Computer Science 112
Data Structures

Lecture 11:
Trees and Binary Search Trees

Midterm Exam

- **This Sunday, March 1**
- **3 – 4:20 pm**
- **Know which recitation you are registered for**
- **Rooms:**

Section 20 (Tues, 3:35 pm):	**Tillet 257**
Section 21 (Tues, 5:15 pm):	**Engineering B120**
Section 22 (Tues, 6:65 pm):	**Engineering B120**

Midterm Exam

- **Closed book**
- **Closed notes**
- **No electronics**
  - **EG: no phone, no calculator, no smart watch**
- **Do bring photo ID with legible picture**

Midterm Exam

- **Topics: Everything in lecture and assigned reading, through Binary Search**
  - **Will <u>not</u> cover Binary Search Trees**

Review: Binary Search

```
??   ??   ??   ??   ??   ??   ??
0    1    2    3    4    5    6
Search for 15
 left   right   mid
  0       6
```

Binary Search

??	??	??	29	??	??	??
0	1	2	3	4	5	6

Search for 15

left	right	mid
0	6	3

Binary Search

??	??	??	29	??	??	??
0	1	2	3	4	5	6

Search for 15

left	right	mid
0	6	3
0	2	

Binary Search

??	13	??	29	??	??	??
0	1	2	3	4	5	6

Search for 15

left	right	mid
0	6	3
0	2	1

Binary Search

??	13	??	29	??	??	??
0	1	2	3	4	5	6

Search for 15

left	right	mid
0	6	3
0	2	1
2	2	

Binary Search

??	13	15	29	??	??	??
0	1	2	3	4	5	6

Search for 15

left	right	mid
0	6	3
0	2	1
2	2	2

Review: Binary Search

??	??	??	??	??	??	??
0	1	2	3	4	5	6

Search for 10

left	right	mid
0	6	

Review: Binary Search

??	??	??	29	??	??	??
0	1	2	3	4	5	6

Search for 10

left	right	mid
0	6	3

Review: Binary Search

??	??	??	29	??	??	??
0	1	2	3	4	5	6

Search for 10

left	right	mid
0	6	3
0	2	

Review: Binary Search

??	13	??	29	??	??	??
0	1	2	3	4	5	6

Search for 10

left	right	mid
0	6	3
0	2	1

Review: Binary Search

??	13	??	29	??	??	??
0	1	2	3	4	5	6

Search for 10

left	right	mid
0	6	3
0	2	1
0	0	

Review: Binary Search

12	13	??	29	??	??	??
0	1	2	3	4	5	6

Search for 10

left	right	mid
0	6	3
0	2	1
0	0	0

Review: Binary Search

	12	13	??	29	??	??	??
	0	1	2	3	4	5	6

Search for 10

left	right	mid
0	6	3
0	2	1
0	0	0
0	-1	

See BinarySearch.java

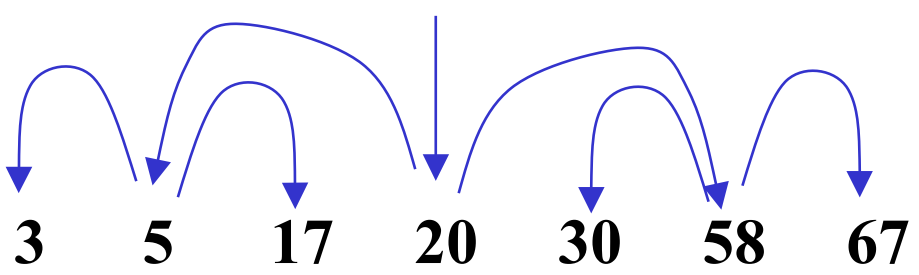
See RecursiveBinSearch.java

How Many Comparisons?

- **How many elements of the array do we have to compare with the target?**
  - **For each element we look at, size of remaining region is cut in half**
  - **When remaining region is 1 element, we are done**
  - **O(log(n))**

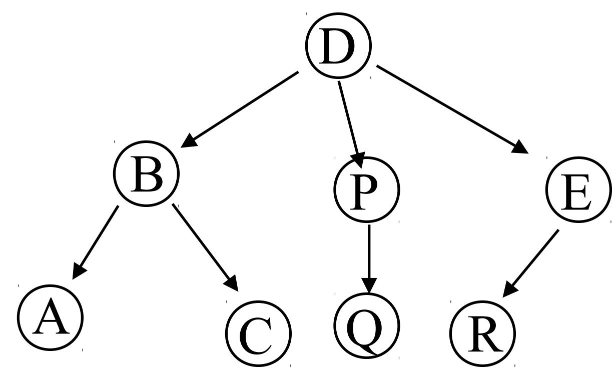
Binary Search Trees

- Why can’t we do binary search on a linked list?
  - can’t jump to middle
- Lets keep a pointer to the middle!
- Then what?



New: Tree Terminology

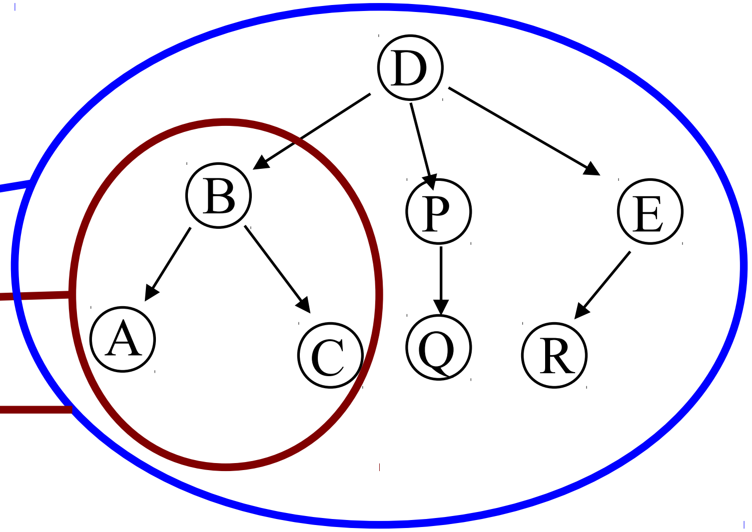
- Nodes (vertices) and arcs (edges)
- Relationships:
  - Parent and Child
    - E is a child of D
    - D is the parent of E



Trees

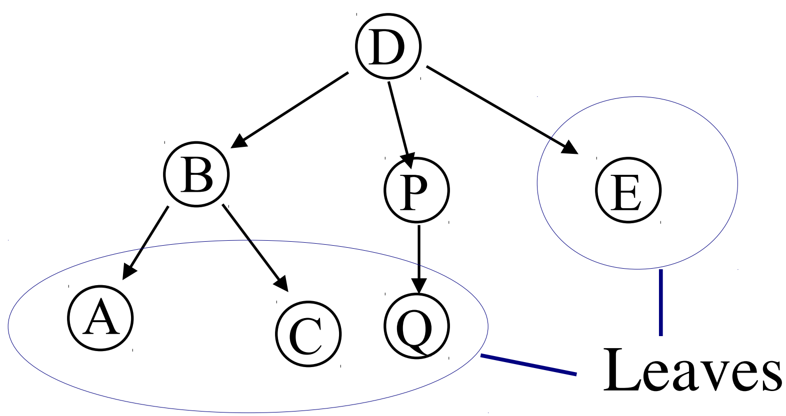
- Nodes and arcs (edges)
- Relationships:
  - Parent and Child
    - E is a child of D
    - D is the parent of E
  - Root and Subtree
    - D is the root of this tree
    - This

      is a subtree of this.
    - The root of the subtree is B



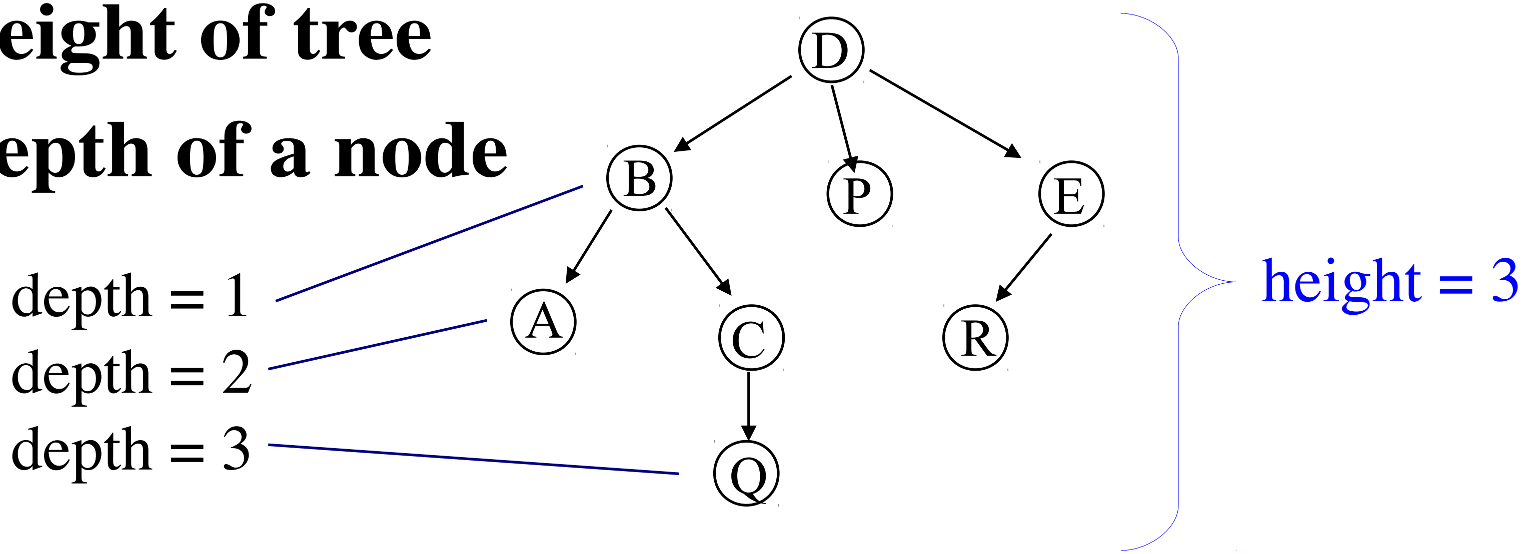
Trees

- **Root has no parents**
- **All nodes except the root have a single parent**
- **Leaf node has no children**
- **There is exactly one path from root to any node**



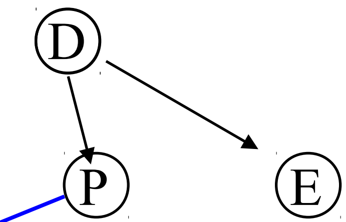
Trees

- **Height of tree**
- **Depth of a node**



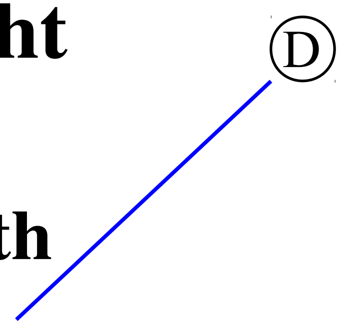
Trees

- **What is the height of this tree?**
  - **What is the depth of this node?**



Trees

- **What is the height of this tree?**
  - **What is the depth of this node?**



Trees

- **What is the height of the empty tree?**

Null

Binary tree

- **each node has at most 2 subtrees**
  - **left and right subtree**

  root of left (right) subtree is called the left (right) child

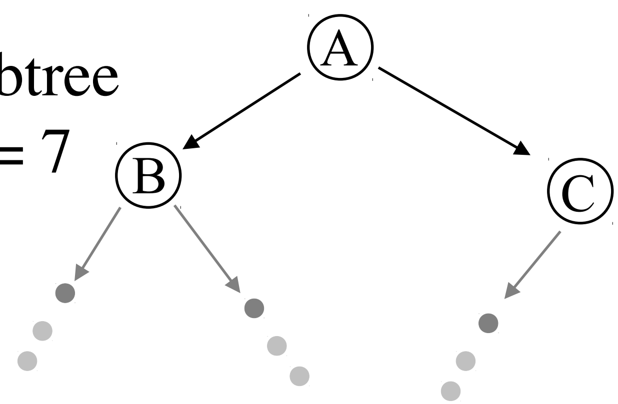
Recursive Data Structures

- **Recursive definition of a binary tree**
  - **empty (i.e. null)**
  - **not empty**
    - **data at the root**
    - **a left subtree, which is a binary tree**
    - **a right subtree, which is a binary tree**

height

What is the height
of the whole tree?

Left subtree
height = 7



Right
subtree
height = 5

Recursive functions
height

```
height(tree):
   if (tree = = null) return  -1
     else  return  1 + max   (  height  (tree.lst),
                                height   (tree.rst))
```

Recursive functions

- Common form of function on a tree is recursive

f(tree):

if (tree = = null) return

else return (data, f(tree.lst), f(tree.rst))

Where is a value and

is a function

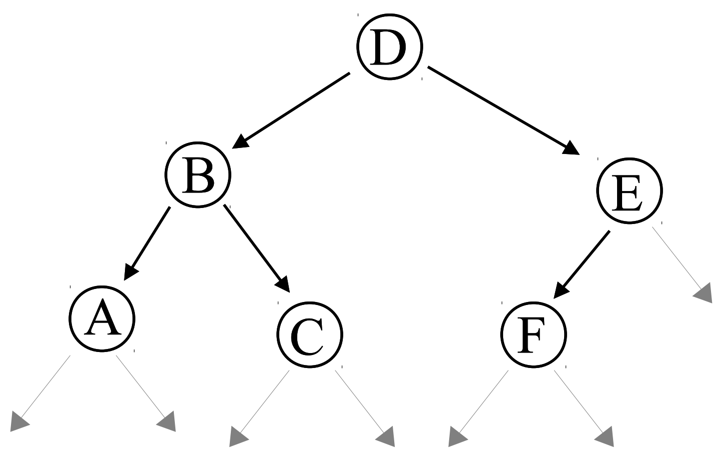
Recursive functions
height

```
height(tree):
   if (tree = = null) return -1
     else  return  1 + max  (  height  (tree.lst),
                               height   (tree.rst))
```

Recursive functions nodeCount

```
nodeCount(tree):
   if (tree = = null) return 0
     else  return 1 + sum  (nodeCount(tree.lst),
                            nodeCount(tree.rst))
```

Recursive functions nodeCount



Recursive functions
Sum

```
sum(tree):
    if (tree = = null) return
    else return      (tree.data,
                       sum( tree.lst ),
                       sum( tree.rst ))
```

Recursive functions
has0

```
has0(tree):
    if (tree = = null) return ◯
    else return □ (tree.data,
                   has0( tree.lst ),
                   has0( tree.rst ))
```

Recursive functions
has0

```
has0(tree):
    if (tree = = null) return false
    else return  or  (tree.data = = 0,
                      has0( tree.lst ),
                      has0( tree.rst ))
```

Binary Search Tree

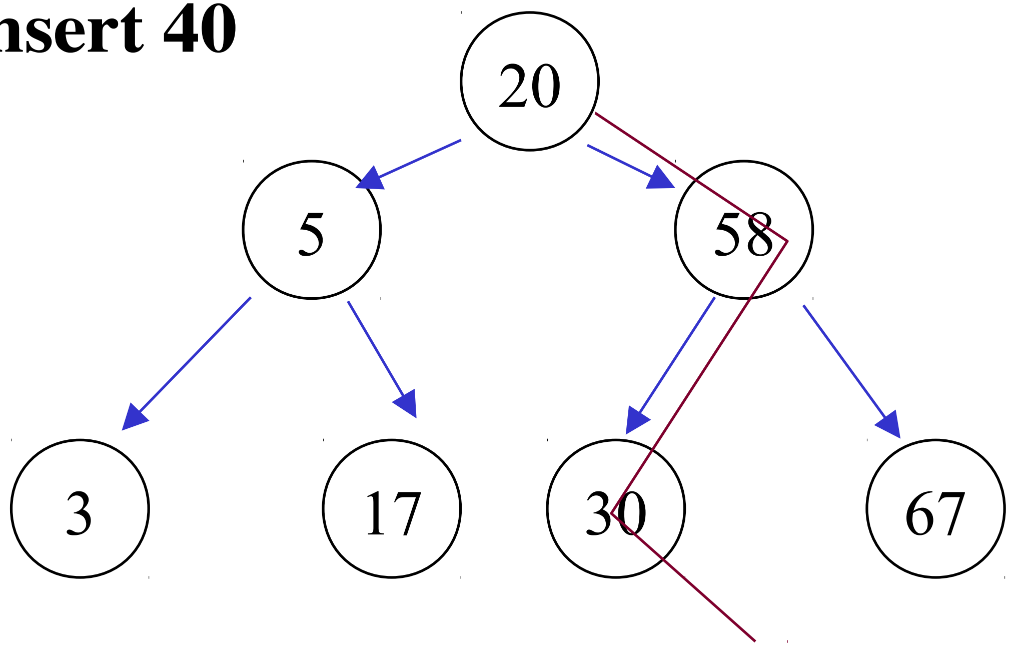
- data at a node is > any data in left subtree
- data at a node is < any data in right subtree
- Therefore, to print a BST in data order:
  - Print left subtree in data order
  - Print data
  - Print right subtree in data order

Search

- **Searching a BST is easy**
  - **if node = null, search fails**
  - **if node.data equals target, found**
  - **if target < node.data, search on left subtree**
  - **else search on right subtree**

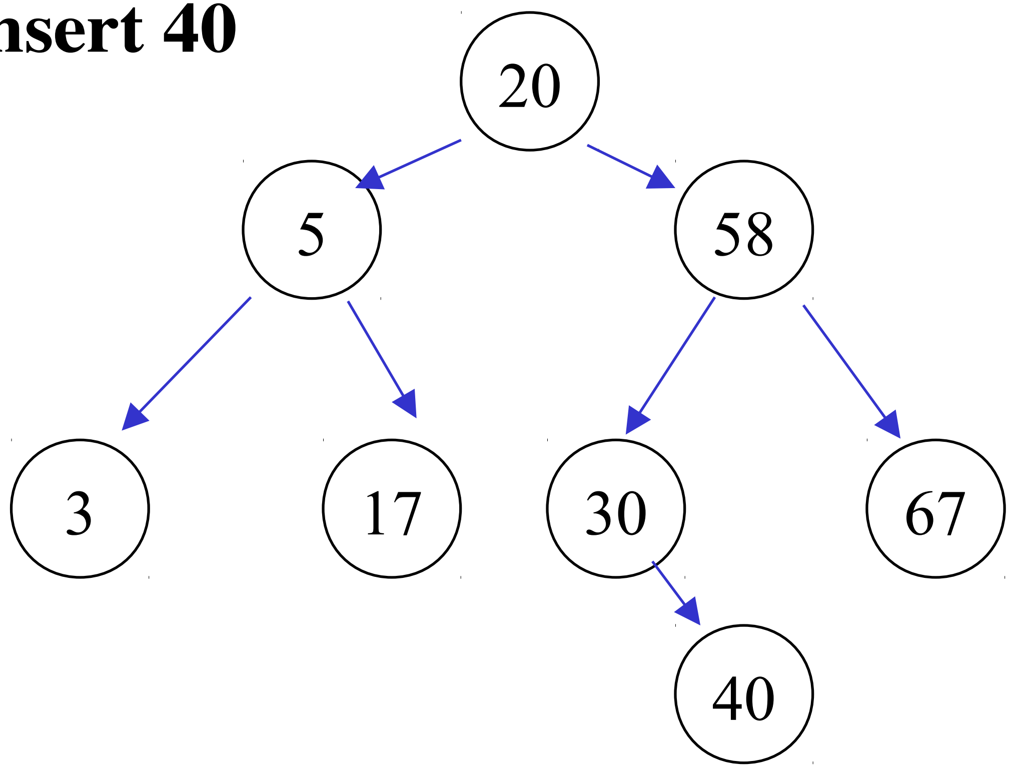
Insert

- **Search, fail, insert where failed**
  - **Insert 40**



Insert

- **Search, fail, insert where failed**
  - **Insert 40**

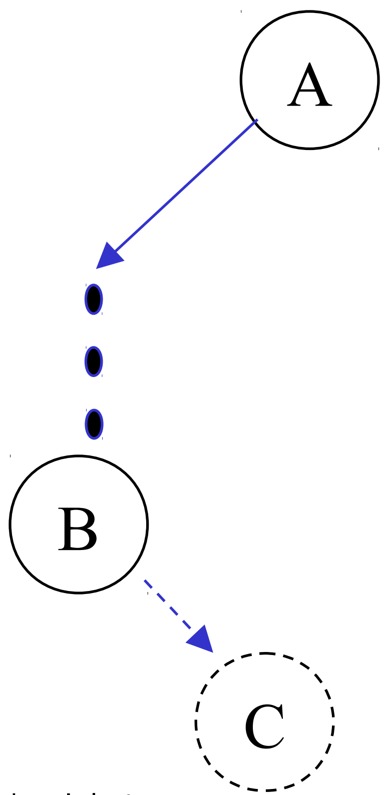


Delete

- **Three cases**
  - **node to delete has no children => delete it**
  - **node to delete has 1 child => replace node with child**
  - **node to delete has 2 children**

Deleting node with 2 children

- **Observation: for node with left child, inorder predecessor has no right child**



If C exists, C > B and A > C

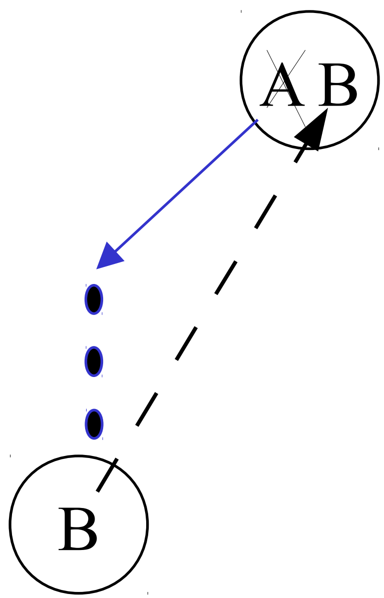
So B cannot be inorder predecessor of A

Deleting node with 2 children

- **Replace data at node with data of inorder predecessor**
- **Delete inorder predecessor (which must have either 0 or 1 child)**

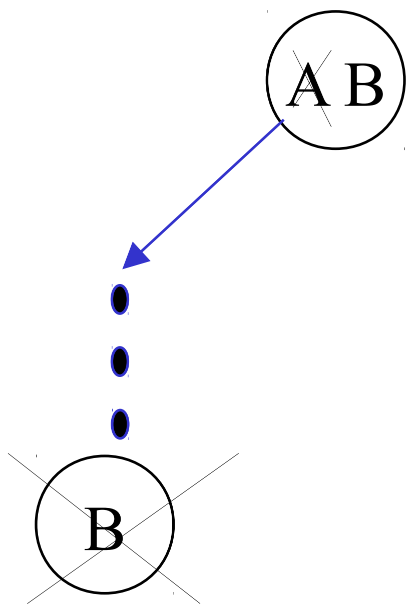
Deleting node with 2 children

- **Replace data at node with data of inorder predecessor**



Deleting node with 2 children

- **Delete inorder predecessor (which must have either 0 or 1 child)**



See BSTNode.java,
BST.java, and
BSTApp.java

Repeated Keys

- **What do you do if you can have two nodes with the same data?**

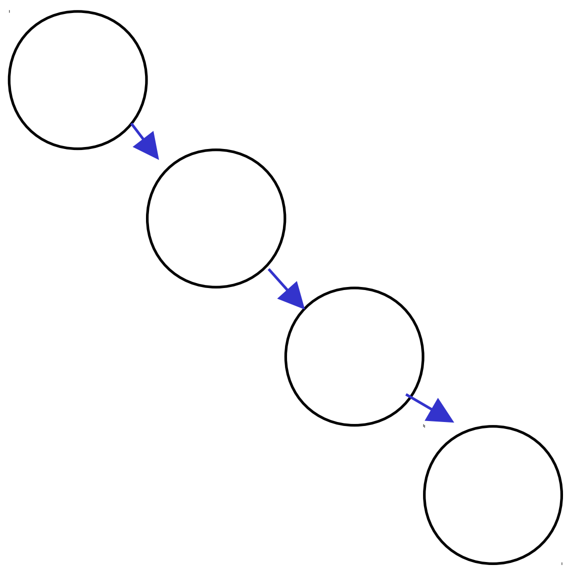
Cost of using BST

Search, insert delete: O(depth)

- **What is depth of tree?**
  - **with n nodes, best depth is log n**
  - **but worst depth is n**

Binary Search Trees

- **Problem: insertion & deletion can give tree of any shape - even**



Binary Search Trees

- **Problem: insertion & deletion can give tree of any shape**
- **Solution: AVL trees**