L19. Binary Search Trees

Friday, 24 September 2021 10:49 AM

Ordered maps : ie a map whose elements are drawn from a totally ordered set.

A Ordered Map → Ground set is $2^{(X_K \times X_V)}$ (value)

where X_A is the totally ordered ground set of adt A.

- Insert, Delete, Find

Simple way of implementing an ordered map

DS : Sorted array

eg $\{3, 17, 2, 64, 87, 13\}$

store it as

2	3	13	17	64	87

Find 88 No (87, ∞)

Find ? query 22 x → No (17, 64) Find 1 No (-∞, 2)

In the ordered Find(x) returns either Yes (with value)

or No (x_-, x_+) where

x_- is the largest number $\leq x$ which is in S

x_+ is the smallest number $> x$ which is in S

Time complexity of Find : $\Theta(\log n)$

Claim: $\Theta(\log n)$ is "optimal" as far as find time is concerned

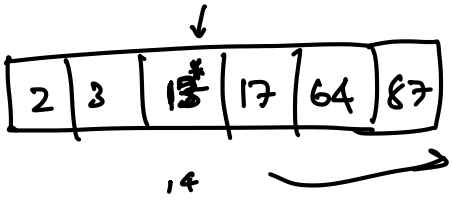
ie we can't do better.

- If I assign $< \log n$ bits to an id then some two elements in a set of n elements will get the same id. ∴ Each element is described by an id of length at ~~most~~ least $\log n$
- To find an element is equivalent to computing the $\log n$ bits of its id.
- Computing each bit will take $\Theta(1)$ time.

∴ $\Omega(\log n)$

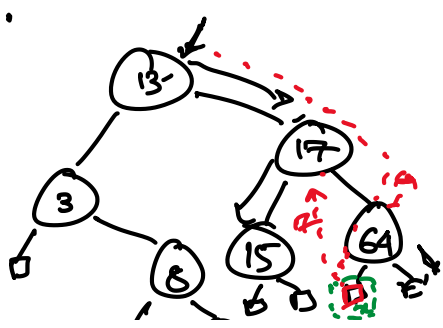
Caveat: This argument is NOT rigorous. c.f. Information Theory.

The issue with the sorted array: harvest can be $\Theta(n)$



Solve this problem using trees

Def: An integer tree is called a binary search tree (BST) if the keys in the left subtree of node u are $\leq$ the key at u and the keys in the right subtree of node u are $>$ than the key at u for every node u of the tree.



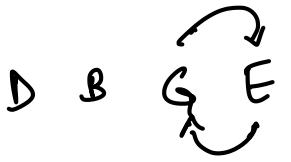
Also we use dummy nodes to make this a proper tree [Goodrich&T]

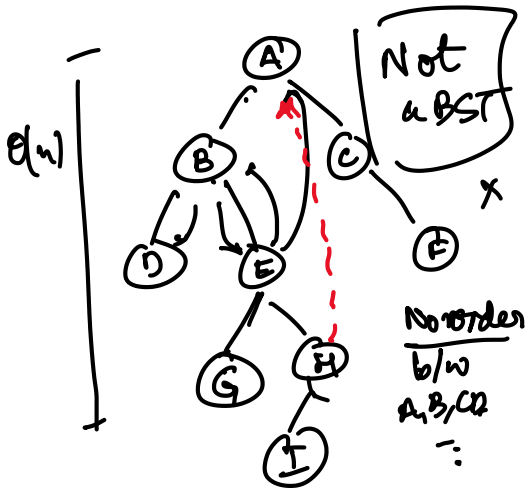
Claim1: The inorder traversal of a BST is a sorted order of the keys in the BST.

Pf: BST: inorder(L) $\leq$ k $\leq$ inorder(R) — inorder(L) is sorted, inorder(R) is sorted, k in between ⇒ sorted

○

Def: Given a node u in a rooted (binary) tree we define its inorder predecessor (resp. successor) as a node v that appears right before it (resp. after it) in an inorder traversal of the tree.





Claim2: If k is the key in the inorder predecessor (resp successor) of a node containing k' in a BST then there are no keys between k and k' in the BST.

Proof: It follows from Claim 1.

Q: Find (x) ?

- If the tree is empty say No. ($x_{\neq}, x_{\neq}$?)

Else

- Start from the root.
  - If key(root) = x then say Yes!

    Else if key(root) $\geq x$

    then recurse on left subtree

    else recurse on rt subtree.

To find (x_-, x_+) change base case as follows

- If root is a dummy then
  return No (inorder pred(root), ✓
  inorder succ(root))

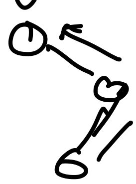
Why is this correct

1. If I insert x at the dummy then the new tree is a BST (by the correctness of find)
2. By Claim 2, nothing lies b/w x and IP(x) and nothing lies b/w x and IS(x) in the extended tree. ∴ nothing lies b/w IP(x) and IS(x) in the original tree.
3. By correctness of find x lies b/w IP(x) and IS(x). ∎

One of the IP / IS of the dummy is easy: its parent.

Finding the other:

- If we are looking for IP → keep going up using left child links till you can go up with a right then stop



||ly for IS

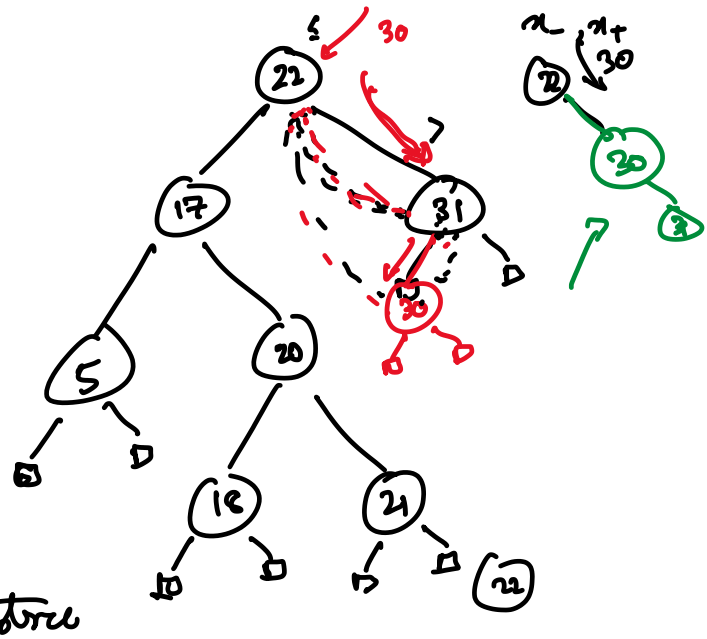
- may for is

Why is this correct ? Watch video.

L20. BSTs continued

Tuesday, 28 September 2021 10:59 AM

30

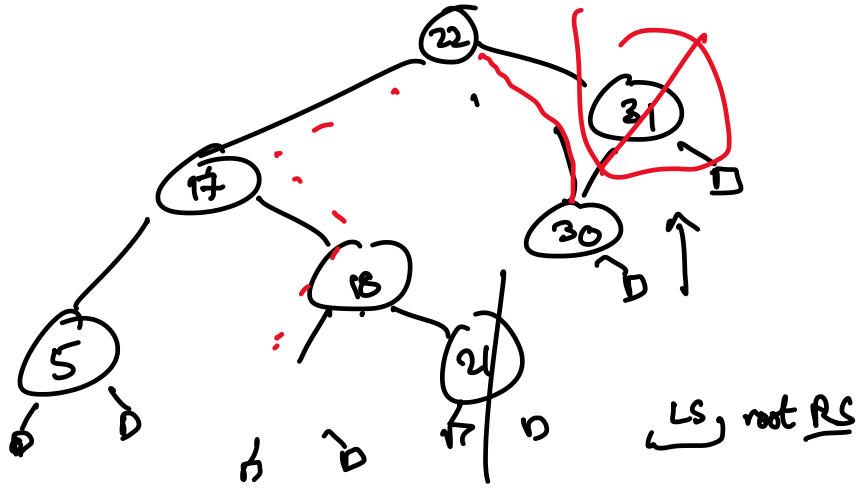


Insertion: (x)

- Start from the root
- At each non-dummy node compare x to to the key k in the node
  - If $x \leq k$ then recurse into left subtree
  - If $x > k$ then recurse into right subtree
- Create a node with x in it and attach it where the dummy was (with 2 dummies of its own)

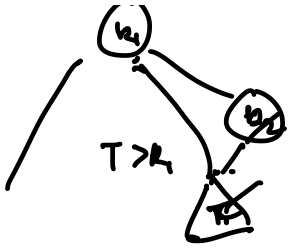
Time complexity: $\Theta(h)$

Deletion: (x)

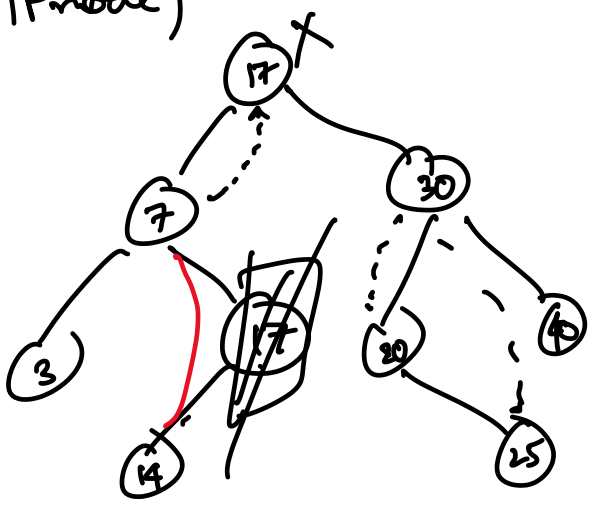


- First find (if not "Error")
- If x is a leaf delete it.
- If x is not a leaf but has only one subtree, delete x and attach its subtree to parent(x)

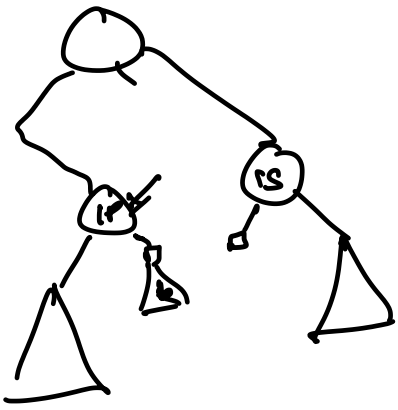
✓ [Verify that the BST property is maintained]



- If x is not a leaf and has 2 subtrees
  - Discard x
  - Copy the key from IS node (or IP node) into x's node
  - Delete the IS (or IP).



Note
- A node with 2 subtrees is an ancestor of both IP and IS.
- The IP/IS of such a node can have at most one subtree. Only left for IP and only right for IS

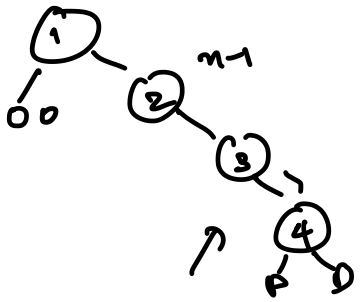


∴ the algorithm is not recursive, it only looks like it is.

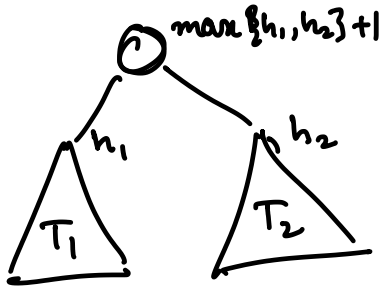
Time : $\Theta(h)$

Problem is everything is $\Theta(h)$ in time and h is $\Theta(n)$

eg. if I insert 1, ... n in sorted order, tree height is exactly n.



How to lower the height of a tree

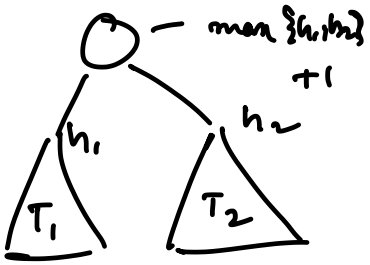


Claim: By "equalising" the heights of the root's subtrees we can reduce the height of the tree.

Candidate for "equalization".

invariant A: At each node the height of the left subtree and the ht of the rt subtree differ by at most 1.

$|h_1 - h_2| \leq 1$

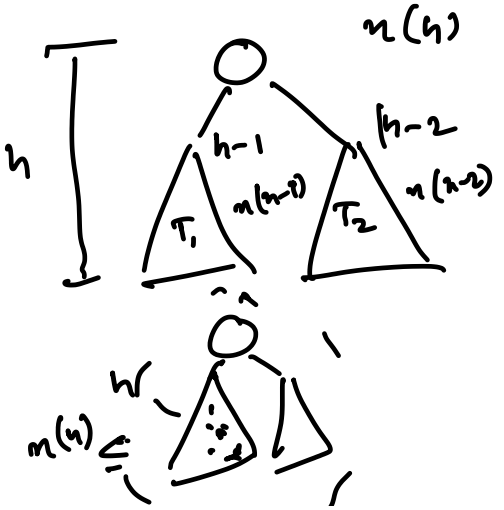


Claim: A tree with n nodes that satisfies invariant A has height $O(\log n)$.

Def: Let $n(h)$ be the minimum no of nodes in a tree of ht h (which satisfies Inv A)

Claim 2 $n(h) < n(h+1)$ $\forall h > 0$

Proof: Any tree of ht $h+1$ will (strictly) contain a tree of ht h.



$$n(h) = n(h-1) + n(h-2) + 1 \rightarrow$$

$$n(n) \geq 2\, n(h-2) + 1 \rightarrow$$

$$\Rightarrow n(h) \geq 2^{h/2} \quad \text{for any tree of ht } h$$

$$\Rightarrow n \geq 2^{h/2}$$

$$\Rightarrow \log n \geq \frac{h}{2}$$

$$\Rightarrow h \leq 2 \log n$$

Other way is $n(h) = \phi^h - \hat{\phi}^h$ where ϕ is $\left(\frac{1+\sqrt{5}}{2}\right)^h - \left(\frac{1-\sqrt{5}}{2}\right)^h$

$$\left(\frac{1+\sqrt{5}}{2}\right)^{\log n}$$

$$h = \Theta(\log n)$$

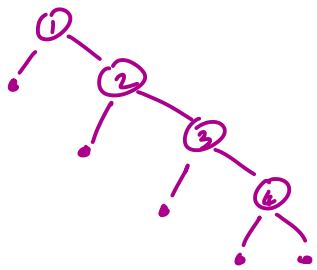
Inv A was first given in 1956 by Adelson Velskii and Landis. so such trees are called AVL trees.

□

L21. AVL Trees - Part 1

29 September 2021 10:06

Recap :



BSTs

- Insert(x)
- Delete(x)
- Search(x)

} O(height of tree) in worst case

Idea : Equalize the heights of subtrees

↓

Invariant A : At each node v,

$|\,ht(\text{left subtree}(v)) - ht(\text{right subtree}(v))\,| \leq 1$

Qn : can we maintain Invariant A 'efficiently' after every delete / insert / search?

↓ Yes

Claim : IF a tree with n nodes satisfies Invariant A, then the tree has $ht \leq 2 \log n$

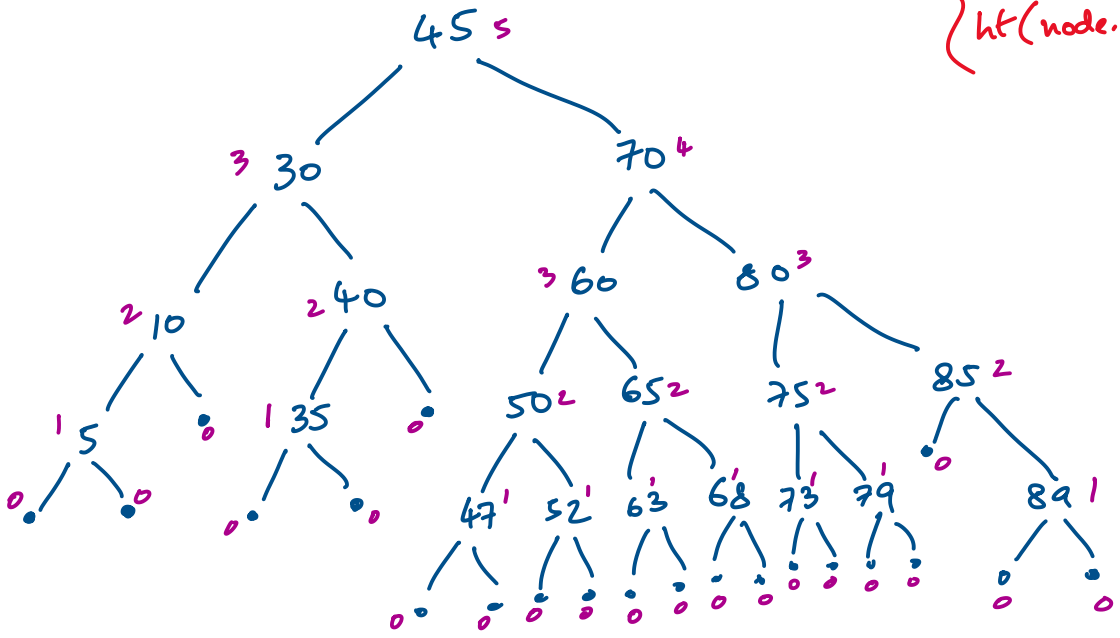
Defn AVL trees [Adelson-Velskii 57]

Def.: AVL trees (Landis)

A binary search tree is called an AVL tree if, at every node, the diff. in heights of left/right subtrees is at most 1.

ht(node) — 0 if dummy node; 1 + max { ht(node.left), ht(node.right) }

Example:



nodes in 30- subtree – 5

nodes in 70- subtree – 14

Discussion Qn for tomorrow:

T : AVL tree

T_1 : left child of root

T_2 : right child of root

How small can $\frac{|T_1|}{|T_2|}$ be?

Operations on AVL Trees

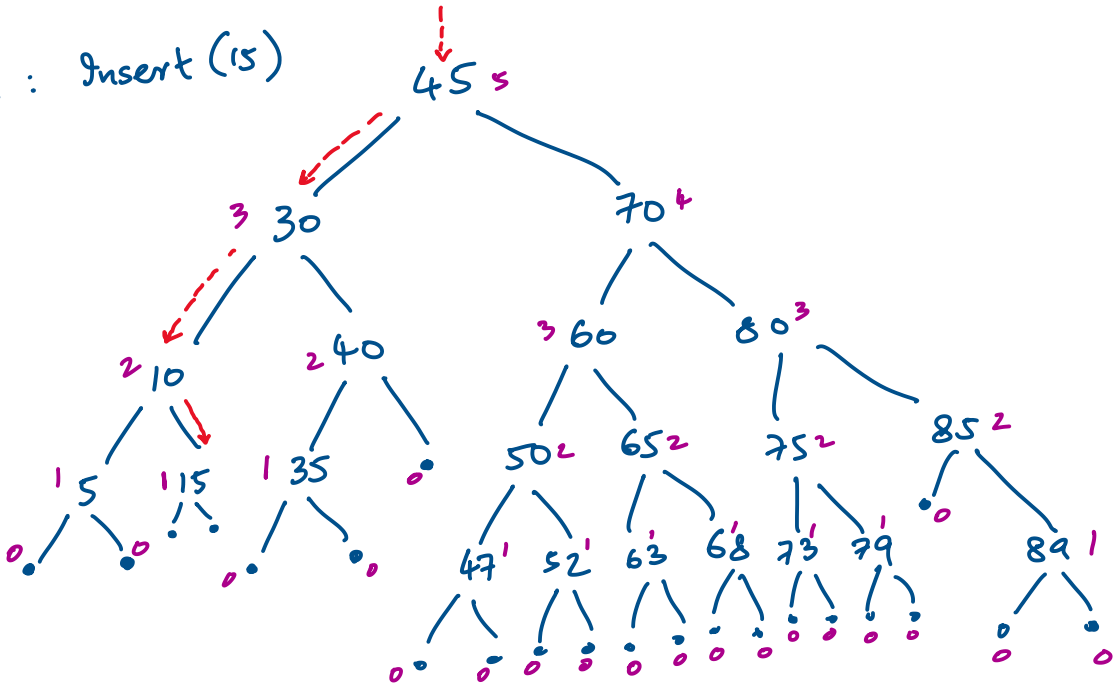
- Search (x) < Yes / No (x_-, x_+)

Doesn't change the tree,

⇒ if T is an AVL tree before search op. then it remains AVL tree after search.

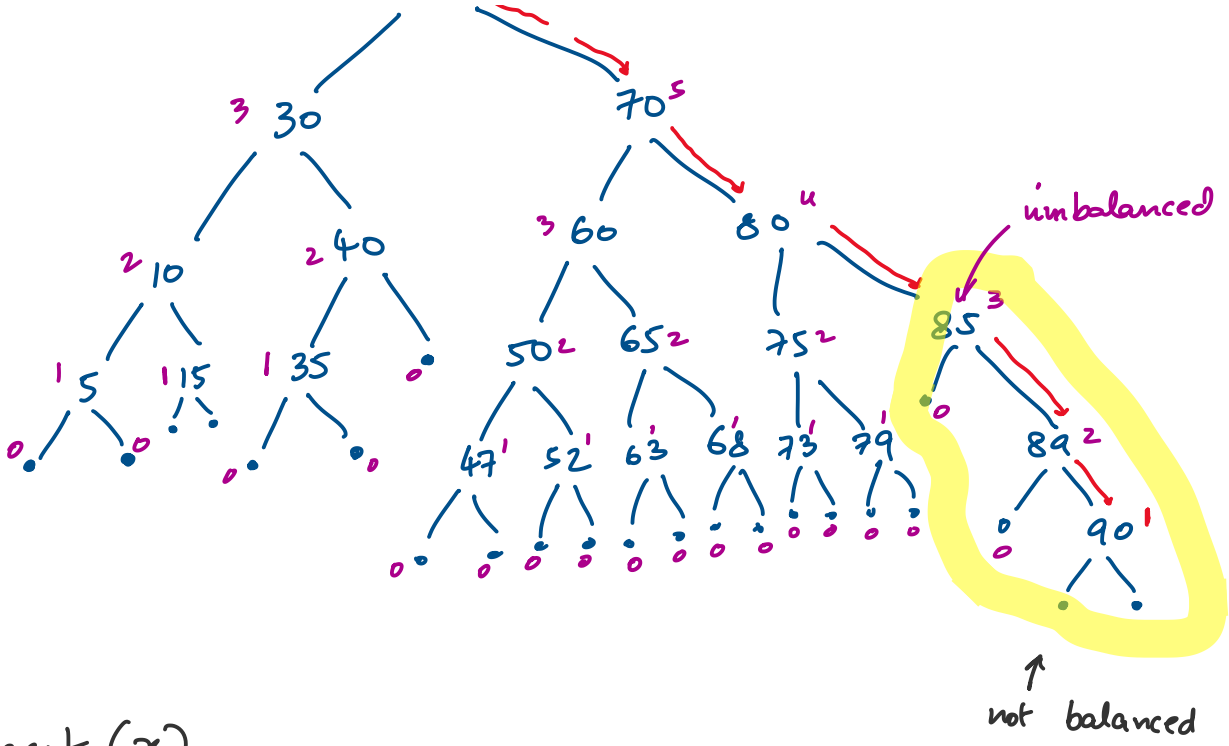
- Insert (x) :

Example : Insert (15)



Observation : on inserting x, only the heights of ancestors of x might change.

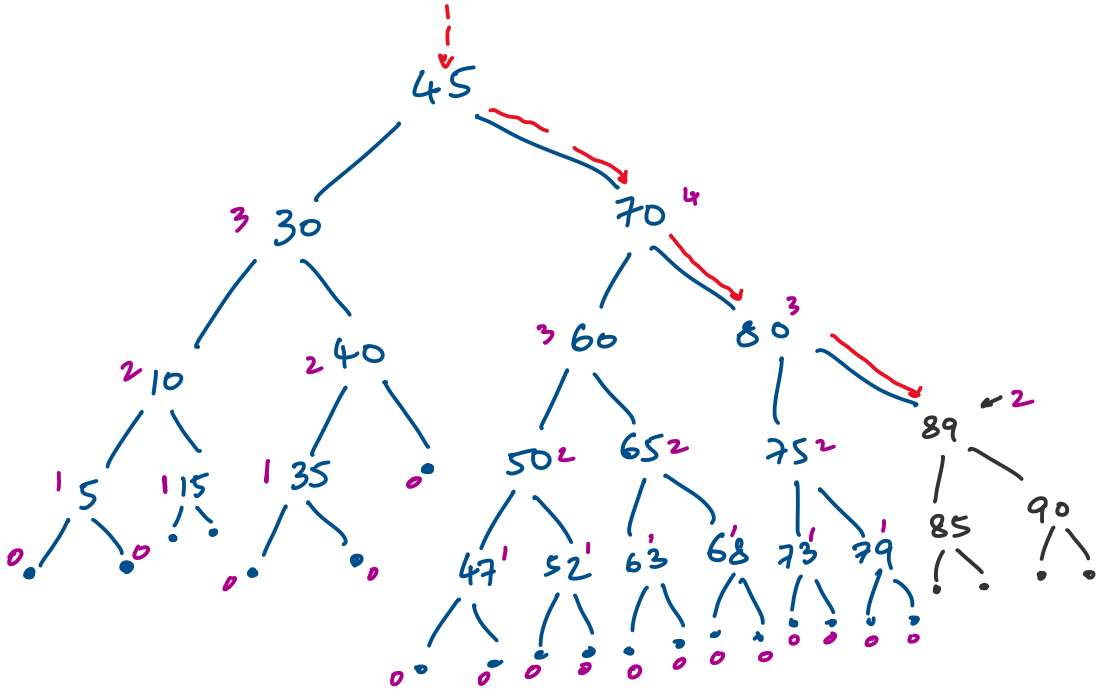
Insert (90) :



insert (x)

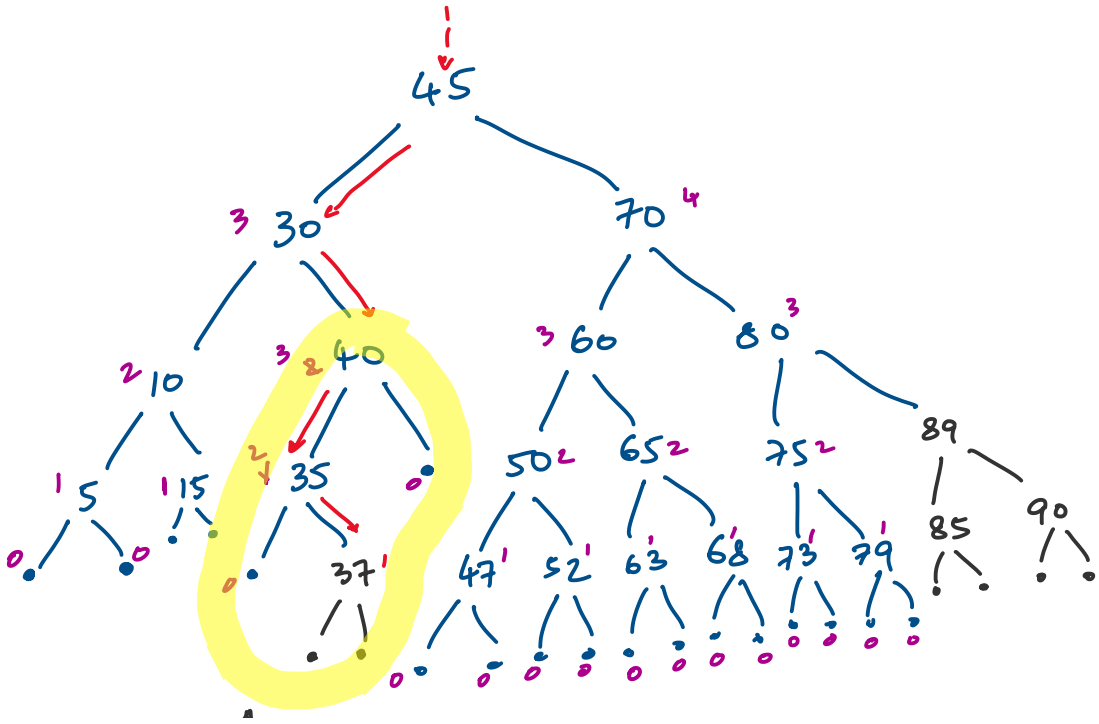
- First perform simple BST insertion
- Move up using parent pointer, update height attr. as required
  - if height of an ancestor doesn't change, done
  - if invariant A is violated, "restructure node". ------

RESTRUCTURING UNBAL. NODES:

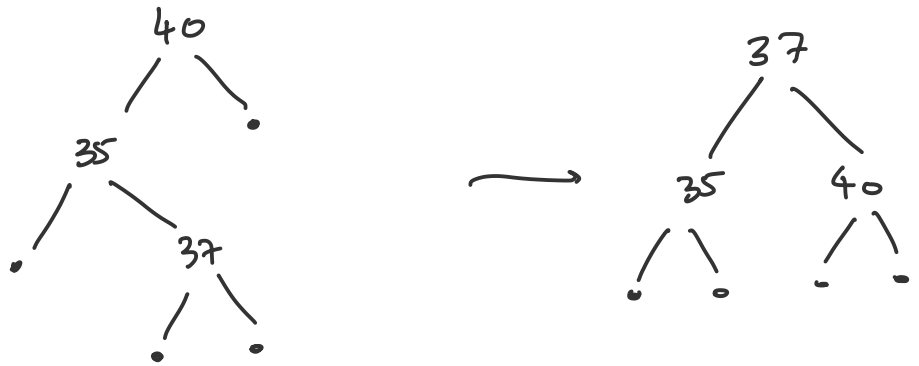


Restructuring Example 2:

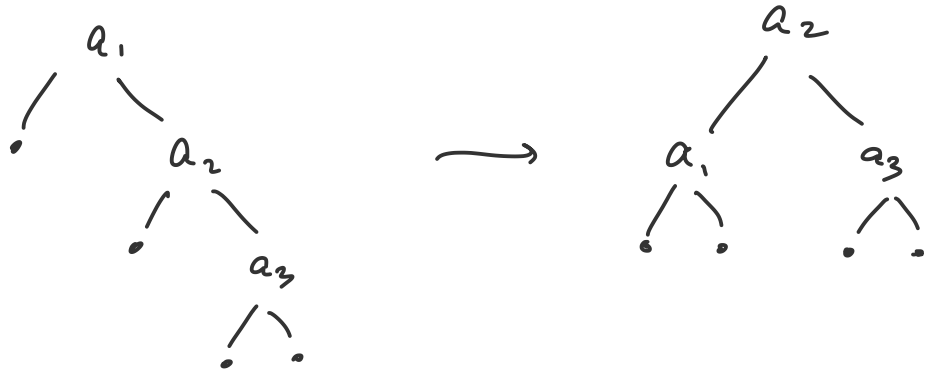
Insert (37):

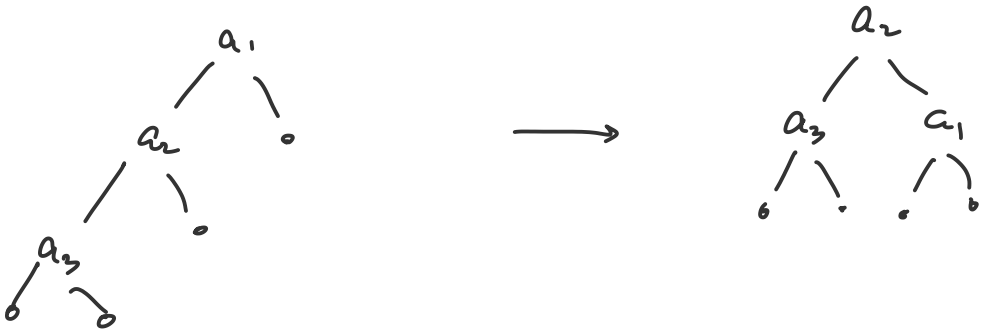


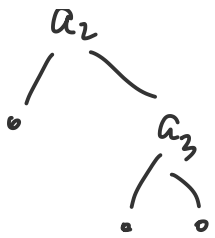
↑
restructure this subtree



After restructuring, put new subtree back in the original tree.







a_2

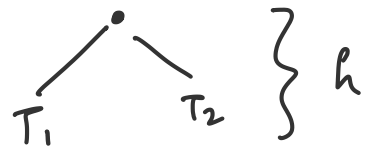
Qn : Is grandparent of inserted no

the are that's always restruc

L22. AVL Trees - Part 2

Thursday, September 30, 2021 9:38 PM

Discussion Qn from L21 :



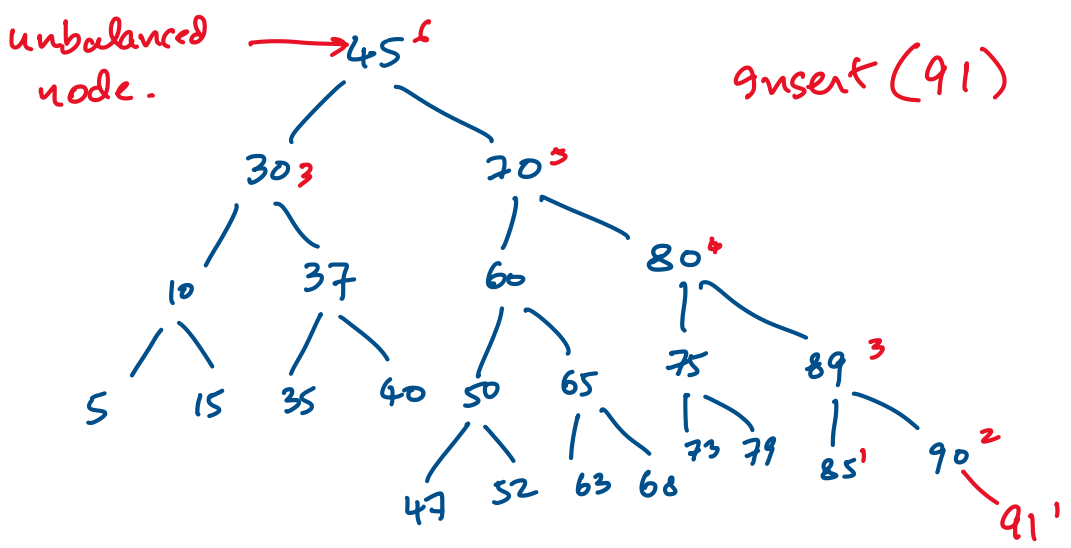
T: AVL tree

min. possible value of $\frac{|T_1|}{|T_2|}$

want: $ht(T_1) = h-2$

$ht(T_2) = h-1$

$\underbrace{\min |T_1|}_{\text{Fibonacci number}}$ and $\underbrace{\max |T_2|}_{2^{h-1}-1}$



Qn: When node x is inserted, is it possible that grandparent(x) is balanced, but some higher up ancestor isn't balanced?

Yes, possible.

Insert Algorithm [Informal]

1. Perform BST insertion.
2. Starting with inserted node, keep moving up (using parent ptr)

   2a. Update height of parent. If no need to update height, then we're done.

   2b. If node is unbalanced, restructure it

Plan :

→ – Restructuring : General case

– Insert alg.

2a. why are we done in 2a if no need to update height?

2b. what to do after restructuring? Done? Or should we check for unbal. ancestors?

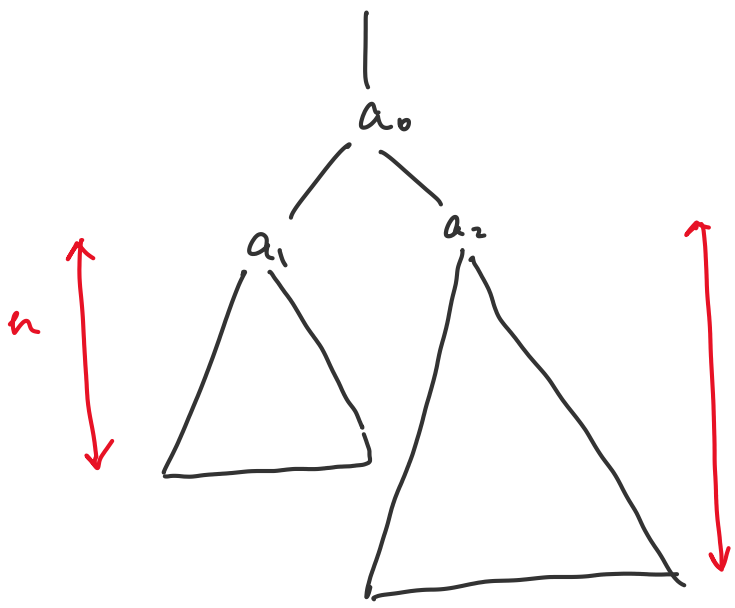
Running time of Insert

– Delete !

Restructuring : General case.

Suppose a_0 is unbalanced (bal. factor is ± 2)

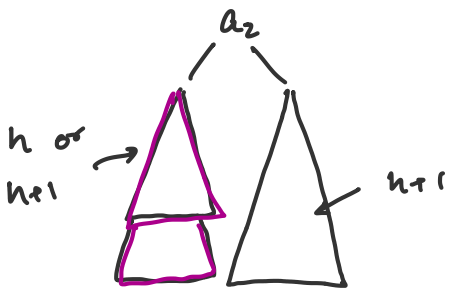
left and right subtrees of a_0 are balanced.



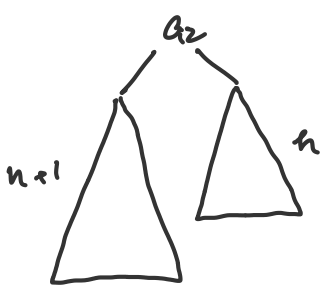
$ht(a_0) = h+3$
$ht(a_2) = h+2$
$ht(a_1) = h$

a_1, a_2 : AVL Trees

case 1 :

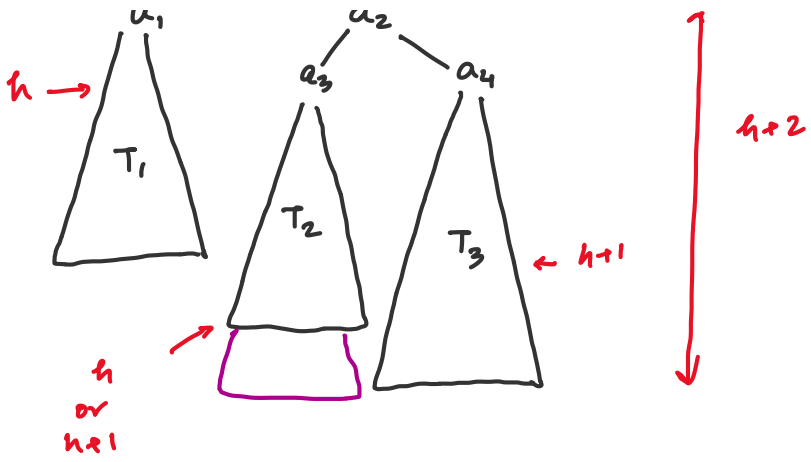


case 2 :

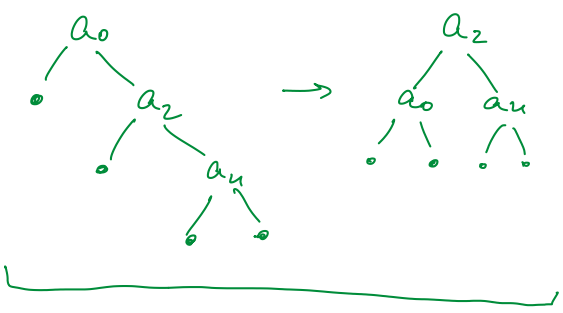


Case 1 :

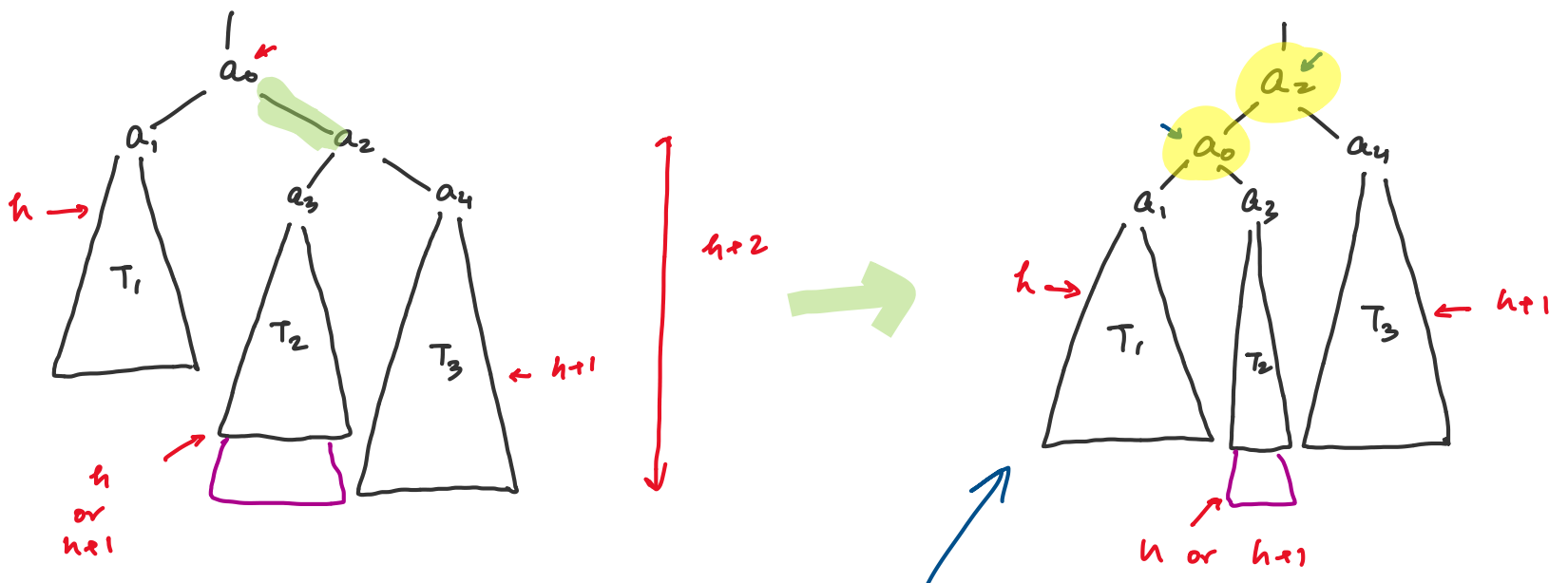
We have already seen a special case of case 1.



Restructuring Ex. 1



use Restructuring Ex. 1 to Restructure Case 1 [Single Rotation]



T : unbalanced

$ht(a_0) = 1 + ht(a_2)$

$= h+3$

T' is balanced.

$ht(a_0) = h+1$ or $h+2$

$ht(a_4) = h+1$

$ht(a_2) = h+2$ or $h+3$

$= 1 + \max\{ ht(a_0), ht(a_4) \}$

$= 1 + h+1$ or $1 + h+2$

$ht(a_2) = h+2$ or $h+3$

Obs: all nodes in T' are balanced.

→ check a_0, a_2 are bal.
use init. cond.: T_1, T_2 T_3 are balanced.

Claim 1: If T is a BST, then so is T'.

Proof sketch

→ $keys(T_1) \le a_0 < keys(T_2)$

use fact that T is a BST.

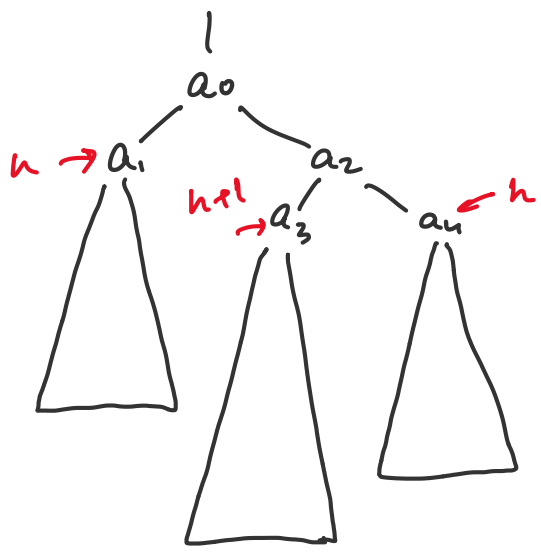
→ $keys(\text{tree rooted at } a_0) \le a_2 < keys(T_3)$

Restructure in $O(1)$ time

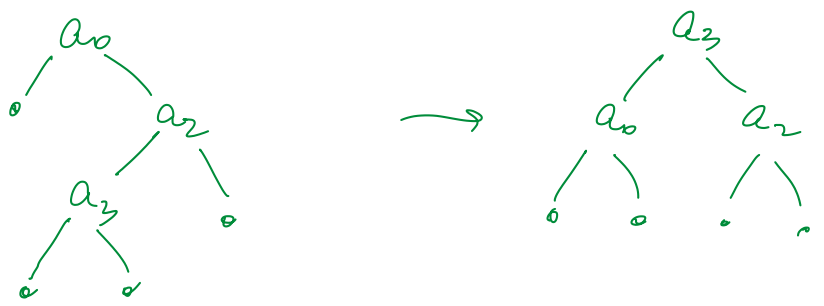
Claim: If $ht(T_2) < ht(T_3)$, then $ht(T') = ht(T) - 1$

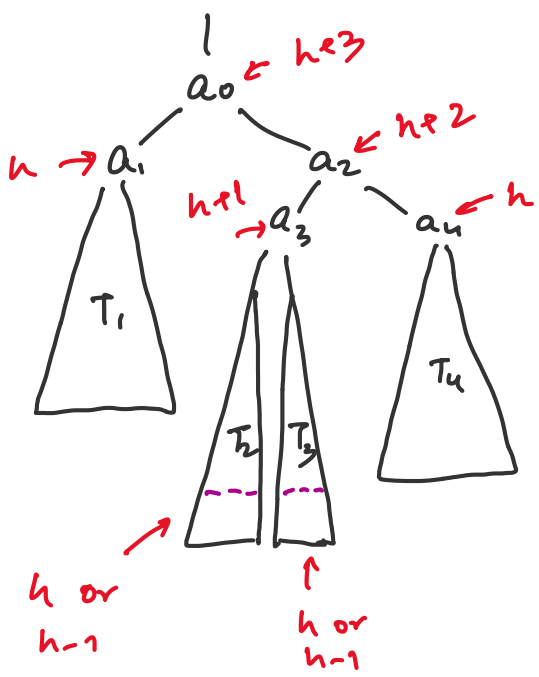
If $ht(T_2) = ht(T_3)$, then $ht(T') = ht(T)$.

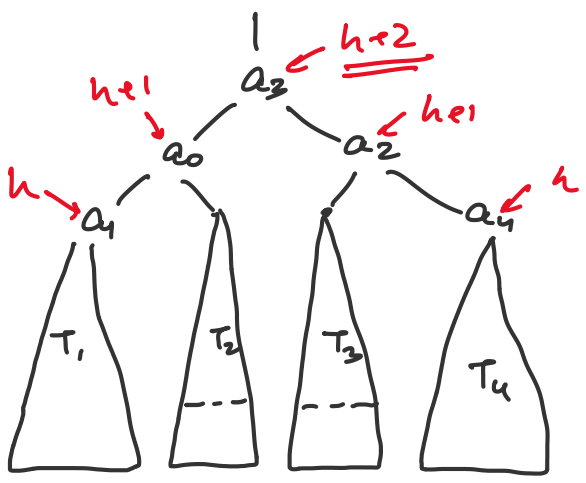
Case 2: [Double Rotation]



we've already seen spl. case of case 2: Restructuring Ex. 2







Claim 3: If T is BST, then so is T'.

Same as before

Claim 4: $ht(T') = ht(T) - 1$

Conclusion: Restructuring can be done with $O(1)$ operations

Discussion Qn: What if bal. factor ~~was~~ at

root was k?

↳ Suppose we want to merge 2 AVL trees. How to do that?

L23. AVL Trees - Part 3

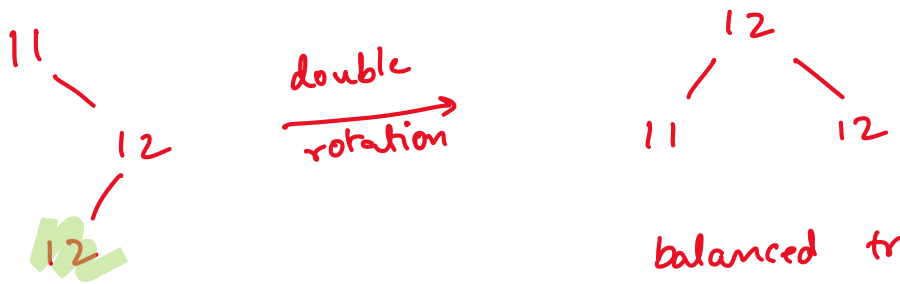
Monday, October 4, 2021 2:09 PM

More test cases for LM 4

Soumil's qn: from L22 qn BST,

left subtree ≤ root < right subtree. (*)

(*) may be violated due to double rot.



balanced tree
doesnt satisfy (*)

balanced BST

Qs if even possible to have ~~double rotation~~ in the presence of duplicates?

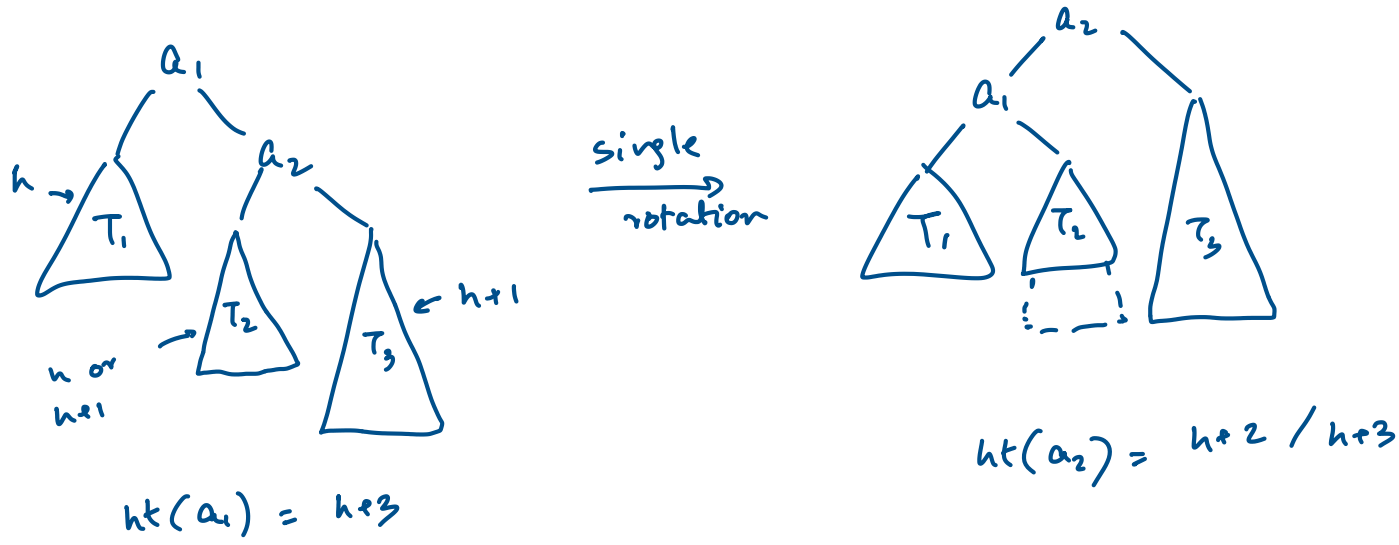
Not possible. Lots of copies of the same key k.

Qn: why should we have this strict requirement (*)? TBD next class

For today: left subtree ≤ root ≤ right subtree suffices for Find, Insert, Delete.

↳ Find(x): If there are multiple copies, just find one of them.

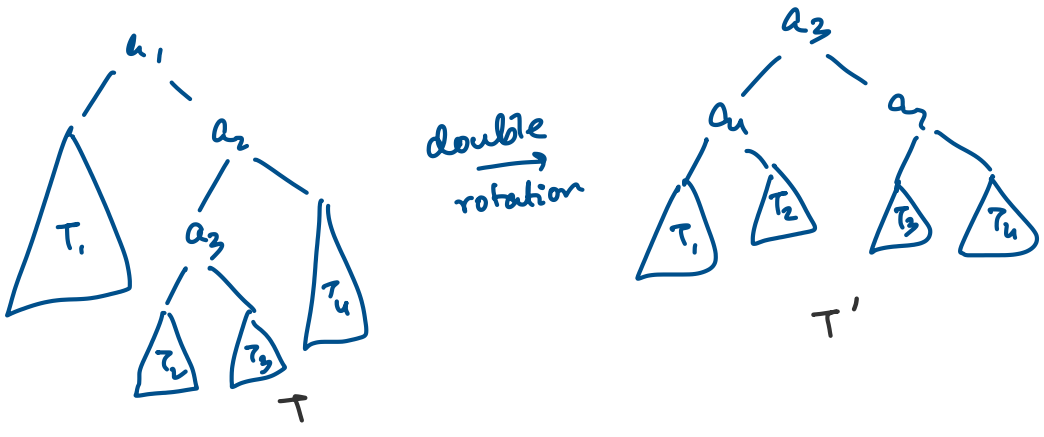
Recap of rotations:



. or $ht(T_3) = ht(T_2) - 1$ then

Claim : If ht(T2) = ht(T1) + 1

ht(new subtree) = ht(old subtree) − 1



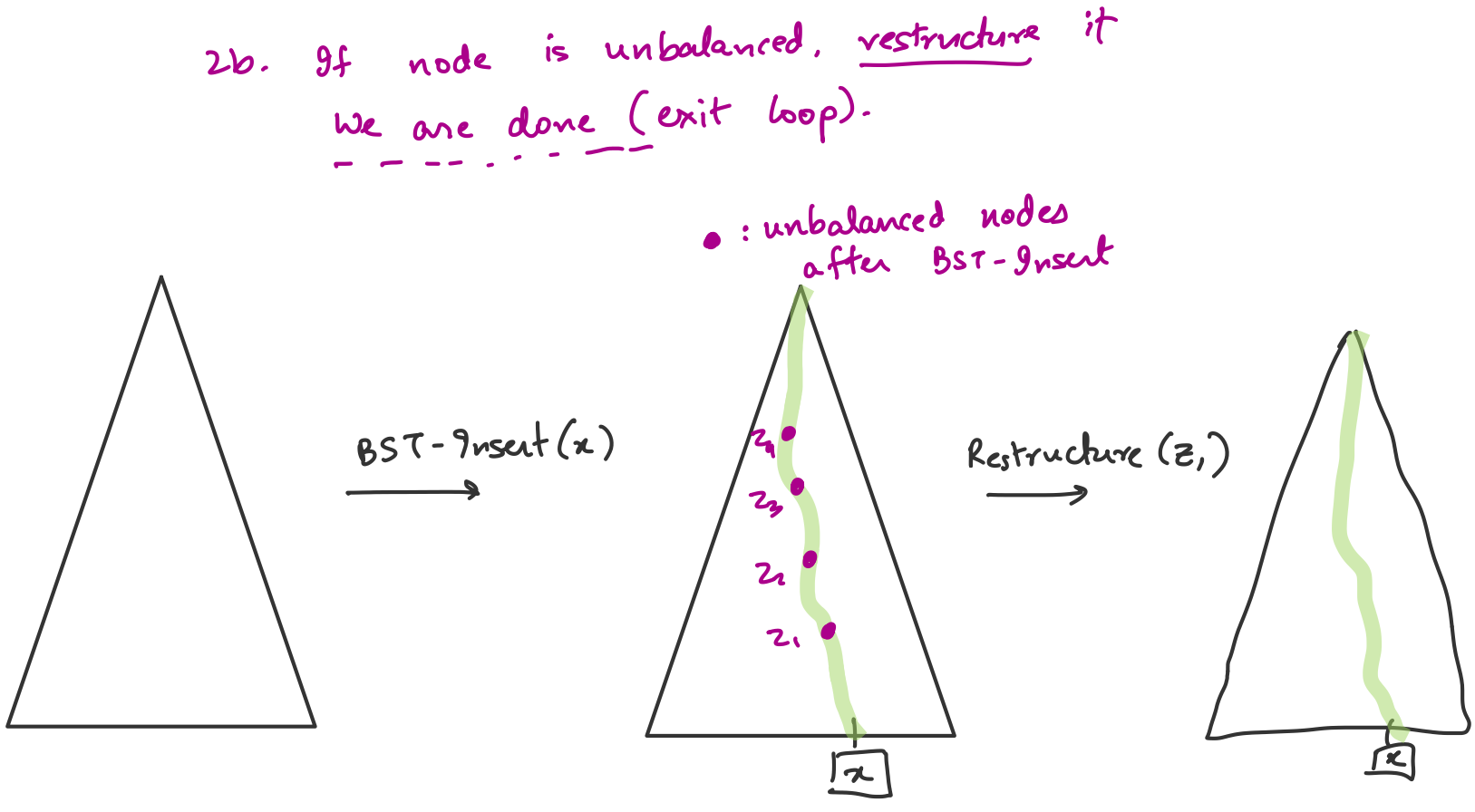
Claim: ht(T') = ht(T) − 1

Insert Algorithm [Informal]

1. Perform BST insertion. (BST-Insert(x))
2. Starting with inserted node, keep moving up (using parent ptr)

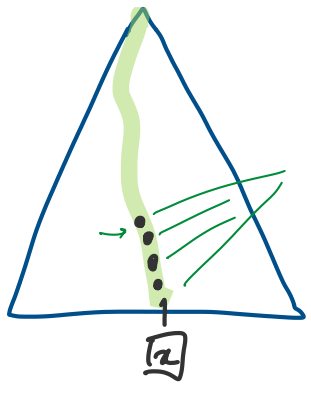
   2a. Update height of parent. If no need to update height, then we're done.

   2b. If node is unbalanced, restructure it we are done (exit loop).



- heights of ancestors of x may increase by 1
- heights of non-ancestors of x stays the same.

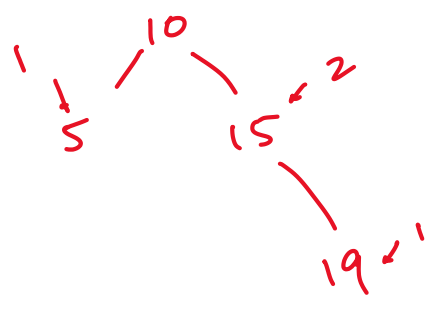
Q1: why are we done if no need to update ht?



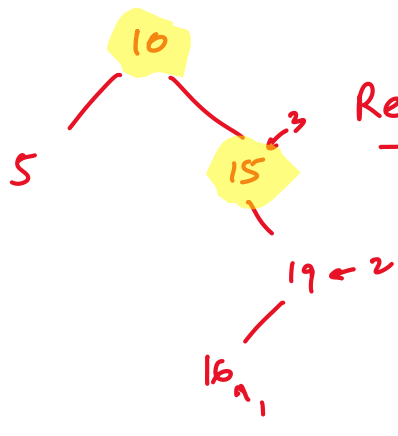
heights updated, but nodes still balanced.

Q2: why are we done after restructuring the lowest unbal. ancestor?

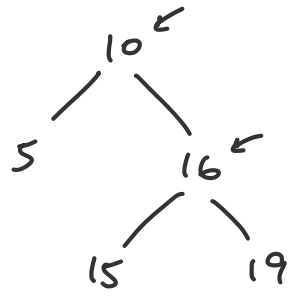
- Insert may result in multiple unbal. nodes.



BST Insert 16



Restructure(15)



Property of lowest unbal. node after insert:

T: lowest unbal. subtree

Claim 1: Insert(x):

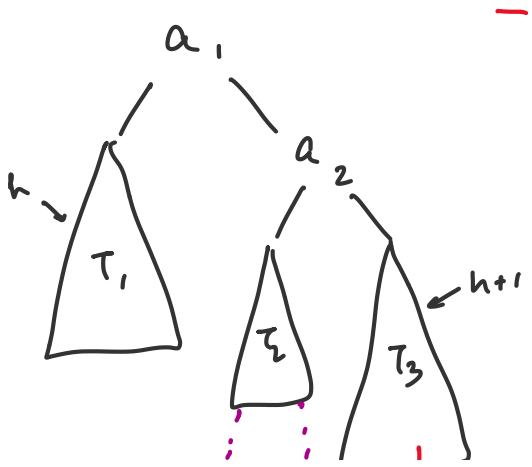
T': after restructuring T.

$ht(T') = ht(T) - 1$

Pf: After BST-Insert(x), case 1 or case 2.

If case 2, $ht(T') = ht(T) - 1$. –

Case 1:



Q: can x be in T_2?

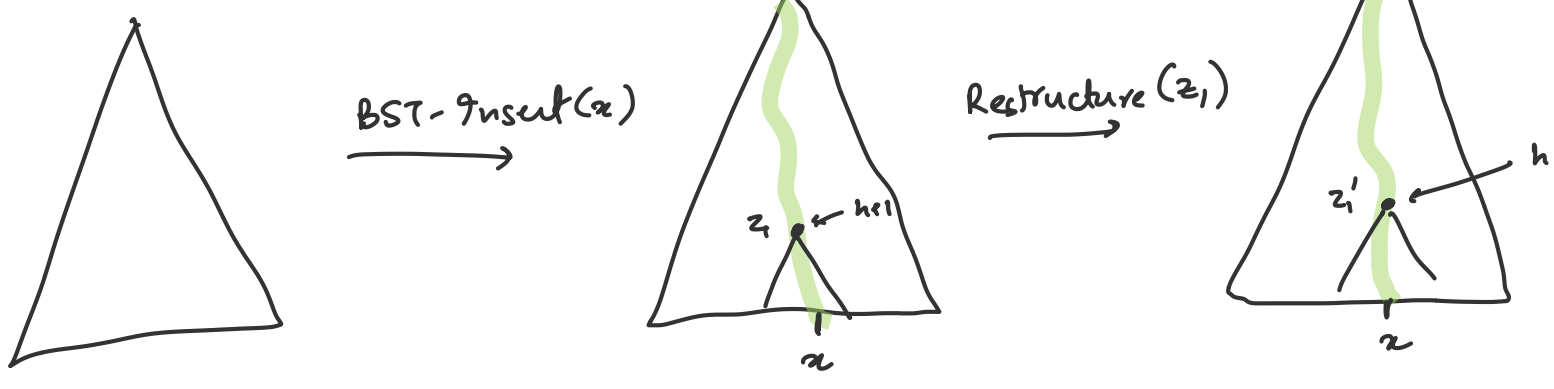
No. If this was the case, then imbal. before insert.

Q Is it possible that $ht(T_2) = h+1$?

Not possible. Because

this implies imbal. before insert

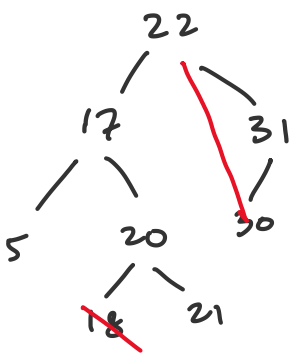
Claim 2: For insert, restructuring lowest unbal. node fixes imbal. of all its ancestors.



Obs: Consider any ancestor w of z_1. Before BST-Insert(x), and after Restructure(z_1), ht(w) is same.

Delete (x).

- Perform BST-deletion. (BST-Delete(x))
- start with deleted node, move up
  - update ht. of parent.
  - If node is unbal, restructure it. Move up and continue.



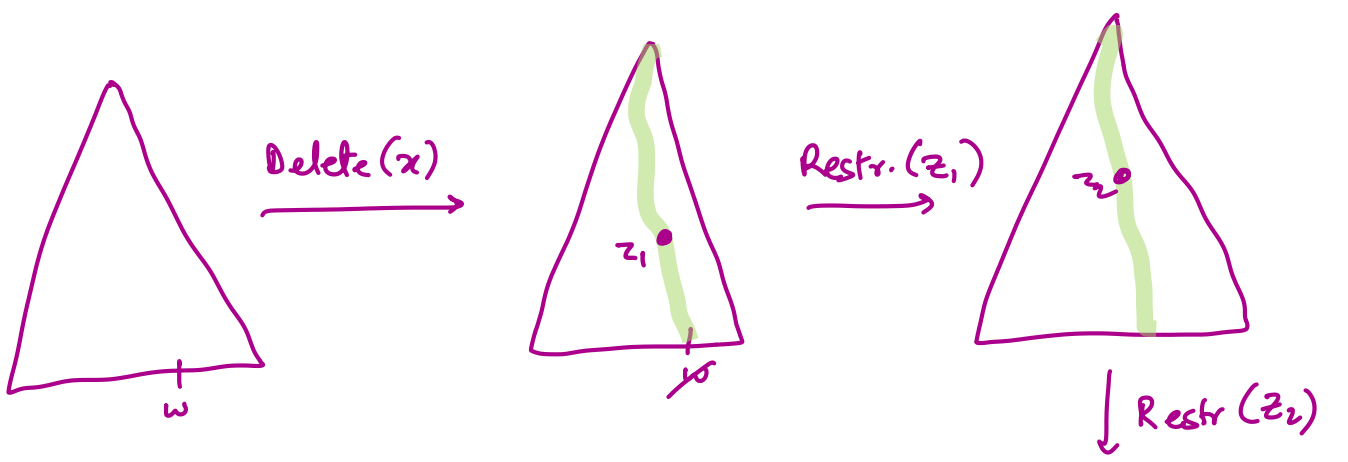
Delete(18) -

Delete(31)

Delete(22)

↳ delete 21

Replace 22 with 21



If root, then done.

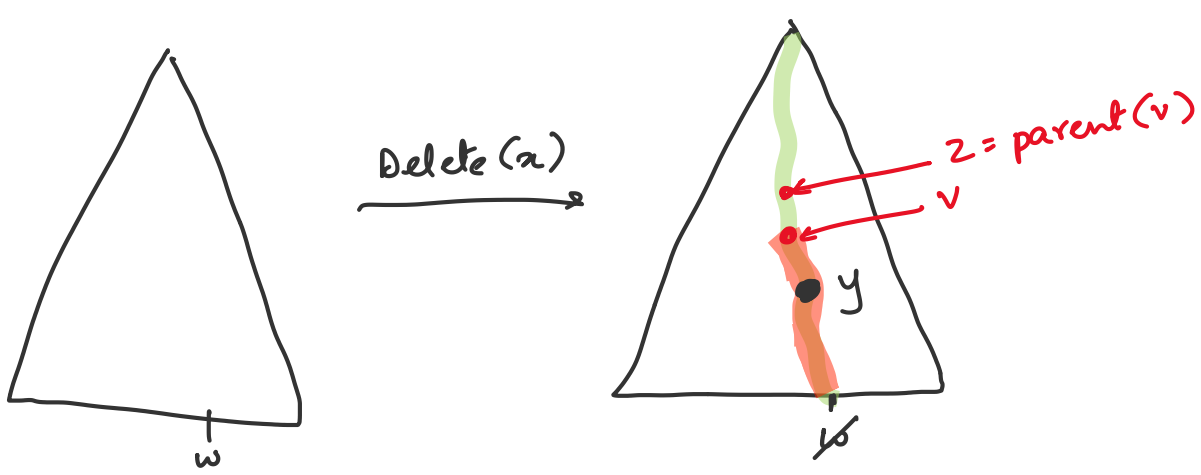
Insert

- many unbal. nodes after insert
- Restructure lowest one, fixes all above

Delete

- exactly one unbal. node after BST-delete
- Restructuring may result in ~~Unab~~ new unbal. nodes.

Q1: Why is there exactly one unbal. node after BST-Delete(x)?



hts of ancestors of w may decrease by 1

: ancestors whose hts decreased by 1.

Due to change in hts, some nodes may get unbal.
↑ y

Q: can y be above z?

No

Q: Can y be below v?

No. Think why?

⇒ unbalanced node has to be z.

Q2> After restructure(z), why are we not done?

↳ ht(restructured subtree) may decrease ← will decrease if in case 2.

↳ may result in new imbal.

L24. kd Trees

Thursday, October 7, 2021 3:32 AM

AVL Trees : Summary of Insert / Delete

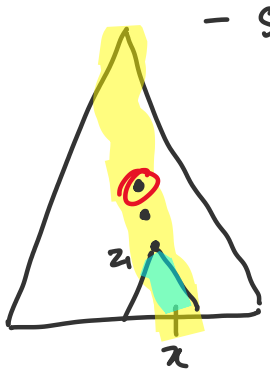
Two types of rotations:

Single rotation : O(1) operations : ht of restructured tree may decrease

Double rotation : O(1) operations : ht of restructured tree will decrease.

Insert (x)

- BST-Insert(x) — O(ht of tree) / O(log n)



- starting with inserted node, move up
  - If ∄ imbal. restr., exit loop.
  - update ht if needed

: nodes whose ht. increased by 1

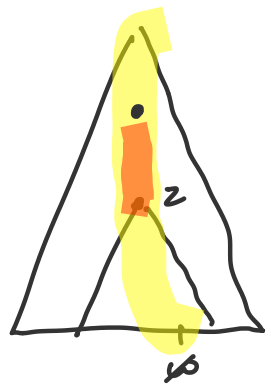
- O(log n) ops.
- ≤1 restr. needed.

Q1: what if restr. reqd. O(log n) time?

Q2: why not just restr. the topmost unbal. node?

Delete (x)

- BST-Delete(x) — O(log n)
-



nodes whose ht. dec. by 1

- Start with the deleted node, move up.
  - If ∄ imbal, restr. — O(1)
  - update ht if needed. — O(1)

O(log n) operation

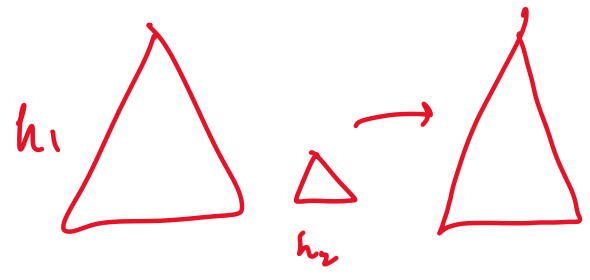
might need O(log n) restr.

Q1: If restr. requires O(log n) ops, then delete might require $O(\log^2 n)$ ops.

✓ Summary doc : AVL trees

Cool stuff

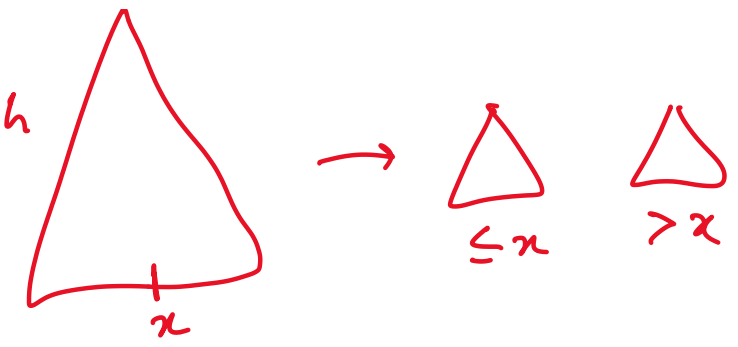
→ Merge AVL trees



→ Split AVL trees

↳ $O(h^2)$ time?

↳ $O(h)$

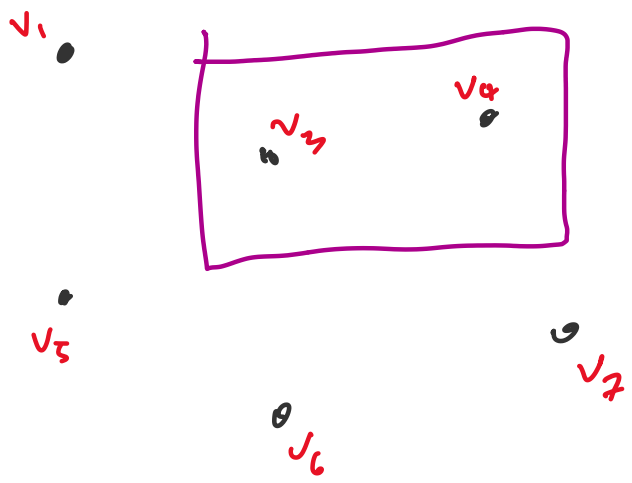


Range - search:

Geometric problem: Given n pts in a 2-d space. ($\mathcal{U}$)

want to construct a d.s.

using these n pts s.t.



→ Find All in Range

Range: specified by a rect.

Output all pts in $\mathcal{U}$

Brute Force: $O(n)$

→ Find Nearest Pt

Input: a pt in the sp.

Output: closest pt in $\mathcal{U}$

Real life applications:

- database queries

  Find All in Range query

- computer graphics

- classification

  Learning step: map data to 2-dim. pts.

  classification step.: figure out which is

the nearest d-dim pt.

---

1 Dimensional Version.

Given: n integers

build a d.s. on these n integers

s.t.

Find All in Range queries can be answered efficiently.

range = (left, right)

basic alg:

Preprocessing - Store $a_1, \ldots a_n$ in sorted order.

- for each (l, r) range query:

$O(\log n)$ - find location i of first pt in array $\geq l$.

$O(\log n)$ - find location j of last pt in array $\leq r$

$O(k)$ - print/output $a_i, a_{i+1}, \ldots, a_j$

↑ number of output pts.

total: $O(\log n + \text{number of output pts})$

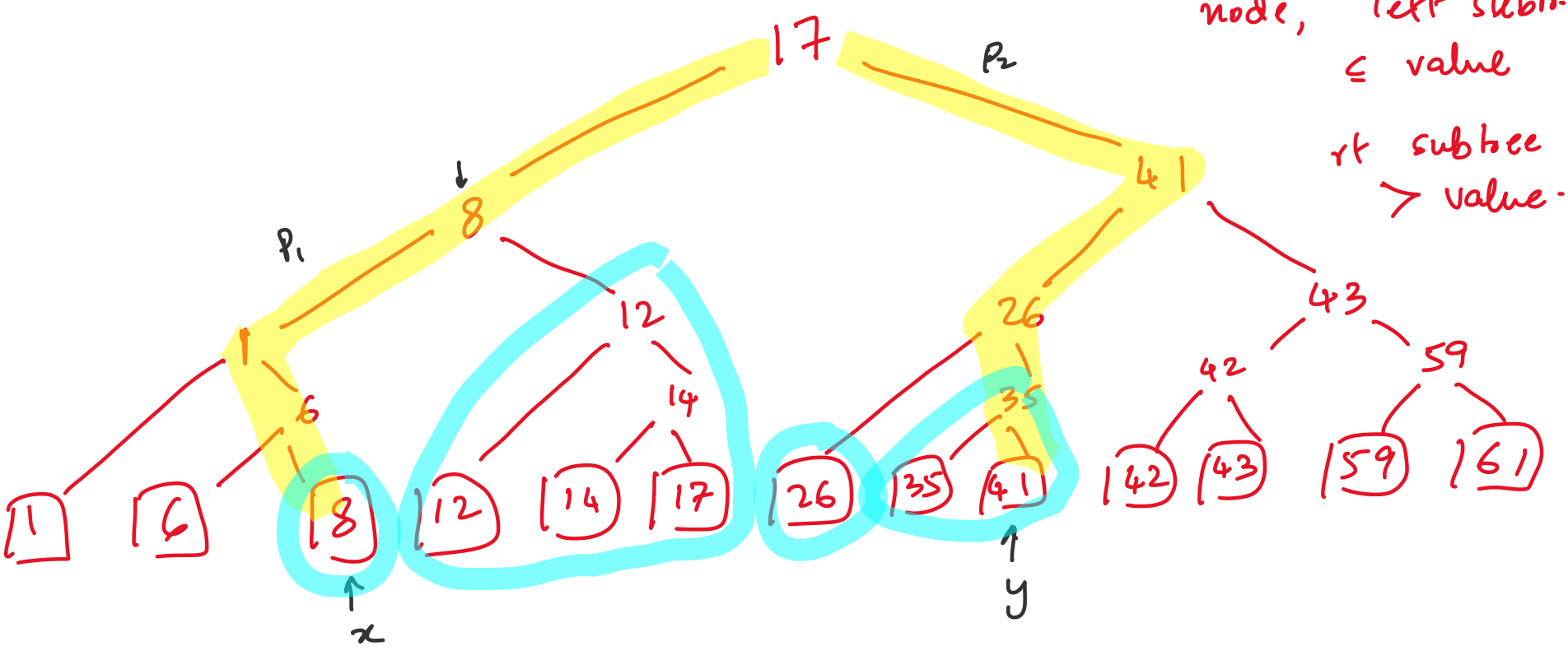
want: solution that extends to 2/3/4 dim.

Solution using BSTs :

- leaf nodes will store the n points.
- at every non-leaf node, the
  left subtree $\leq$ value at the node
  right subtree $>$ value at the node.

Assume all elements are distinct.

Example : (Goodrich's lecture notes)



at any non-leaf node, left subtr. $\leq$ value
rt subtree $>$ value.

range : $[7, 41]$

1. Find smallest leaf x s.t. $x \geq 7$.
   Path from x to root : P_1
2. Find largest leaf y s.t. $y \leq 41$.
   Path from y to root : P_2
3. Find lowest common ancestor (z) of x and y.

P_1 : [x ..... 7 (10 root]
P_2 : [y ..... 6 (10 root]

4. Collect all the subtrees in this range.

$S = \{x, y\}$.

4a - $\forall$ $w \in$ Path (x, z) $\leftarrow P_1$

if right child of $w \notin$ Path (x, z)

include right subtree(w) in S

4b - $\forall$ $w \in$ Path (y, z).

if left child of $w \notin$ Path (y, z)

include left subtree (w) in S.

5. $\forall$ subtrees $T \in S$, print all their leaves.

Q1: Running time of alg.

Q2: How to extend to 2d?

L25. kd Trees continued

11 October 2021 22:38

RECAP :

Range Search :

Given n points, build a data structure to support the following queries :

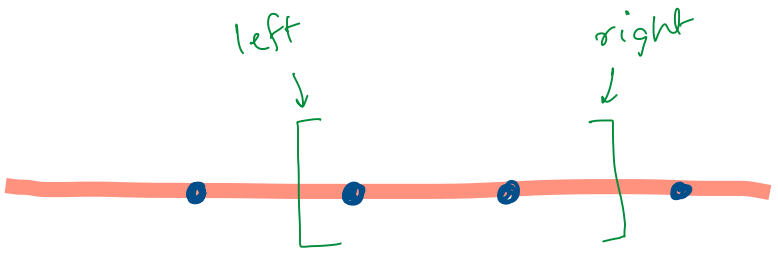
- Find All in Range (range) ↖
- Count All in Range (range) ↙

answer these queries efficiently

1 Dimensional setting :

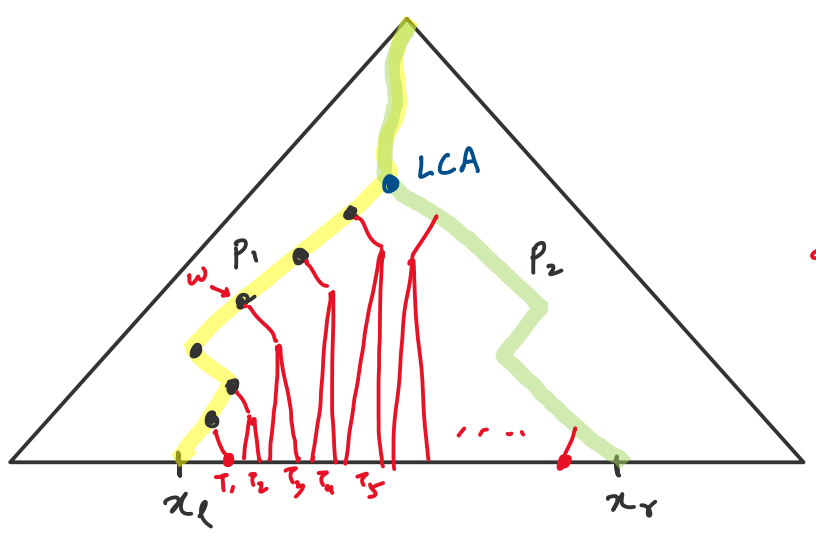
point ≡ integer

range ≡ (left, right)



Solution using BSTs :

- n leaf nodes, one for each point
- left subtree ≤ value at root < right subtree



ht of tree : $O(\log n)$

$S = \{x_l, x_r, T_1, \ldots T_5 T_6 \ldots T_{10}\}$

Find All in Range (left, right)

1 - x_l : smallest leaf value ≥ left ← O(ht of tree)

2 - x_r : largest leaf value ≤ right ↙

3 - Compute $z = LCA(x_l, x_r)$ ← O(ht of tree)

4 - $S = \{x_l, x_r\}$. ← O(ht. of LCA)

a. ∀ $w \in Path(x_l, z)$, if right child of $w \notin Path(x_l, z)$
add right-subtree(w) to S.

b. ∀ $w \in Path(x_r, z)$, if left child of $w \notin Path(x_r, z)$
add left-subtree(w) to S.

_____ : put a pointer to the root of the subtree

S. ∀ trees $T \in S$, output the leaves of T.

Qn: What if we only wanted the count?

- store the number of leaves in every subtree
- ∀ $T \in S$, add T.num_leaves attr

Running time:

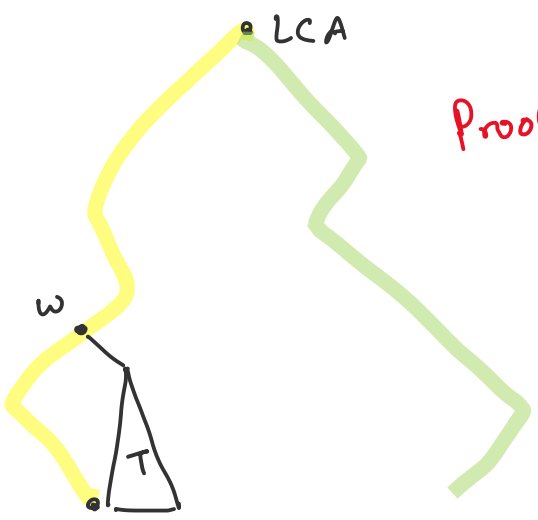
→ O(ht of tree) for steps 1-4 — O(log n) if there are no repetitions

- O(number of output pts) for Step 5 — repetitions: O(n). can have O(log n) if left subtree ≤ root ≤ right subtree. if very few rep. then ≤ ≈ <

Proof of correctness:

1 - all points output by alg. are in range [left, right]

2 - no pts. in this range were skipped.



Proof Outline for (1):

Take any tree $T \in S$. Suppose T was included in Step 4(a). Then $T \leq LCA < x_r$. Therefore, it suffice to show $x_l \leq T$. Since T was included in Step 4a, T is the right child of some node $w \in Path(x_l, LCA)$.

$\Rightarrow$ x_l is in left-subtree of w, and T is the right subtree of w. $\therefore$ $x_l \leq w < T$

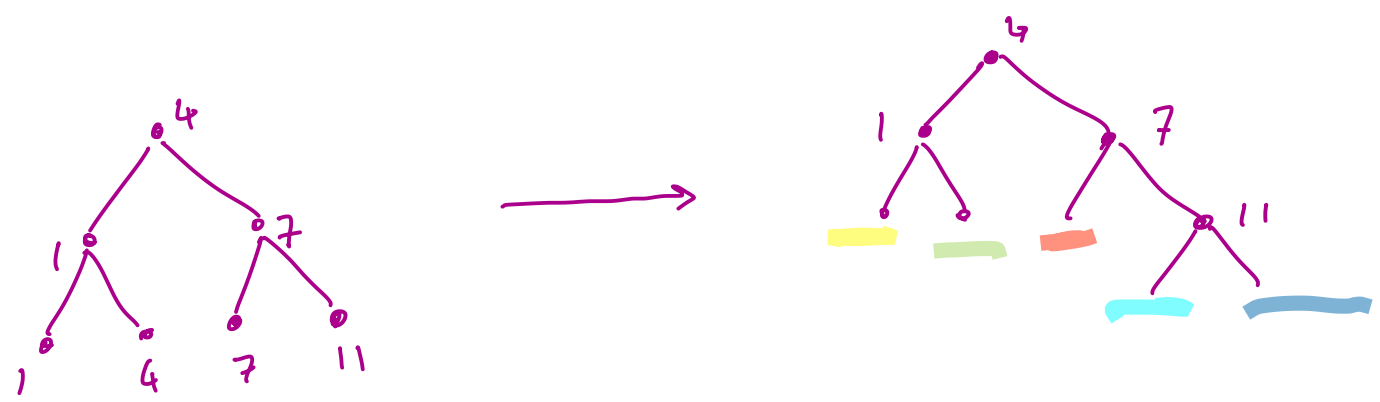
Proof outline of (2):

Consider any $y \in [x_l, x_r]$. Path from y to z will meet either P_1 or P_2. (why?) Suppose it meets P_1, and let z' be the lowest ancestor on path P_1. Then left child of z' is on P_1, and right child of z' is on Path(y, z) (why?).

$\Rightarrow$ right subtree of z' is in S, and therefore y is.

output by the algorithm.

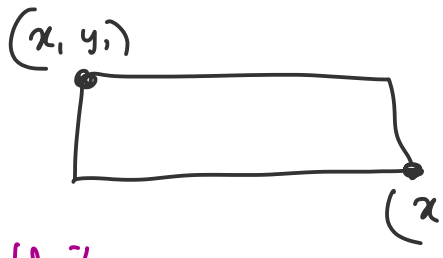
- collect / count all points in a range even in dynamic setting
- running time depends on the output size.



2D Trees :

points : (x, y) : $x, y \in \mathbb{Z}$

orth. range queries :



all range queries are rectangles. with sides parallel to X / Y axes.

Goal: Build a DS given $p_1, \ldots, p_n$ s.t. orth. range queries can be answered efficiently. ← Time to answer queries?

↓ time to build ?

Solution using BSTs [Jon Bentley, 75]

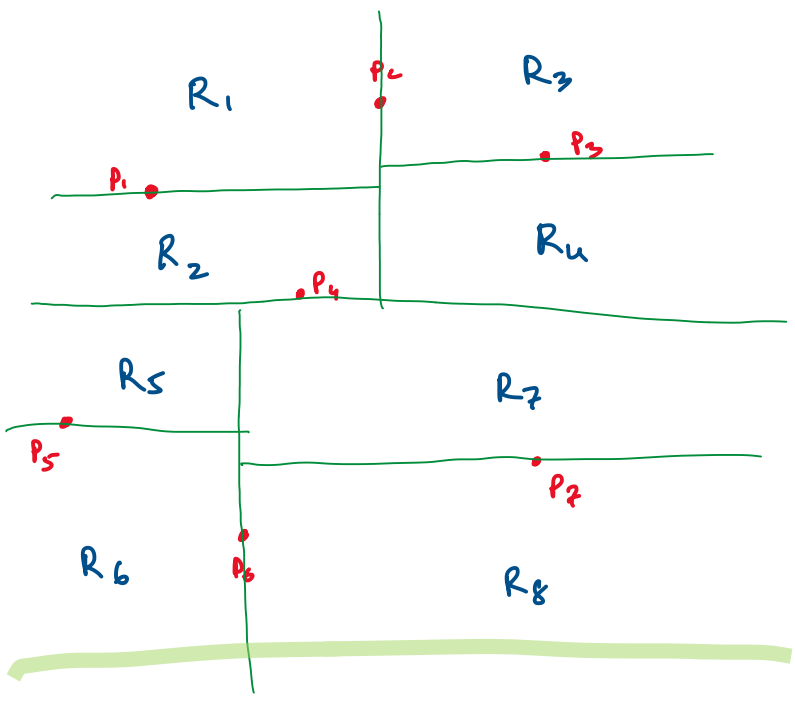
High level idea : use the n pts to partition sp. $\mathcal{C}$ BST is created to

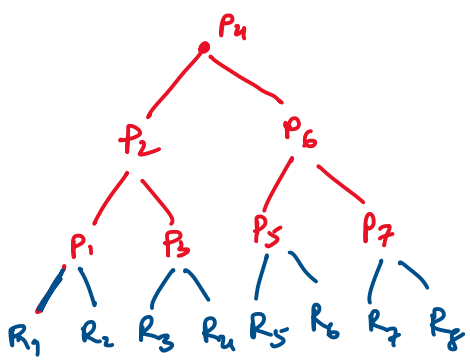
leaf nodes of BST correspond
one of the regions

- odd levels are labelled as X
  even levels labelled as Y

- if n is a node in X-level, then
  all leaves in left subtree are above n
  all leaves in rt. subtree are below n.

- if n is a node in Y-level, then
  all leaves in left subtree are to the left of n
  " " " right " " " rt. of n.





← level 1 (X level)
← level 2 (Y level)

How to construct this tree?

LM-6

Queries:

simplified range: line parallel to X/Y axis

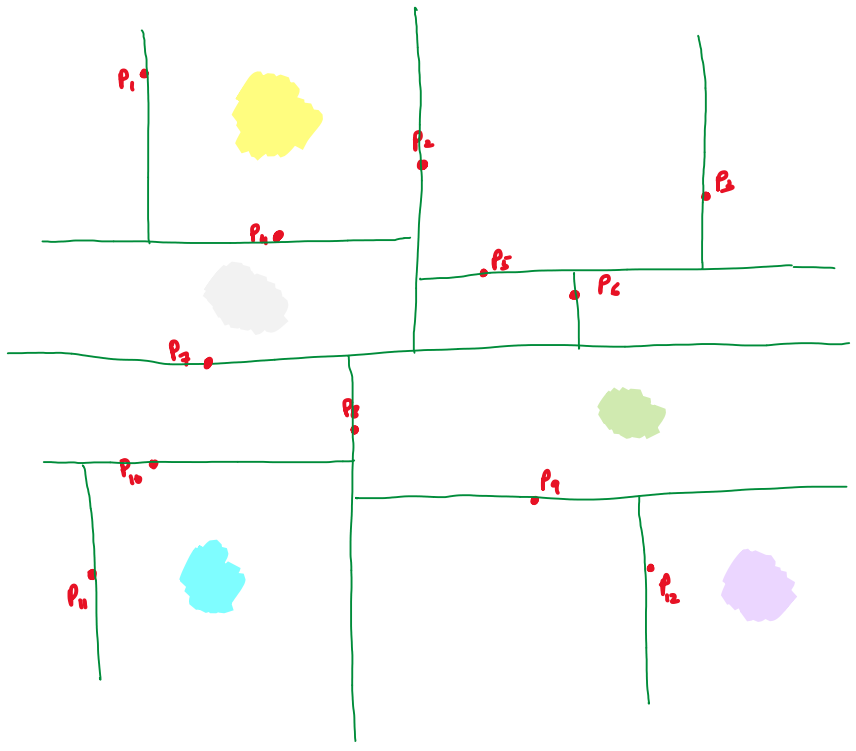
output all points above line: par. to X
left line: par. to Y.

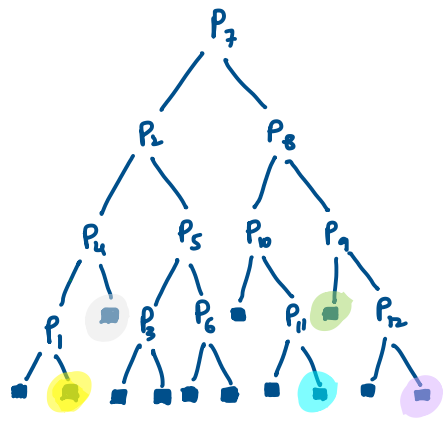
Example of 2D-tree with 12 points.

Internal nodes associated with points

~~Leaf~~ nodes associated with regions (marked using ■)

Ext.





Each ■ represents a region in 2D-space.

L26. Multiway Search Trees

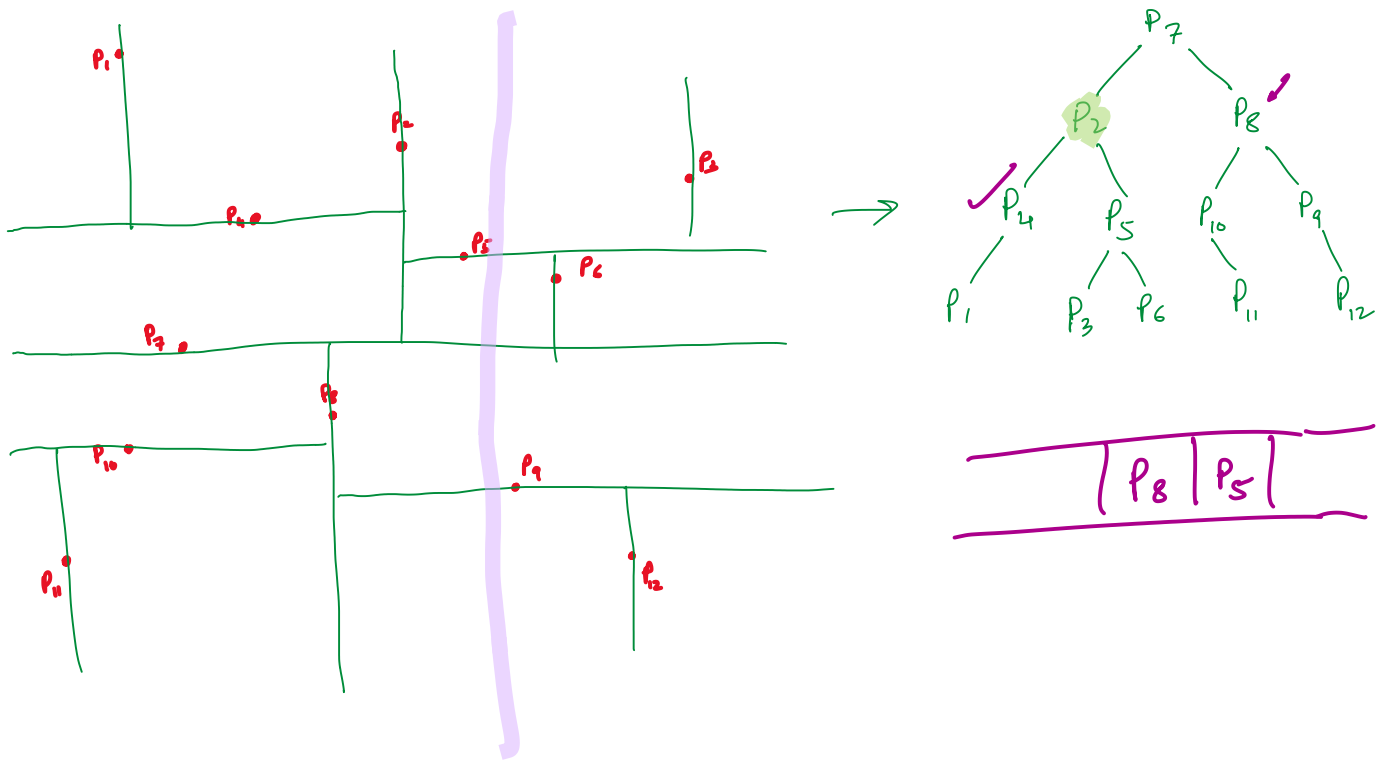
Thursday, October 14, 2021 10:46 AM

kd - Trees contd.

Input: set S of n points

Data structure: tree whose internal nodes correspond to points in S.

Queries: Rectangle / plane parallel to the axes



At odd level: cannot discard if vertical line
At even level: cannot discard if horizontal line

Claim: Suppose this k-d Tree is ~balanced.

$$T(n) = 2T(n/4) + O(1)$$

$2T(n/4)$: to process two of the subtrees at 2 levels below

$O(1)$: to include/discard the remaining two subtrees

$$= O(\sqrt{n})$$

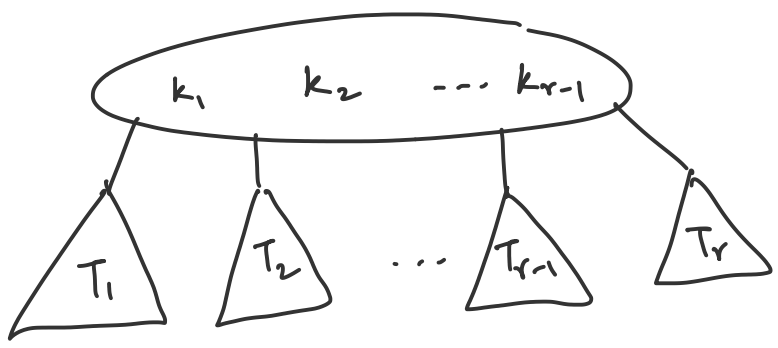
Qn: What should be the recursive relation for k dimensions?

Qn: What happens if we take all horizontal lines first, then all vertical partition lines?

MULTIWAY SEARCH TREES :

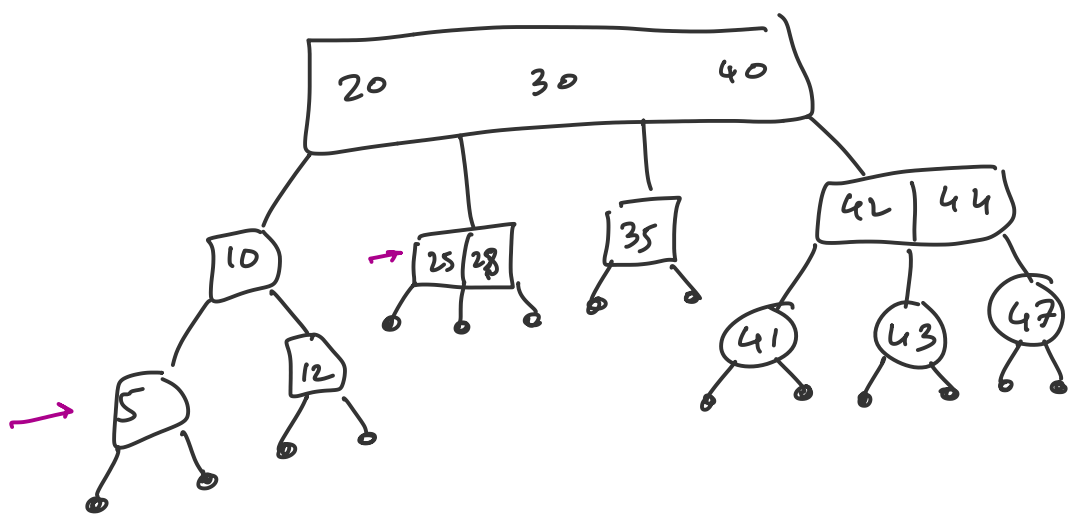
m - multiway search tree

- every node in this tree has
  $r-1$ keys and r subtrees $(r \leq m)$
-



$T_1 < k_1 < T_2 < k_2 \ldots < T_{r-1} < k_{r-1} < T_r$.

$m = 4$

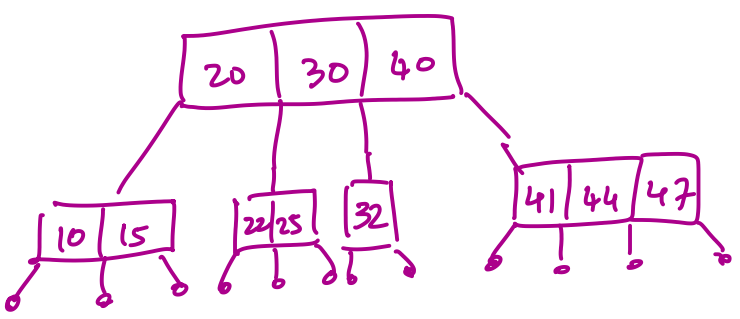


search(x) : $O(\log m \cdot ht)$

m. Multiway Search Trees $\supseteq$ mB - trees $\supseteq$ 2-4 trees

↗ • all leaf nodes are at the same level.

↗ • at least $m/2$ subtrees
at most m subtrees

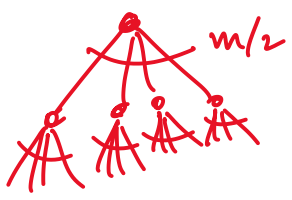


4 - B tree

why care about B-trees / 2-4 trees ?

- what can be the depth of an mB-tree ?

Obs: Depth of an mB-tree must be at least $\log_m(n)$ and can be at most $\log_{m/2}(n)$.



$$n \geq 1 + \frac{m}{2} + \left(\frac{m}{2}\right)^2 + \ldots + \left(\frac{m}{2}\right)^{d-1}$$

- Insert, Search, Delete: depend on m.

Read from disk: at least 64 bits.

but all our keys of size $\leq$ 16 bits.

$m = 4$ 2-4 Tree

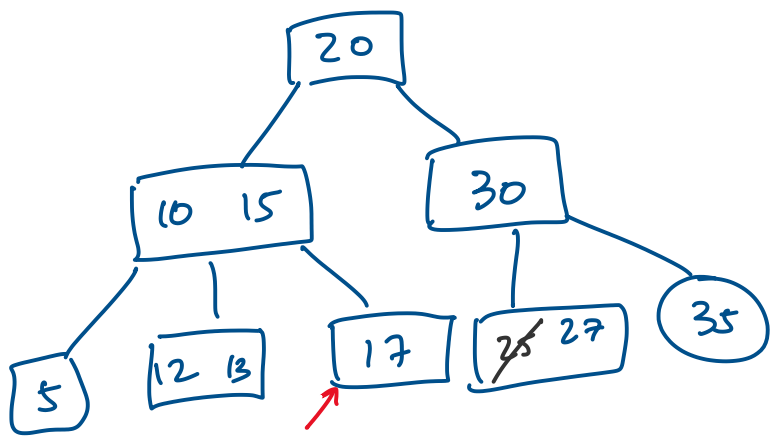
Delete (x):

Delete a leaf node.

Delete (25):

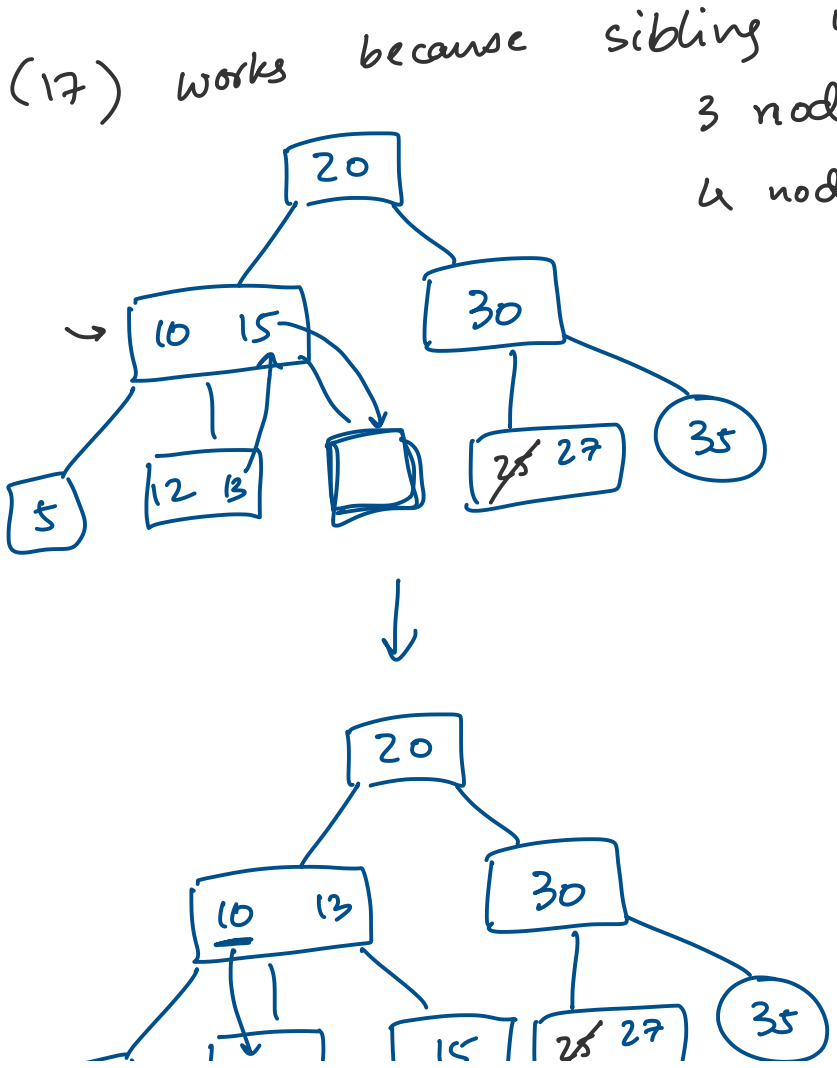
- creates an empty slot,

  but empty slot is in a 3-node, ✓



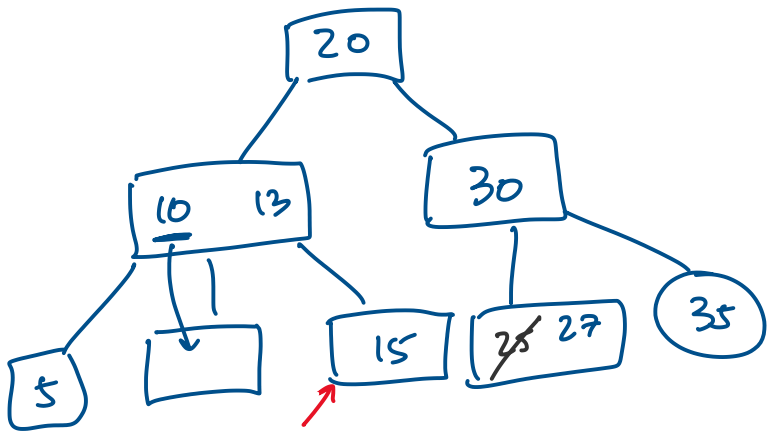
- what if empty slot is in 2-leaf node?

Delete (17) works because sibling was a 3 node or 4 node



- what if only 2-node siblings ?

Delete (12) ?



L27. 2-4 Trees contd.

Wednesday, October 20, 2021 9:54 PM

kd trees : General d recursive relation for time complexity — $O(n^{1-1/k})$

$$T(n) = \ldots T(n/\ldots) + \ldots$$

Recap :

2-4 Trees, m-B Trees

m-B tree :

size property : r-1 keys $r \in [m/2, m]$

r subtrees

depth property : all leaf nodes at same depth

d

$$\log_m n \leq d \leq \log_{m/2} n$$

2-4 Trees : Spl. case of m-B trees, $m = 4$

Why use B-Trees ?

- useful when data stored on disk / database

Search (x) in m-B trees :

log m . height $\geq \log_{m/2} n$

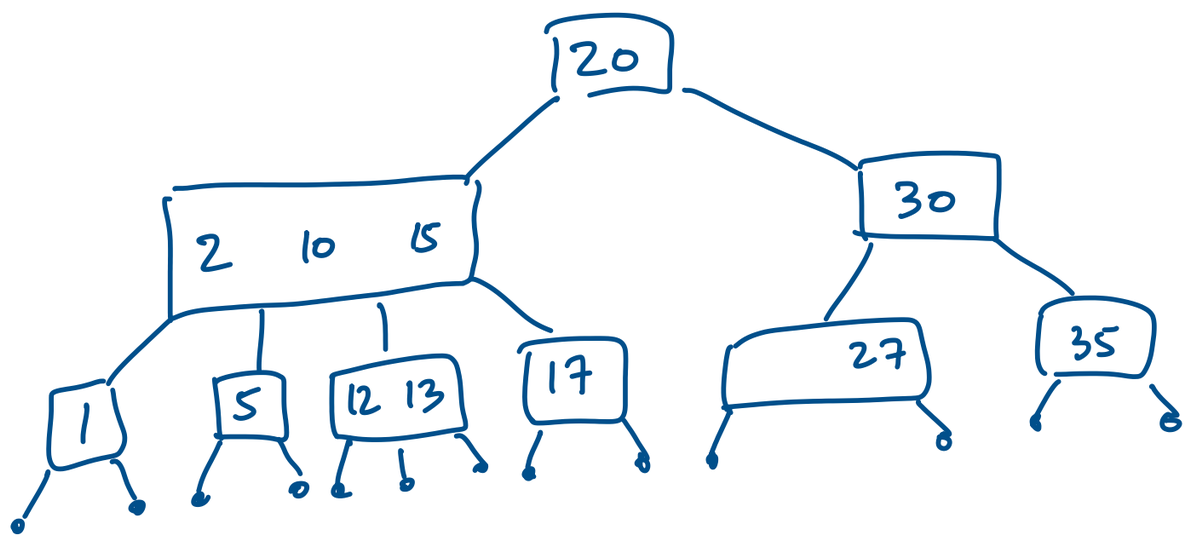
$= \log_2 m \cdot \log_{m/2} n$

Same as BST

$\ln m \cdot \log_2 n \approx \log_2 n$

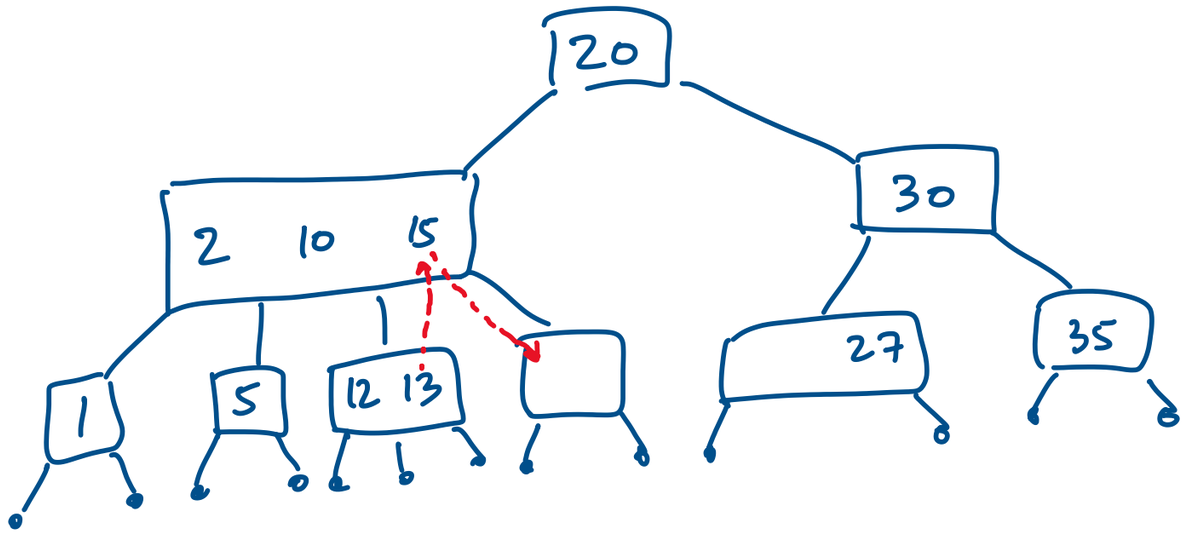
$$= \frac{\log_2}{\log_2(m/2)}$$

Deletion :



Delete (25) [Case 1]

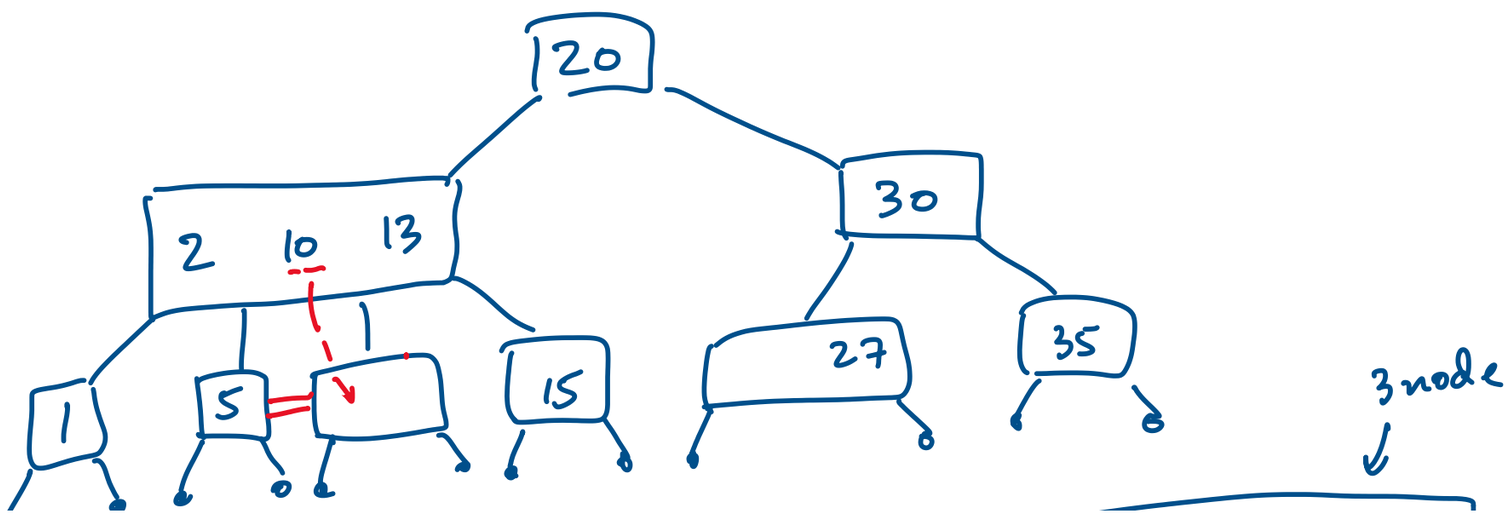
- creates an empty slot
- empty slot is in 3/4 node, so can be deleted.

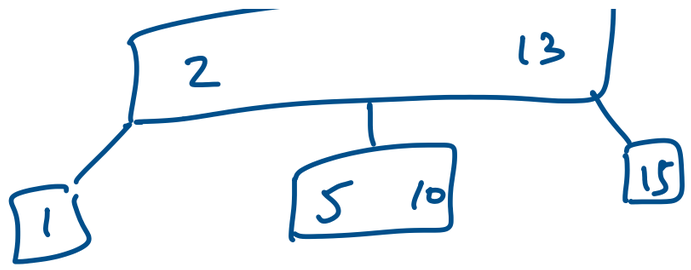


Delete (17) : [Case 2]

[Case 2.1]

- if sibling is a 3 or 4 node, then
  transfer from parent to empty slot
  " " sibling to parent.

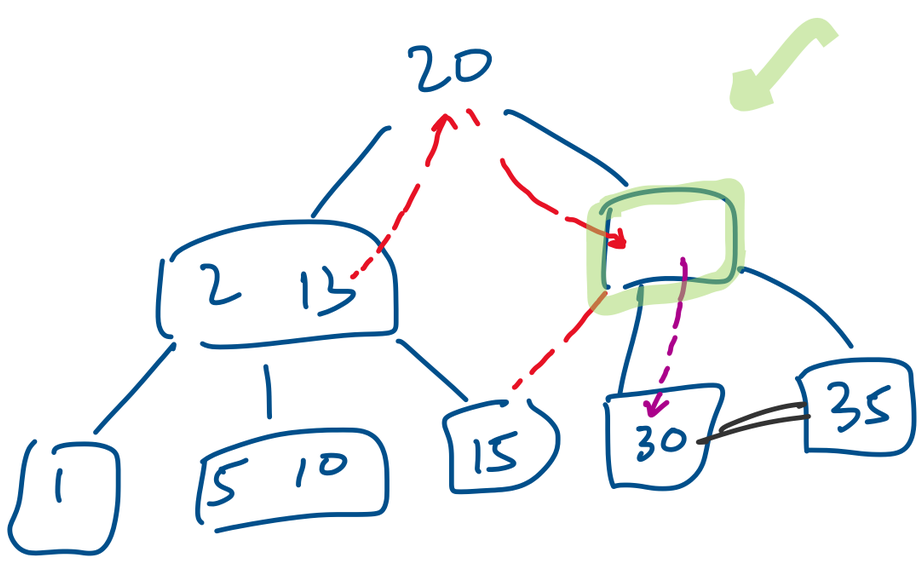


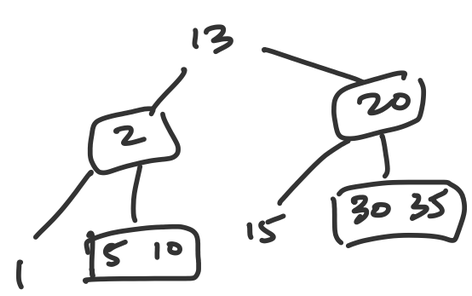


Delete (12) :

empty slot is in 2-node
both siblings are also 2-nodes

[case 2.2]



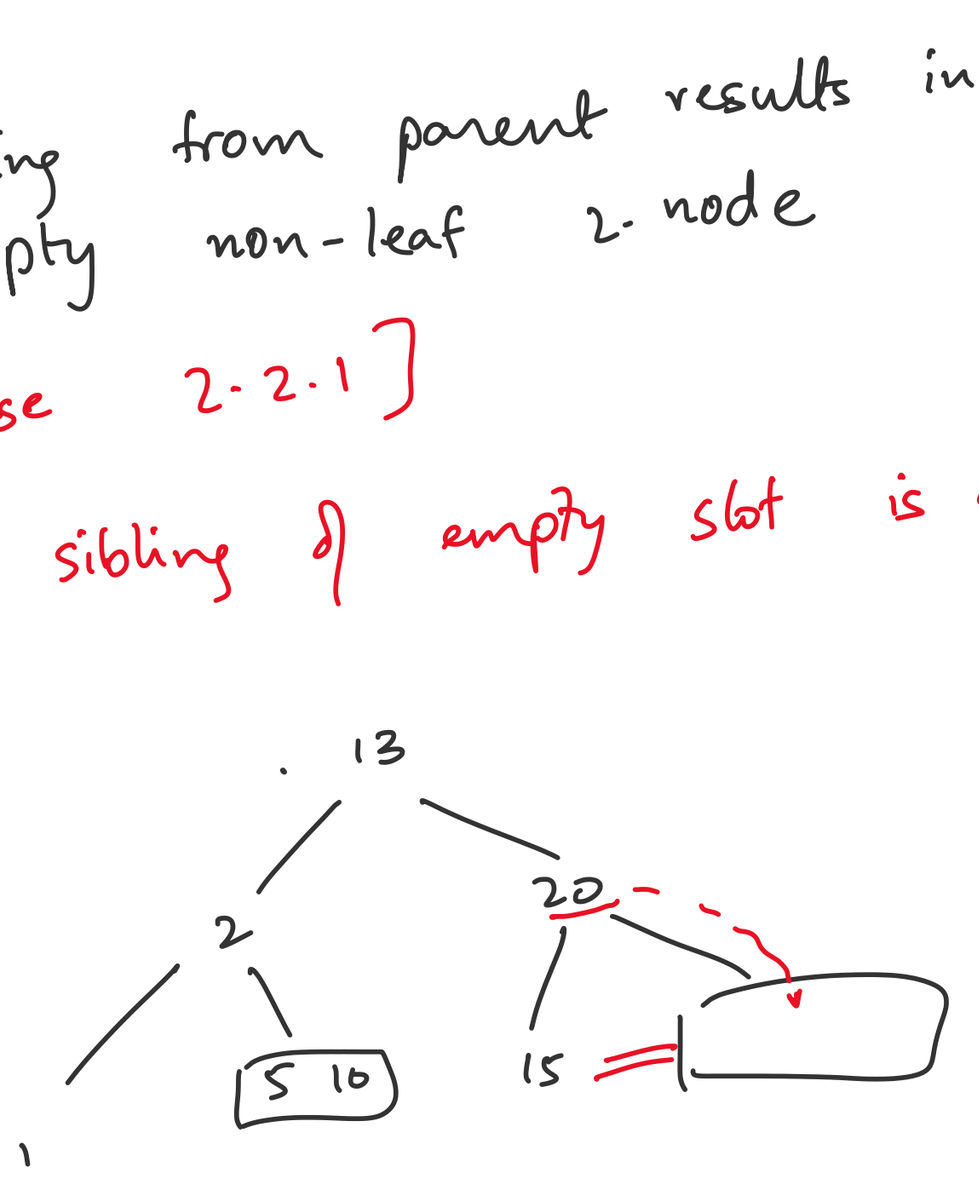


Delete (27) :

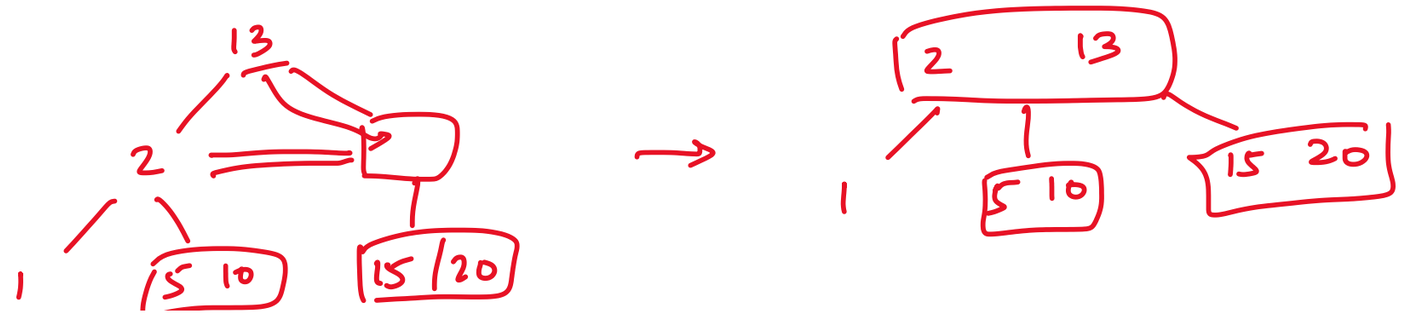
borrowing from parent results in an empty non-leaf 2-node

[case 2.2.1]

- sibling of empty slot is a 3/4 node

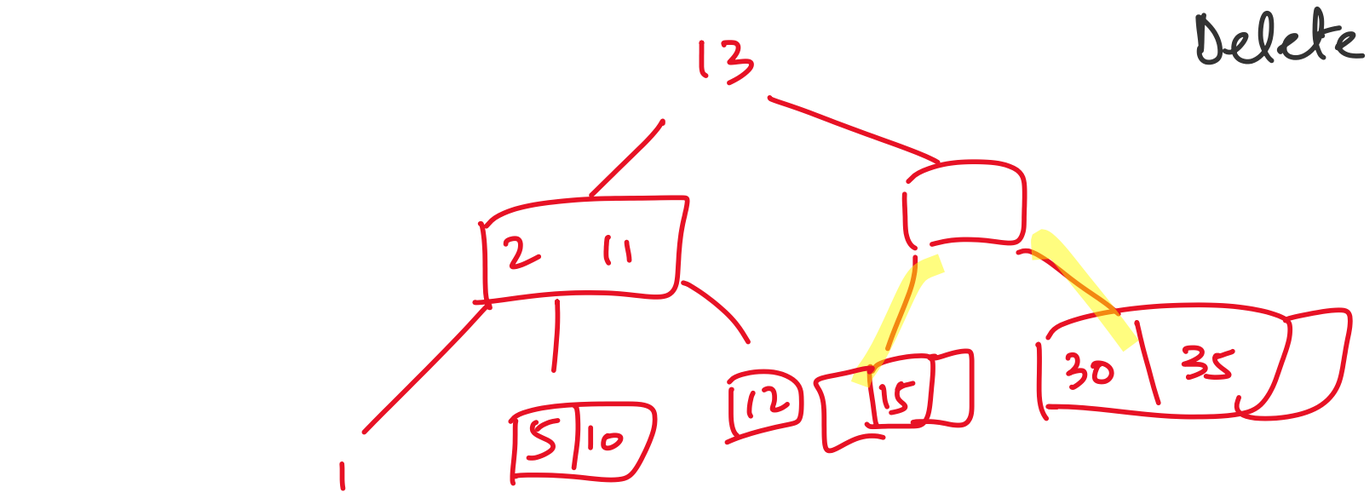


Delete (30) followed by Delete (35)



Delete at leaf

Delete at non-leaf :



Delete (20)

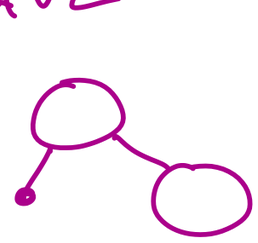
Delete (x): Find IP/IS of x. [always a leaf node]

Replace x with IP/IS.

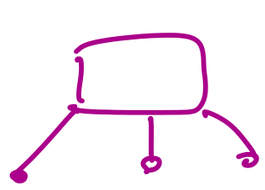
delete IP/IS.

Qn: why is IP(x) or IS(x) always a leaf node ?

AVL



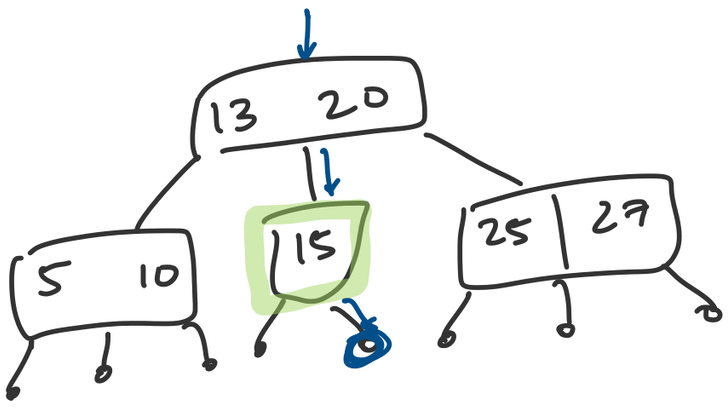
m-B

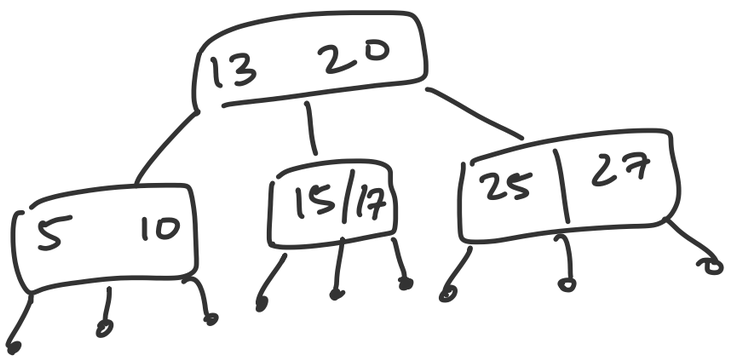


Insert (x)

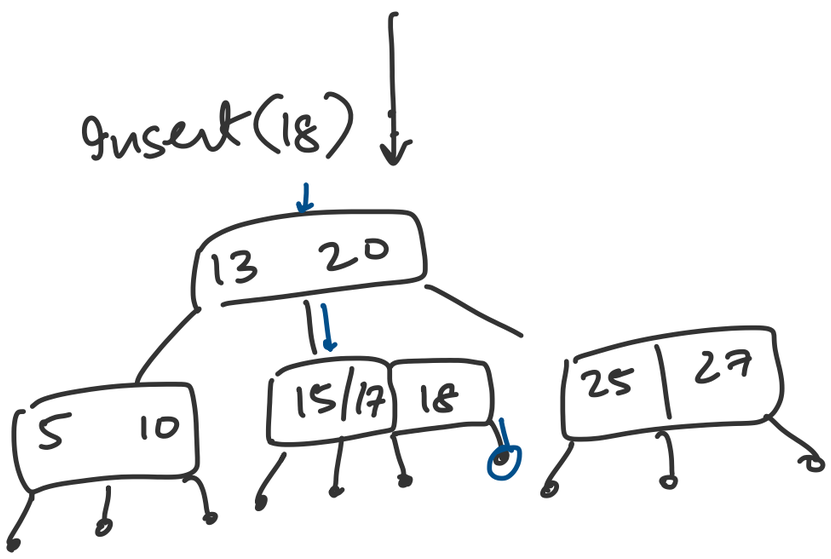
- Find (x) → external node.
- w : parent of this ext. node
- Add x to node w.

Insert(17)



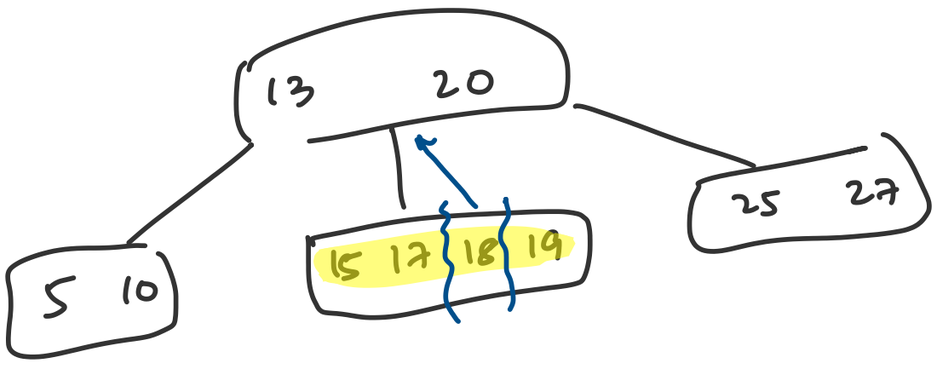


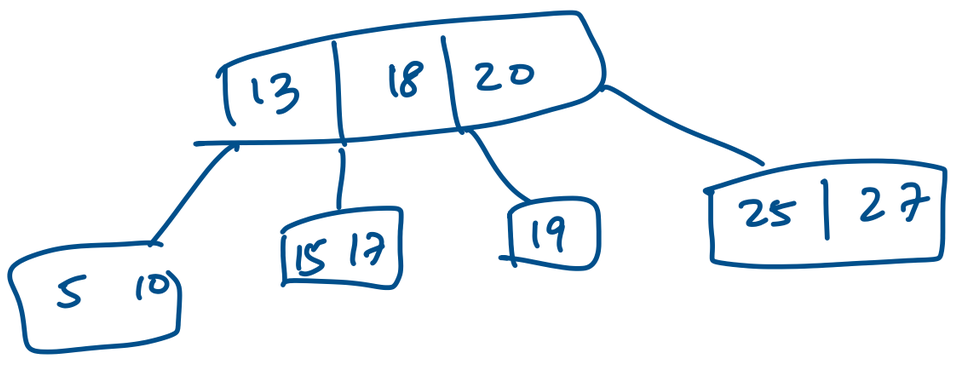
Insert(18)



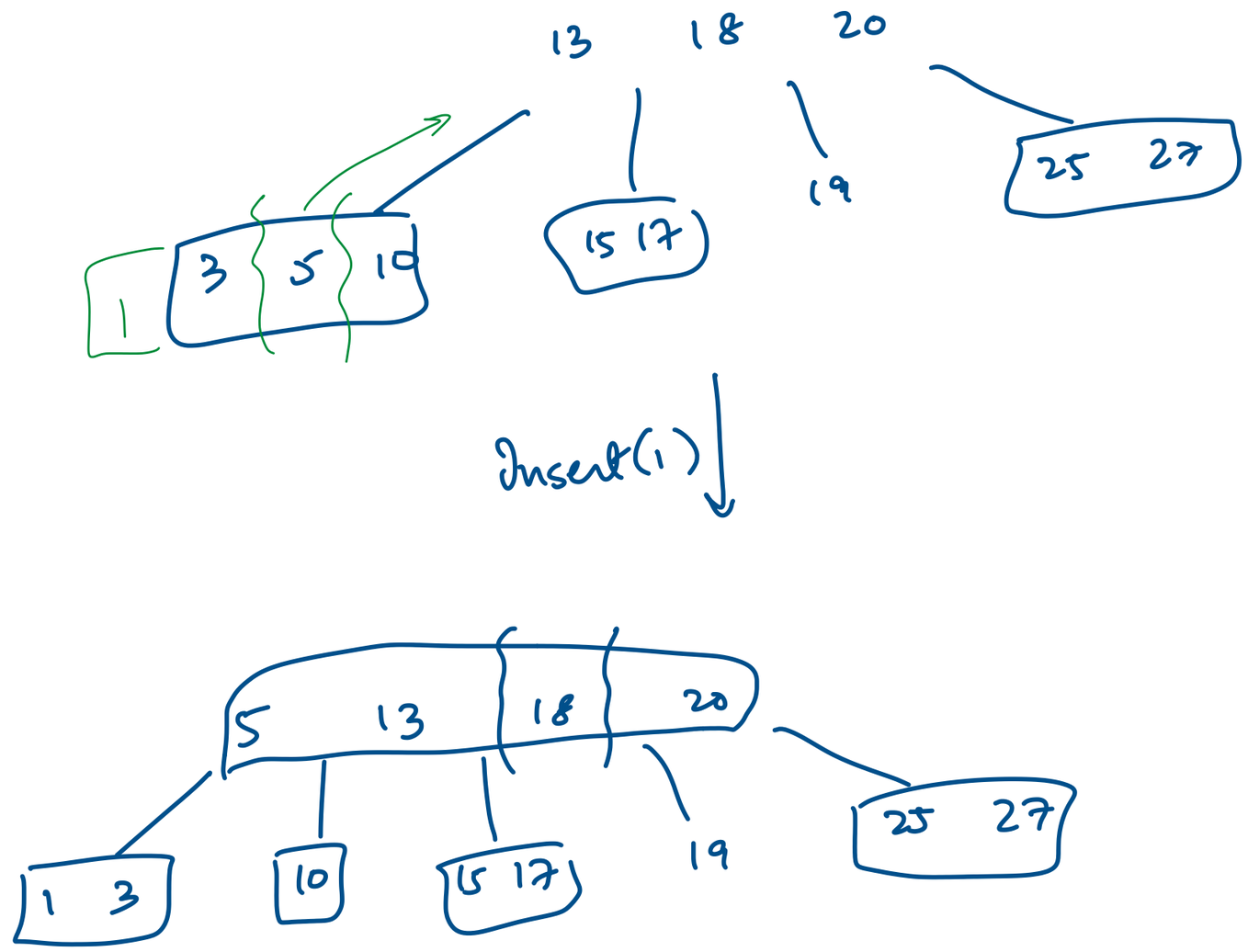
Insert(19)

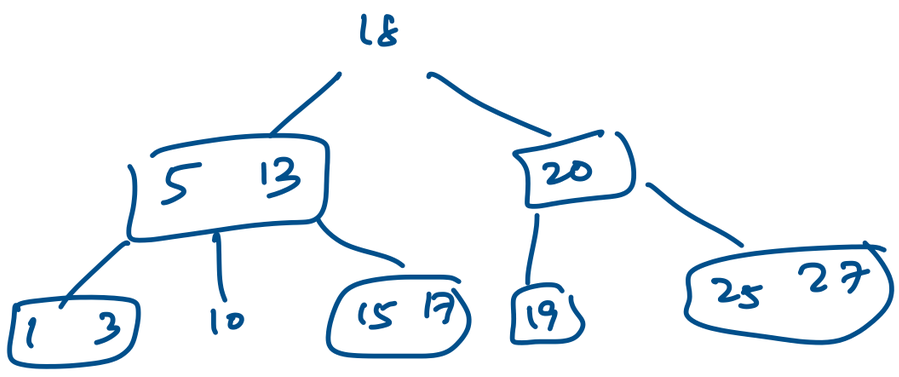
overflow



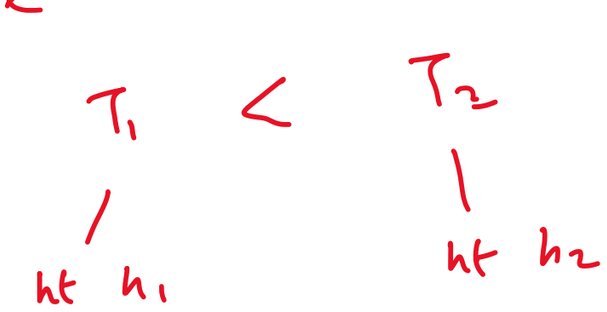


Insert(3)





Qn: Merge two 2-4 trees ?



Qn: mB tree : each node r keys

$r \in [m/2, m]$

$r \in [1, m]$?

$r \in [m/4, m]$?

$\gamma \in [3w/4, w]$

L28. Sorting - Part 1

Thursday, October 21, 2021 9:41 PM

Today: elementary sorting algorithms < insertion sort, selection sort

randomized quick sort

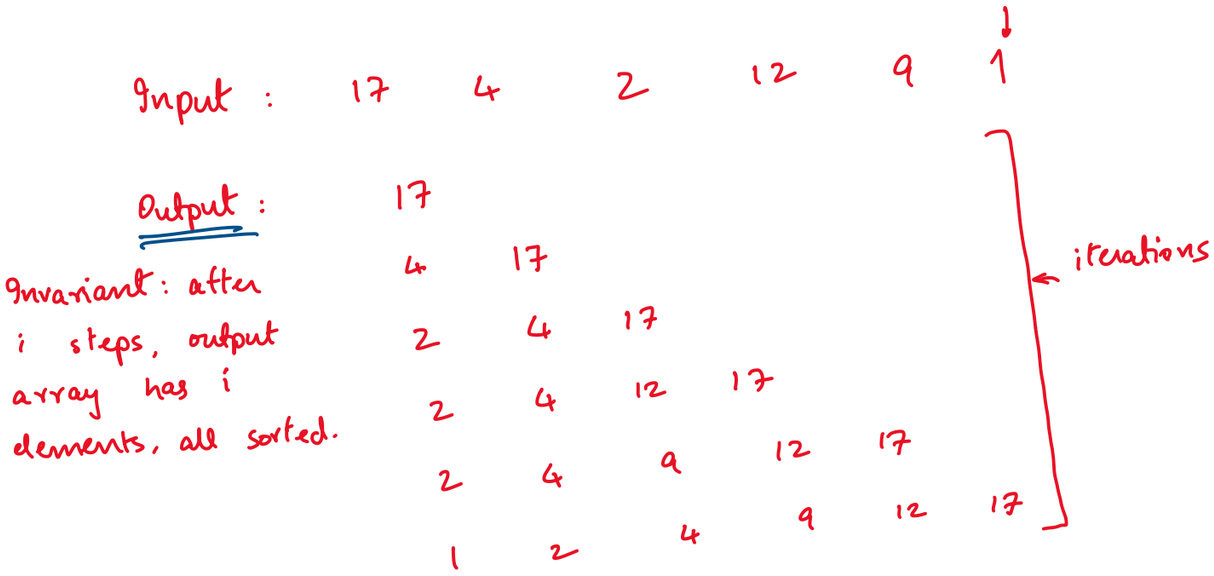
Sort (A) ← n numbers

Permute A s.t. $\forall\, i < j,\ A[i] \le A[j]$.

- Insertion Sort

Qn: Given input A, permute A randomly.

E [running time of in-place insertion sort on A]



Insertion Sort (A):

1. B := empty // output
2. For i = 0 to n-1
   a. Insert A[i] in B ← O(i) time because B is stored as an array.

Worst case time: $\sum i \approx O(n^2)$.

Qn: Can output be stored in a data str. s.t. we can have efficient inserts?

Yes! Store using AVL tree, insert requires only $O(\log i)$. $\sum_i \log i \approx O(n \log n)$

→ $O(n \log n)$ time algorithm for sorting.

Qn: Perform insertion sort without using additional space? - in-place sorting

Worst case : $O(n^2)$

Best case : n steps if input is already sorted.

Average case : $O(n^2)$

Randomized (worst case) expected time : $O(n^2)$

Randomized ...

Selection Sort :

Input : 17 ~~4~~ ~~4~~ 12 9 ~~1~~

Output : 1 2 4 9 12 17

1. B := empty
2. For i = 0 to n-1
   a. x = smallest element in A.
   b. remove x from A
   c. Put x at end of B

O(n-i) time for finding min, removing it from inp. array

If inp. stored using min heap, O(log n)

O(1) time since we always insert at the end.

→ Using min heap to store input, we get $O(n \log n)$ time sorting alg.

store input using array :

Worst case : $O(n^2)$

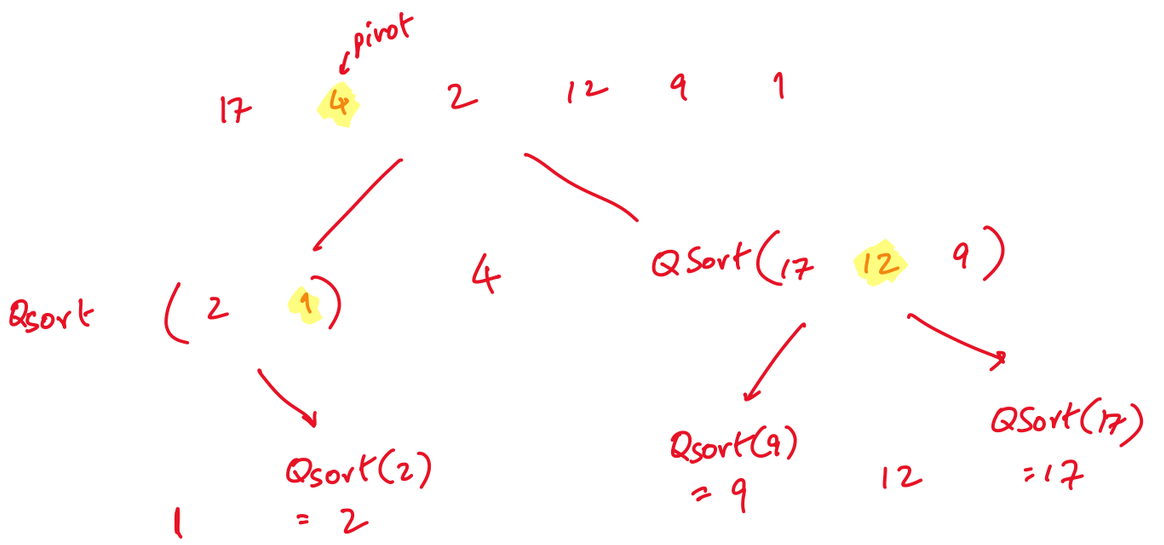
Best case : $O(n^2)$

Average case : ?

Randomized (worst case) expected time : ?

QUICK SORT :

Divide-and-Conquer alg.



QSort :

1. Pick a pivot x from A

?

Remove x from A.

2. B := all elements in A ≤ x
   C := all elements in A > x

O(n)

3. QSort(B) x QSort(C)

interval of A that needs to be sorted

QSortInPlace (A, i, j)

1. Pick a pivot x from A[i,j]
2. Pivoted Rearrange (x, A, i, j) → k

   Permutes A[i,j] s.t.

   A[k] = x

   A[i, k-1] ≤ x A[k+1, j] > x

← O(j - i)

3. QSortInPlace (A, i, k-1)

   QSortInPlace (A, k+1, j)

Fill in the base cases.

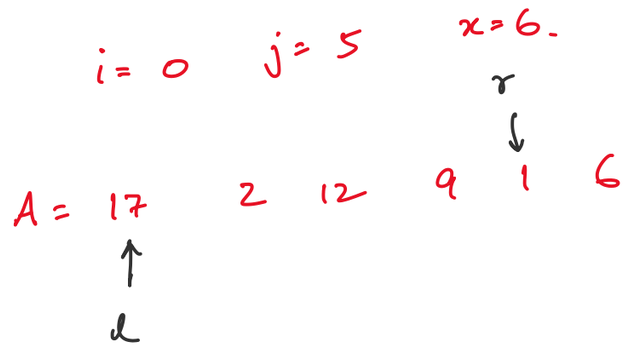
Pivoted Rearrange (x, A, i, j) :

$A = 17 \quad 6 \quad 2 \quad 12 \quad 9 \quad 1$ $\quad i=0 \quad j=5 \quad x=6.$

1. Move pivot to end of range.

   $O(n)$ time.

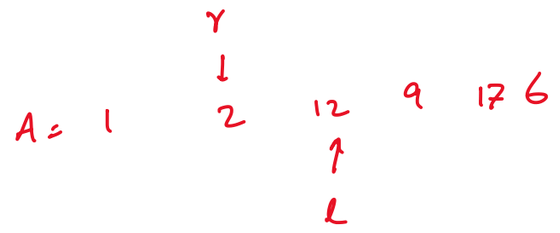
   $l = 0 \qquad r = j - 1$



Loop invariant : $l \le r$

everything to left of $l \le x$

" " rt. of $r > x$.



2. while $(l \le r)$

   2a. while $(l \le r$ and $A[l] \le x)$

   $l++$

   2b. while $(l \le r$ and $A[r] > x)$

   $r--$

   2c. if $l \le r$

   swap$(A[l], A[r])$

   $l++, \quad r--$

Qn : Prove correctness of algorithm

3. swap $(A[l], A[j])$

Qn : How to pick pivot ?

→ Take the last element in range

$O(1)$

$T_{sort}(n) \le O(1) + O(n) +$

$T_{sort}(|\text{left range}|) + T_{sort}(|\text{right range}|)$

↑ as large as $n-1$

$$= \quad n + T(n-1)$$

$$\Rightarrow \quad T_{sort}(n) = O(n^2).$$

L29. Sorting - Part 2

Friday, October 22, 2021 6:51 PM

Errata:

Average case running time ← $\underset{x \in X}{\text{avg}}$ (running time on input x)

vs

Worst case expected running time ↑ $\forall\, x \in X$, $E[\text{running time on input } x] \leq \ldots$

input x

randomness r

Recap:

QSort

- Find pivot
- Pivoted Rearrange : $O(n)$
- QSort (left range), QSort (right range)

Find Pivot :

- take last element as pivot : $O(1)$

$T(n) \leq n + T(n-1)$

$\Rightarrow T(n) = O(n^2)$.

COL 351

- take median : $T(n)$ = Time to find the median ($O(n)$) $+ n + T(n/2) + T(n/2)$

$= 2n + 2T(n/2)$

$T(n) \leq O(n \log n)$

- take a random element in the range

Given : Uniform Rand Gen (number n)

$\downarrow$

$i \in [0, n-1]$ w.p. $1/n$.

Qn: Uniform Rand Bit → 0 w.p. 1/2, 1 w.p. 1/2

How to construct Appx Uniform Rand Gen (n) using Uniform Rand Bit ?

Qn: Arbitrary BitGen → 0 w.p. p, 1 w.p. 1-p for some $p \in [1/4, 3/4]$

Use Arbitrary BitGen to build Uniform BitGen.

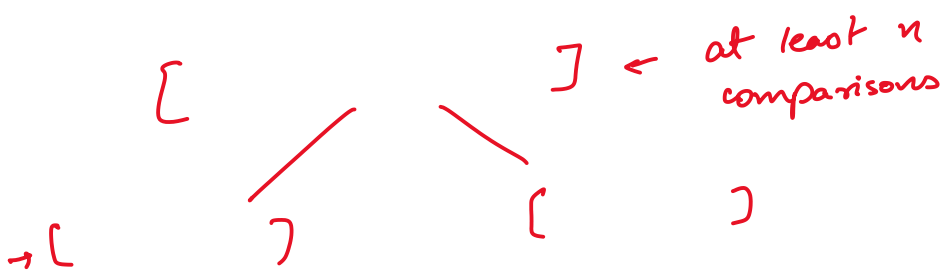
Quick Sort InPlace (A, i, j)

1. Pick pivot x in A[i,j]
   t = UniformNumGen(j - i + 1)
   x = A[i+t]
2. Pivoted Rearrange
3. Qsort(--) Qsort(--)

want: expected running time of our algorithm.
dominated by number of comparisons.

X : random variable denoting number of comparisons made during the exec. of our alg.

x takes values in range $n \log n$ to n^2



$$E[X] = \sum_x Pr[X = x] \cdot x$$

Linearity of Expectation:

$$E[X + Y] = E[X] + E[Y].$$

Thm: $E[X] \leq 2n \log n$.

Proof: Decompose X as sum of simpler r.vs

$$X = \sum Y_i$$

$$E[X] = \sum E[Y_i]$$

$\forall\, i, j \in [n]$,

$$X_{ij} = \begin{cases} 1 & \text{if during alg. execution, } i^{th} \text{ smallest element and } j^{th} \text{ smallest element are compared} \\ 0 & \end{cases}$$

$i \in [n]$. $Pr[X_{i,i+1} = 1]$

Qn: Prove that $\forall i$, $Pr[X_{i,i+1} = 1] = 1$.

Obs: Any two elements will be compared at most once.

$$X = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}$$

$$E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}]$$

Example: 17 4 7 9 13

Obs: If any element bet^n i^{th} smallest and j^{th} smallest are picked, then $X_{ij} = 0$

Qn: $Pr[X_{ij} = 1] = \frac{2}{j-i+1}$

$$E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = 2n \log n.$$

Sorting Lower Bounds :

elements using just

Qn: Can we sort n elements in $O(n)$ operations?

Common feature of all alg. seen so far:

- no structure on inputs reqd, other than the fact that $\exists$ a total ordering

Lower Bounds:

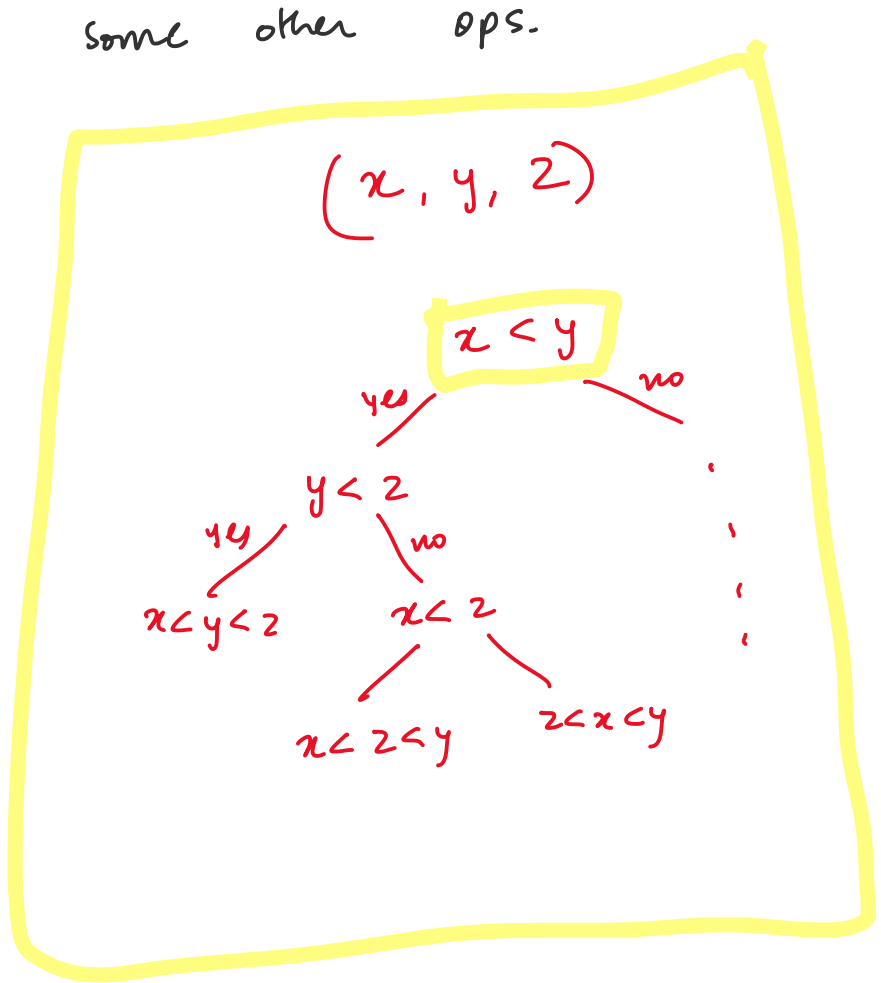
Given: comp. prob. P

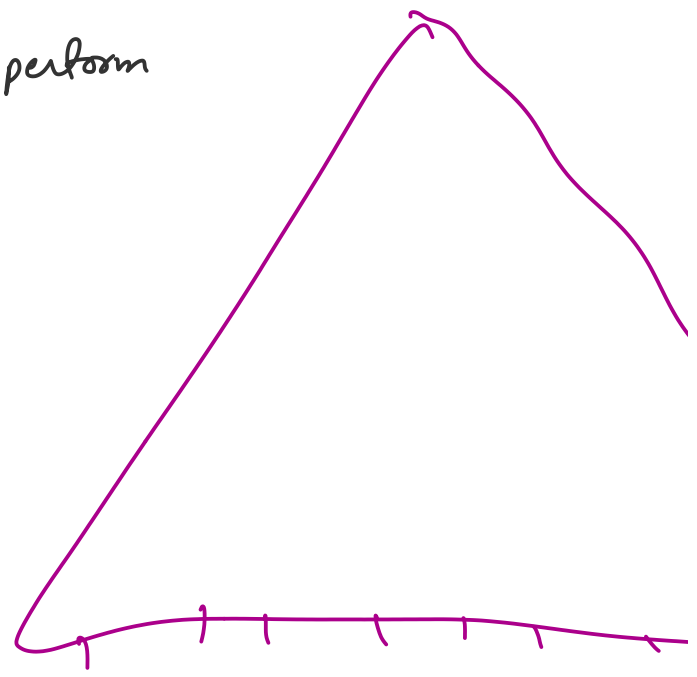
want to prove: ANY alg. (of a certain form) that solves P must use at least ---- time/space.

Defn: comp. based sorting alg.

Input: $[a_1, \ldots a_n]$

Alg. can gain info about input by making queries of the form "is $a_i < a_j$" for any i, j. Based on outcome, alg. can perform some other ops.





Obs: If number of leaves large, then ht is $\geq$.

L30. Sorting - Part 3

Tuesday, October 26, 2021 7:35 AM

Recap:

- comparison based sorting model
- `20 questions'

Def^{n}: Decision tree:

Binary tree

Non leaf nodes: non-decision ops, followed by a decision

Left subtree: Yes

Right subtree: No

Leaves: Outputs.

Alg → decision tree

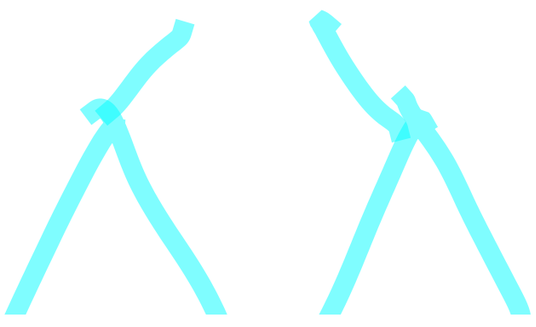
if stmt → non-leaf node

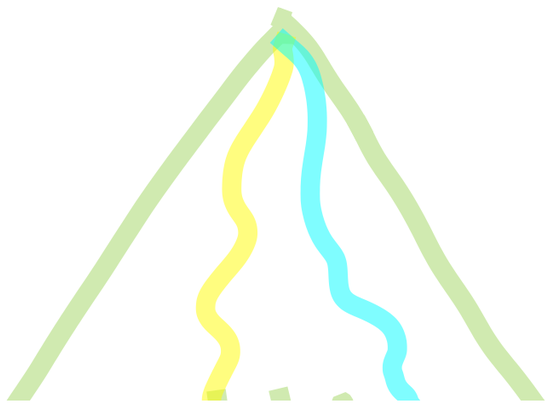
output → leaf node

execution of alg. on input x → path from root to leaf

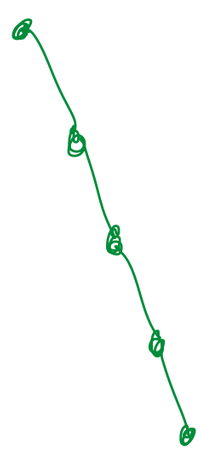
max. number of decisions on any input x. → ht. of tree

- Multiple leaf nodes with same output?





Obs 1 : worst case running time of alg. A in comp. based sorting model
$\geq$ ht. of decision tree corresponding to alg. A.



Obs 2: ht of any decision tree $\geq \log_2$ (number of leaf nodes)

Obs 3: For sorting, output $\equiv$ permutation of the input.

[3 2 7 1] [1 2 3 7]

Fact: Number of leaf nodes
$\geq$ Number of permutations $= n!$

$\Rightarrow$ worst case running time of any alg. in comp. based sorting model
$\geq \log(n!) \geq \frac{n}{2} \log \frac{n}{2}$

$n! \geq \left(\frac{n}{2}\right)^{n/2}$ $\quad n! \geq \underbrace{n \cdot (n-1) \cdots (n/2)}_{\geq n/2}$

For better bound, see Stirling's approx.

Can we have a
[illegible] with Find-Min, Delete-Min in

Qn: Min-heap ... $\sqrt{\log n}$ time?

Qn: Show that worst case expected running time / avg. case running time of any comp. based sorting $\geq \Omega(n \log n)$

Integer Sorting:

↳ input array consists of integer

integers can be added/sub/ used for indexing arrays

- Count Sort:

input: $(a_1 \dots a_n)$

$a_i \in [0, k-1]$

sort array in $O(n+k)$ time

- Radix Sort:

input: $(a_1 \dots a_n)$ $\quad a_i \in [0, k^d - 1]$

$O(d \cdot (n+k))$

Count Sort:

- keep an array to store freq. of each element

$A = [3 \quad 5 \quad 2 \quad 5 \quad 1 \quad 3]$ $\quad k = 6$

$B = [0 \quad 1 \quad 1 \quad 2 \quad 0 \quad 2]$

Use B to output sorted array

- Input: $((k_i, v_i))$ where each $k_i \in [0, k-1]$
- Output s.t. ordering of same key is preserved.

$A: [(3,4), (5,7), (2,1), (5,12), (1,17), (3,14)]$

$\downarrow$

$A' = [(1,17), (2,1), (3,4), (3,14), (5,7), (5,12)]$

Create array of lists B

$B = [\perp, (1,17), (2,1), (3,4), (3,14), \perp, (5,7), (5,12)]$

for $i = 0$ to $n-1$ $\quad O(n+k)$

// $A[i] = (k_i, v_i)$

$B[A[i].key].append(A[i])$ ⇐ $O(n)$ time

↑ we are using lists here. what if arrays?

For i=0 to k-1

B is an array of w

what if B is an array of stacks?

if B[i] is non-empty

output each element of B[i] sequentially.

↖ O(k+n) time

Qn: Sort integers without assuming range is bdd.

$n\sqrt{\log \log n}$ ⇝ $O(n)$

L31. Radix sort, Graph ADT

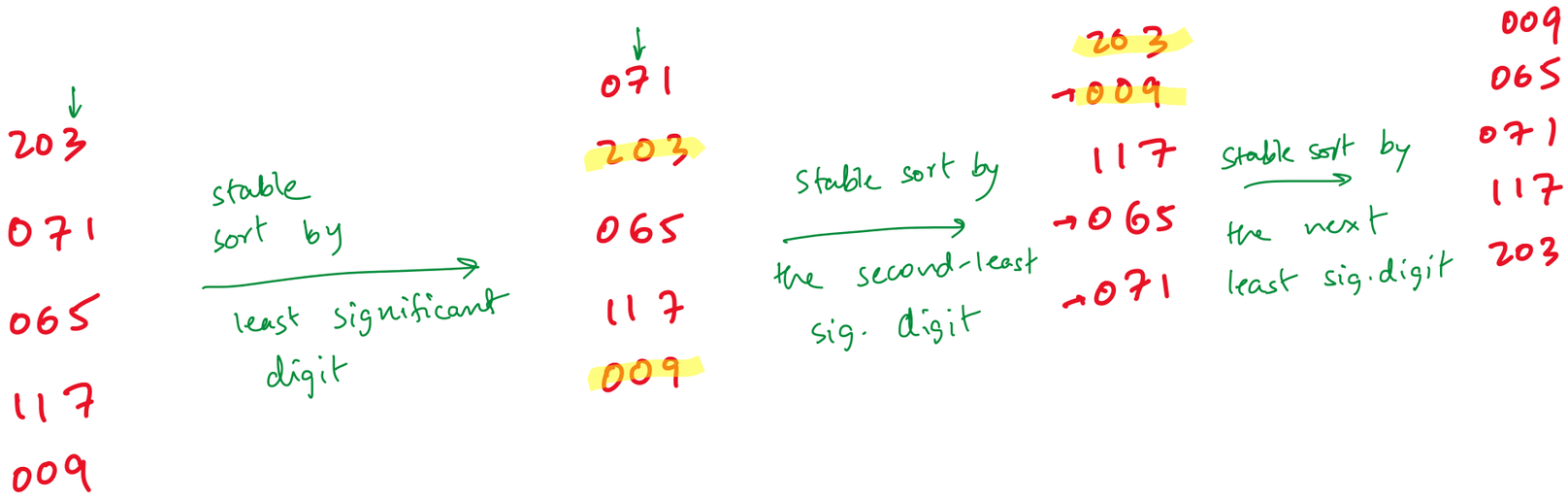
Wednesday, October 27, 2021 11:12 PM

Integer Sorting

- Count Sort : sorts integer arrays
  each $a_i \in [0, k-1]$
  stable sorting alg.
  $O(n+k)$

Qn: QSortInplace - is this a stable sorting alg ?

- Radix Sort : $a_i \in [0, k^d - 1]$
  $O(d \cdot (n+k))$



$a_i \in [0, k^d - 1]$

- express each integer in base k

$$\alpha = \alpha_0 + \alpha_1 \cdot k + \alpha_2 \cdot k^2 + \dots + \alpha_{d-1} \cdot k^{d-1}$$

$$\alpha \equiv (\alpha_{d-1}, \alpha_{d-2}, \dots, \alpha_0)$$

↑ most sig. digit
$(d-1)^{th}$ sig. digit

↑ least sig. digit
0^{th} sig. digit

For each a_i,
$O(d)$ computations to compute the corresponding base k rep^n

- For i = 0 to d-1
  P_1 // At start of i^{th} iteration, the array is stable sorted wrt the last (i-1) digits
  count-sort using the i^{th} base-k digit as key the number as the value

d iterations, each iteration requires $O(n+k)$ operations.

P_2 // At end of i^{th} iteration, the array is stable sorted w.r.t last i digits

convert number back to base 10 rep^n

- convert each number [illegible]

Claim: $P_1 \Rightarrow P_2$, assuming count sort is a stable and correct sorting alg.

Total running time : $O(d \cdot (n+k))$

Qn: Is this a stable sorting alg ?

Graphs and Graph ADT :

Qn: A: adjacency matrix of a directed graph

Does there exist a vertex v with no outgoing edges and n-1 incoming edges?

↳ O(n) time

- Delhi Metro, find the number of stops reqd to go from station A to station B

- Facebook social network

Any two people x, y,

$\exists$ $u_1 \dots u_6$ s.t.

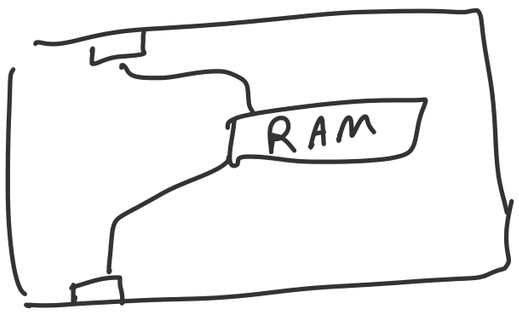
$x = u_1$, (u_1, u_2) are friends on FB

(u_2, u_3) " "

:

(u_5, u_6) " " "

$u_6 = y$

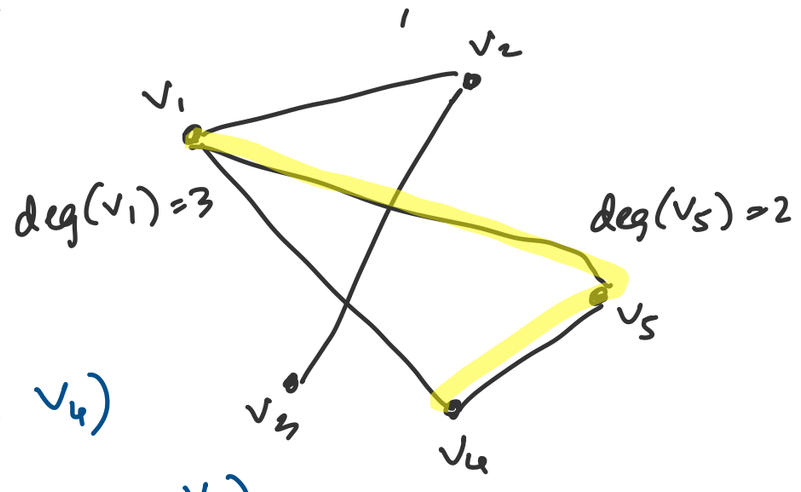


Formally, $G = (V, E \subseteq V \times V)$

vertex set : any set of objects

edge set

$n = |V|$, $m = |E|$.



path: $(v_1\ v_5\ v_4)$

walk: $(v_1\ v_2\ v_1\ v_5\ v_4)$

cycle: $(v_5\ v_4\ v_1\ v_5)$

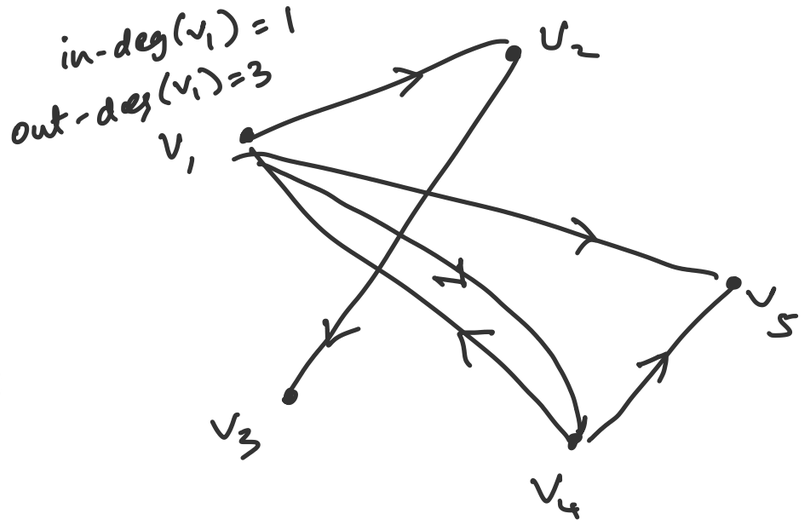
undirected graph

Facebook : undirected graph

Instagram / Twitter : directed graph

Max no. of edges in directed graph : $n \cdot (n-1)$
with no self loops

For undirected graph : $\frac{n(n-1)}{2}$.



directed graph

---

degree of a vertex v of an undirected graph :
number of edges incident on v

in-deg(v) in a directed graph : number of edges 'coming into' v

out-deg(v) : number of outgoing edges

Claim! For a directed graph,

$$\sum_v \text{in-deg}(v) = \sum_\omega \text{out-deg}(w) = m$$

Claim 2 For an undirected graph,

$$\sum_{v} deg(v) = 2m.$$

Qn: ∃ ? undirected graph with 5 vertices

$deg(v_i) = 3$? Not possible,

$\sum deg(v_i) = 15$, not equal to $2m$.

Graph G :

walk = $(v_0, v_1, \ldots, v_k)$ s.t.

each $(v_i, v_{i+1}) \in E$

path : $(v_0, v_1 \ldots v_k)$ is a path if

each $(v_i, v_{i+1}) \in E$, no $v_i = v_j$

cycle : $(v_0, v_1 \ldots v_k)$ is a cycle if

- $v_0 = v_k$
- each $(v_i, v_{i+1}) \in E$
- $v_i \neq v_j \quad \forall \quad i,j > 0$

How to store a graph ?

- Adjacency List representation

Radix sort: $O(d \cdot (n+k))$

Radix Sort ++ :

L32. Graph Traversal

Friday, October 29, 2021 12:36 AM

Recap:

- undirected, directed graphs
- deg of a vertex, in-deg / out-deg of vertex

  neighbours of $v = \{ w : (v, w) \in E \}$

- walks, cycles, paths

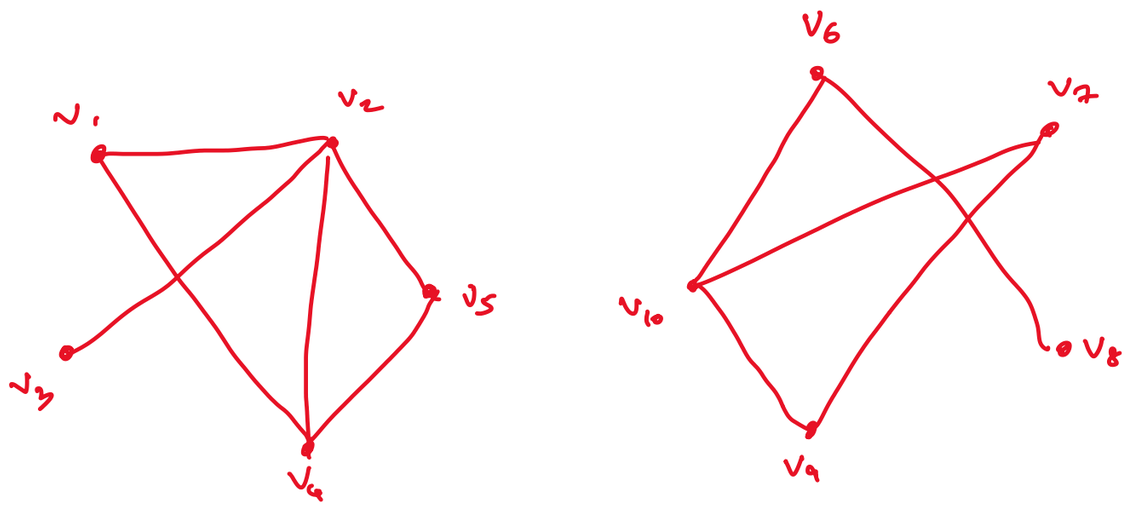
  vertex w is reachable from v if $\exists$ path from v to w.

  $v = v_0 \quad v_1 \quad \dots \quad v_k = w$

Qn: Given v, w, $\exists$? path from v to w?

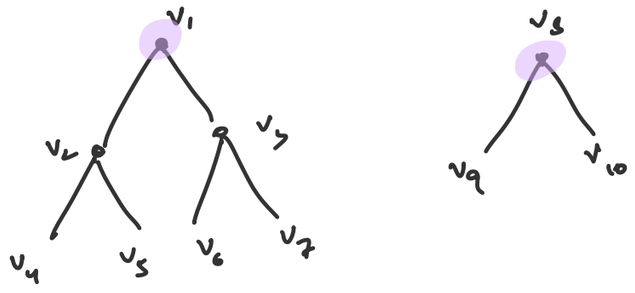
Problem: Given: $v \in V$, $G = (V, E)$ undirected graph

Find all vertices $w \in V$ s.t. w is reachable from v.



set of vertices reachable from $v_1 = \{v_1, v_2, \dots, v_5\}$

1D-Trees - Range Queries



Find all leaf nodes in these trees.

Given v_1, find all nodes reachable from v_1.

1. Q : empty queue
   R : empty list

Can we do the same for general graphs?

// R: all nodes reachable from v_1

2. Q.enqueue(v_1), R.append(v_1)

3. while Q is not empty

3.1. x = Q.dequeue()

3.2. if x has children

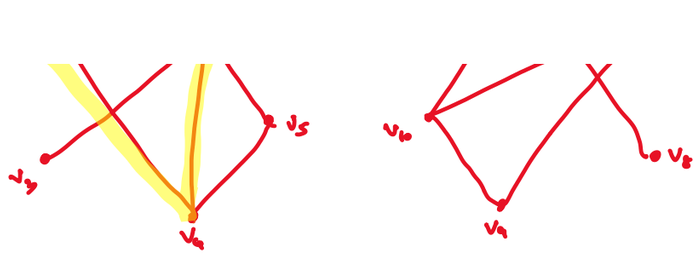
3.2.1. For each child w of x

3.2.1.1. Q.enqueue(w)

R.append(w)

Qn: The above alg. terminates if there are no cycles.

Not true!



children → neighbours

Q = [v_4 v_1 v_3 v_4 v_5]

R = [v_1 v_2 v_4 v_1 v_3 v_8 v_5]

How to handle cycles?

Put a small 'tick' next to each vertex, denoting that its already visited. Add to queue only if unvisited.

Given v_1, find all nodes reachable from v_1.

[BREADTH FIRST SEARCH V.0]

1. Q : empty queue
   R : empty list
   // R: all nodes reachable from v_1
   $\forall v$, v.visited = 0
2. Q.enqueue(v_1), R.append(v_1)
   v_1.visited = 1

O(n) steps.

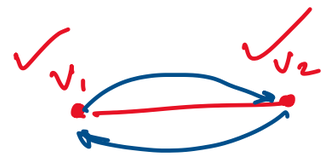
3. while Q is not empty
   // $\forall w \in R$, $\exists$ a path from v_1 to w
   3.1. x = Q.dequeue()
   3.2. if x has ~~children~~ neighbours
   neighbour
   3.2.1. For each ~~child~~ w of x
   3.2.1.1. if w.visited = 0
   Q.enqueue(w)
   R.append(w)
   w.visited = 1
   // $\forall w \in R$, $\exists$ a path from v_1 to w

$\leq$ O(n) steps

O(n) steps

$\Rightarrow O(n^2)$ for both adj. List, adj matrix

Claim: If G is rep. as an adj. List, then



Q = [ ]

why does the alg. terminate?

- every vertex added to queue at most once.
- in every iteration, we dequeue 1 element.

Hence, the loop must terminate.

running time is $O(n+m)$. $n = |V|$, $m = |E|$

Proof of correctness:

1 – R includes only vertices that are reachable from v_1

2 – $\forall w$, if $\exists$ path from v_1 to w $(v_1 = u_0\ u_1 \dots u_k = w)$
then w was added to Q at some point.
and R contains w.

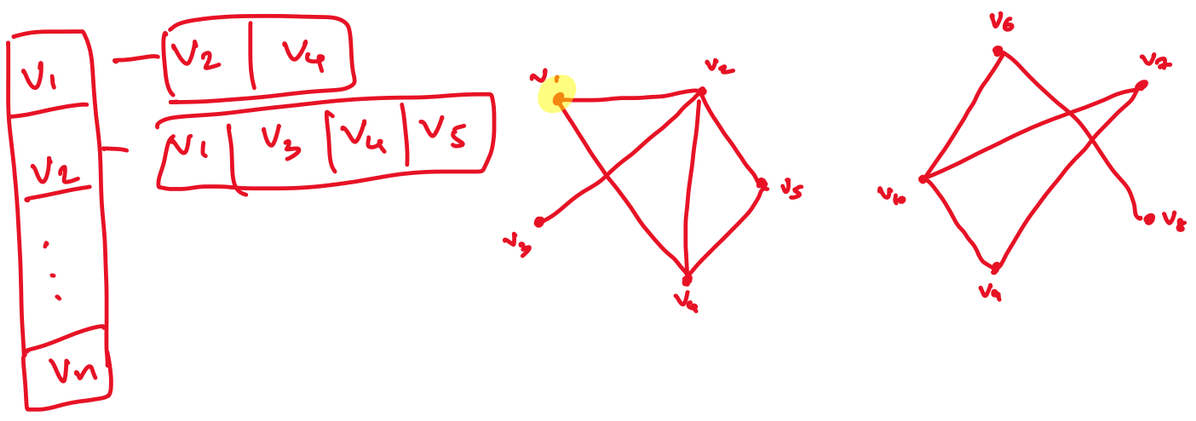
Proof by induction on k.

– Induction hypothesis holds for $k=0$.

Suppose it holds for $k-1$. Then u_{k-1} was in Q at some point. When it is dequeued, check if w (nbr. of u_{k-1}) is visited. If yes, then w is already in Q, R.
If no, then its added to Q, R.

Time Complexity ?

Adjacency List



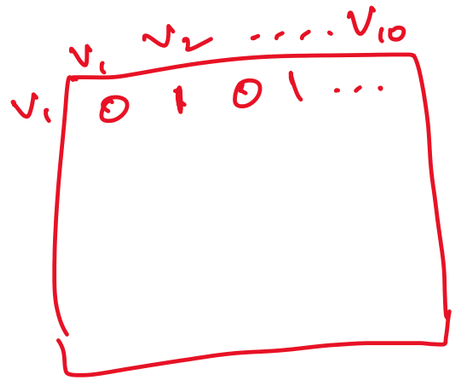
For each vertex, we have a list of nbrs.
– size of ith list : $\deg(v_i)$

size of adjacency list: $2|E| + |V|$.

– Get all neighbours of v
$O(\deg(v))$ ops.

– Check if $\exists$ an edge bet^n v, w
$O($ $)$

Adjacency matrix



A : n x n matrix
$A[v,w] = 1$ if $(v,w) \in E$
$= 0$ otherwise

– Get all nbrs of v
$O(n)$ ops – scan entire row of v

– Check if $\exists$ edge bet^n v, w
$O($ $)$

$O(\deg(v))$

$O(1)$ ops.

Sink node : no outgoing edges.

$n-1$ incoming edges

Qn: Given adj. matrix, find if $\exists$ sink node in $O(n)$ time.

L33. Graph Traversal - Part 2

Monday, November 1, 2021 3:34 PM

Recap:

[BREADTH FIRST SEARCH V.0]

1. Q : empty queue
   R : empty list
   // R : all nodes reachable from v_1
   $\forall v$, v.visited = 0

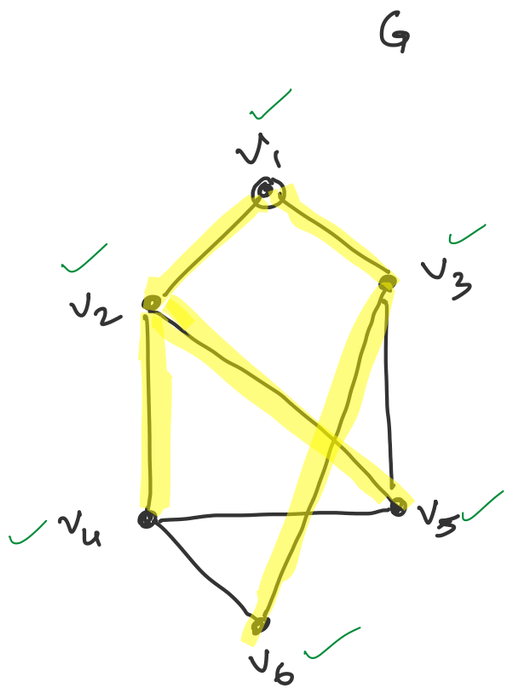
   $O(n)$ steps.

2. Q.queue(v_1), R.append(v_1)
   v_1.visited = 1

3. while Q is not empty
   // $\forall w \in R$, $\exists$ a path from v_1 to w
   3.1. x = Q.dequeue() ← $O(1)$
   3.2. if x has neighbours
      3.2.1. For each neighbour w of x ← $O(\deg(x))$
         3.2.1.1. if w.visited = 0
            Q.queue(w)
            R.append(w)
            w.visited = 1
   // $\forall w \in R$, $\exists$ a path from v_1 to w

   $\leq O(n)$ steps

$O(n)$ steps



Q :

R : v_1 v_2 v_3 v_4 v_5 v_6

Running time analysis:

→ if G stored as an adj. matrix : $O(n^2)$

→ if G stored as an adj. list : $O(n^2)$

Better bound:

$$\sum_x O(1) + O(\deg(x)) \leq O(n) + O\Big(\underbrace{\sum \deg(x)}_{\leq 2m}\Big)$$

$$O(n+m)$$

BFS v.0 can be modified to compute the dist from a fixed vertex v to all other vertices in G.

[BREADTH FIRST SEARCH v.1 ~~]~~]

1. Q : empty queue
   R : empty list

$O(n)$ steps

// R: all nodes reachable from v_1

$\forall v$, v.visited = 0

$\forall v$, v.distance = ∞

2. Q.queue(v_1), R.append(v_1)

v_1.visited = 1

v_1.distance = 0

3. while Q is not empty

// $\forall w \in R$, $\exists$ a path from v_1 to w

3.1. x = Q.dequeue() ← O(1)

3.2. if x has neighbours ← O(deg(x))

3.2.1. For each neighbour w of x

3.2.1.1. if w.visited = 0

w.distance = x.distance + 1

Q.queue(w)

R.append(w)

w.visited = 1

3.2.1.2 else if w.visited = 1 and

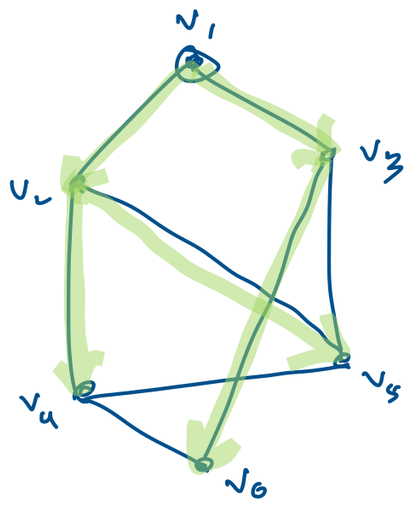
w.distance > x.distance + 1

w.distance = x.distance + 1

Qn: do we need to have 3.2.1.2 ?

Not needed. follows from proof discussed in last class.

BFS tree :



$v_1 \to v_2 \to v_3$

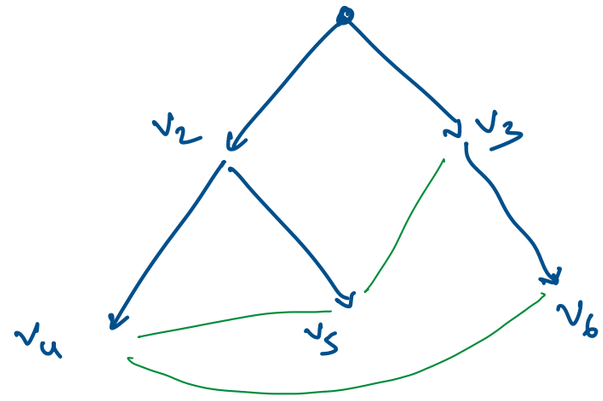
$v_2 \to v_1 \to v_4 \to v_5$

$v_3 \to v_1 \to v_5 \to v_6$

$v_4 \to v_2 \to v_5 \to v_6$

$v_5 \to v_2 \to v_3 \to v_4$

$v_6 \to v_3 \to v_4$

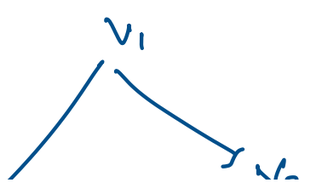


Qn: Is BFS tree unique given root v_1?

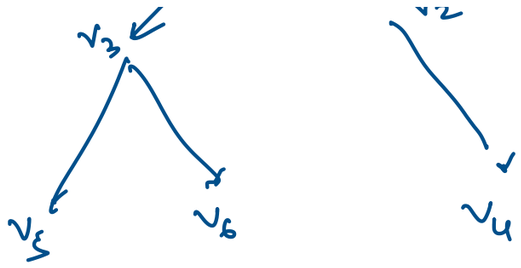
No. If v_2 & v_3 swapped, then

undirected graph :

- non tree edges can either be from level i to level i, i-1



or level i to level $i \pm 1$



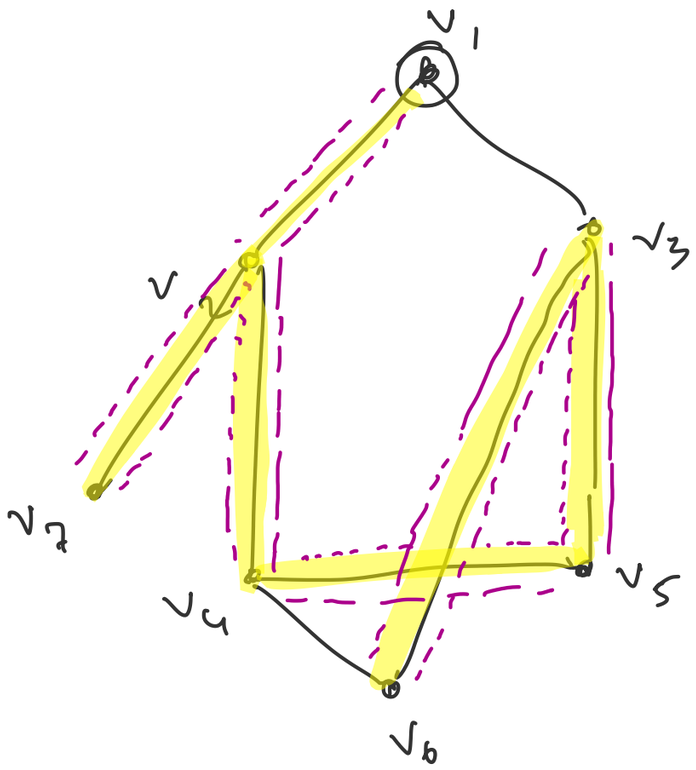
directed graphs:

- layer i to layer $i+1$
- layer i to layer $j \leq i$

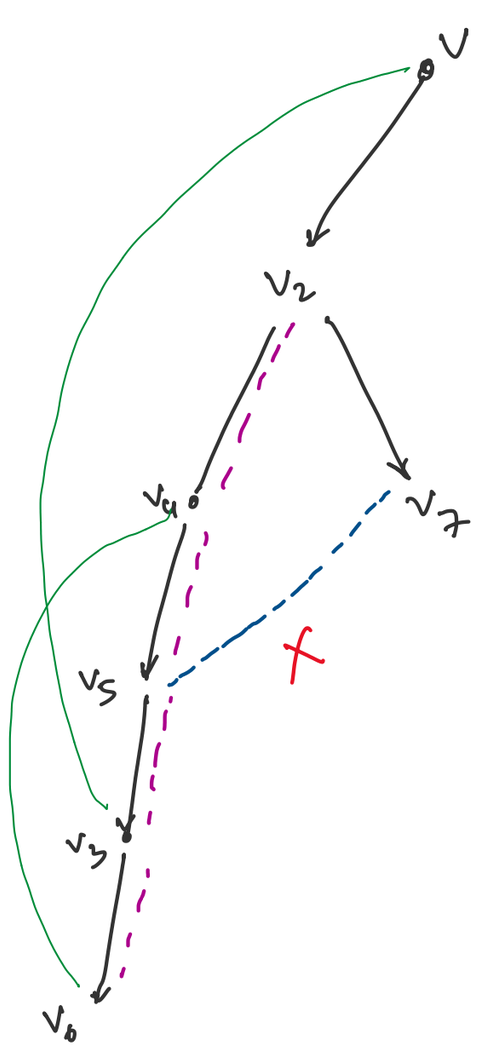
Qn: Use BFS to check if an undirected graph has an odd length cycle.

Depth First Search

- can explore the entire map / graph
- cannot be used for computing distances.



DFS tree:



- all non-tree edges are from a node to ancestor / descendant

$\vdash \quad \forall x. \quad x.\text{visited} = 0$

2. v.visited = 1

DFS(v) : [version v0]

Non recursive pseudocode using stacks

→ Next time

→ Time Complexity

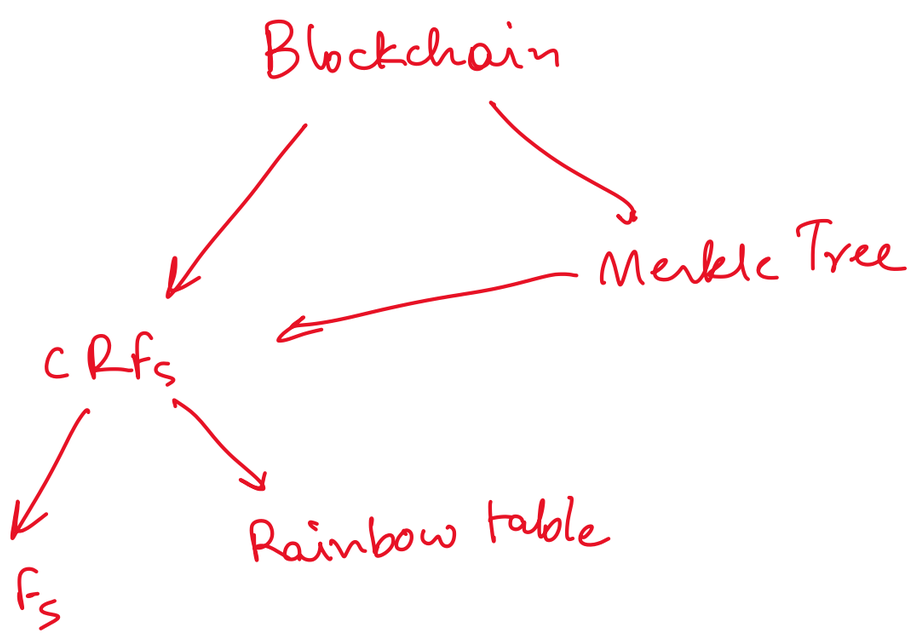
3. for each neighbour x of v

3.1. if x.visited = 0

3.1.1. x.visited = 1

3.1.2. DFS(x).

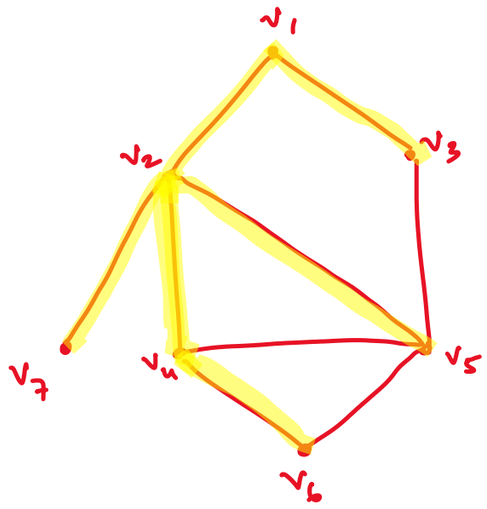
Topological sort



L34. Graph Traversal, SSSP

Tuesday, November 9, 2021 10:54 AM

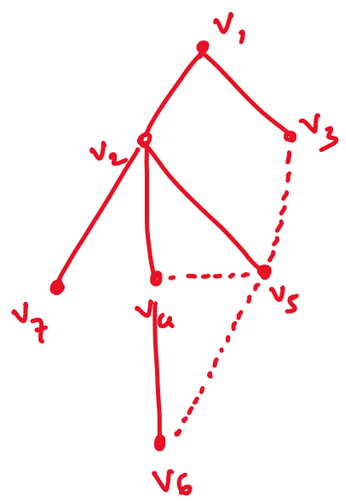
[undirected]



$v_4 \to v_2 \to v_5 \to v_6$

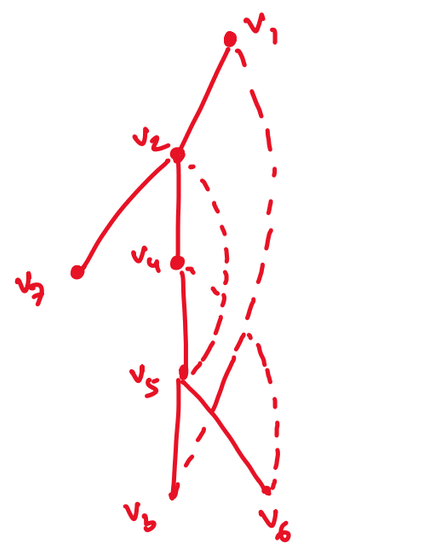
$v_4 \to v_2 \to v_6 \to v_5$

BFS (v_1)



non tree edges bet^n.. level i, i or level i, $i+1$

DFS (v_1)



non tree edge bet^n.. ancestor and descendant

✓

Qn: When is the BFS tree ≡ DFS tree?
Assume graph is connected and undirected.

Qn: Given BFS tree & DFS tree, is the graph unique?

DFS (s) without recursion:

1. $\forall$ u, u.explored = false
2. stack.push(s)
3. while stack is not empty:
   → 3.1. u = stack.pop().
   3.2. if u.explored = false
      3.2.1. u.explored = true
      3.2.2. $\forall$ v in out-nbr(u)

Q1. every vertex is set to explored exactly once.
if u.explored = false, then # ops = $O(1) + O(\deg u)$
⇒ 3.2.1 and 3.2.2 executed at most once for each vertex

Q.2 Total number of push ops $\leq 2m$
↑ number of edges.
Q.2 ⇒ number of times 3.1 is executed is $O(m)$

3.2.2.1 stack.push(v)

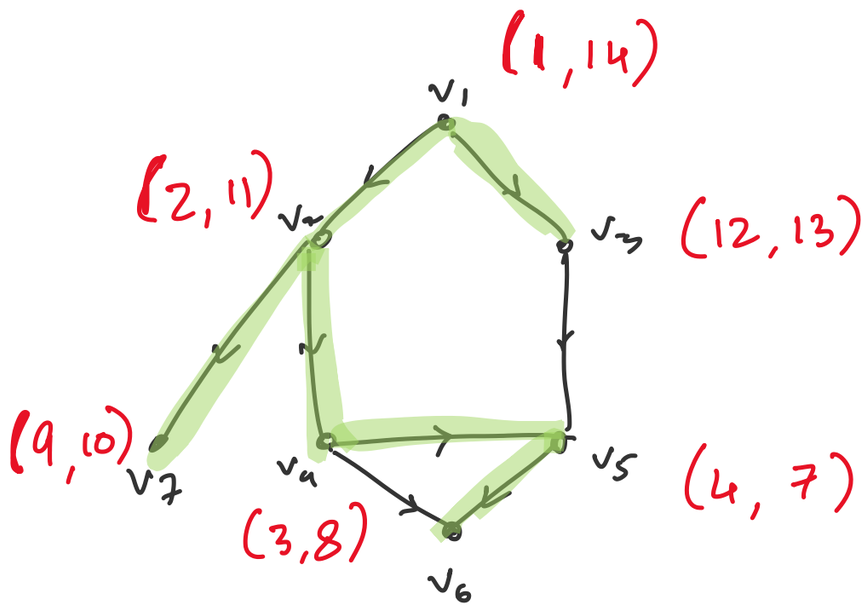
← Qm: do we set v.explored = true?

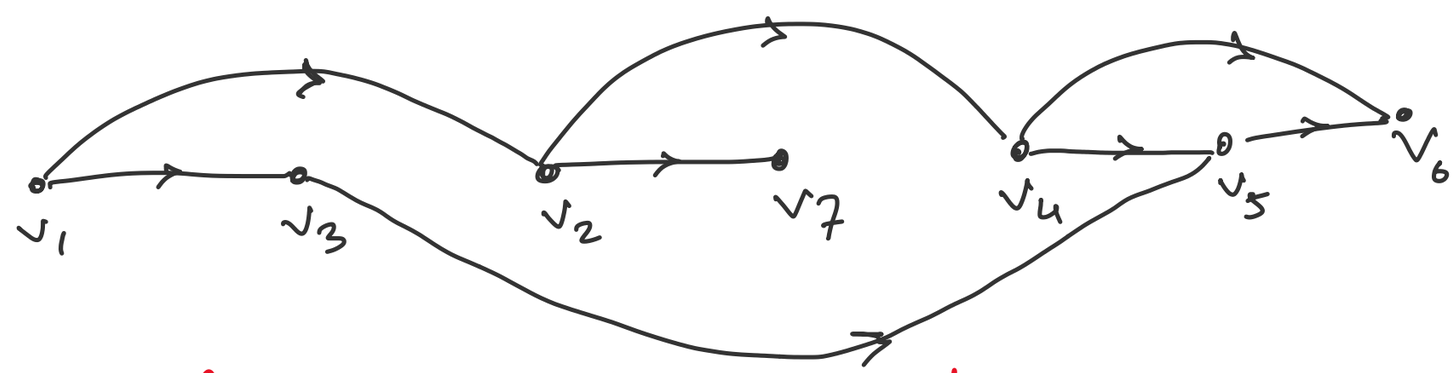
⇒ Total running time: $O(m+n)$.

Application of DFS: Topological ordering [Not in syllabus]

Given: directed graph $G=(V,E)$, no cycles in G

Output: ordering of the vertices from left to right s.t. all edges go from left to right.





Alg. for Topological sort:

1. compute start/finish times [$O(n+m)$ time]
2. Sort according to finish times [$O(n \log n)$ ↓ $O(n)$?]

Q1: How to modify pseudocode to maintain start/finish times?

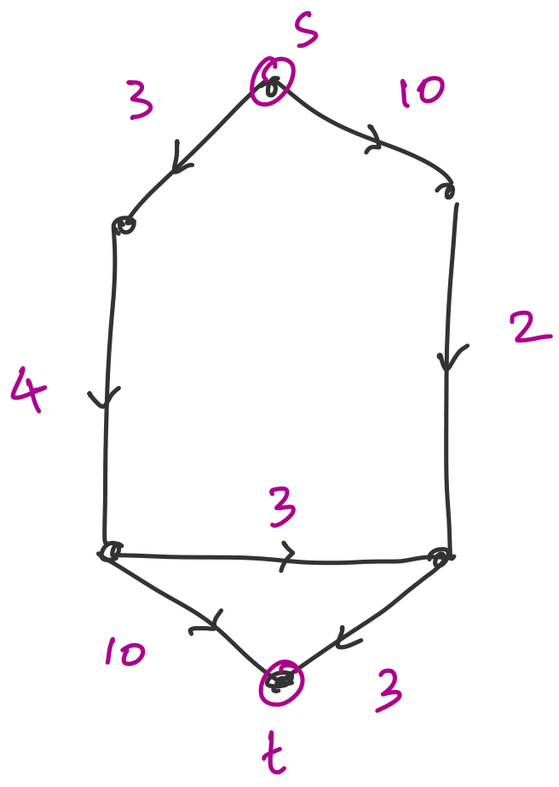
Q2: Why does this guarantee no edges from right to left?

BFS: gives shortest distance from source to all ... UNWEIGHTED graph.

vertices in an [illegible]

Graph with edge weights : $G = (V, E, \underline{wt : E \to \mathbb{R}})$

└ each edge e assigned a weight $wt(e)$



length of path in a wtd. graph

$(v_0 \; v_1 \; \dots \; v_k)$

$= \sum_{i=0}^{k-1} wt(v_i, v_{i+1})$

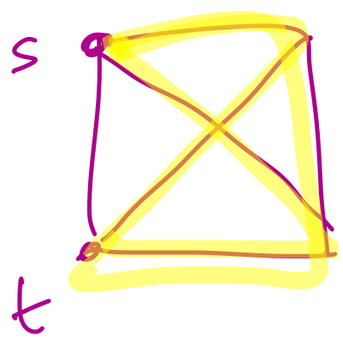
Today : $wt : E \to \mathbb{N}$

Single Source Shortest Path Problem (SSSP) :

Simpler problem :

Input : G (wtd.), s, t

output : length of shortest path from s to t.



Enumerate all paths ?

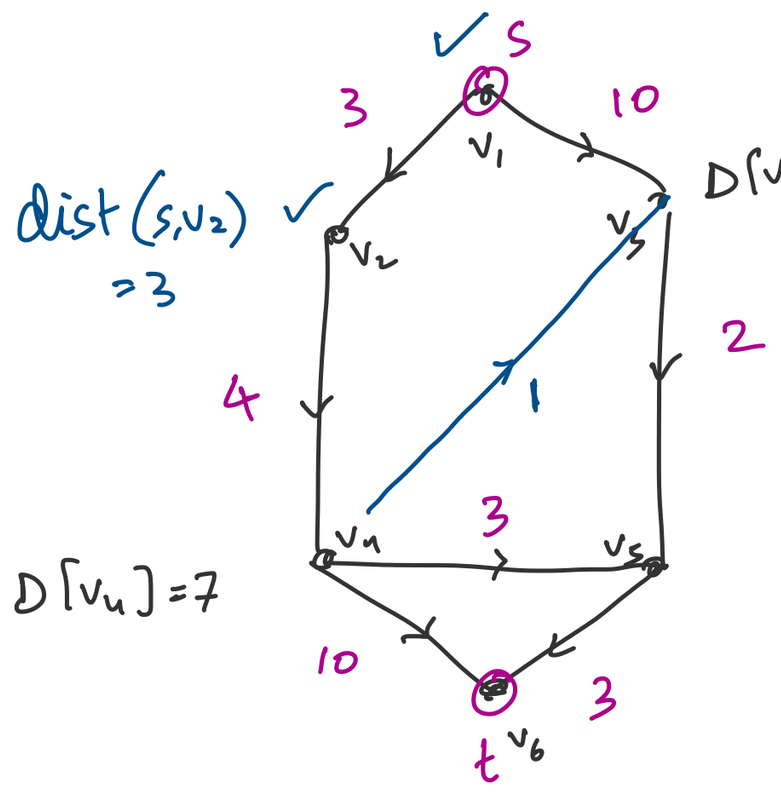
- Too many paths. :(

Qn : tight upper bd. on max. number of paths from s to t ?

Given : Graph $G = (V, E)$, positive edge wts,

$s \in V$

Compute $dist(s, v) \quad \forall \quad v \in V$.



$dist(s, v_2) = 3$ ✓

$D[v_3] = 10$

$D[v_4] = 7$

$\rightarrow dist(s, s) = 0$.

Obs 1: If w is out-nbr of s with min. edge wt, then $dist(s, w) = wt(s, w)$.

(Not true if we had negative edge wts)

$D[\cdot]$: upper bound on dist from s to vertex.

$D[v_3] = wt(s, v_3) \qquad dist(s, v_3) \leq D[v_3]$.

Claim: $S \subseteq V$ s.t. we know $dist(s, v)$ for all $v \in S$. We can find a vertex $w \in V \setminus S$ whose dist from s can be computed using $\{dist(s, v)\}_{v \in S}$ and adj. list. How?

L35. SSSP continued, MST

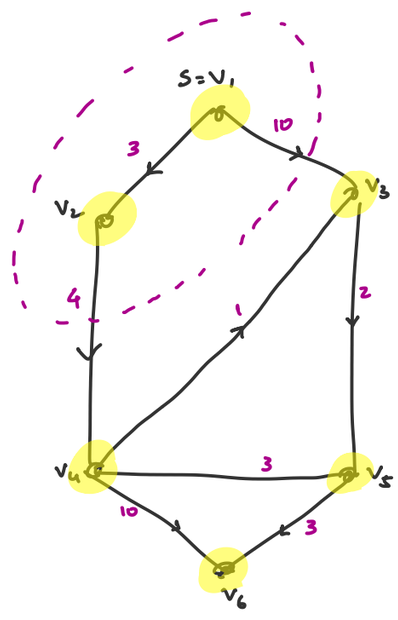
Tuesday, November 9, 2021 7:18 PM

$G = (V, E, wt : E \to \mathbb{R}^+)$

↑ positive real numbers

$|V| = n \qquad |E| = m$

source s



$dist(s, v) = \delta(v)$

$\delta(s) = 0.$

– we can compute dist. to immediate nbrs of s.

Dijkstra's Algorithm

Obs 1: w: nearest out-nbr of s.

Then $dist(s, w) = wt(s, w)$

Proof: Suppose $dist(s, w) < wt(s, w)$.

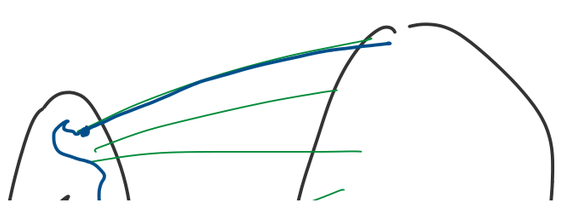
Then $\exists$ path $s = v_0 \; v_1 \ldots v_k = w$

s.t. $dist(s, w) = \underbrace{\sum_{i=0}^{k-1} wt(v_i, v_{i+1})}_{wt(s, v_1) + \sum_{i=1}^{k-1} wt(v_i, v_{i+1}) \geq 0} < wt(s, w)$

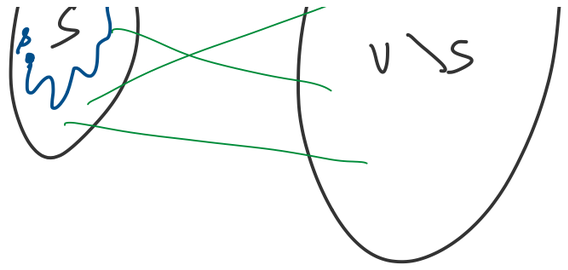
$wt(s, w) \leq wt(s, v_1)$

contradiction ▨

Suppose $\exists$ set $S \subseteq V$ s.t. $\forall w \in S$, we've computed $dist(s, w)$. How to compute $dist(s, v)$ for a new vertex $v \in V \setminus S$?

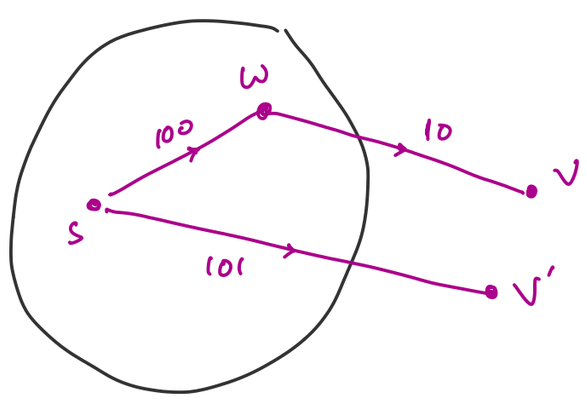


we've solved this for spl. case where $|S| = 1$ $(S = \{s\})$.



Attempt 1: pick v s.t. $\exists\ w \in S$, and edge (w, v) has min wt. among all edges going from S to $V \setminus S$.

Attempt 1 doesn't work.



S

Attempt 2: pick $v \in V \setminus S$ s.t. dist. from s to v using only vertices in S as intermediate vertices, is minimum.

$\text{dist}_S(s, v)$ ← distance of v from set S

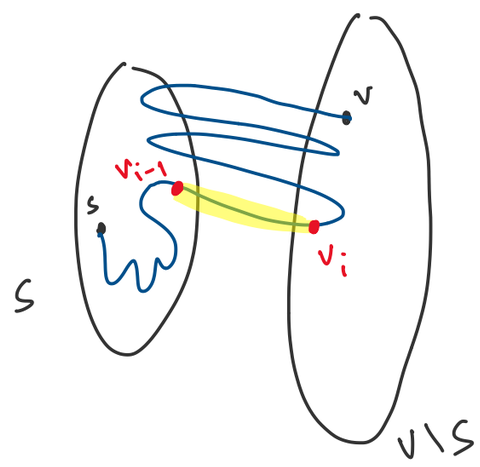
$$\min_{\substack{w \in S \\ v \in V \setminus S}} \text{dist}(s, w) + wt(w, v)$$

claim: $v \in V \setminus S$. If $\text{dist}_S(s, v) = \min_{z \in V \setminus S} \text{dist}_S(s, z)$

then $\text{dist}_S(s, v) = \text{dist}(s, v)$.

Proof: Suppose $\exists$ a shorter path from s to v.

$p = v_0 = s \quad v_1 \quad \dots \quad v_k = v$



$$\text{dist}(s, v) \geq \text{dist}(s, v_i)$$

$$\text{dist}(s, v_i) = \text{dist}(s, v_{i-1}) + wt(v_{i-1}, v_i)$$

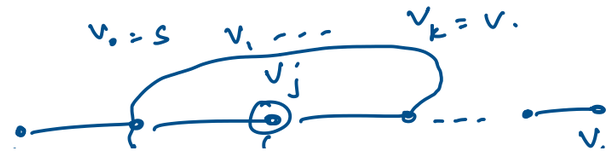
$\sum_{j=0}^{i-1} wt(v_j, v_{j+1})$ $\qquad$ $\sum_{j=0}^{i-2} wt(v_j, v_{j+1})$

Qn: can we always make this claim?

we had chosen vertex v s.t.

$$\text{dist}_S(s, v) = \min_{z \in V \setminus S} \text{dist}_S(s, z).$$

shortest path from s to v.



$$\text{dist}(s,v) \geq \text{dist}(s,v_i)$$
$$= \text{dist}_S(s,v_i)$$
$$\geq \text{dist}_S(s,v)$$

$\therefore$ contradiction. ∎

v_0 v_1) - - -

can we say that

$$\text{dist}(v_j, v_\ell) = \sum_{t=j}^{\ell-1} wt(v_t, v_{t+1})$$

Algorithm v. 0

1. $S = \{s\}$, $\delta[s] = 0$ $\quad \forall\, v \in V \setminus S$, $\delta(v) = \infty$
2. while $S \neq V$
   - 2.1. Find vertex $v \in V \setminus S$ s.t. $\text{dist}_S(s,v)$ is min. among all $v \in V \setminus S$
   - 2.2. $\delta[v] = \text{dist}_S(s,v)$, put v in S.

Obs. Suppose v is selected in ith iteration.
Then, only for the out-nbrs of v in $V \setminus S$,
the $\text{dist}_S(s,\cdot)$ might need to be updated.

Algorithm v1 using Obs

1. $S = \{s\}$, $\delta[s] = 0$, $\forall\, v \neq s$, $\delta[v] = \infty$.

$D[s] = 0$, $D[w] = \begin{cases} wt(s,w) & \text{if } w \in \text{out-nbr}(s) \\ \infty \end{cases}$

↑ stores $\text{dist}_S(s,w)$ at any pt.

2. while $S \neq V$
   - 2.1. Find $v \in V \setminus S$ s.t. $D[v]$ is min. among all $v \in V \setminus S$.] $O(n - |S|)$
   - 2.2. $\delta[v] = D[v]$, put v in S] $O(1)$
   - 2.3 $\forall\, w \in \text{out-nbr}(v)$, $D[w] = \min \begin{cases} D[w] \\ D[v] + wt(v,w) \end{cases}$] $O(\deg(v))$

Running time : $\underbrace{\sum_{i=0}^{n-1} n-i}_{} + \sum_{i=0}^{n-1} 1 + \underbrace{\sum_{v \in V} \deg(v)}_{}$

$O(n^2)$ $O(n)$ $O(m)$.

$O(n^2)$

If $m = O(n^2)$, then this is optimal.

What if $m = O(n)$

Qn: Can we use min-heap to store $D[\cdot]$?

ops in steps 2.1 and 2.3 ?

L36.

Thursday, November 11, 2021 11:00 AM

Algorithm v1 using Obs

1. $S = \{s\}$, $\delta[s] = 0$, $\forall\, v \neq s$, $\delta[v] = \infty$.

   $D[s] = 0$, $D[w] = \begin{cases} wt(s,w) & \text{if } w \in \text{out-nbr}(s) \\ \infty \end{cases}$

   ↑ stores $dist_S(s,w)$ at any pt.

2. while $S \neq V$

   2.1. Find $v \in V \setminus S$ s.t. $D[v]$ is min. among all $v \in V \setminus S$.] $O(n - |S|)$

   2.2. $\delta[v] = D[v]$, put v in S] $O(1)$

   2.3 $\forall\, w \in$ out-nbr(v), $D[w] = \min \begin{cases} D[w] \\ D[v] + wt(v,w) \end{cases}$] $O(\deg(v))$

Implementation using arrays: $O(n^2)$ running time

what if $m \ll n^2$?

- use min-heap to store D ?

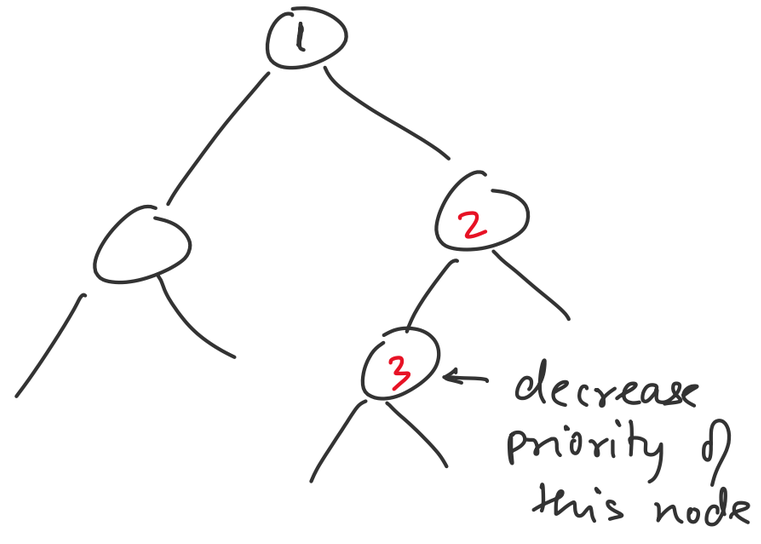
$O(\log n)$ for Step 2.1

$\rightarrow O(\log n \cdot \deg(v))$ for Step 2.3.

Total running time: $O(m \log n + n \log n)$

Fact: Fibonacci heaps: $O(1)$ update time
$O(\log n)$ delete-min

$\Rightarrow$ $O(m + n \log n)$ if Fibonacci heaps used.



update them

number of edges

number of vertices

bubble up.

Array based imp : $O(n^2)$

Binary heap imp. $O((m+n)\log n)$

Fibonacci heaps : $O(m + n \log n)$

↖ almost optimal since $O(m+n)$ ops. are required to read input.

Dijkstra's algorithm: Greedy algorithm.

Given : n size instance.

make some choice

n size $\longrightarrow (n-1)$ size instance.

Given : source,

store a shortest path from s to v,

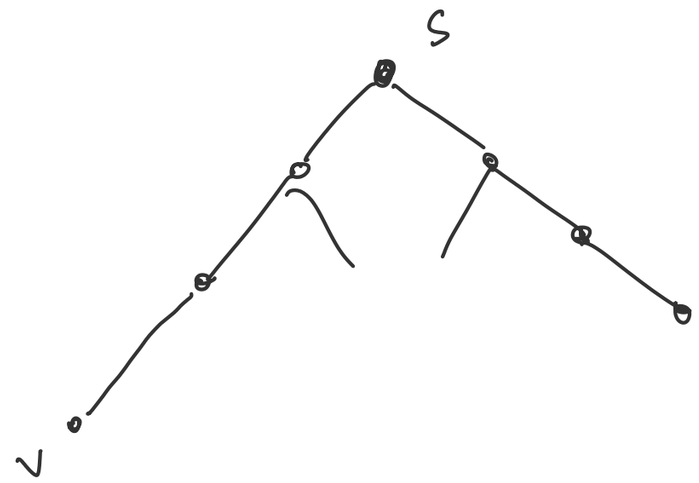
for all v.

Approach 1 :

For each vertex v, store the path separately.

$\Rightarrow$ $O(n^2)$ storage

SSSP tree :



$O(n)$ space

2nd Half Review:

- Binary Search Trees

  $\forall$ node v, $v.left.key \leq v.key \leq v.right.key$.

  Insert; Delete, Search : $O(height)$

- AVL trees : BSTs with $O(\log n)$ height

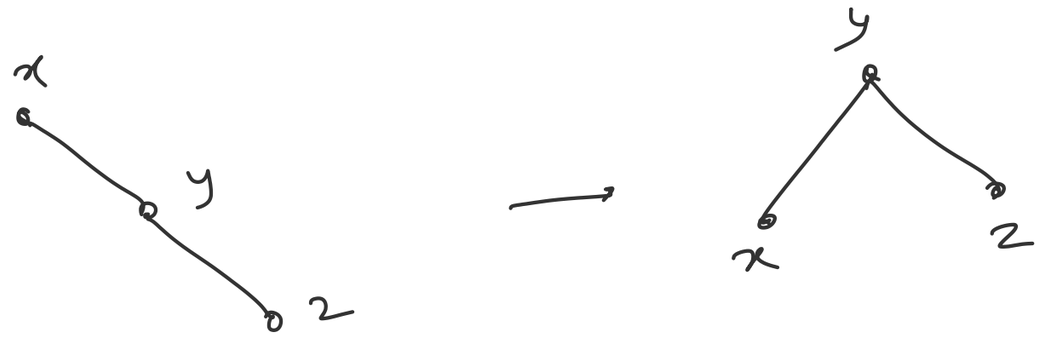
  AVL invariant
  $\Rightarrow$ ht. of tree : $O(\log n)$

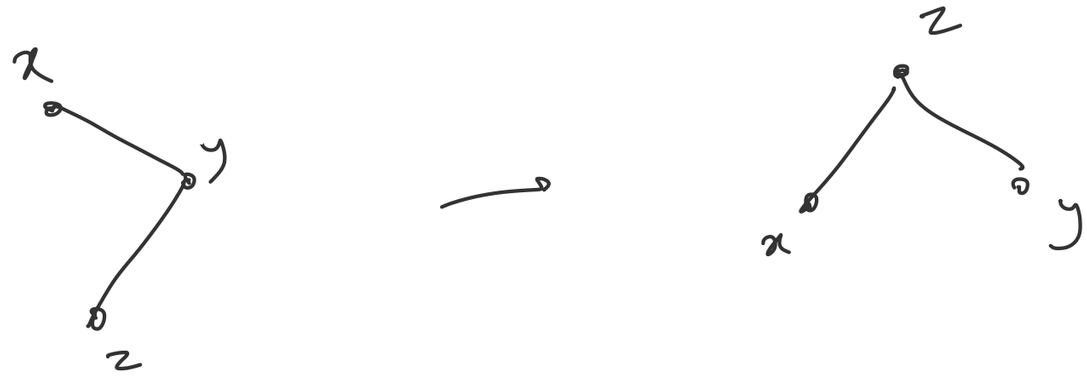
  Insert / Delete ?

  ↳ insert / delete acc. BST

  If invariant violated at some node,

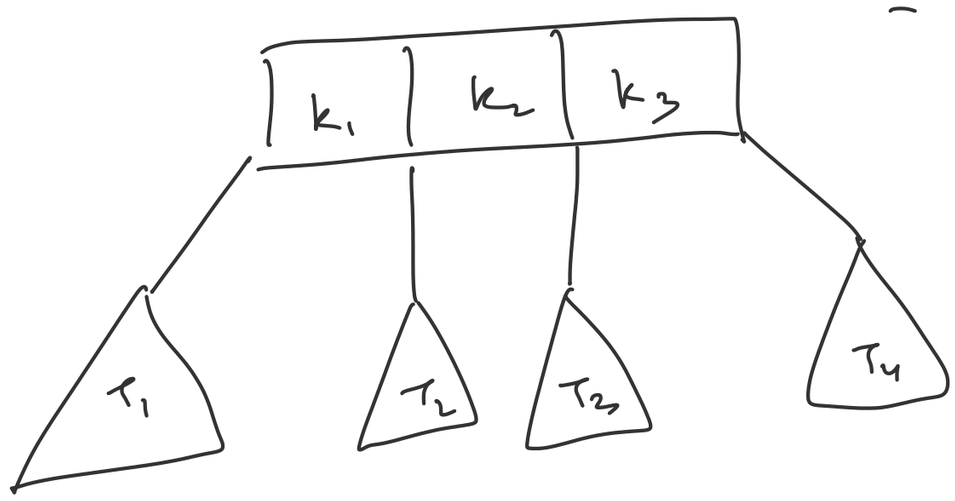
  then perform single / double rotations.





Application of BSTs :

- Range queries
  - 2D trees
  - range increments / decrements [Lab test 5/6]
  - m-B trees / 2-4 Trees



- r-1 keys at every node
  r children
- all leaf nodes at same depth

$$T_1 \le k_1 \le T_2 \le k_2 \le T_3 \le k_3 \le T_4.$$

Sorting : Insertion Sort
Sel. Sort
Quick Sort

Lower bounds for comp. based sorting

Integer sorting alg. : $O(k+n)$

$O(d(k+n))$: radix sort

Graphs :

BFS – computing distances in unwtd. graphs

DFS – Topological sorting.

SSSP

L 36