DATA STRUCTURE AND ALGORITHM

Dr. Khine Thin Zar
Professor
Computer Engineering and Information Technology Dept.
Yangon Technological University

Lecture 12

Advanced Tree Structures

Outlines of Class (Lecture 12)

- Introduction
- Tries
- Balanced Trees
- AVL Trees
- The Splay Tree
- The K-D Tree
- The PR Quadtree

Introduction

- Designed for use in specialized applications
- Examples of self-balancing search trees
- Good performance

Tries

- The shape of a BST is determined by the order in which its data records are inserted.
- The value of the key stored in the root node splits the key range into two parts:
  - ✓ those key values less than the root's key value, and
  - ✓ those key values greater than the root's key value.
- The resulting BST might be balanced or unbalanced

Tries (Cont.)

- Object space decomposition:
  - ✓ The decomposition of the key range is driven by the objects stored in the tree.
- The alternative is to predefine the splitting position within the key range for each node in the tree.
- Key space decomposition:
  - ✓ Splitting based on predetermined subdivisions of the key range
- Trie: A data structure based on key space decomposition

Tries (Cont.)

- Object space decomposition:
  - ✓ The decomposition of the key range is driven by the objects stored in the tree.
- The alternative is to predefine the splitting position within the key range for each node in the tree.
- Key space decomposition:
  - ✓ Splitting based on predetermined subdivisions of the key range
- Trie: A data structure based on key space decomposition
  - ✓ can be built with any branching factor.

Tries (Cont.)

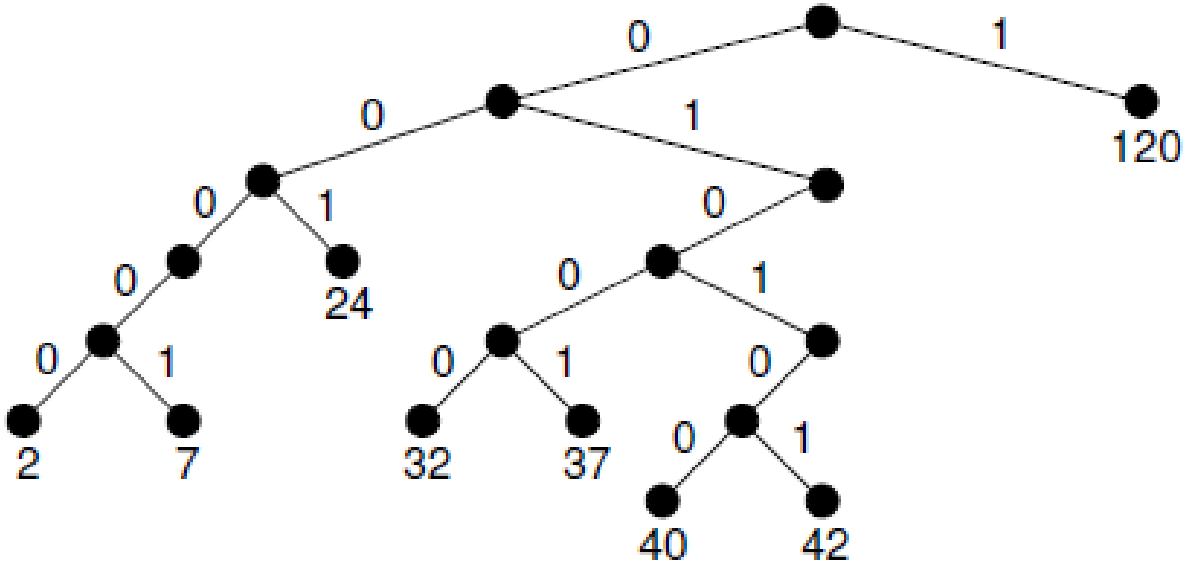

Figure: The binary trie for the collection of values 2, 7, 24, 31, 37, 40, 42, 120. All data values are stored in the leaf nodes. Edges are labeled with the value of the bit used to determine the branching direction of each node. The binary form of the key value determines the path to the record, assuming that each key is represented as a 7-bit value representing a number in the range 0 to 127.

Tries (Cont.)

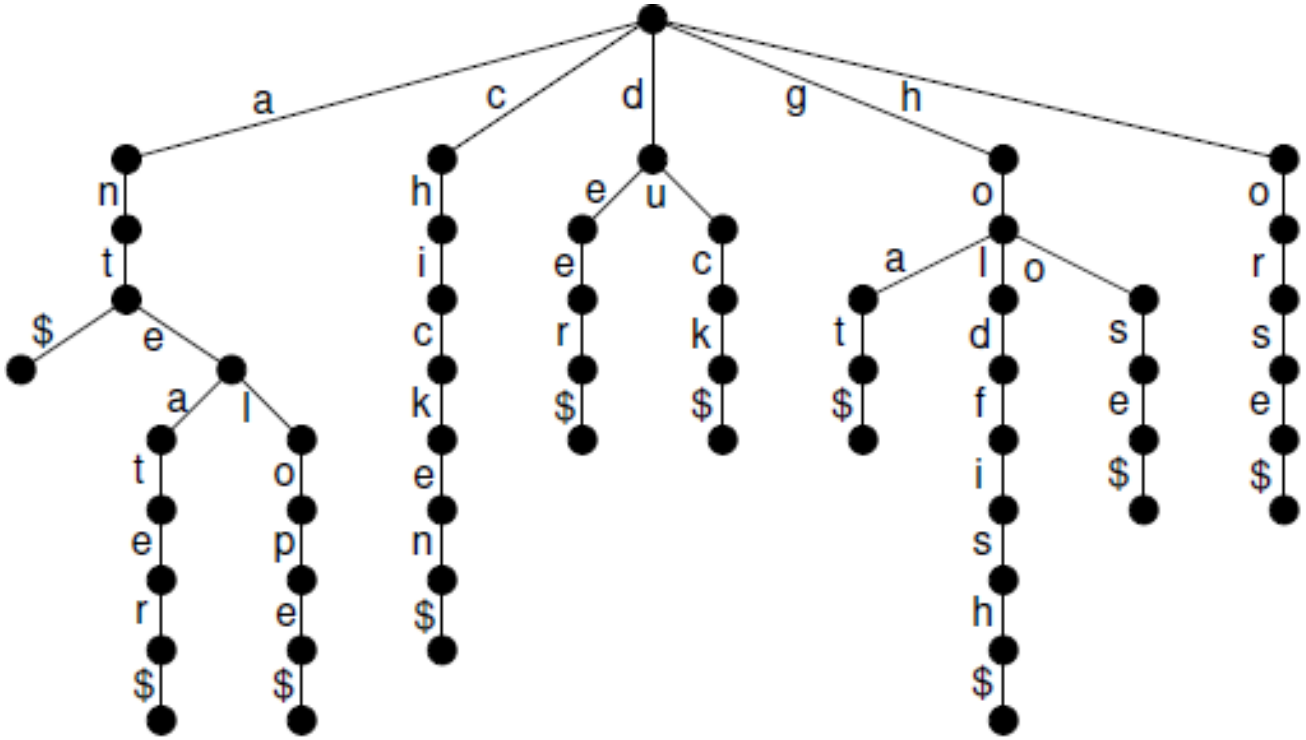

Figure: Two variations on the alphabet trie representation for a set of ten words. (a) Each node contains a set of links corresponding to single letters, and each letter in the set of words has a corresponding link. “$” is used to indicate the end of a word. Internal nodes direct the search and also spell out the word one letter per link. The word need not be stored explicitly. “$” is needed to recognize the existence of words that are prefixes to other words, such as ‘ant’ in this example.

Tries (Cont.)

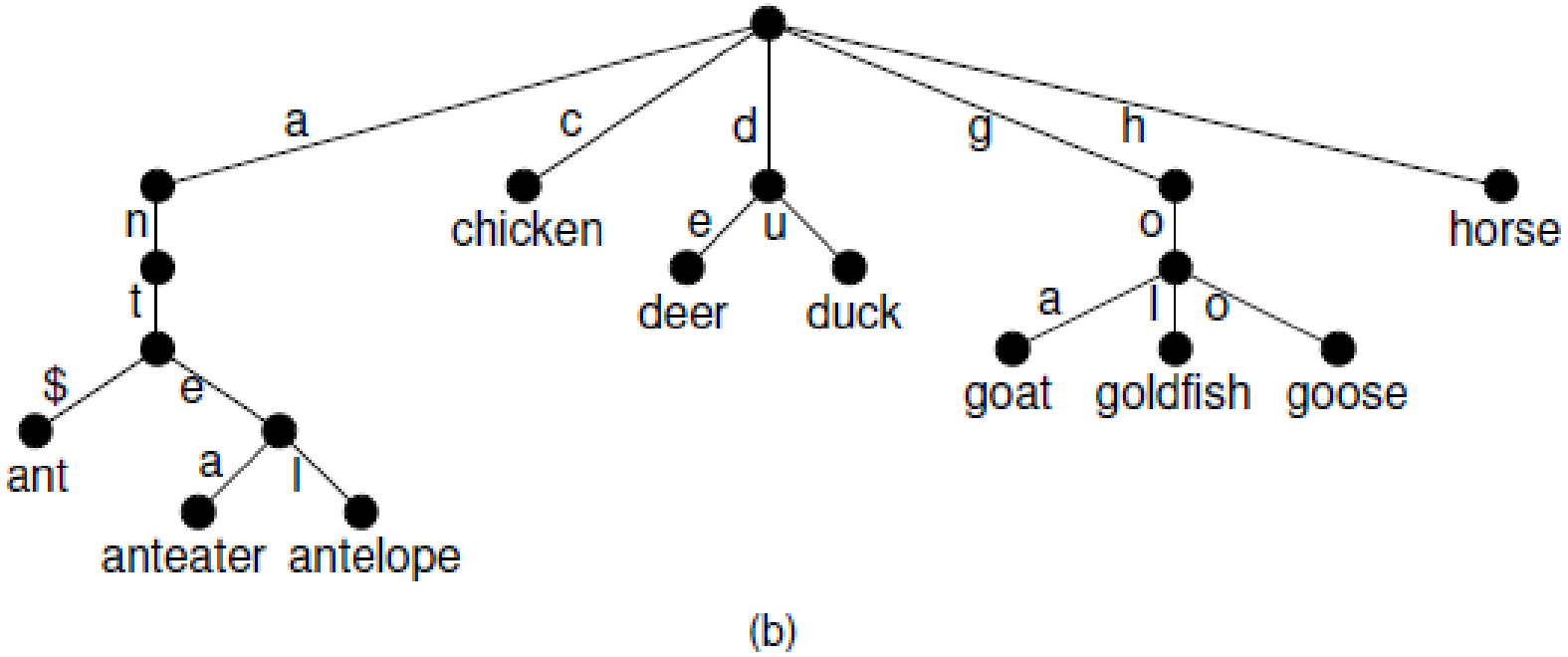

Figure 13.2 Two variations on the alphabet trie representation for a set of ten words. (b) Here the trie extends only far enough to discriminate between the words. Leaf nodes of the trie each store a complete word; internal nodes merely direct the search.

Tries (Cont.)

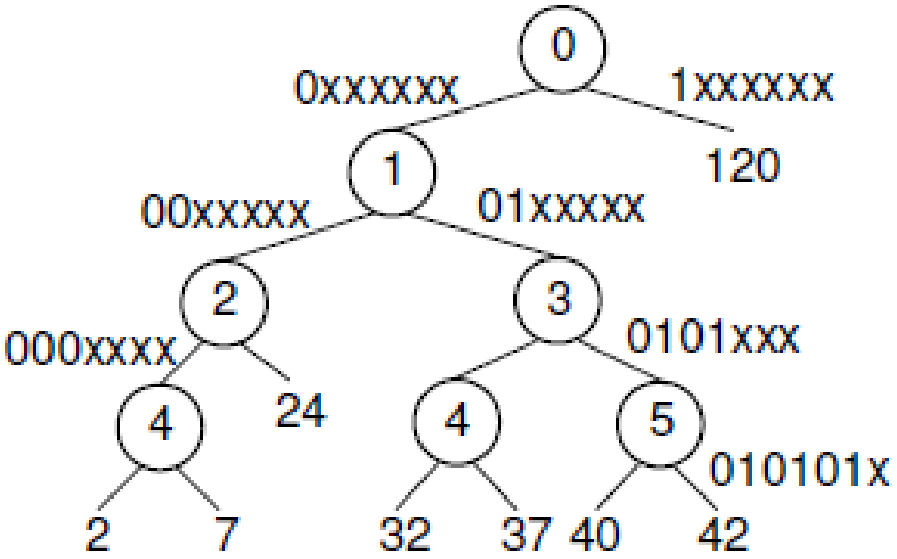

Figure: The PAT trie for the collection of values 2, 7, 24, 32, 37, 40, 42, 120.

Balanced Trees

- BST has a high risk of becoming unbalanced
- Expensive search and update operations
- Solution
  - Adopt another search tree structure such as the 2-3 tree or the binary trie
  - Modify the BST access functions in some way to guarantee that the tree performs well

AVL Trees

- Named for its inventors Adelson-Velskii and Landis
- The AVL tree: using insertion and deletion routines altered from those of the BST to ensure that, for every node, the depths of the left and right subtrees differ by at most one.
- Additional property: For every node, the heights of its left and right subtrees differ by at most 1.

AVL Trees (Cont.)

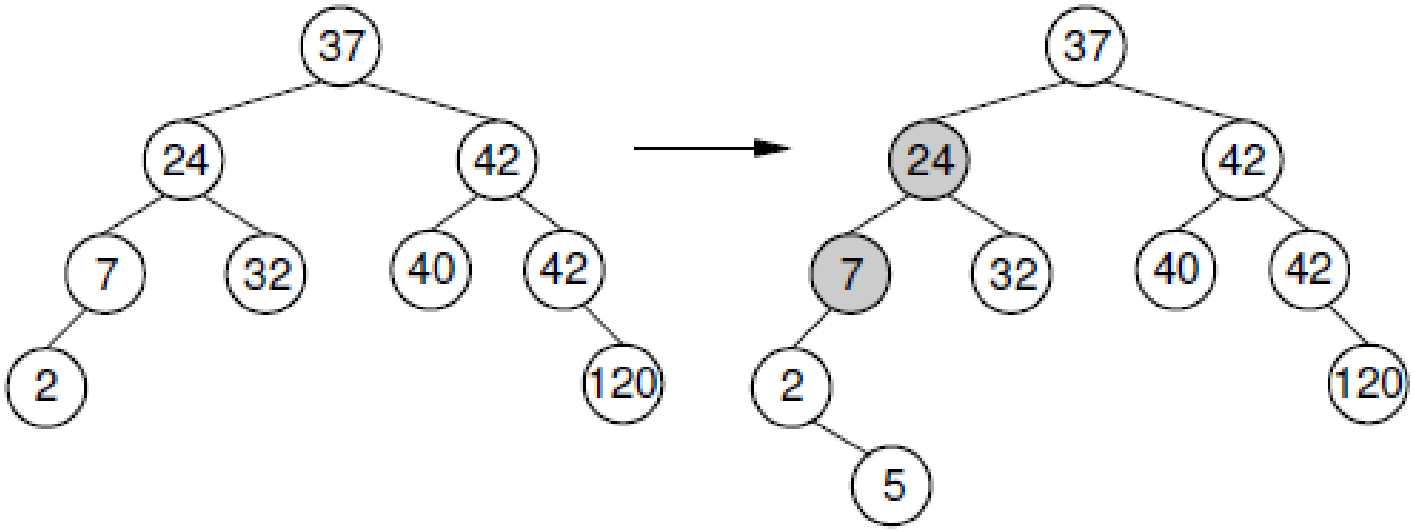

Figure: Example of an insert operation that violates the AVL tree balance property. Prior to the insert operation, all nodes of the tree are balanced (i.e., the depths of the left and right subtrees for every node differ by at most one). After inserting the node with value 5, the nodes with values 7 and 24 are no longer balanced.

AVL Trees (Cont.)

- For the bottommost unbalanced node, call it S, there are 4 cases:

1. The extra node is in the left child of the left child of S.
2. The extra node is in the right child of the left child of S.
3. The extra node is in the left child of the right child of S.
4. The extra node is in the right child of the right child of S.

AVL Trees (Cont.)

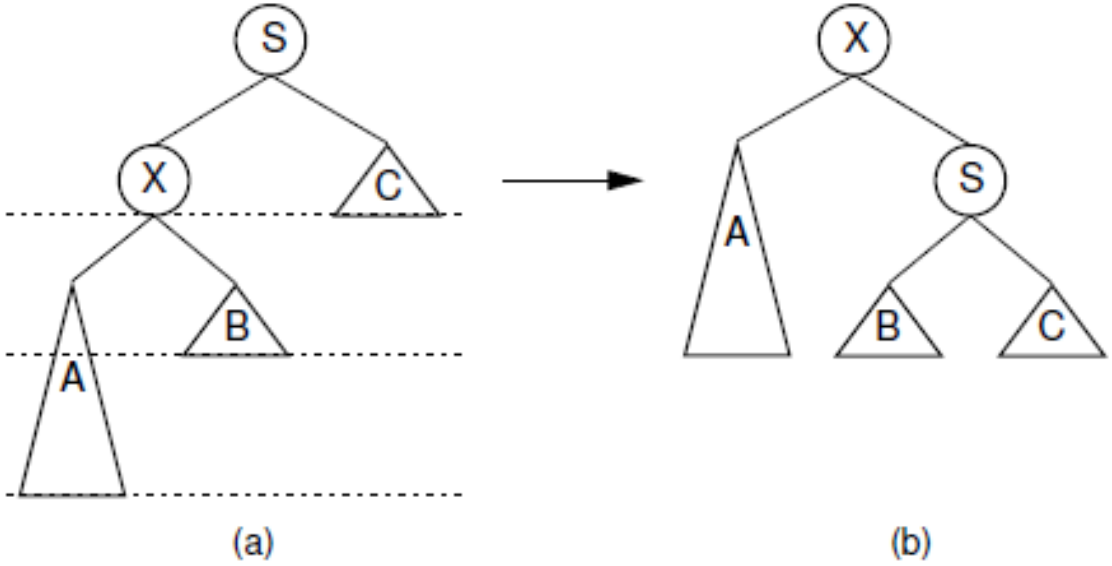

Figure: A single rotation in an AVL tree. This operation occurs when the excess node (in subtree A) is in the left child of the left child of the unbalanced node labeled S. By rearranging the nodes as shown, we preserve the BST property, as well as re-balance the tree to preserve the AVL tree balance property. The case where the excess node is in the right child of the right child of the unbalanced node is handled in the same way.

AVL Trees (Cont.)

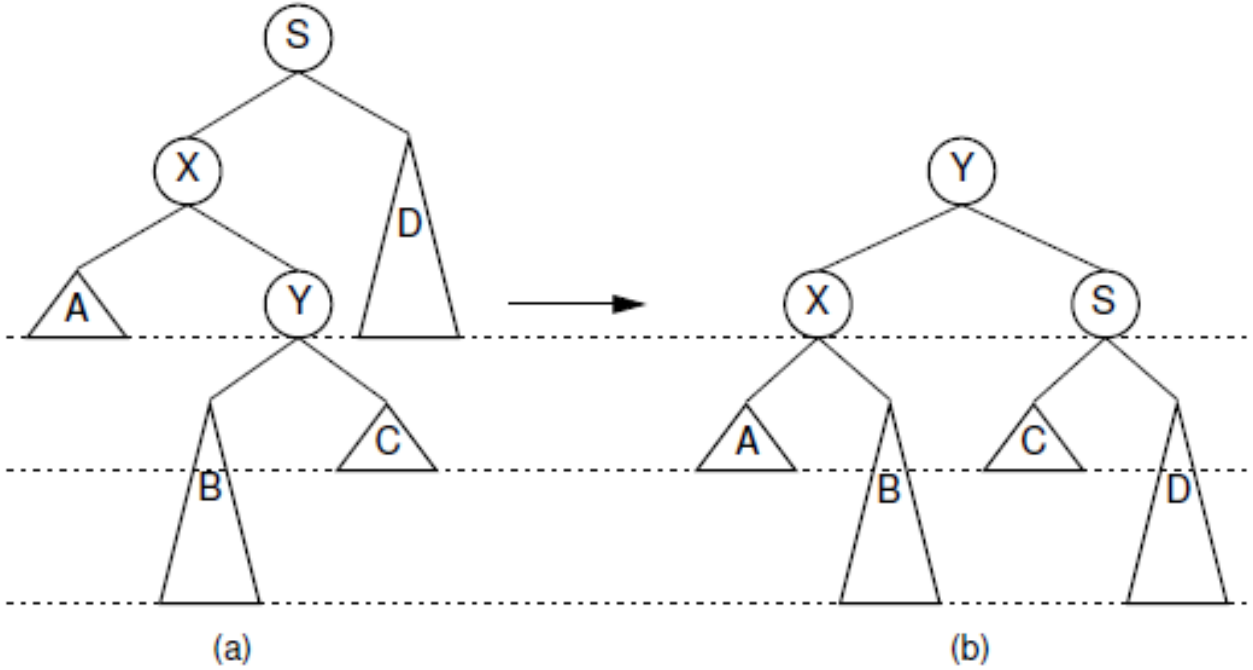

Figure: A double rotation in an AVL tree. This operation occurs when the excess node (in subtree B) is in the right child of the left child of the unbalanced node labeled S. By rearranging the nodes as shown, we preserve the BST property, as well as re-balance the tree to preserve the AVL tree balance property. The case where the excess node is in the left child of the right child of S is handled in the same way.

The Splay Tree

- Any M consecutive tree operations starting from an empty tree take at most O(MlogN) time
- O(logN) amortized cost per operation
- The basic idea of the splay tree
  - ✓ after a node is accessed, it is pushed to the root by a series of AVL tree rotations.
  - ✓ if a node is deep, there are many nodes on the path that are also relatively deep
  - ✓ have practical utility (when a node is accessed, it is likely to be accessed again in the near future)
  - ✓ do not require the maintenance of height or balance information

The Splay Tree (Cont.)

- to improve the performance of a BST.
- to provide guarantees on the time required by a series of operations
- to avoide the worst-case linear time behavior of standard BST operations

The Splay Tree (Cont.)

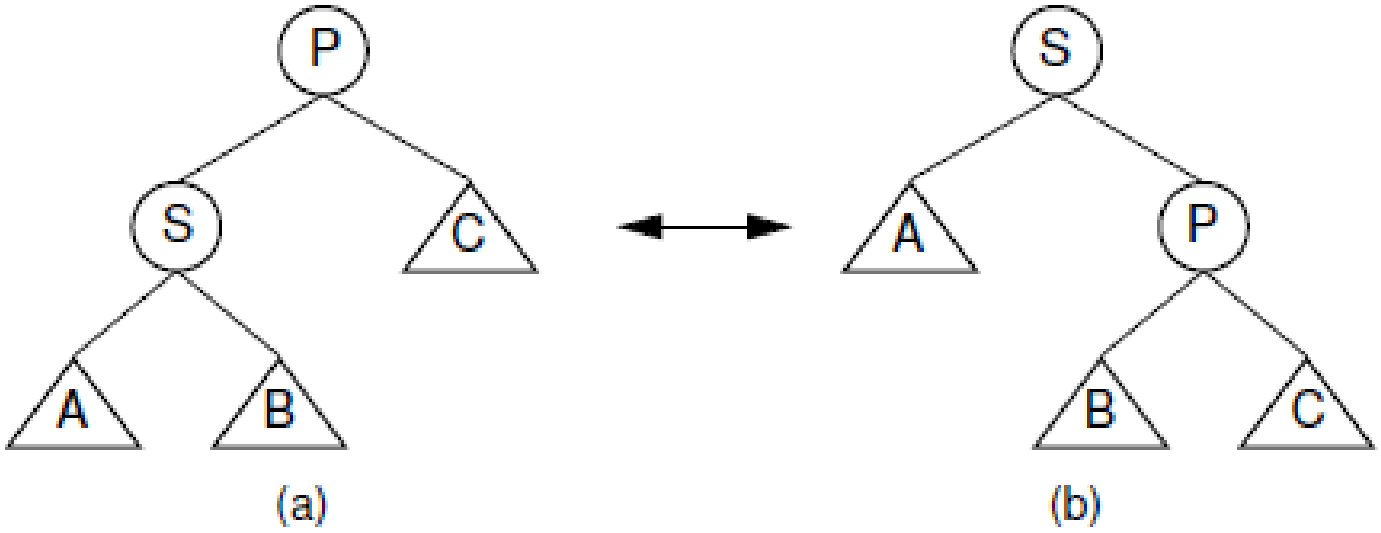

Figure: Splay tree single rotation. (a) The original tree with P as the parent. (b) The tree after a rotation takes place. Performing a single rotation a second time will return the tree to its original shape. Equivalently, if (b) is the initial configuration of the tree, then (a) shows the result of a single rotation to splay P to the root.

The Splay Tree (Cont.)

- zigzag rotation:
  - ✓ It takes place when either of the following two conditions are met:
    1. S is the left child of P, and P is the right child of G.
    2. S is the right child of P, and P is the left child of G.

The Splay Tree (Cont.)

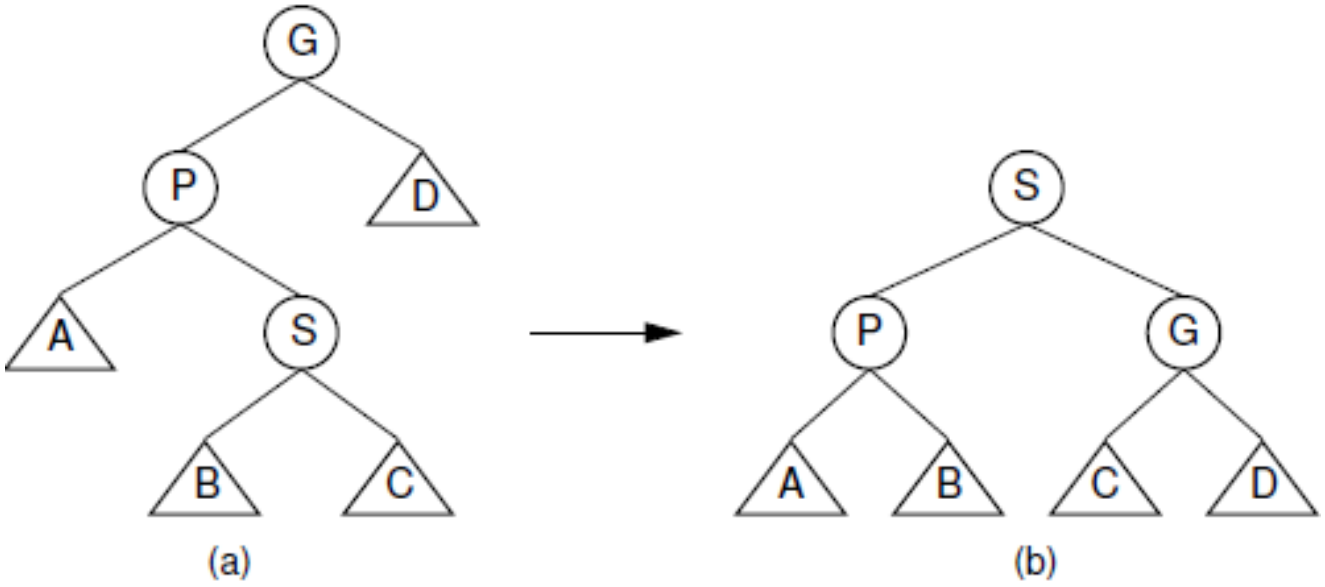

Figure: Splay tree zigzag rotation. (a) The original tree with S, P, and G in zigzag formation. (b) The tree after the rotation takes place. The positions of subtrees A, B, C, and D are altered as appropriate to maintain the BST property.

The Splay Tree (Cont.)

- zigzig rotation:
  - ✓ It takes place when either of the following two conditions are met:

1. S is the left child of P, which is in turn the left child of G.

2. S is the right child of P, which is in turn the right child of G.

The Splay Tree (Cont.)

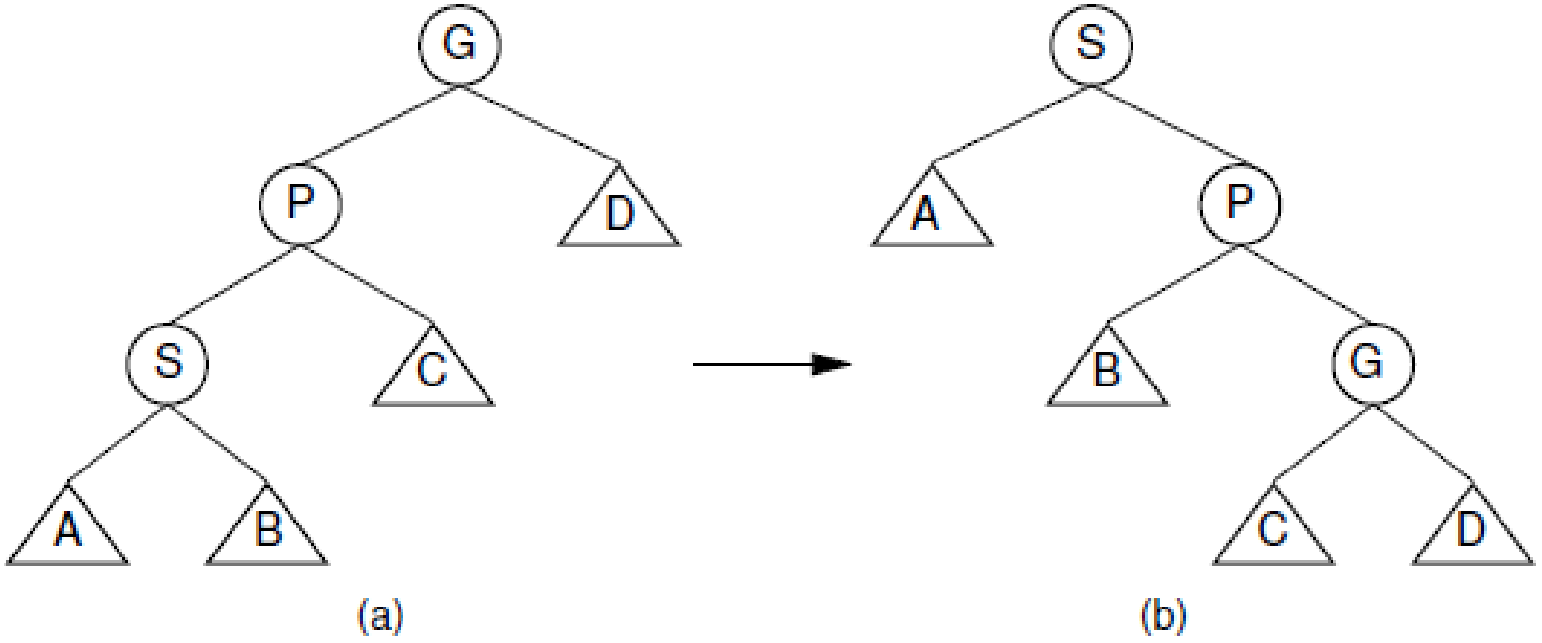

Figure: Splay tree zigzig rotation. (a) The original tree with S, P, and G in zigzig formation. (b) The tree after the rotation takes place. The positions of subtrees A, B, C, and D are altered as appropriate to maintain the BST property.

Spatial Data Structures

- BSTs, AVL trees, splay trees, 2-3 trees, B-trees, and tries: designed for searching on a one-dimensional key.
- For a multidimensional search key:
  - ✓ Separate BSTs could be used to index the x- and y-coordinates.
  - ✓ To combine the xy-coordinates into a single key
  - ✓ Allow search by coordinate, but would not allow for efficient two-dimensional range queries
- Feature of a spatial application: Multidimensional range queries
- To implement spatial applications efficiently requires the use of spatial data structures.
- used in geographic information systems, computer graphics, robotics, and many other fields

The K-D Tree

- Efficient processing of multidimensional keys
- Each level of the k-d tree makes branching decisions based on a particular search key associated with that level, called the discriminator
- Unify key searching across any arbitrary set of keys such as name and zipcode
- The discriminator at level i to be i mod k for k dimensions.

The K-D Tree (Cont.)

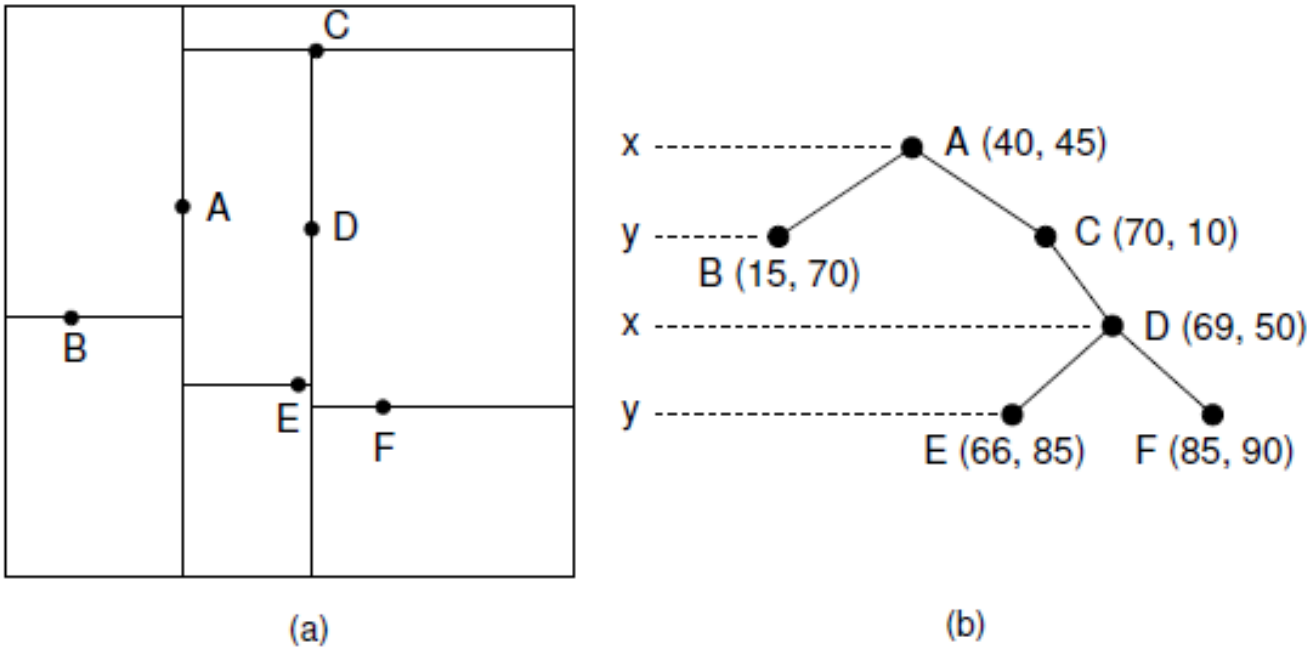

Figure: Example of a k-d tree. (a) The k-d tree decomposition for a 128 128-unit region containing seven data points. (b) The k-d tree for the region of (a).

The K-D Tree (Cont.)

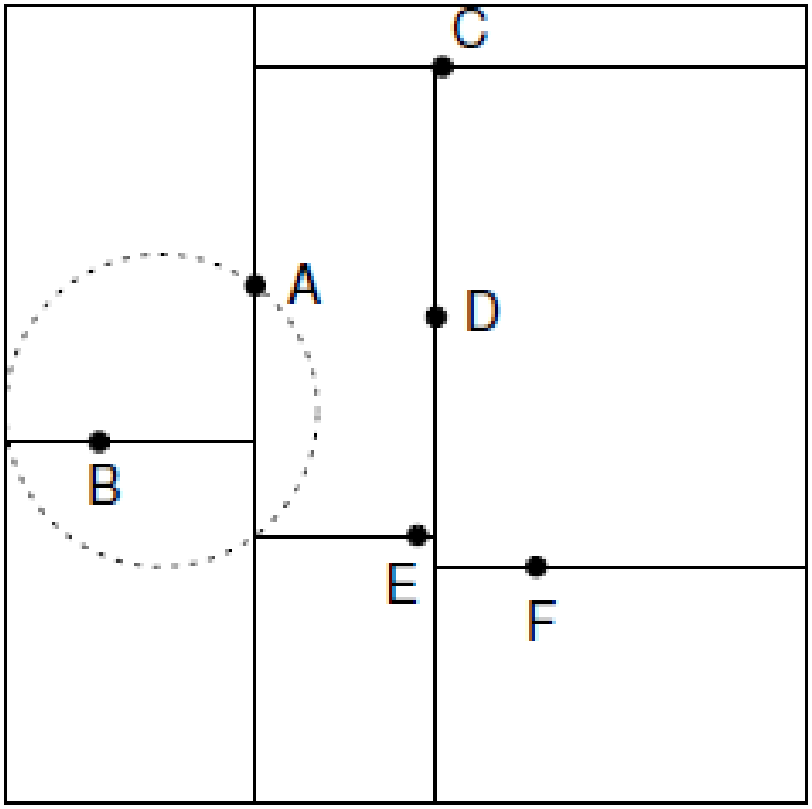

Figure: Searching in the k-d tree

The K-D Tree (Cont.)

```
// Find the record with the given coordinates
bool findhelp(BinNode<E>* root, int* coord,
              E& e, int discrim) const {
  // Member "coord" of a node is an integer array storing
  // the node's coordinates.
  if (root == NULL) return false;      // Empty tree
  int* currcoord = (root->val())->coord();
  if (EqualCoord(currcoord, coord)) { // Found it
    e = root->val();
    return true;
  }
  if (currcoord[discrim] < coord[discrim])
    return findhelp(root->left(),coord,e,(discrim+1)%D);
  else
    return findhelp(root->right(),coord,e,(discrim+1)%D);
}
```

The K-D Tree (Cont.)

- If the search process reaches a NULL pointer, then that point is not contained in the tree.
- Equivalent to the findhelp function of the BST class
- KD class private member D stores the key's dimension.

The PR Quadtree

- Each node either has exactly four children or is a leaf.
- A full fourway branching (4-ary) tree in shape
- Represents a collection of data points in two dimensions
- Decomposing the region containing the data points into four equal quadrants, subquadrants, and so on
- The corresponding PR quadtree then contains an internal node and four subtrees
- The four quadrants of the region (or equivalently, the corresponding subtrees) are designated (in order) NW, NE, SW, and SE.

The PR Quadtree (Cont.)

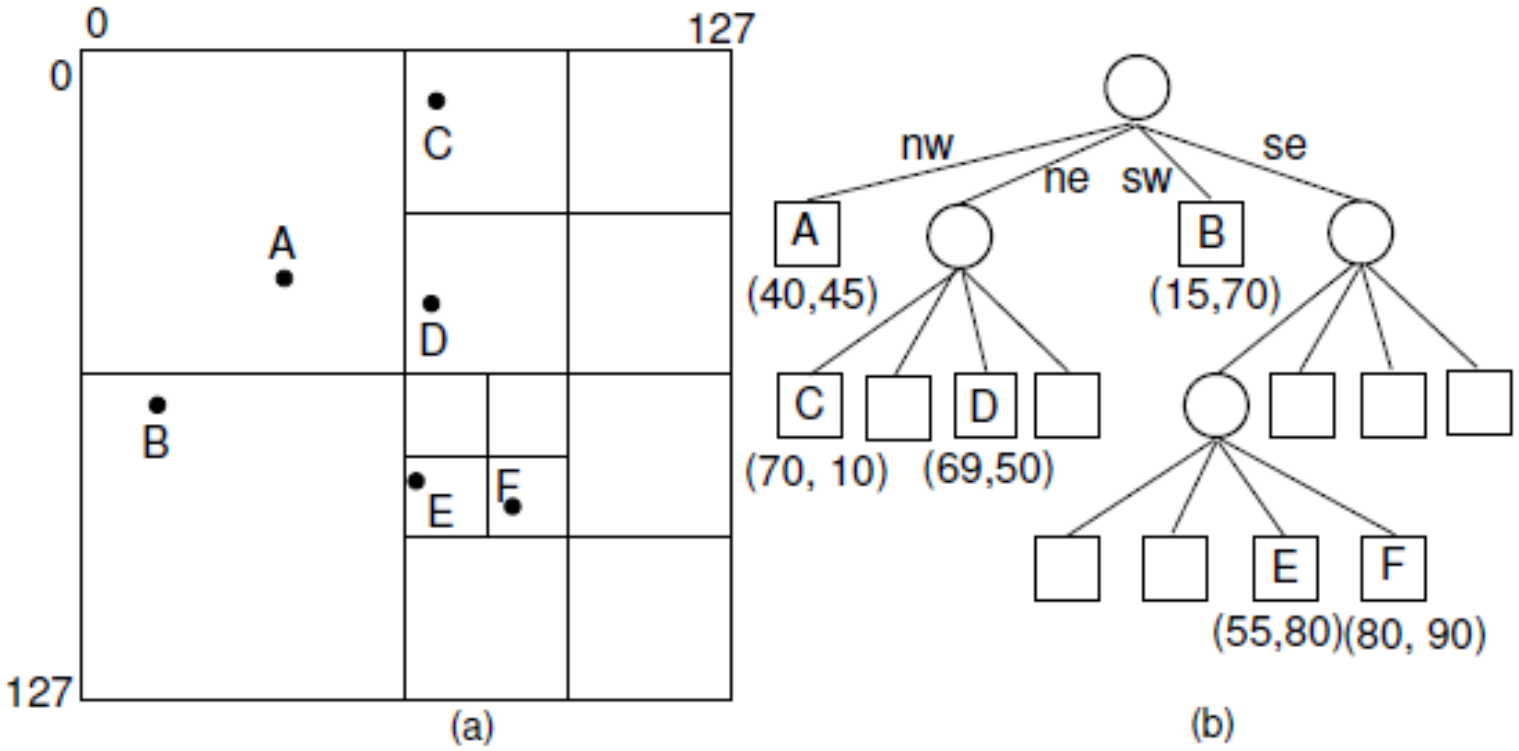

Figure: Example of a PR quadtree. (a) A map of data points. We define the region to be square with origin at the upper-left-hand corner and sides of length 128. (b) The PR quadtree for the points in (a).

Bintree

- A binary trie that uses keyspace decomposition and alternates discriminators at each level in a manner similar to the k-d tree.

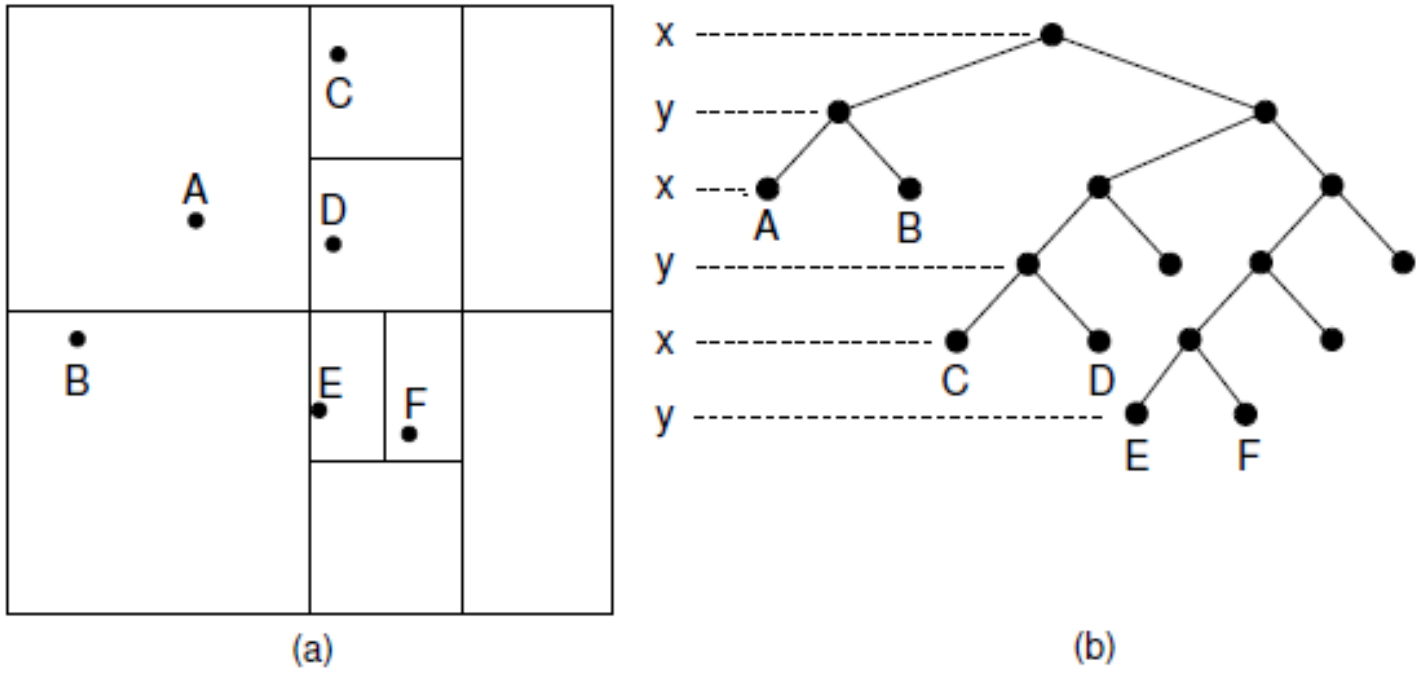

Figure: An example of the bintree

Point Quadtree

- Use a four-way decomposition of space centered on the data points.
- Resulting from such a decomposition is called a point quadtree

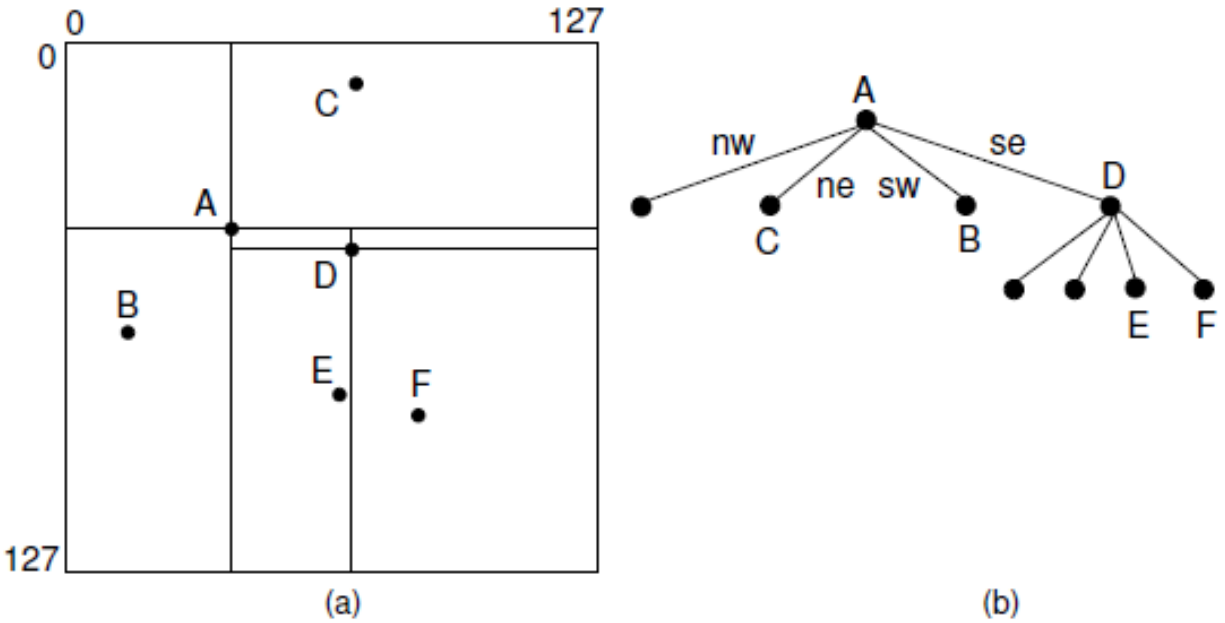

Figure: An example of the point quadtree

Next Week Lecture (Week 13)

Lecture 13: Algorithm Design Techniques

Thank you!