

Advanced Data Structures

Introduction:

The main disadvantage with binary search is that it requires that the array remain sorted. Keeping an array sorted requires an insertion every time an element is added to the array. If the list is large enough, the processing time to do an insertion is considerable, as most of the list has to be recopied each time an element is added. Hashing is a technique for storing information for fast retrieval that does not require that the list remain sorted. Hashing is also faster than binary search. To find an element is independent of the number of elements in the array! In other words, it takes the same length of time to find an element among 50 as it does to find the element among 50,000,000 items.

Sets - a collection of elements with no duplicates. Set is an interface in Java

I. Hash Tables (20.3) - implements the Set interface

A) Simplistic Implementation of a Hash Table

1. To implement

- a. Generate hash codes for objects - The hashing technique requires a function that converts the key's data into an index into the array where the data is stored. This function is called the **hash** function and the result from the function is called the key's hash **value**. A *hash function* computes an integer value (called the *hash code*) from an object. We say that the hash function maps the key to the hash value or that the key is mapped **into** the hash value. A *hash table* can be used to implement sets and maps.
- b. Make an array - An array (or other data structure like ArrayList or a linked list) that stores the data based on the hash values of the element is called a hash **table**.
- c. Insert each object at the location of its hash code - To add an item to the array of data, you store the item at the index specified by the item's hash value.

2. To test if an object is contained in the set. Hashing can be used to find elements in a data structure quickly without making a linear search

- a. Compute its hash code
- b. Check if the array position with that hash code is already occupied

B) Problems with Simplistic Implementation

1. It is not possible to allocate an array that is large enough to hold all possible integer index positions
2. It is possible for two different objects to have the same hash code

C) Computing Hash Codes - A good hash function minimizes **collisions** (identical hash codes for different objects)

- A hash function computes an integer (really a nonnegative integer) hash code from an object
- Choose a hash function so that different objects are likely to have different hash codes.

ex.1) Storing strings

ex a) Bad choice for hash function for a string:

```
int h = 0;
for (int i = 0; i < s.length( ); i++)
    h = h + s.charAt(i);
```

Permutations ("eat" and "tea") would have the same hash code

ex.b) Good choice for hash function for a string from the from Javastandard library (using the unicode (ASCII) values of the characters in the string)

```
final int HASH_MULTIPLIER = 31;
int h = 0;
for (int i = 0; i < s.length( ); i++)
    h = HASH_MULTIPLIER * h + s.charAt(i)
```

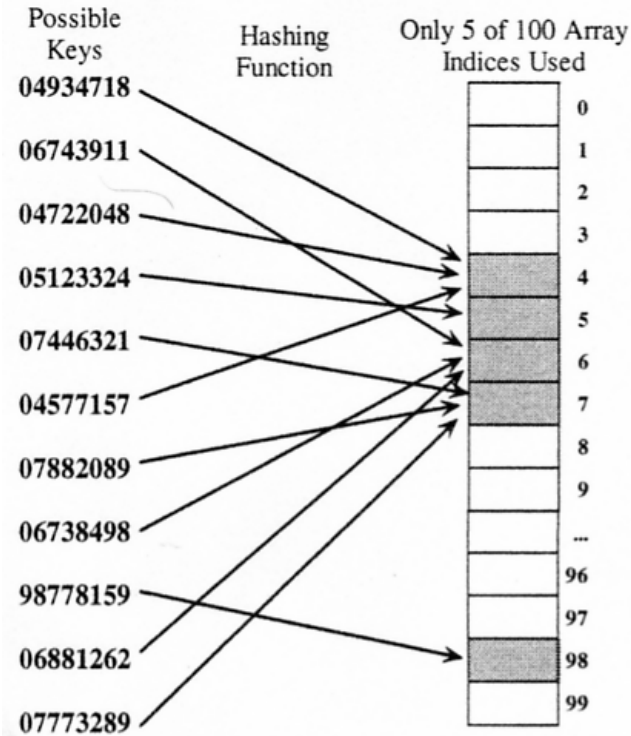
For example, the hash code of "eat" is

$$31 * (31 * 'e' + 'a') + 't' = 100184$$

The hash code of "tea" is quite different, namely

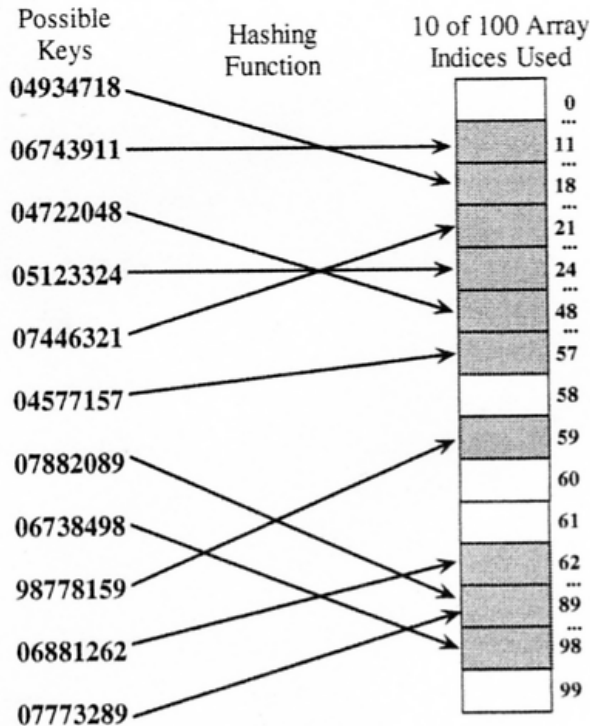
$$31 * (31 * 't' + 'e') + 'a' = 114704$$

ex.2) Storing student info using the ID code as the hashing function
 ex.a) Bad one using just the first 2 digits (the year of starting school)



A Bad Hashing Function – 6 Collisions in 11 Keys

ex.b) Good one using just the last 2 digits



A Good Hashing Function – 1 Collision in 11 Keys

D) Handling Collisions

Pick a reasonable array size and reduce the hash codes to fall inside the array where the array should be only 30% filled.

```
int h = x.hashCode( );  
if (h < 0)  
    h = -h;  
h = h % size;
```

Method 1 - Linear Probing

If an array position is not empty at the position calculated by the hash function put the item in the next empty array position. If you reach the end of the array loop to the beginning.

Method 2 - Chaining

When elements have the same hash code: Use a node sequence to store multiple objects in the same array position (like a linked list). These node sequences are called **buckets**

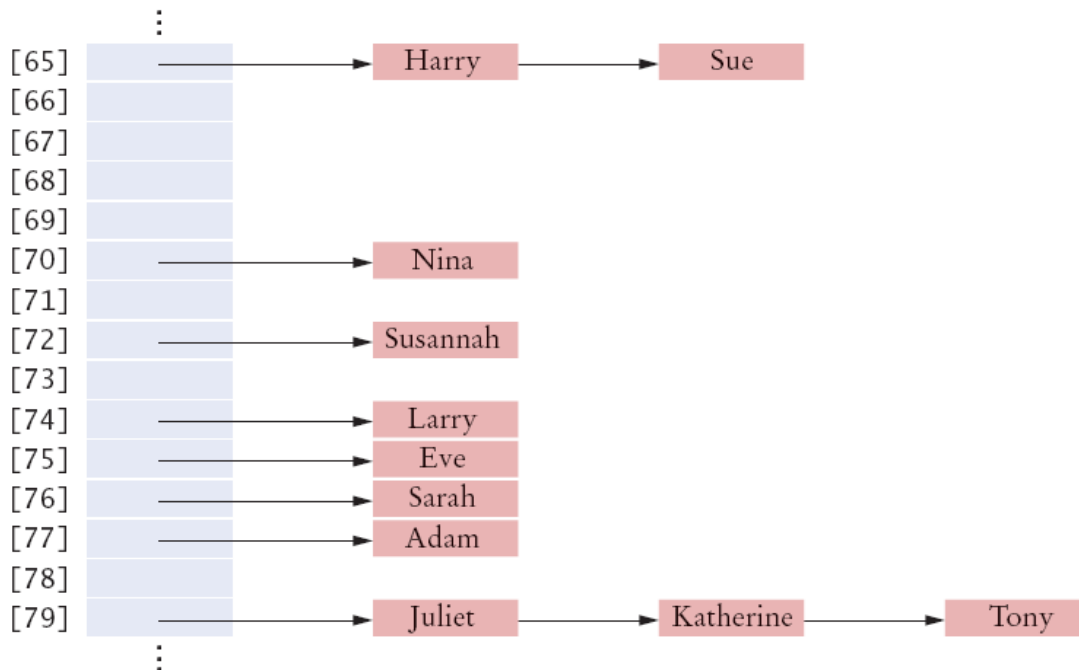


Figure 6 A Hash Table with Buckets to Store Elements with the Same Hash Code

Chaining has several advantages over linear probing:

1. The hash table cannot overflow. With linear probing, if you add more entries to the hash table than the size of the hash table, the hash table is filled and the program stops working or has an infinite loop. With chaining, entries that collide are not stored in the table itself.
2. Collisions at one hash value will not cause collisions at another hash value. With linear probing a collision at one location causes an entry to be stored at a different location, which can then collide with different hash value. With chaining, only the entries for a particular hash value are stored in the linked list associated with that value.
3. Entries can be removed from a hash table more easily. With linear probing, removing an entry is fraught with difficulties as leaving an empty space in the table could cause the linear probing algorithm to incorrectly report that an item was not found.

II. Binary Search Trees - All Java tree structures implement the Set interface

Introduction

- Binary search trees allow for fast insertion and removal of elements
- They are specially designed for fast searching
- A binary tree consists of two nodes, each of which has two child nodes
- All nodes in a binary search tree fulfill the property that:
 - Descendants to the left have smaller data values than the node data value
 - Descendants to the right have larger data values than the node data value

ex.1)

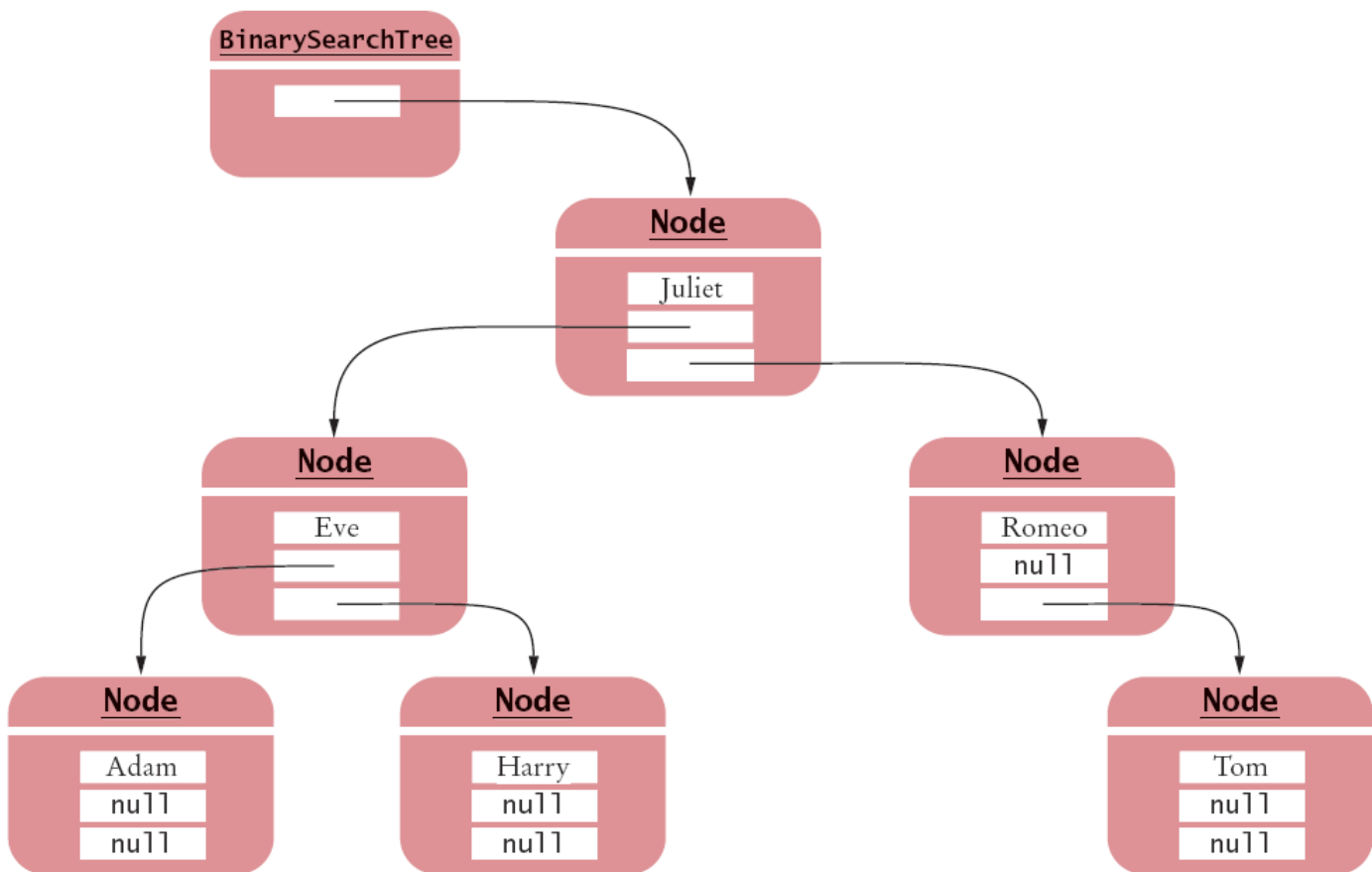


Figure 7 A Binary Search Tree

Terminology:

- 1) Width: 3
- 2) Depth: 3
- 3) Root (single topmost node): Juliet
- 4) Leaves (a node with no child): Adam, Harry, Tom
- 5) Parent (a node with 1 or 2 children):
- 6) Left and/or Right Child:
- 7) Siblings (have a direct common parent):

ex.2) a binary tree that is not a binary search tree

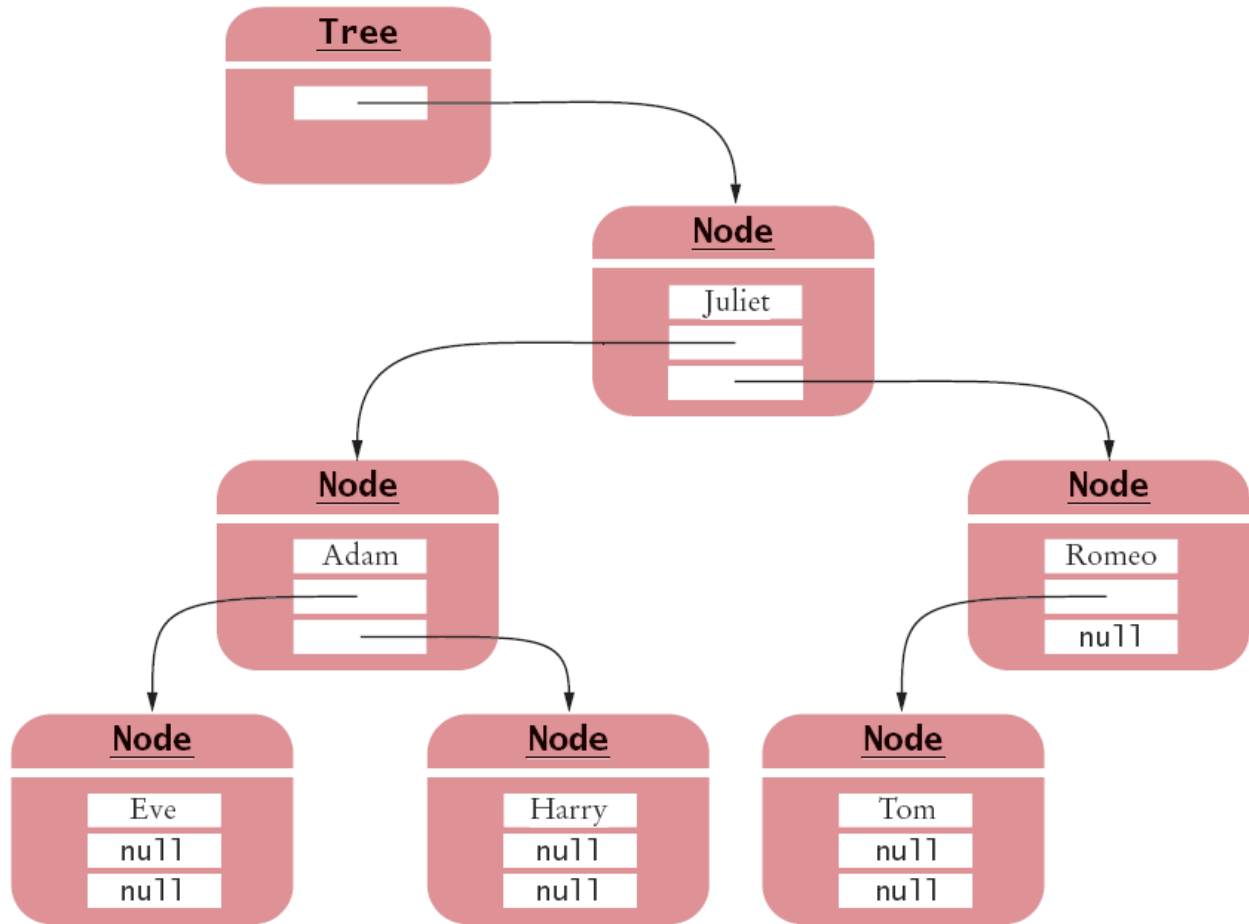


Figure 8 A Binary Tree That Is Not a Binary Search Tree

Why is this not a Binary Search Tree?

Answer: Eve > Adam and Tom > Romeo

A) Insertion Algorithm

- If you encounter a non-null node reference, look at its data value
 - If the data value of that node is larger than the one you want to insert, continue the process with the left subtree
 - If the existing data value is smaller, continue the process with the right subtree
- If you encounter a null node pointer, replace it with the new node

Code:

```
public class BinarySearchTree
{
    ...
    public void add(Comparable obj)
    {
        Node newNode = new Node( );
        newNode.data = obj;
        newNode.left = null;
        newNode.right = null;
        if (root == null) root = newNode;
        else root.addNode(newNode);
    }
    ...
}

private class Node
{
    ...
    public void addNode(Node newNode)
    {
        int comp = newNode.data.compareTo(data);
        if (comp < 0)
        {
            if (left == null) left = newNode;
            else left.addNode(newNode);
        }
        else if (comp > 0)
        {
            if (right == null) right = newNode;
            else right.addNode(newNode);
        }
    }
    ...
}
```

ex.1)

```
BinarySearchTree tree = new BinarySearchTree( );
```

```
tree.add("Juliet"); ①
```

```
tree.add("Tom"); ②
```

```
tree.add("Dick"); ③
```

```
tree.add("Harry"); ④
```

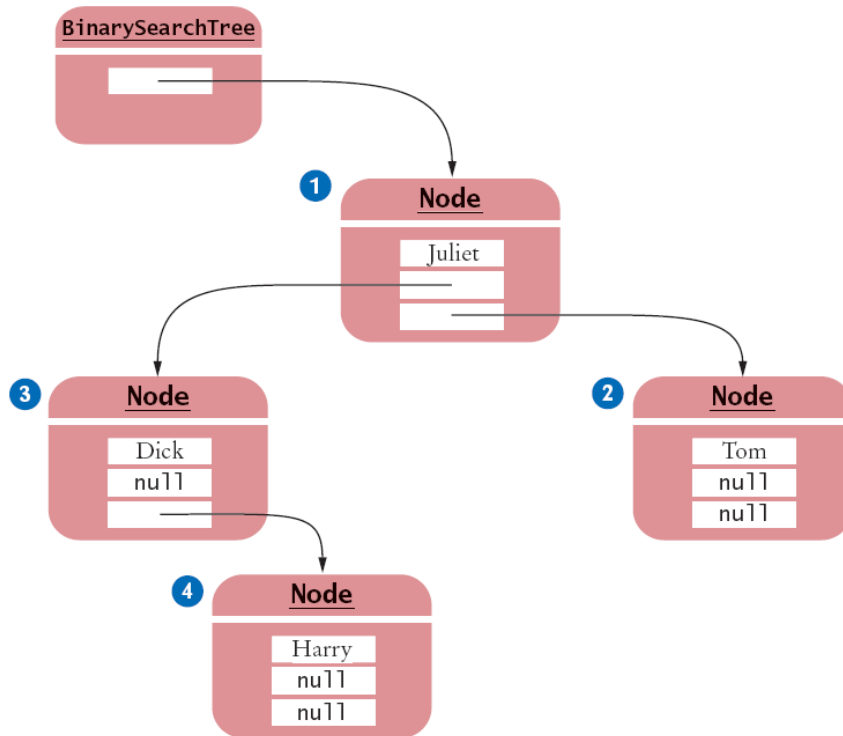


Figure 9
Binary Search Tree
After Four Insertions

ex.1) continued

```
tree.add("Romeo");
```

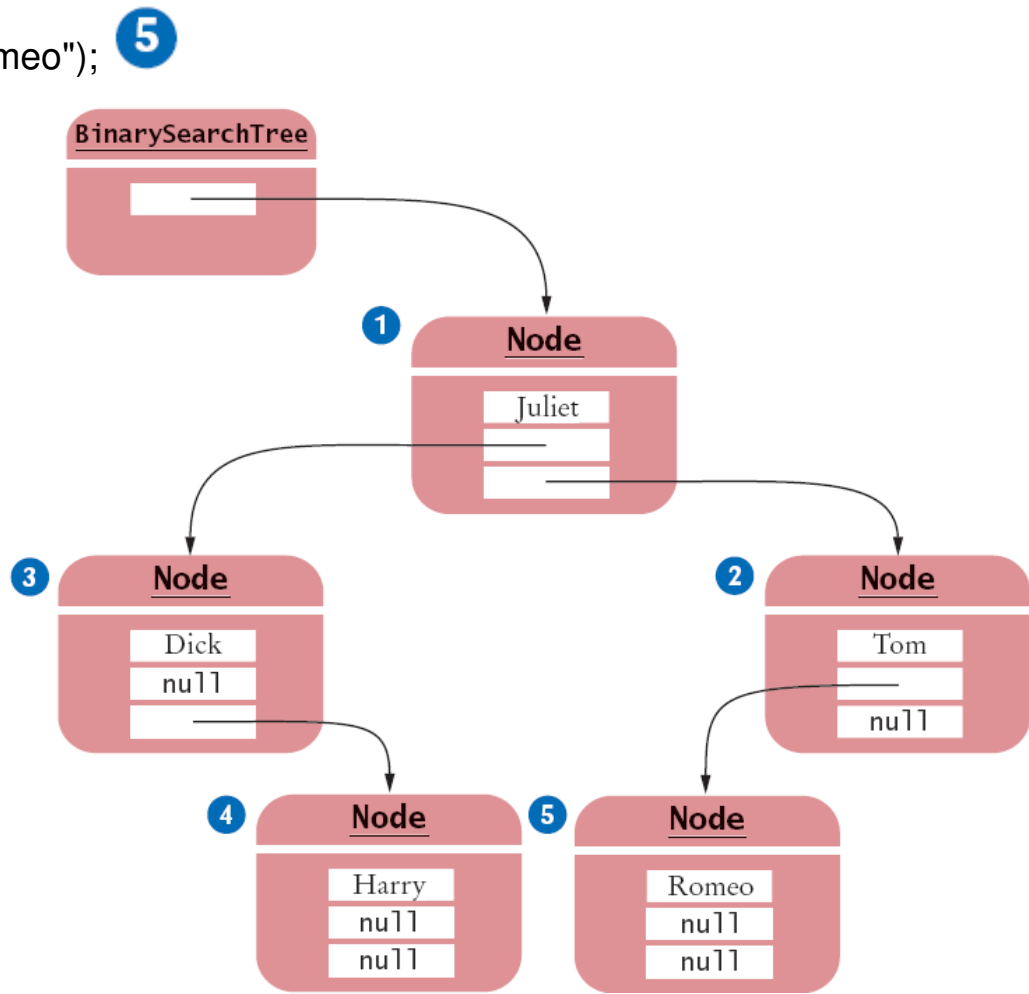


Figure 10
Binary Search Tree
After Five Insertions

B) Removing a node

- 1) When removing a node with only one child, the child replaces the node to be removed.

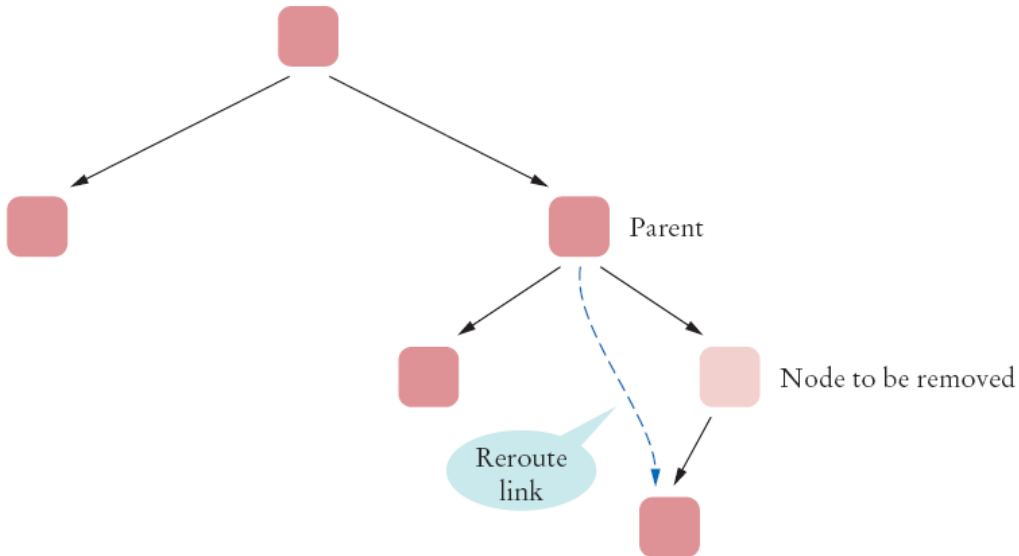


Figure 11 Removing a Node with One Child

- 2) When removing a node with two children, replace it with the smallest node of the right subtree

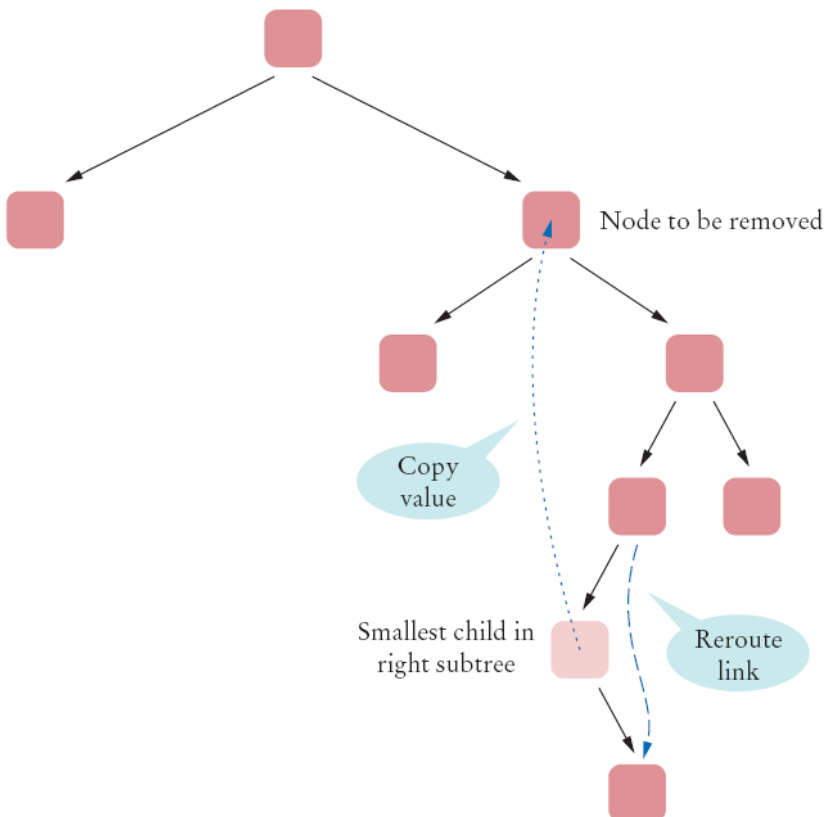


Figure 12 Removing a Node with Two Children

C) Balanced Binary Search Trees - each node has approximately as many descendants on the left as on the right

Note: If the tree is unbalanced, insertion can be slow, perhaps as slow as insertion into a linked list.

ex.1) an unbalanced binary search tree

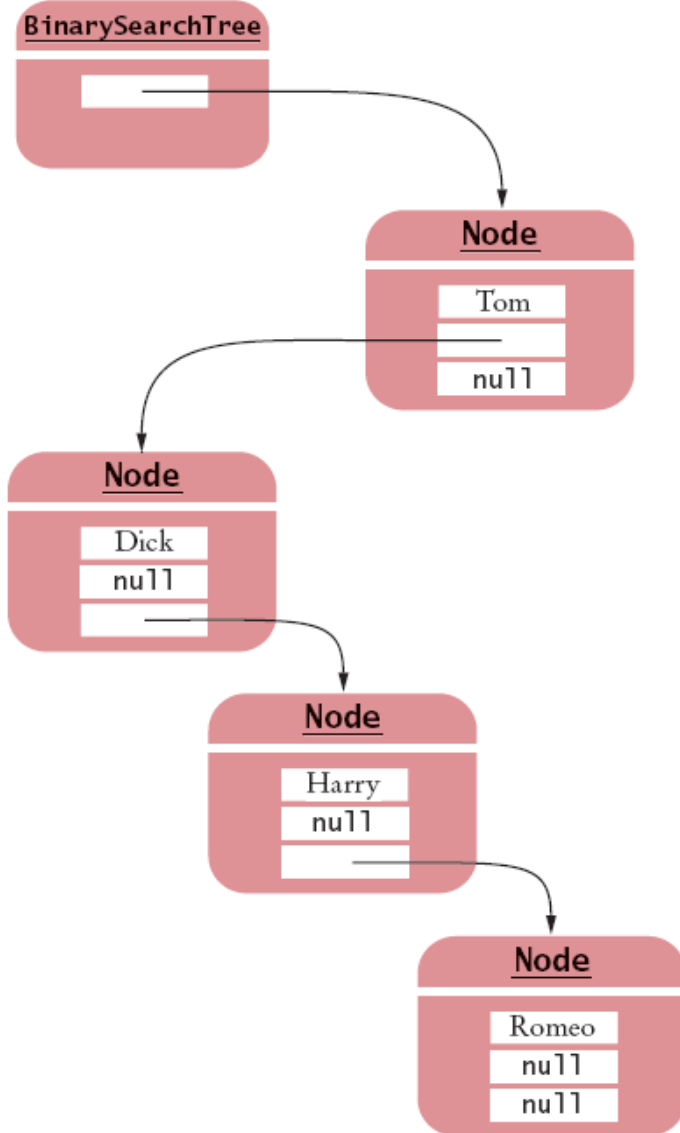


Figure 13 An Unbalanced Binary Search Tree

D) Tree Transversals

1. Inorder traversal

- Visit the left subtree
- Visit the root
- Visit the right subtree

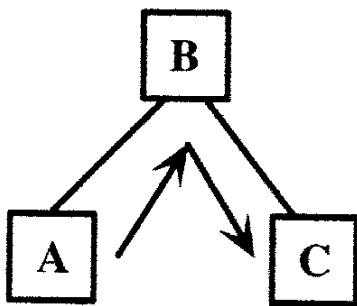
2. Preorder traversal

- Visit the root
- Visit the left subtree
- Visit the right subtree

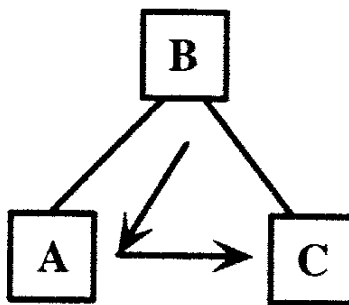
3. Postorder traversal

- Visit the left subtree
- Visit the right subtree
- Visit the root

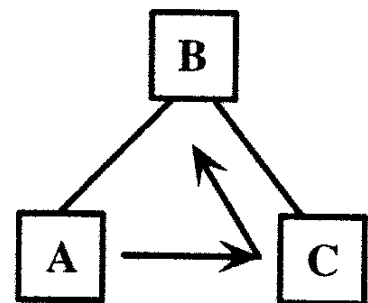
ex.1)



In-order
A - B - C



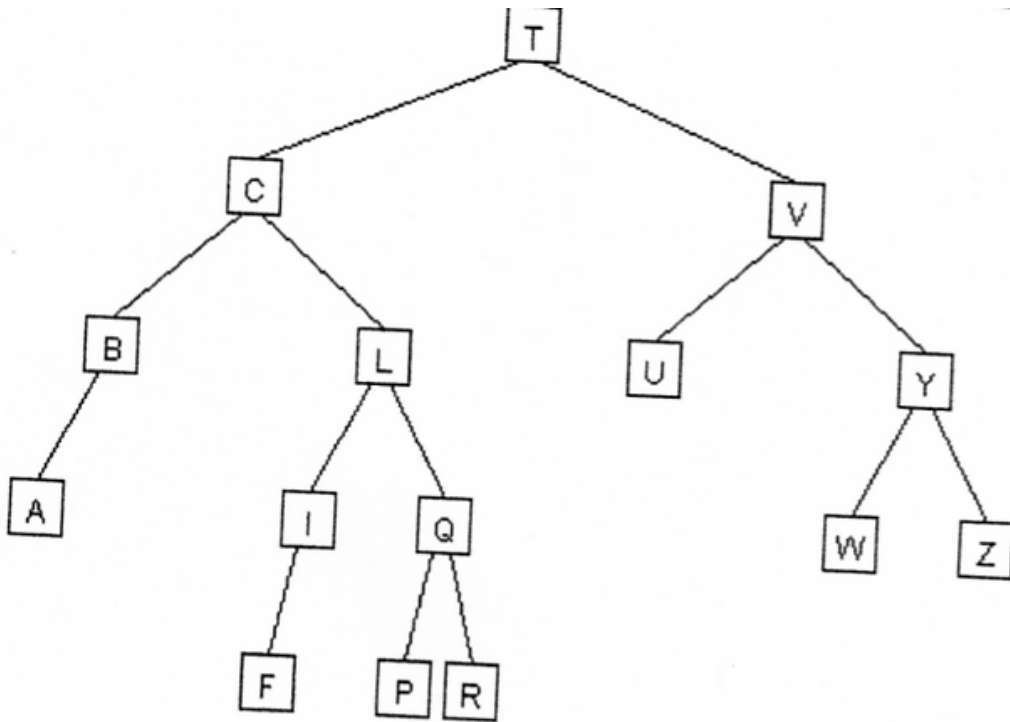
Pre-order
B - A - C



Post-order
A - C - B

AP Programming - Chapter 20 Lecture

ex.2) List the nodes in the figure below using an inorder, a preorder, and a postorder traversal.



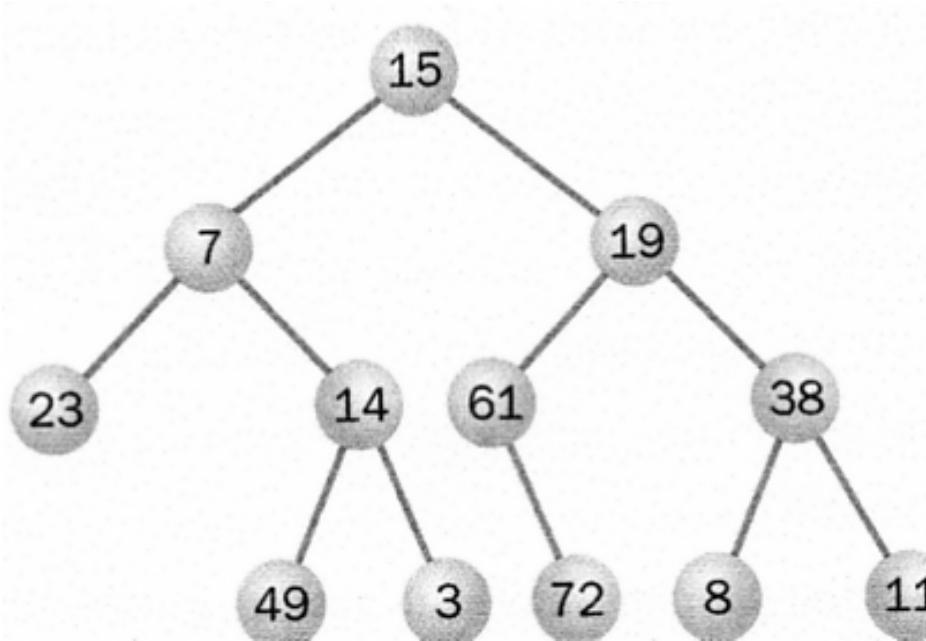
InOrder: A B C F I L P Q R T U V W Y Z

PreOrder: T C B A L I F Q P R V U Y W Z

PostOrder: A B F I P R Q L C U W Z Y V T

AP Programming - Chapter 20 Lecture

ex.2) List the nodes in the figure below using an inorder, a preorder, and a postorder traversal.



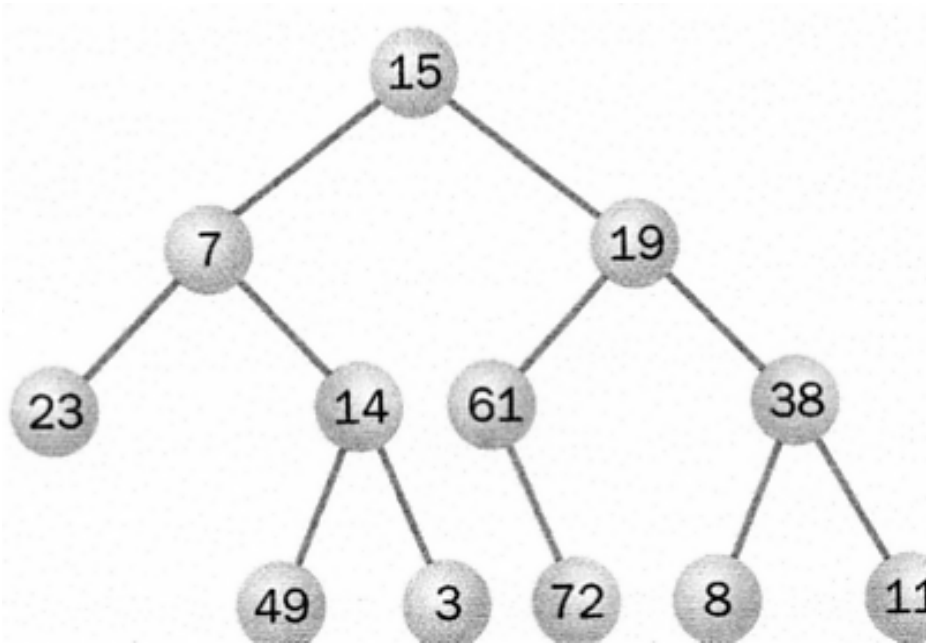
InOrder: 23 7 49 14 3 15 61 72 19 8 38 11

PreOrder: 15 7 23 14 49 3 19 61 72 38 8 11

PostOrder: 23 49 3 14 7 72 61 8 11 38 19 15

AP Programming - Chapter 20 Lecture

ex.2) List the nodes in the figure below using an inorder, a preorder, and a postorder traversal.



ex.3) Postorder traversal of an expression tree yields the instructions for evaluating the expression on a stack-based calculator

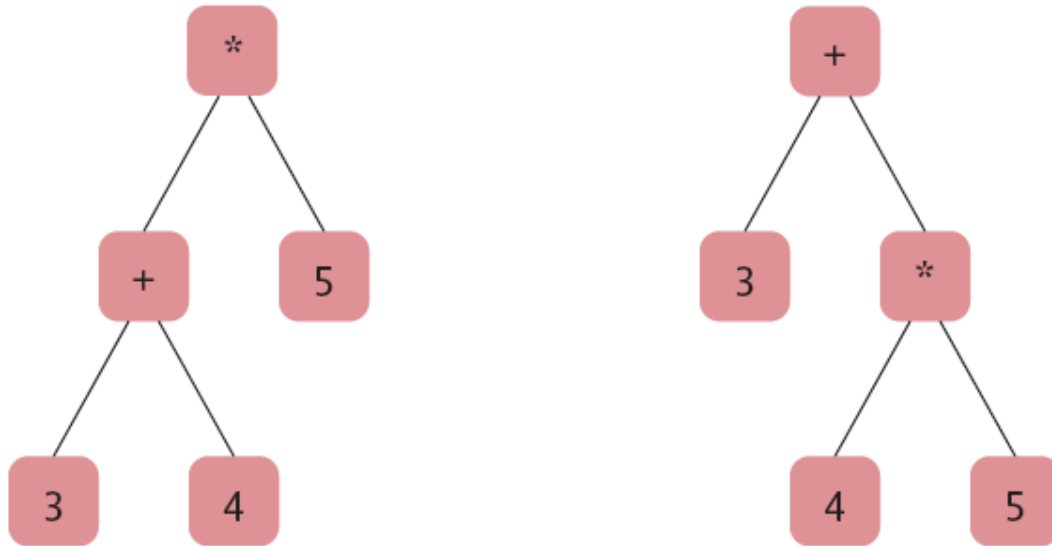


Figure 14 Expression Trees

The first tree ((3 + 4) * 5) yields: 3 4 + 5 *

Whereas the second tree (3 + 4 * 5) yields: 3 4 5 * +

4) Sorted Binary Tree Traversal using Inorder traversal:

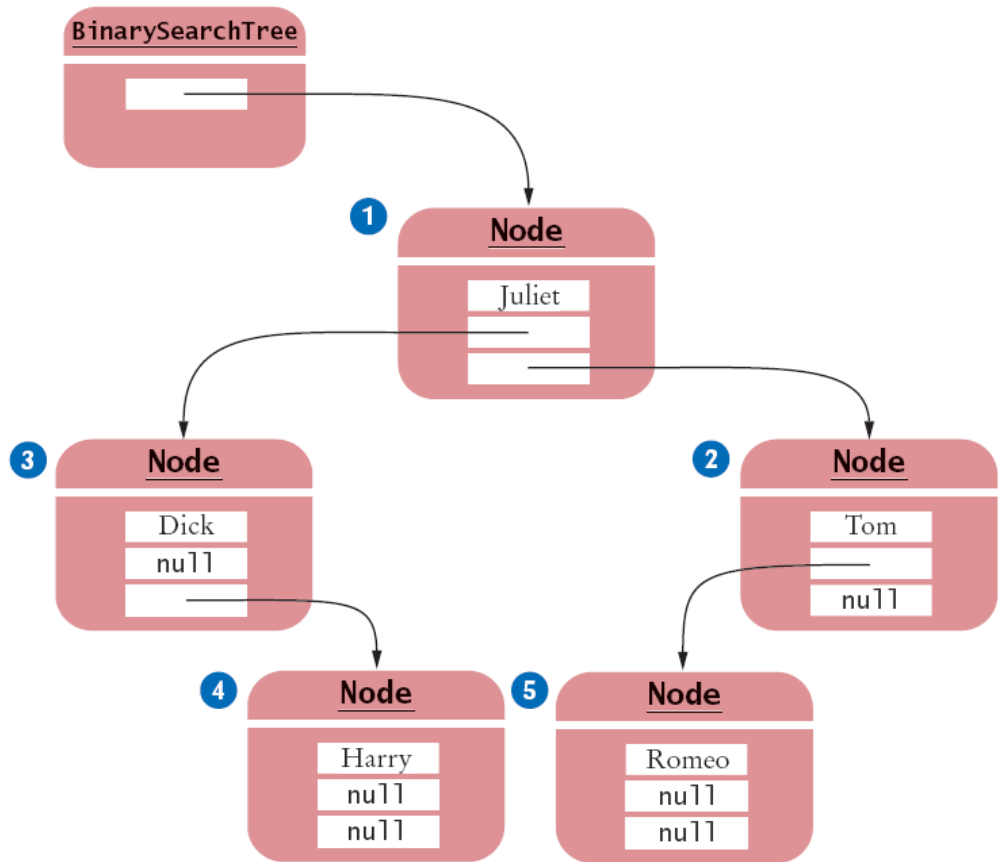


Figure 10
Binary Search Tree
After Five Insertions

- Let's try this out with the tree in Figure 10. The algorithm tells us to
 1. Print the left subtree of Juliet; that is, Dick and descendants
 2. Print Juliet
 3. Print the right subtree of Juliet; that is, Tom and descendants
- How do you print the subtree starting at Dick?
 1. Print the left subtree of Dick. There is nothing to print
 2. Print Dick
- Print the right subtree of Dick, that is, Harry

Output: Dick Harry Juliet Romeo Tom

*The data comes out in sorted order.