CS F211
Data Structures & Algorithms

Dictionary Data Structures – Search Trees

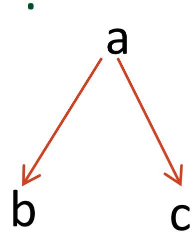
Binary Search Trees

- **Rank Queries**
- **In-order Traversal**
- **Complexity of Operations:**
  - **find, add, and delete**
  - **Rank Queries**

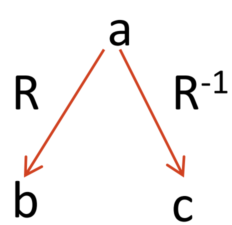
3/31/2017 Sundar B. CSIS, BITS, Pilani

TREES VS. SEARCH TREES

- Trees capture partial order.
- Example:
  - a R b and a R c :



  - b ? c
- What do search trees capture?



  - If R is ***transitive***, then (b R a) AND (c R^{-1} a) ==> b R c
- Conclusion:
  - *Left-to-Right traversal of a BST yields an ordered list*

BINARY SEARCH TREES (BSTS)

- BSTs store data in order:
  - i.e. if you traverse a BST such that for any node v,
    - Visit all nodes in the left sub tree of v
    - Visit v
    - Visit all nodes in the right sub tree of v
  - then you are visiting them in sorted order.
- This is referred to as in-order traversal:

```
inorder(BinaryTree bt) {
  if (bt != NULL) {
      inorder(bt->left));
      visit(bt);
      inorder(bt->right);
  }
}     // Time Complexity??  Space Complexity??
```

Binary Search Trees – Order Queries

- Exercises:
  - Write a procedure to find the minimum element in a BST.
  - Write a procedure to find the maximum element in a BST
  - Write a procedure to find the second smallest element in a BST.
  - Write a procedure to find the K^{th} smallest element in a BST.
  - Write a procedure to find the element in a BST with key closest to a given key.

ADT Ordered Dictionary – Complexity of operations

- Time Complexity:
  - *find*, *insert*, and *delete*
    - **θ(h)**, where **h** is the height of the tree
- Height of binary tree (by induction):
  - Empty Tree ==> 0
  - Non-empty ==> 1 + max(height(left), height(right))
- Balanced Tree
  - **h = logN**
    - Why?
- Unbalanced Tree
  - **h = N**
    - Example?
- How do you ensure balance?