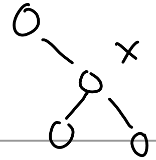
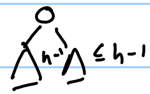
28.08.2018

Some properties of Binary trees

 × proper

Def^n: A proper binary tree is one in which each node has either 0 or 2 children.

Thm: If T is a (proper) binary tree of ht h and with n nodes

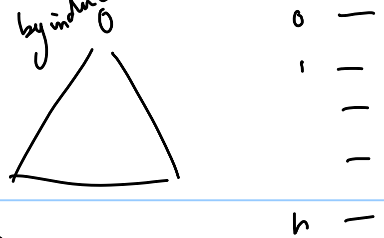
i) $h+1 \le \#\text{ext nodes} \le 2^h$ ✓ — 

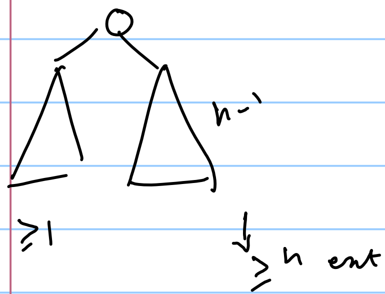
ii) $h \le \#\text{int nodes} \le 2^{h-1}$

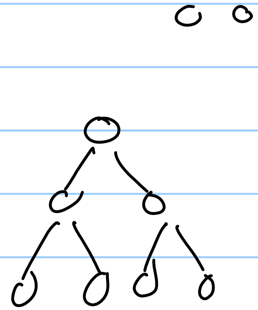
iii) $2h+1 \le \#\text{nodes} \le 2^{h+1}-1$

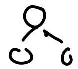
iv) $\log(n+1)-1 \le h \le (n-1)/2$

Proof of (i) by induction





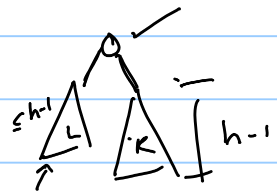




Proof of (ii) by induction

IH: $\forall$ trees of ht $k \leq h-1$ # int nodes $\geq k$

IS: Consider a tree of ht h



By IH: R has $\geq h-1$ internal nodes

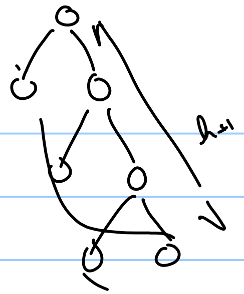
L has ≥ 0 internal nodes

because T has ht ≥ 1

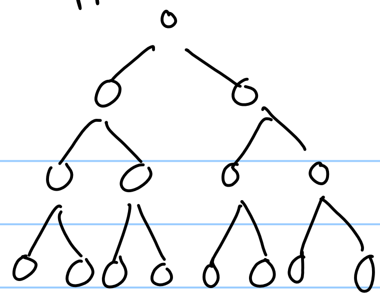
So, T has int nodes

$$= 1 + \#\mathrm{int}(R) + \#\mathrm{int}(L)$$

Lower bound tree

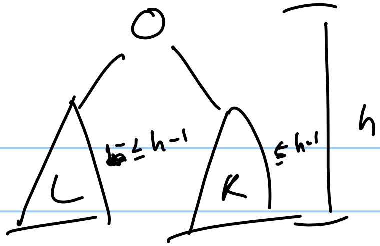


Upper bound



Prop: In a proper binary tree T, $\#\text{ ext} = \#\text{ int} + 1$.

Pf:



$\#\text{ ext}(L) = \#\text{ int}(L) + 1$

$\#\text{ ext}(R) = \#\text{ int}(R) + 1$

$\#\text{ ext}(T) = \#\text{ ext}(L) + \#\text{ ext}(R)$

$= \#\text{ int}(L) + \#\text{ int}(R) + 2$

$\#\text{ int}(T) = \#\text{ int}(L) + \#\text{ int}(R) + 1$

Euler Traversal.

30.08.2018

$A = 2^A$

"$\subseteq$" $\subseteq 2^A \times 2^A$

"$\leq$" $\subseteq \mathbb{Z} \times \mathbb{Z}$

isEqual(x, y)

Beyond isEqual( )

eg. set containment

$A \subseteq B$

$\{1, 2, 3\}$

$\{2, 3, 4\}$

$A \nsubseteq B$ and $B \nsubseteq A$

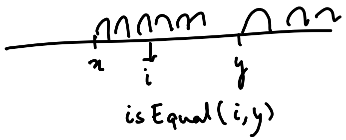
eg $\mathbb{Z}$, $\leq$

$x, y \in \mathbb{Z} \Rightarrow x \leq y$

or $y \leq x$.

- Given a set X, a relation R is a subset of $X \times X$
- A relation R is called
  - i) - reflexive if $(x, x) \in R \; \forall x \in X$
  - ii) - antisymmetric if $(x, y) \in R$ & $(y, x) \in R \Rightarrow x \equiv y$
  - iii) transitive if $(x, y) \in R, (y, z) \in R \Rightarrow (x, y) \in R$
- A relation R is called a partial order if it has (i), (ii) & (iii)
- A partial order is called a total order if $(x, y) \in R$ or $(y, x) \in R \; \forall x, y \in X$

Priority Queue



$\mathcal{X} = A \times \underline{T}$

where T is a totally ordered set.

A is ~~the~~ ground set of some ADT.

(a, k)

a → data

k → priority or key.

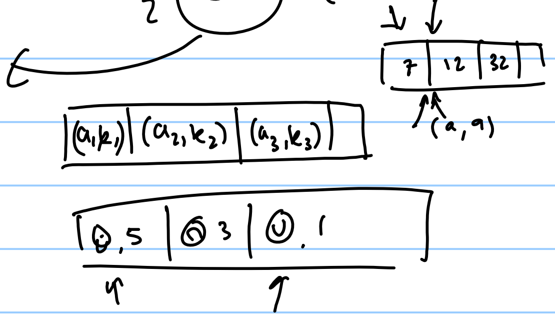
A set T is called a T.O. set if ∃ a relation $\leq$ which is a total order on T.

$PQ = \{(a_1, k_1) \cdots (a_m, k_m)\}$, $a \in A$, $k \in T$

Operations:

1. Insert(PQ, a, k)
2. RemoveMin(PQ) = removes (a_i, k_i) where $i = \underset{j}{\text{argmin}}\ k_j$
3. InspectMin(PQ)

$\{(①, 1), (②, 3), (③, 5)\}$



Implement

i) Insert (a_1, k_1) — $O(1)$

→ RemoveMin — $O(n)$

ii) Insert (a_1, k_1) — $O(n)$

RemoveMin — $O(1)$

Sort using a PQ

Given a set H of numbers

✓ – Insert all $|H|$ numbers in PQ as (i,i)

✓ – Remove Min $|H|$ times.

i) Insert $O(1)$
Remove $O(n)$

selection sort

Insert : $|H|$ steps

Remove : $\sum_{i=1}^{|H|} c\, i = c\,\frac{|H|(|H|+1)}{2}$

$\Theta(|H|^2)$

ii) Insert $O(n)$
Remove $O(1)$

insertion sort

Insert : $\sum i$

Remove : $|H|$

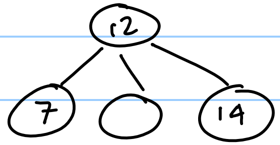
$\Theta(|H|^2)$

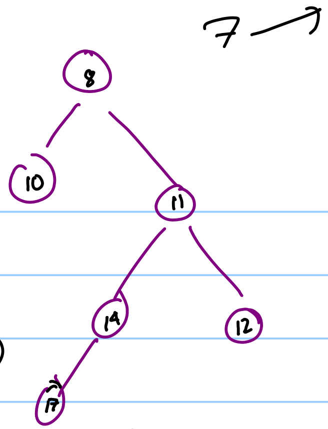
31 08.2018

Heap order property:

∀ nodes: Key at node ≤ keys at all children

Remove_min:
- Return root value and empty the root
- Move min of children to root and recursively fix the subtree whose root is now empty.





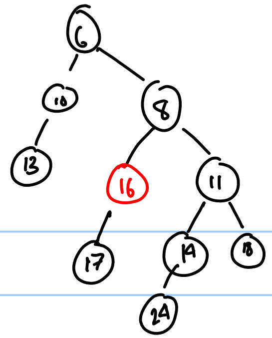
Time ≤ ht × max #children

$O(ht(T) \cdot max(T))$

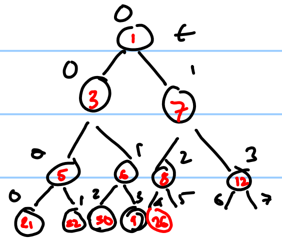
Insert

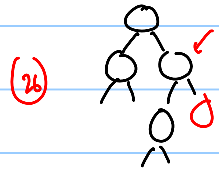
- Attach new key at leaf
- If new key ≥ parent key stop
  else
  swap with parent and ~~continue~~

Time $O(ht(T))$



Heap structure property





Remove min

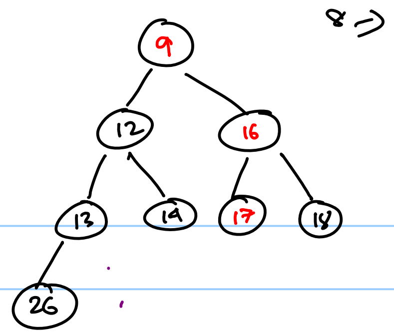
Remove "last" node on lowest level and put it in the empty root.

- "Bubble down"
  - While the key violates the HO property swap with min of children

Time = $O(ht(T) \; \# child(T))$

$$ht(T) = \lceil \log(\# nodes(T) + 1) \rceil - 1$$

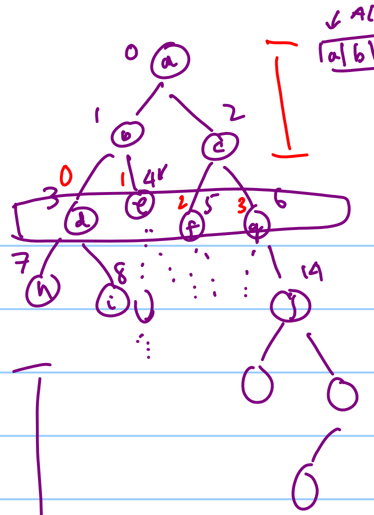
$$= \Theta(\log n) \qquad n = \# nodes(T)$$

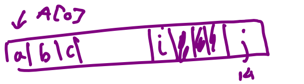


01.09.2018

Lchild (i) = 2i+1

Rchild (i) = 2i+2



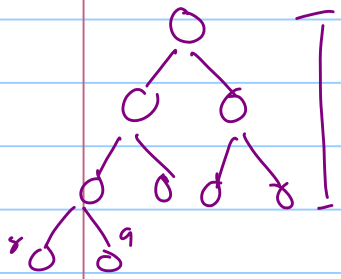


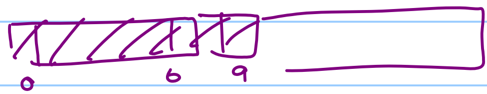
d

$$\sum_{i=0}^{d-1} 2^i$$

$$= 2^d - 1 + k + 1$$

$$= 2^d + k - 1$$





Heapsort

- Successively insert n items
- Successively delete n items

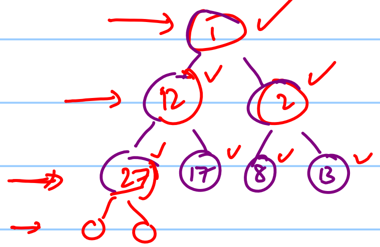
$$\sum_{i=1}^{n} \left((\log i) + 1 \right)$$

$$\geq \sum_{i=n/2}^{n} \log i$$

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

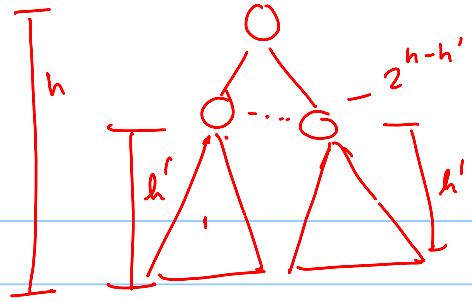
27 12 2 1 17 8 13

27	12	2	1	17	8	13	

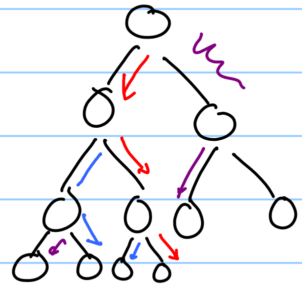


$c \quad \sum_{h'=1}^{h} h' \cdot 2^{h-h'}$

$= 2^h \boxed{\sum_{h'=1}^{h} h' \cdot 2^{-h'}} \le 2$



$\le 2^{h+1} = \Theta(n)$



04.09.2018

$f(n) \geq \frac{f(n)}{6}$ ✓

Post minor wrapup

1. How do computer scientists compare $f, g : \mathbb{N} \to \mathbb{R}_+$?

A. Three options

i) $f(n) \in \Theta(g(n))$ — same (why?)

f(n) is cheaper/faster than g(n) (why?)

ii) $f(n) \in o(g(n))$

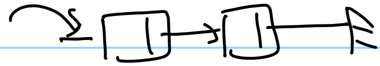
iii) $g(n) \in o(f(n))$

Conceptually the same.

2.

$T' = parent(Root(T))$

IntTree := IntTreeNode | EmptyTree



IntLinkedList := IntNode | EmptyList

Dictionary ADT : A-Dictionary

set of all dictionary on ADT A.

Operations : Given $x \in \mathcal{X}_A$, $D \in \mathcal{X}_{A\text{-Dictionary}}$

- Insert (x, D)
- Delete (x, D)
- Find (x, D)

$\mathcal{X}_A$ — unordered [isEqual exists]

$\mathcal{X}_A$ — ordered [$\leq$ exists and is a Total order]

Implementation 1: Log files

Each $x \in A$ has a key k and data d associated with it. The k comes from a ~~an~~ ADT ~~that~~ has isEqual() defined)

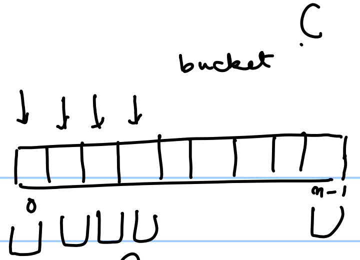
$$\boxed{(k_1, d_1), (k_2, d_2) \ldots (k_\ell, d_\ell)}$$

06.09.2018

Implementation 2: Hash maps.

Dictionary : Bucket Array

Storage : "Bucket" array



$f : X_A \to \{0, \ldots n-1\}$

Given $x \in X_A$: $f(x)$ is the index of the bucket used to store x

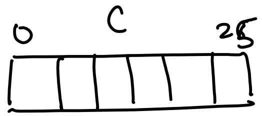
Insert(x, D) =

Bucket-Insert$(C[f(x)], x)$

If Bucket is implemented with lists then A-List-Insert$(C[f(x)], x)$

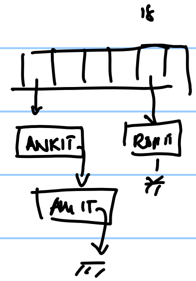
Find (x, D) : AList-isMember $(C[f(x)], x)$

Ex X_A = names in Roman script



$f(x)$ = letter position of first letter of x.

$f(\text{"ANKIT"}) = 0$, $f(\text{"ROHIT"}) = 18$



Insert (D, "ANKIT")

Insert (D, "ROHIT")

Insert (D, "AMIT")

Hash function

Computing a hash code $h: x_A \to S$

takes at least $\log S$

bit operations (why?)

x_A

Universal $\Downarrow$ Hash code (h)

S

Depends on size of array $\Downarrow$ Compression map (c)

n

$f = c \circ h$

time $= \frac{\text{Largest}(S)}{n}$

07.09.2018

Hash codes :

- Casting to an Integer : $(\text{int})\ x$ where x is of some type A.

- Summing components : $x = (x_1, x_2 .. x_n)$
  $h(x) = \sum x_i$

- Polynomials : $x = (x_1 .. x_n)$ n
  $h(x) = \sum_{i=1}^{n} x_i a^i$

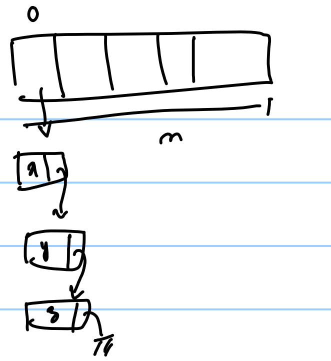
Compression maps

- $h(x) \bmod n$.
- $(a\, h(x) + b) \bmod n$

If x, y have the property that $f(x) = f(y)$ we say that a "collision" has occured.

$|X_A| \geq n$

Collision handling



1. Chaining: If $f(x) = f(y) = f(z) = 0$ then we make a linked list / array / hash map of these elements and store it at $C[0]$.

Load factor:

Given set size $= M$

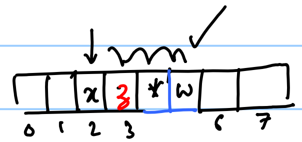
Array size $= m$

$\left\lceil \frac{M}{m} \right\rceil$ is the load factor

Load factor is a lower bound on worst case operation time for all dictionary operations.

Open addressing

1. Linear probing



x : f(x) = 2

y : f(y) = 4

z : f(z) = 2

w : f(w) = 2

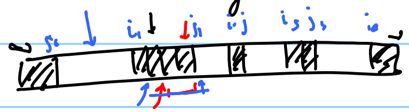
u : f(u) = 3

Find(u) . Go to f(u) and move rt till you find u or empty

yes   no

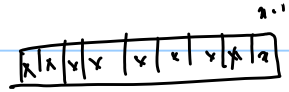
$(i_1, j_1), (i_2, j_2) \ldots$

(i_k, j_k)



1 Insert(x): Go to C(f(x)) and insert if it is empty or has a *
else go right till you find empty / * or
return to f(x) (Exception)

2. Delete (x): Find(x) and replace with * if found.



$f(x)=0$

Variation (a) Quadratic probing.

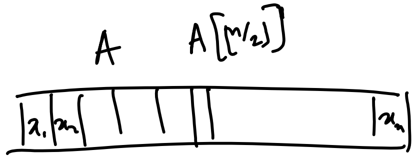
$f(x),\ f(x)+1,\ f(x)+4\ \ldots\ f(x)+i^2$

(b) Double hashing

PS (11 09)
In case the load factor gets very high we may need to rehash (move to larger array)

11.09.2018

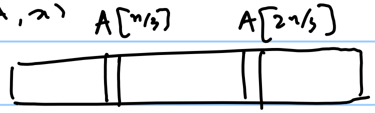
Ordered dictionaries



1. Look-up table

: Maintain key in sorted order $x_1 \le x_2 \le \ldots x_n$

Binary search : Find (A, x)



$\Theta(\log_2 n)$

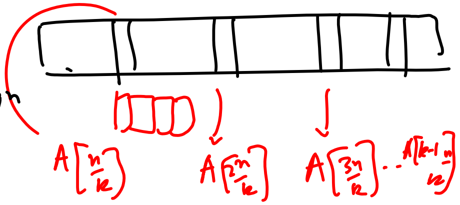
$O(\log_2 n)$

$n, \frac{n}{3}, \frac{n}{3^2} \ldots \frac{n}{3^i} \le 1$

$i = \log_3 n$

k-ary search takes : $(k-1)\log_k n = \frac{k-1}{\log_2 k} \cdot \boxed{\log_2 n}$

$\cancel{\log_2(k-1)}$. $\dfrac{\log_2 n}{\cancel{\log_2 k}} \le \dfrac{\log_2 2}{\log_3 3} \cdot \log n$



$\dfrac{\log k-1}{\log k}$ $\dfrac{\log 2}{\log 3}$ 1

1	3	7	12	14	16	18

Find (8, A)

→ Not Found (7, 12)

$x-$, $x+$

$x-$: greatest key < x which is in A

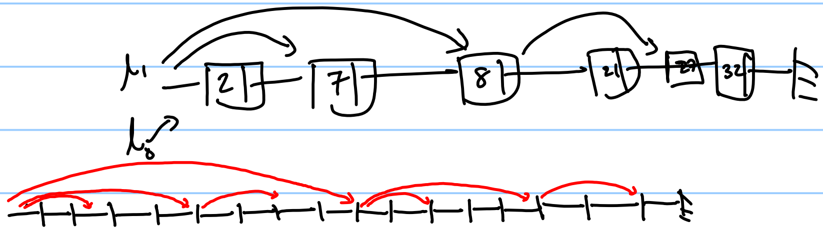
$x+$: least key > x which is in A.

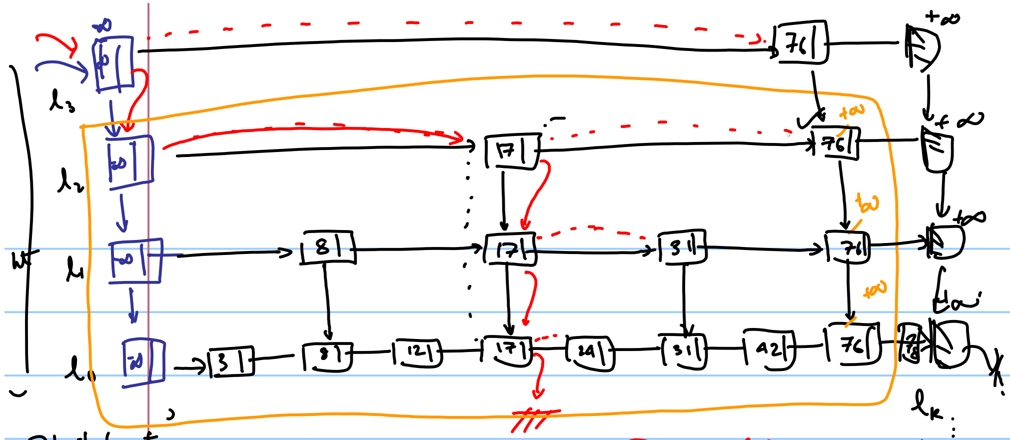
PS: Maintaining the sorted array is $\Theta(n)$ in case of insert & delete.

13.09.2018

Can we do binary search with linked lists (or some modification of linked lists) so that insert/delete becomes easier?

Find (D, 4) , Find (D, 10)





Skip list

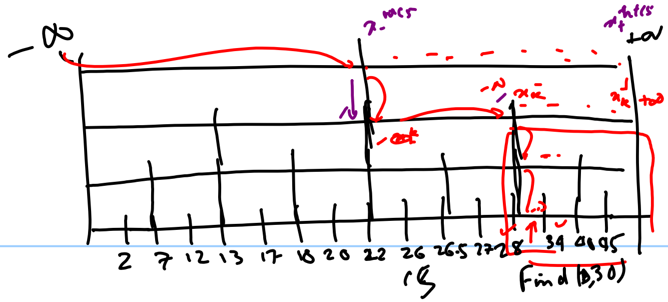
Find (D, 20)

Find (D, x)

→ LinSearch (l_k, x)

No (x^k_-, x^k_+)

linSearch (l_{k-1} (at x^k_-), x)



14.09.2018

How to insert in a skiplist.

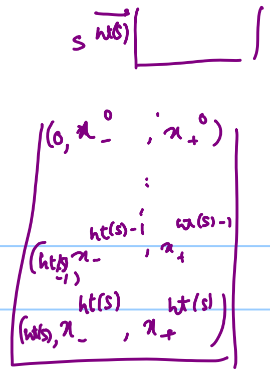
Insert (S, x) — i) Decide $ht(x) \geq 0$.

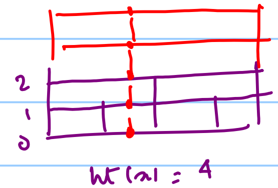
ii) Insert x with height $ht(x)$.

Where $ht(x) = \max \{k : x \in l_k\}$

Insert as follows:

- Find (s, x) "with a stack"
- $i = 0$
- Until Stack empty and $i \le ht(x)$
  - Pop (Stack) and get (k, x^k_-, x^k_+)
  - Insert x between x^k_- & x^k_+ in l_k
  - i++
- While $i \le ht(x)$
  - Make a new list l_i and insert x in it
  - i++





How to choose $ht(x)$: Randomly.

$p^{\log_{1/p}} = \frac{1}{(1/p)^{2\log_{1/p} n}}$

$= 1/n^2$

Pick $p \in (0,1)$

- x is in l_0 with probability 1
- For all $i > 0$,

  If $x \in l_{i-1}$ then put x in l_i w/prob p.

  $i \le 2\log_{1/p} n$

Height analysis: Given n elements $x_1, \ldots x_n$

$P[ht(x_i) \geq k] = p^k$

$P[\underbrace{\bigcup_{i=1}^{n} ht(x_i) \geq k}_{(\max_i ht(x_i)) \geq k}] \leq n p^k = n \cdot \frac{1}{n^2}$

if $k = 2\log_{1/p} n$

$\leq \frac{1}{n}$

$k = 2\log_{1/p} n = \log_{1/p} n + \omega(1)$

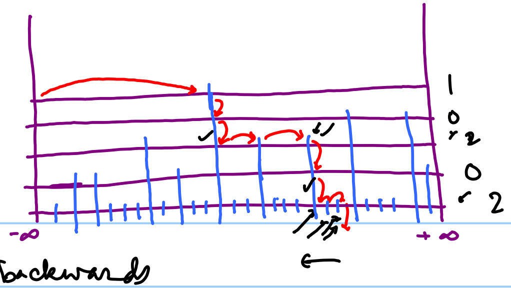
Given events A & B

$P(A \cup B) = P(A) + P(B) - P(A \cap B)$

$\leq P(A) + P(B)$

20.09.2018

Skiplist with parameter p. Ht is h.



✓ Distance travelled backwards before encountering 1st key promoted to level 1 is k w. pr. $(1-p)^{k-1}p$.

$$\left.\begin{aligned} E[\#\text{ of items seen in list } 0] &= \frac{1}{p} \\ E[\#\text{ of items seen in list } i] &= \frac{1}{p} \end{aligned}\right\} \Rightarrow$$

X : No of items in level 0 met before 1st item of level ≥ 1

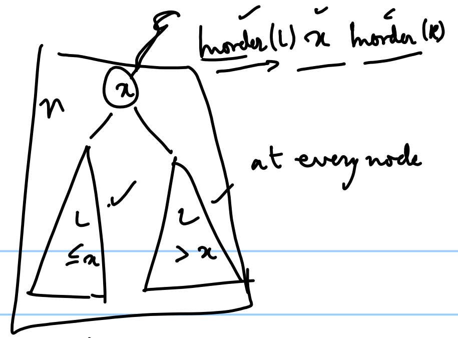
$$P[X = k] = (1-p)^{k-1} p$$

$$E[X] = \sum_{k=1}^{\infty} k (1-p)^{k-1} p = p \cdot \frac{1}{(1-(1-p))^2} = \frac{1}{p}$$

Time taken to find in expectation $= \frac{1}{p} \, ht(S)$

$$= \Theta\left(\frac{1}{p} \log_{1/p} n\right)$$

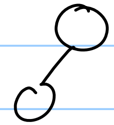
Binary Search Trees ADT



A-BinTree where

A is a totally ordered datatype with comparison operator ≤.

Def: An A-BinTree st at every node q, the keys in the left subtree of q are ≤ the key at q and the keys in the rt subtree of q are > " " ".



Find (19)

- Find(x, T)
  - If T is Empty say NO
  - If key(root(T)) = x

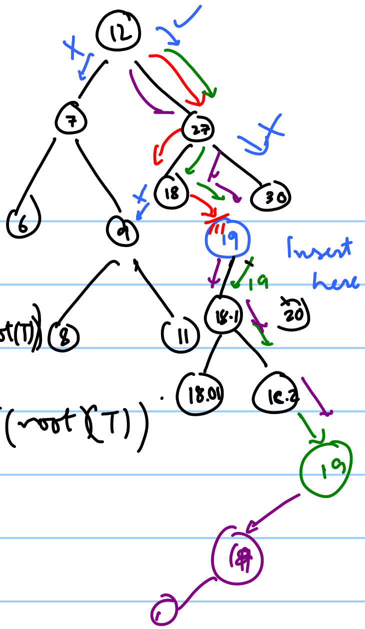
    say Yes

    else if x < key(root(T))

    find(x, leftSubT(root(T)))

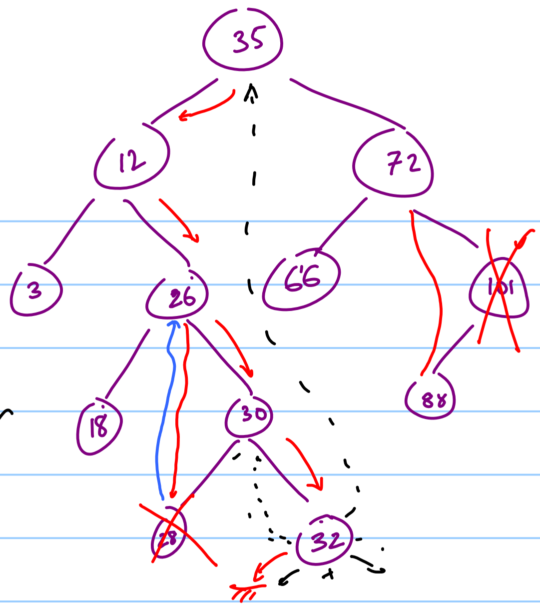
    else

    find(x, rtSubT(root(T)))



25.09.2018

Find (31)



$30^+ = 32$

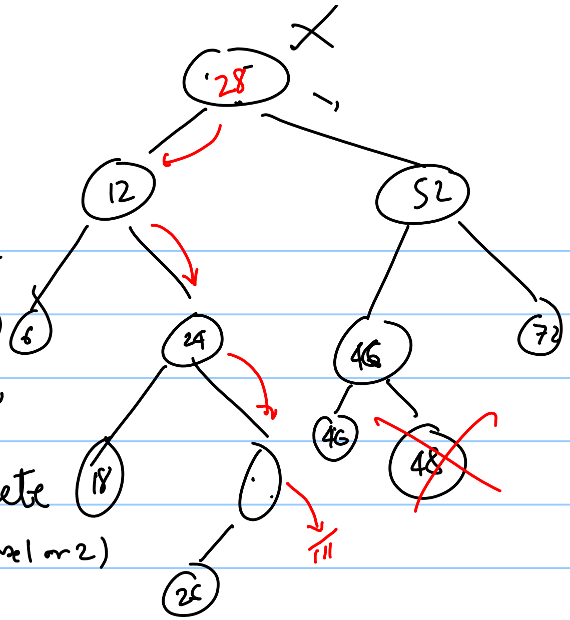
$30^- = 30$

Define: inorder successor
inorder predecessor
of each node

Case 1: Delete leaf

Case 2: Delete node w/ 1 child (attach subtree to parent of deleted node)

Case 3: Pop up inorder predecessor and then delete its node (Case 1 or 2)

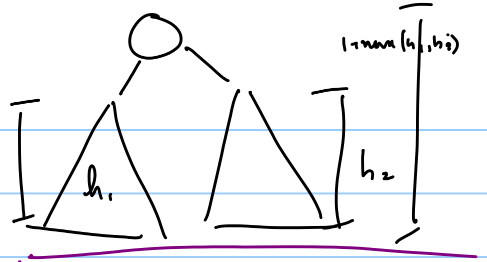


AVL Trees (Adelson, Velskii, Landis, 1957)

$|h_1 - h_2| \le 1$

A binary search tree is called an AVL tree if it is a BST and the difference in heights of subtrees is at most one at every node.

"structural invariant"



Find operation:

Exactly like BST

(no structural invariant is broken so nothing extra to do)

Height analysis:

$n(h)$: no of nodes in AVL tree of height h.

Convention

○ → ht 0

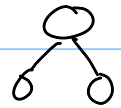
ht = max depth

$\bar{n}(h)$ = max no of nodes of AVL tree of ht h

$\underline{n}(h)$ = min " " "

$$\underline{n}(h) \le n(h) \le \bar{n}(h)$$

$n(0) = 1$



$n(1) = \{2,3\}$

$n(2) = \{4,5,6,7\}$

Claim: $\bar{n}(h)$ increases with h.

Pf: True since $\bar{n}(h) = 2^{h+1} - 1$

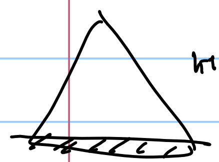
= # nodes in CBT of ht h.

Claim: $\underline{n}(h)$ increases with h.

Claim $\underline{n}(h)$ increases with h.

Pf: Consider a tree of ht h that has $\underline{n}(h)$ nodes.

- Remove all nodes of depth h.
- The resulting tree is an AVL tree of ht $h-1$



$\underline{n}(h) = \underline{n}(h-1) + 1$

$\underline{n}(h) = \underline{n}(h-1) + \underline{n}(h-2) + 1$

the no of nodes here $= n(h-1) \geq \underline{n}(h-1)$

$\Rightarrow \underline{n}(h) \geq n(h-1)$

Claim 3: $\underline{n}(h) = 1 + \underline{n}(h-1) + \underline{n}(h-2)$

Pf: Follows since $\underline{n}(h)$ increases with h

Soln 1: Claim 3 $\Rightarrow$ $\underline{n}(h) \geq 1 + 2\,\underline{n}(h-2)$

$$\geq 1 + 2\left(1 + 2\,\underline{n}(h-4)\right)$$

$$\text{RHS} = \sum_{i=0}^{h/2} 2^i .$$

$$\Rightarrow \underline{n}(h) \geq 2^{h/2+1} - 1$$

$\forall\, n(h)$

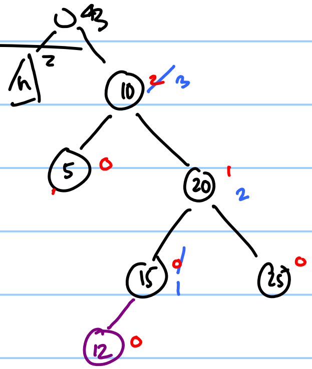
$$\Rightarrow \frac{h}{2} + 1 \leq \log_2\left(\underline{n}(h) + 1\right) \leq \log_2\left(n(h) + 1\right)$$

$$\Rightarrow h \leq \boxed{2}\left(\log_2\left(n(h) + 1\right) - 1\right) = O\left(\log_2 n(h)\right)$$

2 — becomes ϕ

Given an n : let $h(n)$ be the set of heights of AVL trees on n nodes ie $h(n) = \{h_1, h_2 .. h_k\}$

$\forall i: h_i \in O(\log_2 n)$



Insertion

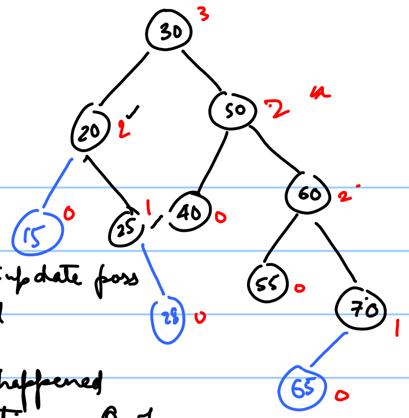
- Start with simple BST insertion
- Move up (using parent pointers) updating height as required stopping either when ht doesnt change or when AVL property is violated.

28.09.2018

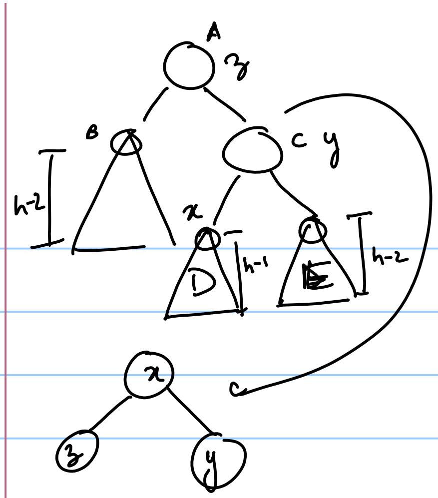
Stage 1 : BST insertion

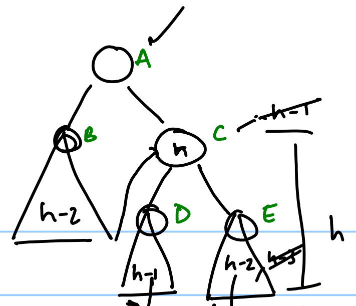
Stage 2 : Moving up from inserted node (a) update heights & (b) check for balance property violation.

- Stop if — no further height update poss
  — violation found



Stage 3 : If stop in stage 2 happened with no invariant violation ⇒ Quit.





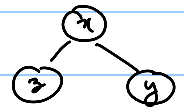
Since it caused ht(c) to increase

Can't be h-3 since violation would be observed at C before A.

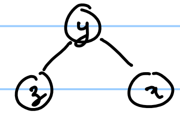
Restructuring procedure (trinode restructuring)

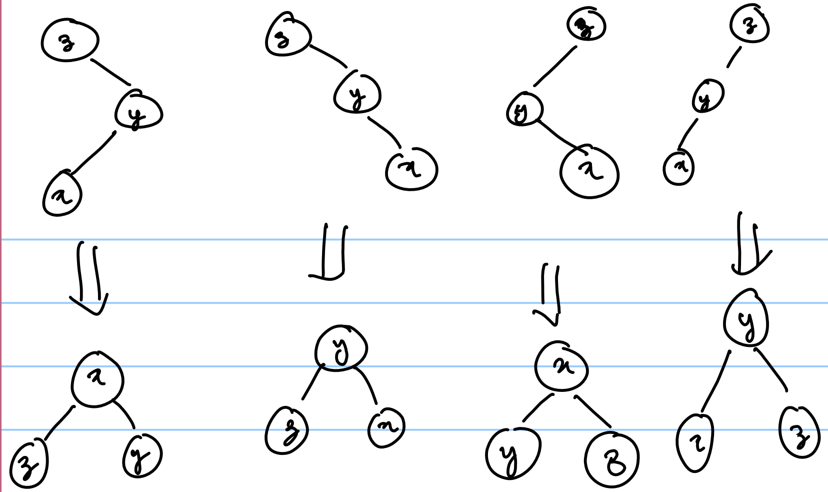
1. Identify 3 nodes
   - z : 1st node at which imbalance is observed
   - y : higher child of z in term of subtree height
   - x : higher child of y in terms of subtree height

If $x \leq y$: make x the root

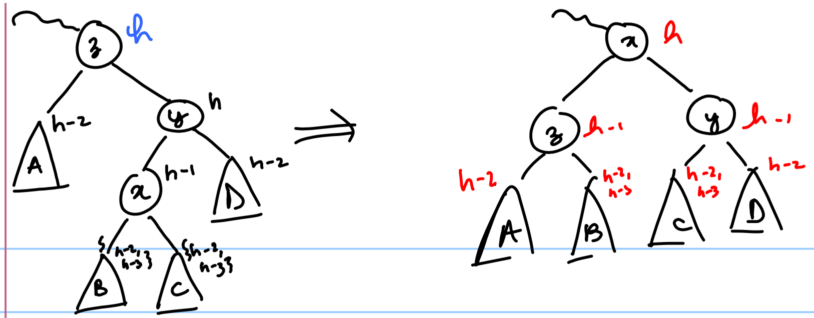


$x > y$: make y the root

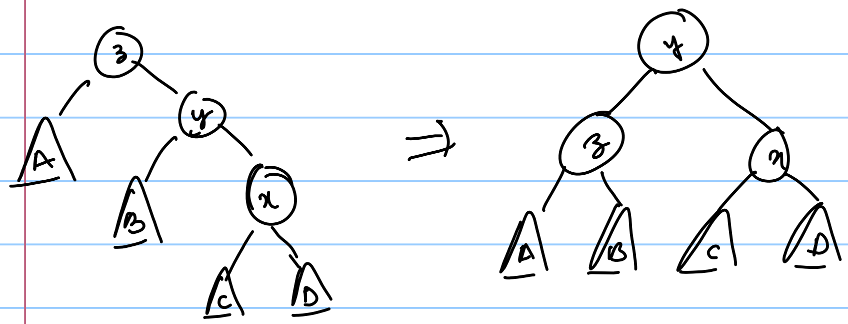




z
y
x
z
y
x
z
y
x
z
y
x
x
z
y
y
z
x
x
y
z
y
x
z



z
h
y
h
h-2
A
x
h-1
D
h-2
B
C
x
h
z
h-1
y
h-1
h-2
A
B
C
D



z
y
A
B
x
C
D
y
z
x
A
B
C
D