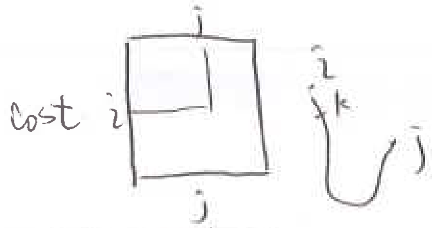
proof: if all edges are distinct, there is only ONE MST.

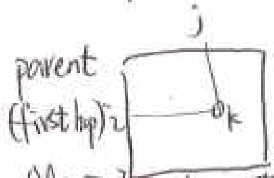
suppose there're two MSTs, choose the shortest edge in one MST but not in another



 Oct 30 CS381

APSP. All Pair Shortest Path.





M_0 = Input matrix

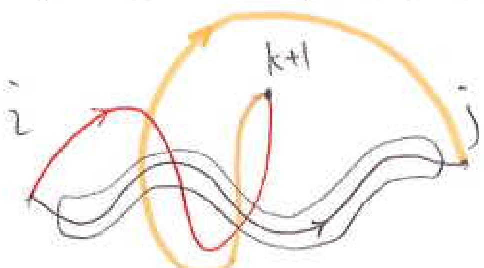
for k = 1 to n $\xrightarrow{?}$ 2n

  for i = 1 to n

    for j = 1 to n

$$M_{k+1}(i,j) = \overset{\text{OR}}{\min} \begin{cases} M_k(i,j) \\ M_k(i,k+1) \overset{\text{AND}}{+} M_k(k+1,j) \end{cases}$$

$M_k(i,j)$: ~~whether there's a path~~ shortest path from i to j that uses intermediate vertices from 1,2, to k.



$\{1, 2, \ldots, k\}$ ← k+1 too!

$M_k(i,j)$

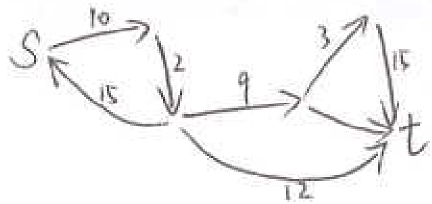
$M_k(i, k+1) + M_k(k+1, j)$

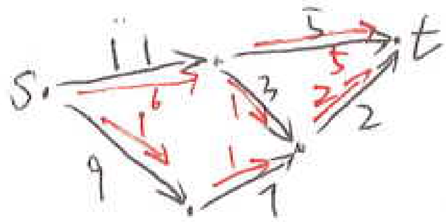
Can't use similar algorithm to find longest path (due to self intersection) (walk, not path)

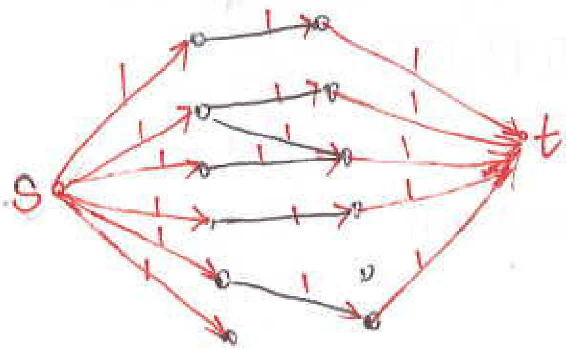
if acyclic (no ~~self~~ cycles). changing min to max to find longest path works! (no self intersection)

 Nov 1 CS381

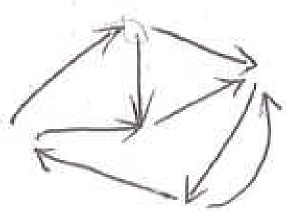
Network flow





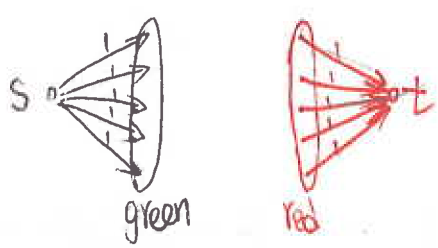


Solve matching bigraph using network flow

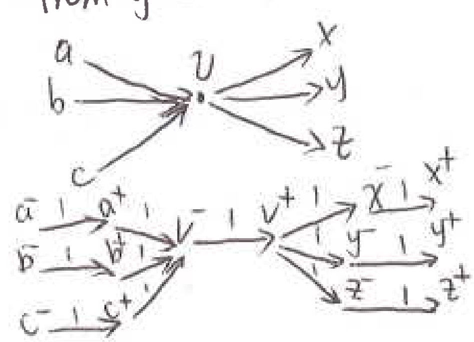


k green vertices
k red vertices

find k edge-disjoint path from green to red



find vertex-disjoint path from green to red



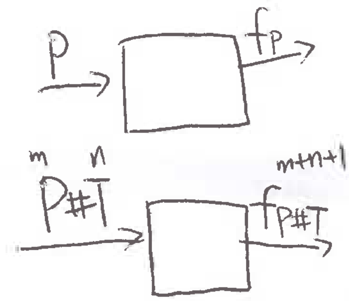
Pattern matching KMP

Text $t_1 \cdots\cdots t_n$

Pattern $P_1 \cdots P_m$

$m < n$

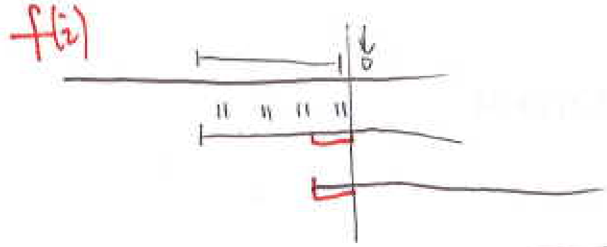
Naive solution : $m(n-m)$ - mn

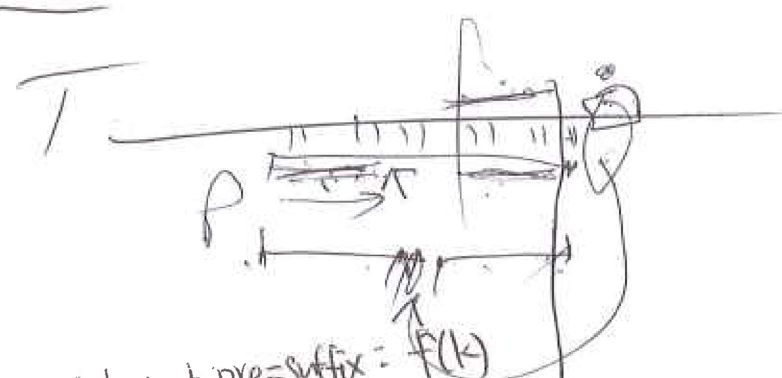


is any $f_{P\#T}(i)$ equal to m

$P_{f(f(i))+1}$ $P_1 \cdots P_i$

$f(i)$ $P_{f(i)+1}$





$\boxed{O(n)}$ T

a b r a b r a h a
0 0 0 1 2 3 4 0 1

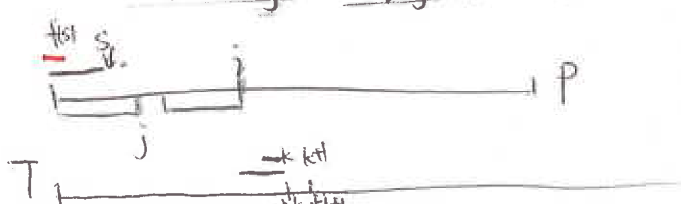
length of longest pre=suffix : $f(k)$

length of third longest pre=suffix : $f(f(f(k)))$

"failure function" f.

$f(i) = $ ir $j < i$

$P_1 \cdots P_j = P_{i-j+1} \cdots P_i$



$P_{s+1} \stackrel{?}{=} t_{k+1}$

Yes → $s+1$

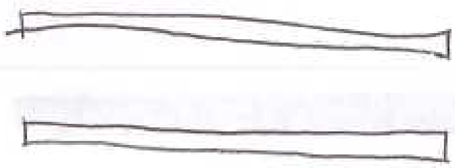
No → $t_{k+1} \stackrel{?}{=} P_{f(s)+1}$

Yes → $f(s)+1$

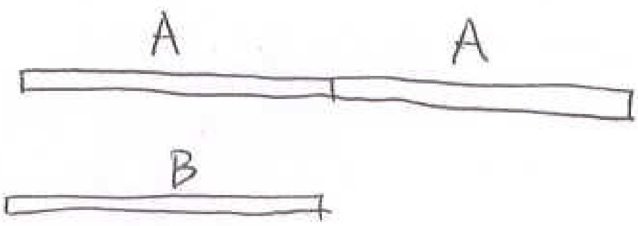
No →

 Nov 8 CS381

Find circular array.

A  abcdef

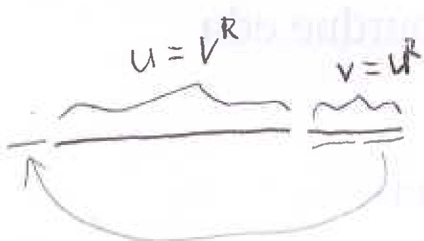
B efabcd

Text


Pattern

then KMP.

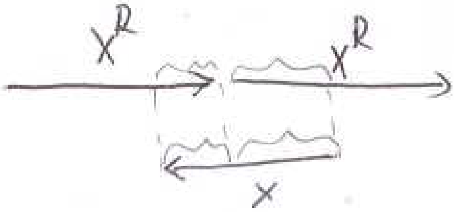
Palindrome.



$T = x^R x^R$

$P = x$

Counter Example:



$\overbrace{\text{c d a b a}}^{5}\ \overbrace{\text{d}}^{1}$ c d a b a d

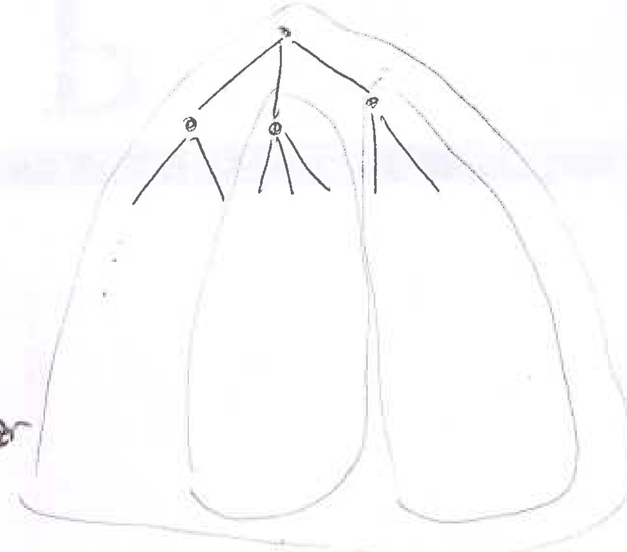
d a b a d c

at least one of the two part must have even length

if both odd length, not the one we want

2-3 B-tree.

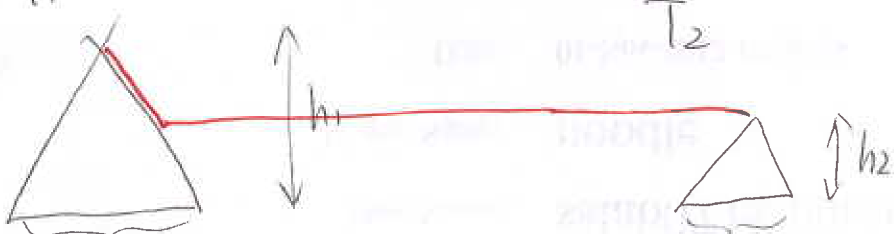
same depth
at least 2 children
at most 3 children



among children, in increasing order
store maximum underneath ...

logn. Search: if bigger than a node, don't go down, if less than a node, go down
logn. insert: ...
delete

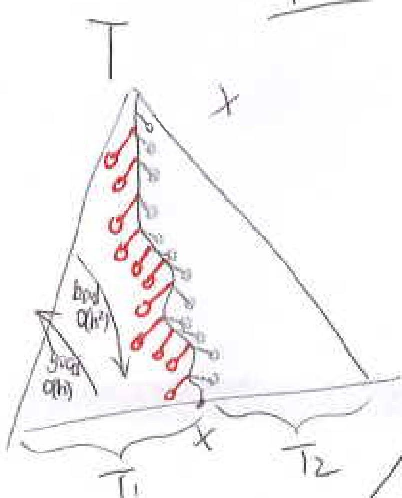
concatinate



$O(h_1-h_2)$

Concatinate (T_1, T_2)

split.

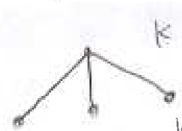


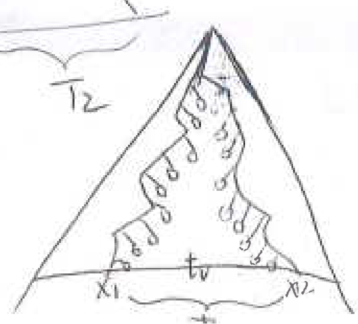
pop from a stack: good way go up.

concatinate $(.....) \longrightarrow T_1$
concatinate $(....) \longrightarrow T_2$

Select(k) in 2-3 tree. $O(\log n)$

augment tree: store number of leaves





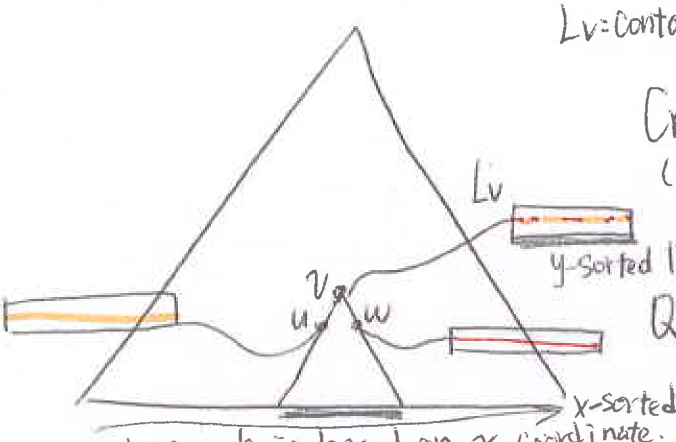
Rang Query (x_1, x_2) $O(\log n + t)$
traverse each node on stack
also look at leaves

Range Query

2-dimension

$Q(x_1, x_2, y_1, y_2)$

Lv: contain all the leaves, y sorted.



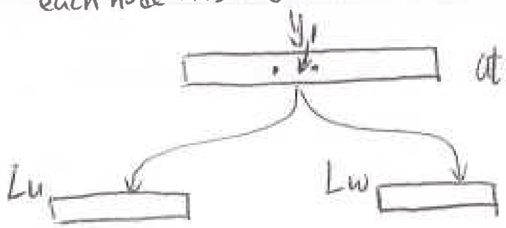
tree node is based on x-coordinate.
each node has a y-sorted list.

Create: $O(n\log n)$ time
(merge) $O(n\log n)$ space

Query $= O((\log n)^{\cancel{2}} + t)$

y_1 & y_2 in L

at root: binary search (once)

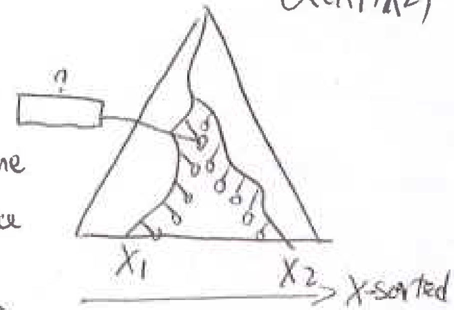


at children of root:

constant time to get rank of children if location in parent is found.
then location in children could be found by following the pointer

1-dimension

$Q(x_1, x_2)$

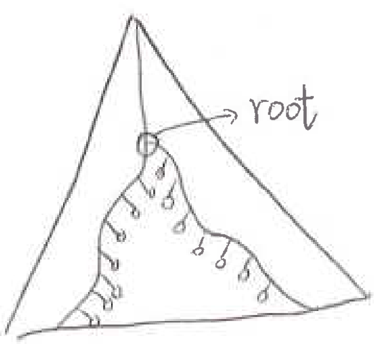


Stupid: binary search on Lv.

$O((\log n)^2 + t)$

good: Store corresponding y rank of left list and right list in Lv. (when creating)

$O(\log n + t)$



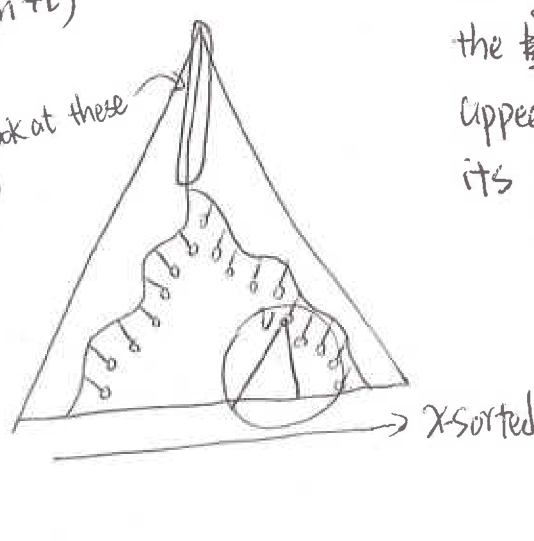
1.5-dimension.

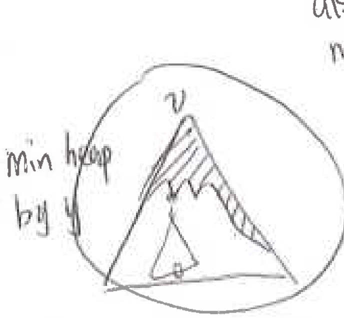
$Q(x_1, x_2, y_2) = Q(x_1, x_2, -\infty, y_2)$

Creat: $O(n)$

Query $= O(\log n + t)$

each node: store extra info
the ~~largest~~ smallest y that has not appear in anywhere of its parents extra info.





traverse v using y_2 as cut off point.
(if minumum is too big, its subtree is too big)

 Nov 15 CS381

locate x in every list.

L_1 L_2 ... L_k

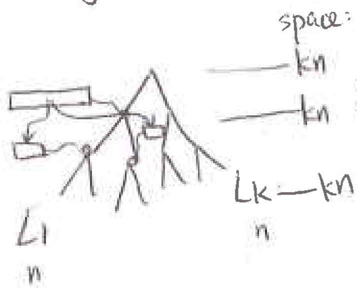
n n n

time

① $O(k \log n)$ (do nothing, binary search each List)

② space: $O(kn \log k)$

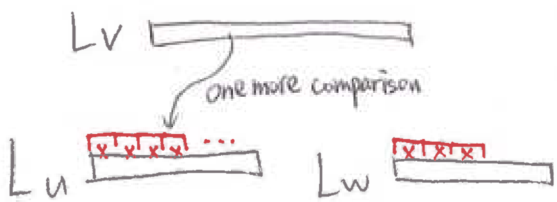
time: $O(k + \log n)$

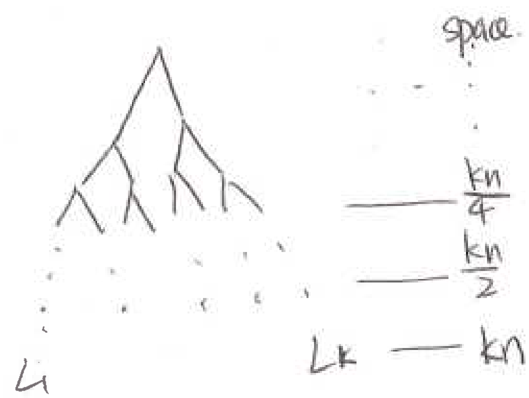


maintain a list in every node, also the ranks.

③ space: $O(kn)$

time: $O(k + \log n)$





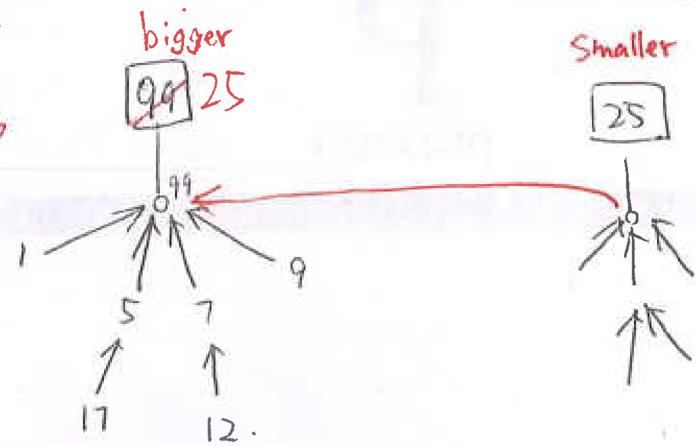
total space: $O(kn)$.

when merging, pick only odd number node thus halfing the space needed for each node

Union-find

size(99) = ~~7~~ 25 ~~13~~

~~size(25) = 6~~



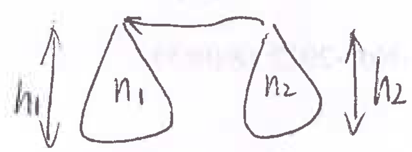
Union(99, 25)

Using such union, in every tree, $h \le \log n$

Induction: when $n = 1$. $h = \log n = 0$.

when $n > 1$.

$n_1 > n_2$



$h_1 \le \log n_1$ $\quad$ $h_2 \le \log n_2$.

Case 1: $h = h_1$

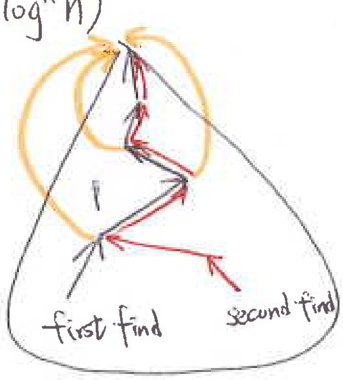
$h = h_1 \le \log(n_1 + n_2)$

Case 2: $h = h_2 + 1$

$h = 1 + h_2 \le \log n_2 + 1 = \log(n_2 + n_2) \le \log(n_1 + n_2)$

Find: $O(\log n)$

Find: $O(n \log^* n)$



(use stack to connect each node directly to root)

Path compression

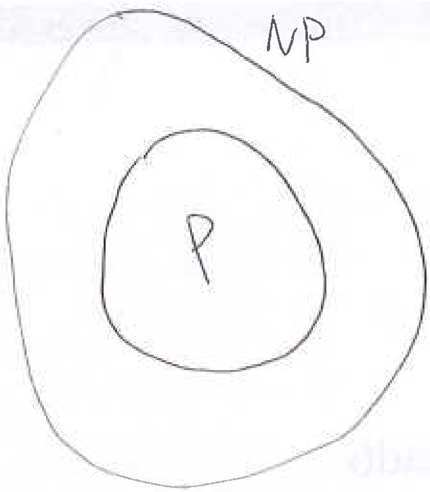
$\log^* n$ looks constant. (grow very slow)

if $\log^* n = 5$.

then n is

$2^{2^{2^{2^{2}}}}$

NP



P:
compute: polynomial

NP:
verify: polynomial

Compute: (exponential) difficult.
non-polynomial

NP-complete

CNF. **3SAT.**

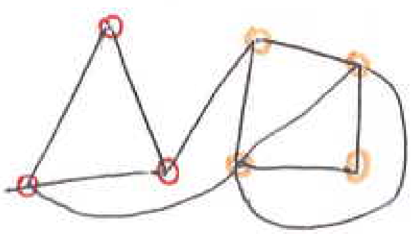
boolean: $X_1 \cdots X_n$

$(\ \vee \ \vee \ ) \wedge (\ \vee \ \vee \ ) \wedge (\ \vee \ \vee \ )$

Given any instance of 3SAT. construct in polynomial time find an instance of clique suth that the solution to Clique is a polinomial solution

$\overset{\#1}{(X_1 \vee \neg X_2 \vee X_{17})} \wedge (\neg X_1 \vee \neg X_2 \vee X_{19}) \wedge \cdots \overset{\#k}{(\quad)}$

Clique



NP-complete is NP-hard + "in NP"

G. int k.

① in NP

② use 3SAT

G. k

Clique is NP-complete

Find largest clique: NP-hard

Nov 29 CS381

Vertex Cover

Graph G, int k

Is there a subset S of the vertices such that

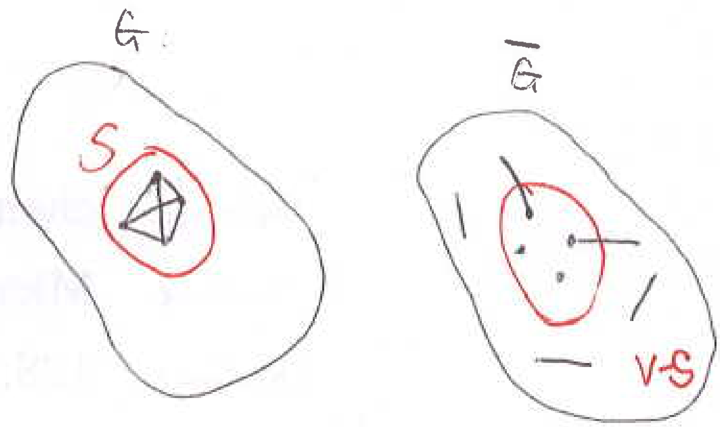
① $|S| = k$

② every edge touches S

Given any instance of Clique create in poly time an instance of vertex cover such that ...

clique of size k in G

vertex cover of size $n-k$ in $\overline{G}$



if S is a clique in G

then $V-S$ in $\overline{G}$ is a vertex cover

2-approximation of min vertex cover