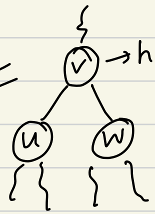
Lecture-13 Empirical Comparision Unbalanced BST vs AVL. Heaps and Priority Queues & Implementing Heaps

- **Last times:**

$h(\text{null}) = -1$

T is AVL if every node v is height-balanced.



$v \rightarrow$ height balanced if we have $|h(u) - h(w)| \leqslant 1$

height = max. distance of descendent leaf.

Consequence: If T is AVL tree with n nodes, then

$$h(T) = O(\log n)$$

$\hookrightarrow$ If T is AVL then find, add, remove can be performed in $O(\log n)$ time.

See lecture13 code for implementation

Updating: If T was AVL before add/remove then AVL property can be restored in time $O(\log n)$ after performing the operation.

$\hookrightarrow$ each node has to additionally store its height, we can update the height of ancestors on add/remove. $O(\log n)$

$\hookrightarrow$ check for imbalance.

$\hookrightarrow$ restore balance by restructuring. $O(\log n)$

Conclusion: We can implement sorted sets where all of my operations can be done in $O(\log n)$ time.

Queue specializes SSet where we can only add the largest/latest element and only remove the smallest which is oldest.

Previously, we have some version of BST → sorted sets
- Queues: Access FIFO

add, find, remove

Today: further restriction of sorted set functionality.

functionality: Still generalizing a queue.

FORMALISED DESCRIPTION IN HW-04

→ Priority Queues
- add/remove elements
  - specify a priority for element
  - remove highest priority element.

Priority of Sorted Set

- Define pair datatype that stores:
  - priority: k (= key) → numeric long in Java
  - element: v (= value)

  } = (k, v)

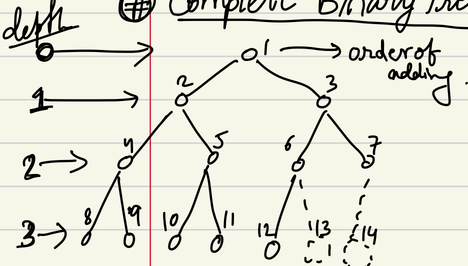
- Comparision of pairs

$$(k,v) < (k',v') \text{ if } k < k'$$

- SSet stores pairs
  - to add v with priority k, create pair (k, v) and add to SSet.

removal: removeMin()
- removeMin returns (k, v)
- PQ returns "v"!

Complete Binary Trees: CBT of depth d if:



1. All nodes at depth ≤ (d−2) have exactly 2 children.
2. At most one node at depth (d−1) with one child
3. If u & v at depth (d−1), u to left of v, & v has child, then u has 2 children

⇒ there is a unique location to add a new node to any complete binary tree.

⊙ Observations of CBT (depth d) has atleast

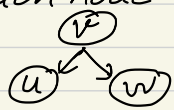
$$\underbrace{1+2+2^2+2^3+\cdots+2^{d-1}}_{(2^d-1)}+1$$

$$(2^d-1)+1 = 2^d \text{ nodes}$$

(round down)

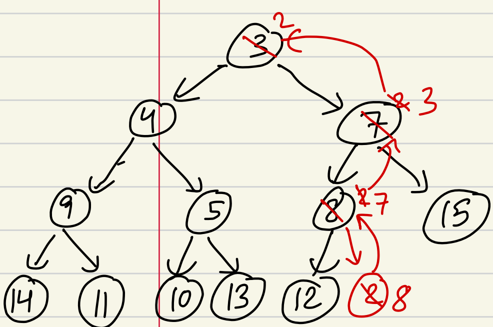
⇒ If T CBT with n nodes, then $\text{depth}(T) \leq \lfloor \log n \rfloor$

A Binary Heap is a CBT where each node stores a comparable element and if



[v has children u & w]

then $(v \leq u)$ and $(v \leq w)$.



⇒ Smallest element is always at the root.

→ How to add(2)?
- 2 must end up at the root
- other elements must be pushed down to unique space for new leaf.

Implement with bubble up...
- add to at unique new location
- Repeat: if new value < parent, swap, move to parent

WORK: Convince yourself that this maintains heap property.

→ How to remove min?
- copy value "e" last leaf (in this case 8) to root and remove leaf. then "bubble down": swap root value with smaller child, repeate this until root value < both children.