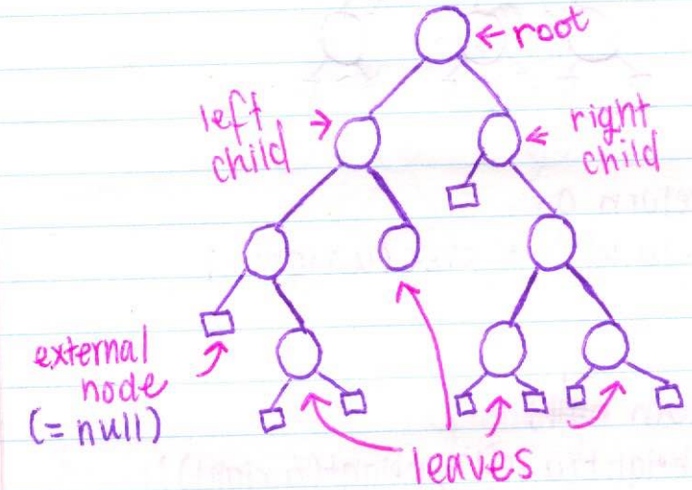
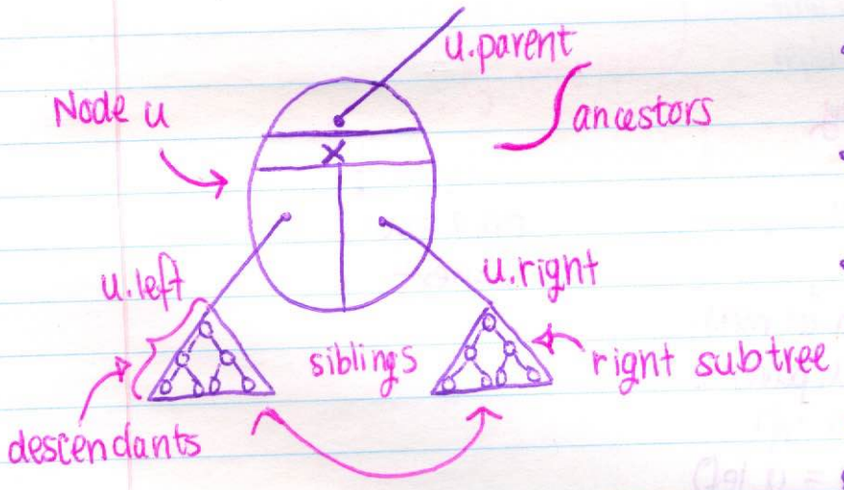
Nov 3, 2017
COMP 2402

Binary Search Trees

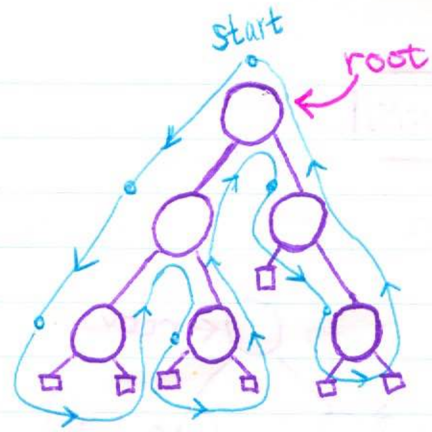
Binary Trees



Binary tree: { null, or a node w/ exactly two children which are binary trees



- depth(u): distance from u to the root
- level(i): the set of nodes with depth = i
- height(r): maximum depth in subtree rooted @ r
- size(r): # of nodes in subtree rooted @ r




```
size(u);
    if (u==null) return 0;
    return 1 + size(u.left) + size(u.right);

height(u);
    if (u== null) return ~~null~~ -1;
    return 1+ max (height(u.left), height(u.right));
```

iterative

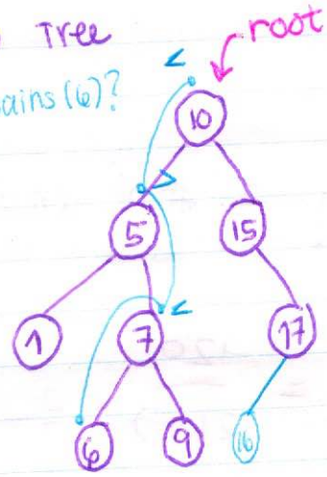
```
depth(u);
    d = 0;
    while(u != root) {
    u = u.parent
    d ++
    }
    return d;
```

recursive

```
depth(u);
    if (u == root) return 0;
    return 1+ depth (u.parent)
```

```
traverse(●)   <- empty
    prev, next = null
    u = root
    while (Not back at root)
        if( prev = u.parent)
            next = u.left
        else if (prev = u.lef)
            next = u.right
        else
            next = u.parent
                ⋮
                                ⋮
                        prev = u
                        u=next
```

→ Binary Search Tree



left.value < this.value ≤ right.value

add(16)

```
add(x):
  y = findLast(x)
  if (y=x) return false
  if (x<y) y.left = x
  else y.right = x
```

eg: remove(9)
```
remove(u): // case 1, u was
  find(u)    a leaf
  cut from tree
```

eg: remove(17)
```
remove(u): // case 2, u had
  find(u)    one child
  replace u with only child
```

eg: remove(5)
```
remove(u): // case 2, u has
  find(u)    two children
  replace u with next largest
    value in u's subtree, w
                  ^
                 right
  w is case 1 or case 2
```

Hilroy