No.

Date :

DES221 : Data Structure and Algorithm Final.

Binary Tree

- root ⇒ node on top (indegree = 0)
- parent ⇒ outdegree ≠ 0
- child ⇒ indegree ≠ 0
- leaf ⇒ no children
- level ⇒ distance from root to node
- height ⇒ level + 1
- balance factor ⇒ $B = H_L - H_R$ (left - right)

  balanced. only when B is in {-1, 0, 1} only
- complete ⇒ if and only if every level is full
- nearly complete ⇒ if and only if every level excepts the last is full and last level are filled from left to right (consecutively)
- expression tree : parent = operator / leaf = operand.
  - create tree & operator stack (high over low)
  - if operand, push to tree stack

    if operator and such has priority than the top., push in operator stack

    if not, pop 2 from tree (first be right, second be left) and pop 1 from operator and built a tree

    repeat until operator stack is empty

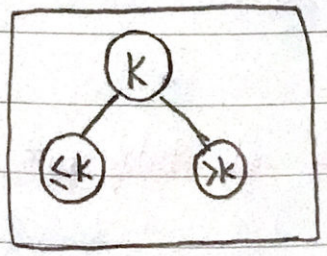
Traversal

- Recursion
  - Inorder : left root right
  - Preorder : root left right
  - Postorder : left right root
- first traversal
  - BFT (Breadth) : use queue (left enq first)

    (deq - print - enq)
  - DFT (Depth) : use stack (right push first)

    (pop - print - push)

Binary Search Tree



left subtree : less or equal

right subtree : greater.

- Insertion : find an appropriate spot.
- Deletion : if leaf ; delete immediately

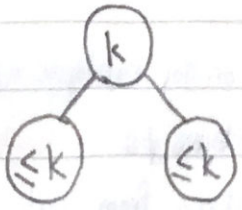
  if has 1 child ; connect child to appropriate position

  if has 2 children ; find maxleft then replace

  then reconnect.
- find smallest / largest : go all the way to left (small) / right (large)

L-11

- **Heap**
  - must be complete
  - parent must be greater than children
- Do it in array structure is easier / BFT Fashion



Parent of i is at index $\lfloor (i-1)/2 \rfloor$

left child of i is at index $(2*i)+1$

right child of i is at index $(2*i)+2$

- Insertion
  - insert at next position available
  - Reheap Up! if newNode is greater than its parent, swap. Repeat until Heap structure is received.
- Deletion
  - delete from the top
  - last element be replaced on the top.
  - Reheap Down! if replaced is less than its (maximal) child, swap. Repeat until heap structure is regained.

- **Graph**
  - degree : number of lines incident to it
  - outdegree : number of line "leaving" vertex
  - indegree : number of line "coming" vertex
  - path : sequence of vertex and edge start and end at different.
  - cycle : path that have same endpoint as startpoint.

data can be stored in 2 type

- adjacency matrix
- adjacency list.

- Traversing a graph
  - DFT : use stack.
  - BFT : use queue

} solution is not unique b/c order of data and starting point.

- Spanning Tree

subtree underlying in a graph, spanning tree are not unique. must contain no cycle.

6-11

No.

Date :

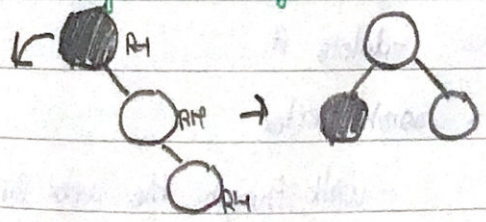
- Minimum spanning tree.
  - Prim-Jarnik's
    - start anywhere
    - pick next, by using minimum edge that adjacent to previous vertex.
  - Kruskal's
    - order all edges by weight from min to max
    - choose minimum and check if it creates cycle or not, if yes, skip that.
    - repeat until get n-1 edges for n vertex graph.

- Shortest path
  - Dijkstra's Algorithm
    - compute cumulative distance from starting point
      collect data as [from where, distance from starting point]
    - if finish, backtrack using the collected data.

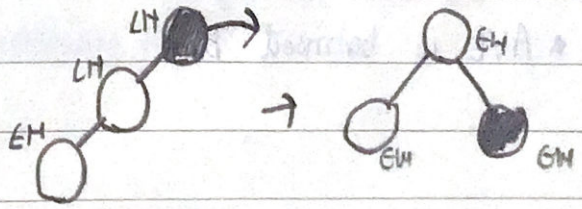
- AVL -
  - height-balanced BST
  - improving efficient by rotation until tree is balanced.
  - must be BST and balanced factor of all node is between [-1, 0, 1]

Terminology RH → B < 0 , LH → B > 0 , EH → B = 0

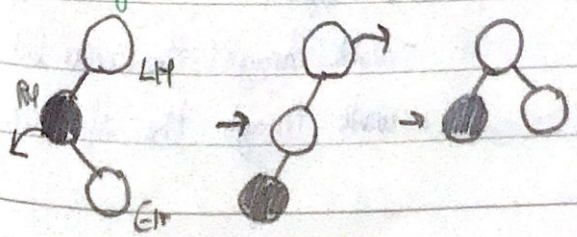
- do the normal operation but if becomes unbalanced, rotate appropriately.
- Right of Right (\\)



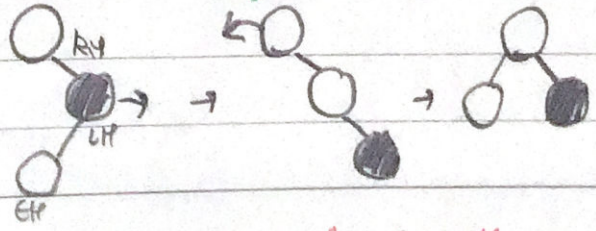
- Left of Left (//)



- Right of Left (<)



- Left of Right (>)



- Sometimes, deletion won't fall down in any of these, just ignore it will be fixed by itself

• Big-O Comparison •

• BST

	Balanced	Unbalanced
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$
Search	$O(\log n)$	$O(n)$
Find Extrems	$O(\log n)$	$O(n)$

• Heap

Insert + Reheap Up	$O(\log n)$
Delete + Reheap Down	$O(\log n)$

• Graph

	# of Vertex	# of Edge
Insert	$O(1)$	$O(n)$
Delete	$O(n^2)$	$O(n^2)$
Search	$O(n)$	$O(n^2)$

• AVL

Insert	$O(\log n)$
Delete	$O(\log n)$
Search	$O(\log n)$
Find Extrems	$O(\log n)$

* AVL is balanced BST

* insert vertex at the end of list
* insert edge
  - find vertex
  - insert at the last
* delete vertex
  - delete vertex list
  - delete the entry of the vertex
* delete edge
  - find edge
  - delete it
* search vertex
  - walk through the vertex list
* search edge
  - walk through the vertex list
  - walk through the edge list.