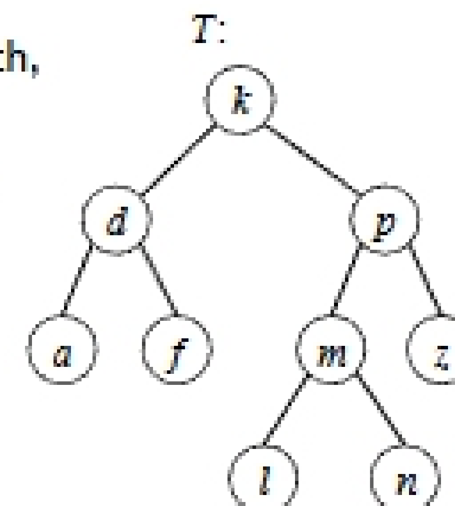
eecs281 DATA STRUCTURES AND ALGORITHMS

Lecture 7: BST Range Search
k-d Trees
Binary Space Partitioning Trees
Nearest-Neighbor Search

BST Range Search

Instead of finding an exact match, find all items whose keys fall between a range of values, e.g., between *m* and *z*, inclusive

Example applications:

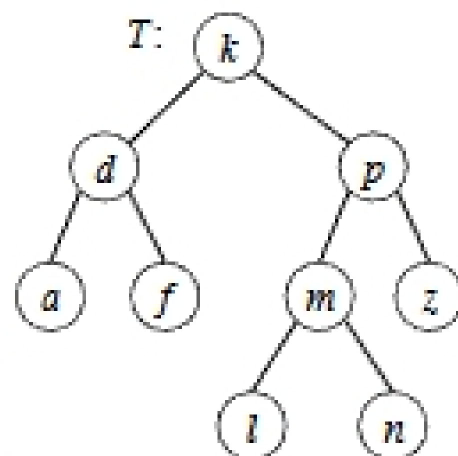


BST Range Search: Algorithm

```
void
rangesearch(Link root, Key searchrange[],
            Key subtreerange[], List results)
```

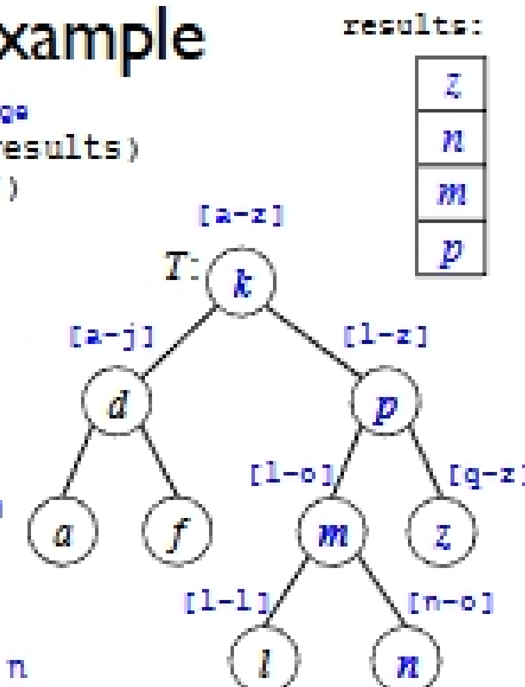
1. if root is in search range, add root to results
2. compute range of left subtree
3. if search range covers all or part of left subtree, search left
4. compute range of right subtree
5. if search range covers all or part of right subtree, search right
6. return results

(Other traversal orders are also ok)



BST Range Search: Example

```
               searchrange, subtreerange
rangesearch(T, [m-z], [a-z], results)
(T's range is "whole universe")
is k in [m-z]?
does [a-j] overlap [m-z]?
does [l-z] overlap [m-z]?
 search p's subtree
 is p in [m-z]? results ← p
 does [l-o] overlap [m-z]?
   search m's subtree
   is m in [m-z]? results ← m
   does [l-l] overlap [m-z]?
   does [n-o] overlap [m-z]?
    search n's subtree
    is n in [m-z]? results ← n
 does [q-z] overlap [m-z]?
   search z's subtree
   is z in [m-z]? results ← z
```



BST Range Search: Support Functions

1. if root is in search range, i.e.,
   `root->key <= searchrange[MAX]`, and
   `root->key >= searchrange[MIN]`
   add node to results

2. compute subtree's range: replace upper (lower) bound of left (right) subtree's range by `root->key-1 (+1)`

3. if search range covers all or part of subtree's range, search subtree
   - each subtree covers a range of key values
   - compute overlap between subtree's range and search range
   - no overlap if either
     `searchrange[MAX] < subtreerange[MIN]` or
     `searchrange[MIN] > subtreerange[MAX]`

BST Range Search: Other Details

How to express range when the keys are floats?
- be careful with numerical precision and floating point errors [one of this week's discussion topic]

How to support duplicate keys?
- be consistent about using $\leq$ or $<$
- if $\leq$, the range for the left subtree would be closed, e.g., $[-\infty, 0]$, and the range for the right subtree half open, e.g., $(0, +\infty]$

Multidimensional Search

Example applications:

A k-d tree can handle all these queries with $O(\log n)$ insert and search times (it can also handle partial, range, and approximate matches)

k-d Trees

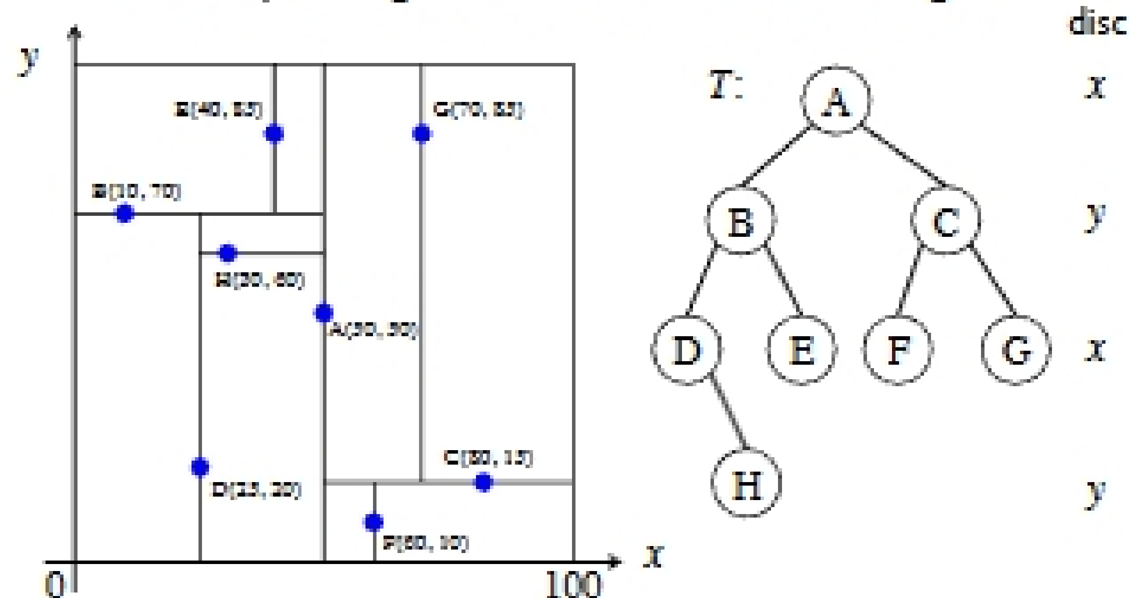
A k-d tree is a binary search tree (not covered in textbook, link to original 1975 paper on syllabus)

At each level of the k-d tree, keys from a different search dimension is used as the discriminator
- keys for any given level are all from the same dimension indicated by the discriminator
- nodes on the left subtree of a node have keys with value $<$ the node's key value along this dimension
- nodes on the right subtree have keys with value $>$ the node's key value along this dimension

We cycle through the dimensions as we go down the tree
- for example, given keys consisting of x- and y-coordinates, level 0 could discriminate by the x-coordinate, level 1 by the y-coordinate, level 2 again by the x-coordinate, etc.

k-d Trees: Example

Given points in a Cartesian plane on the left,
a corresponding *k*-d tree is shown on the right



k-d Tree Insert

```
void kdTree::
insert(Link &root, Item newitem, int disc)
{
  if (root == NULL) {
   root = new Node(newitem);
   return;
  }
  if (newitem.key[disc] < root->item.key[disc]) // or <=
   insert(root->left, newitem, (disc+1)%dim);
  else if (newitem.key[disc] > root->item.key[disc])
   insert(root->right, newitem, (disc+1)%dim);
}
```

If new item's key is smaller than root's along the dimension indicated by the discriminator, recursive call on left subtree else recursive call on right subtree

In both cases, switch the discriminator before traversing the next level of the tree, cycling through the dimensions

k-d Tree Search

Search works similarly to insert, using a discriminator to cycle through the dimensions as one recurses down the levels: $O(\log n)$ time

The tree we built was nicely balanced

- if nodes are inserted randomly, on average we will get a balanced tree: $O(\log n)$ time
- if nodes inserted are sorted, recursively insert the median of the range to build a balanced tree: $O(n \log n)$ time

k-d Tree Remove

To remove a node on a level with discriminator along dimension *j*:

- if the node is a leaf, remove it
- else if node has right subtree, find the *j*-minimum node in the right subtree
- replace node with *j*-minimum node and repeat until you reach a leaf, then remove the leaf
- else find the *j*-maximum node in the left subtree, replace, repeat, remove
- *j*-minimum means minimum along the *j* dimension, analogously *j*-maximum
- $O(\log n)$ time