IX. Binary Trees (Chapter 10)

A. Introduction: Searching a linked list.

1. Linear Search

/* To linear search a list for a particular Item */
1. Set Loc = 0;
2. Repeat the following:
    a. If Loc >= length of list
        Return –1 to indicate Item not found.
    b. If list element at location Loc is Item
        Return Loc as location of Item
    c. Increment Loc by 1.

Linear search can be used for lists stored in an array as well as for linked lists. (It's the method used in the `find` algorithm in STL.) For a list of length n, its average search time will be _______________.

2. Binary Search

If a list is ordered, it can be searched more efficiently using binary search:

/* To binary search an ordered list for a particular Item */

1. Set First = 0 and Last = Length of List – 1.
2. Repeat the following:
    a. If First > Last
        Return –1 to indicate Item not found.
    b. Find the middle element in the sublist from locations First through Last
        and its location Loc.
    c. If Item < the list element at Loc
        Set Last = Loc – 1. // Search first half of list
      Else if Item > the list element at Loc
        Set First = Loc + 1. // Search last half of list
      Else
        Return Loc as location of Item
  End Loop

Since the size of the list being searched is reduced by approximately 1/2 on each pass through the loop, the number of times the loop will be executed is __________________ .

It would seem therefore that binary search is much more efficient than linear search. This is true for lists stored in arrays in which step 2b can be done simply by calculating Loc = (First + Last) / 2 and Array[Loc] is the middle list element.

For linked lists, however, binary search is not practical, because we only have direct access to the first node, and locating any other node requires traversing the list until that node is located. Thus step 2b requires:

i. Mid = (First + Last) / 2
ii. Set LocPtr = First;
iii. For Loc = First to Mid - 1
    Set LocPtr = Next part of node pointed to by LocPtr.
iv. Loc is the location of the middle node and the Data part of the node
    pointed to LocPtr is the middle list element.

The traversal required in step iii to locate the middle node clearly negates the efficiency of binary search for array-based lists; the computing time becomes O(n) instead of $O(\log_2 n)$.

However, perhaps we could modify the linked structure to make a binary search feasible. What would we need?

Direct access to the middle node:

22 33 44 55 66 77 88

and from it to the middle of the first half and to the middle of the second half, and so on:

22 33 44 55 66 77 88

and so on:

22 33 44 55 66 77 88

Or if stretch out the links to give it a ___________-like shape:

That is, use a ____

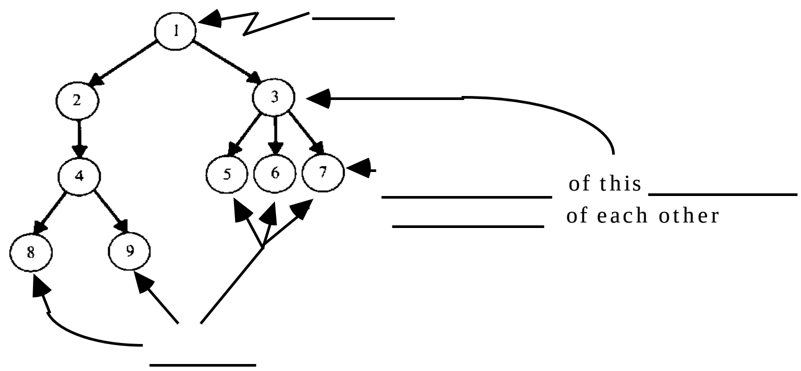
B. Binary Search Trees

1. Definition and Terminology:

A **tree** consists of a finite set of elements called **nodes** (or **vertices**) and a finite set of **directed arcs** that connect pairs of nodes. If the tree is not empty, then one of the nodes, called the **root**, has no incoming arcs, but every other node in the tree can be reached from the root by a unique path (a sequence of consecutive arcs).

A **leaf** is a node with no outgoing arcs.

Nodes directly accessible (using one arc) from a node are called the **children** of that node, which is called the **parent** of these children; these nodes are **siblings** of each other.

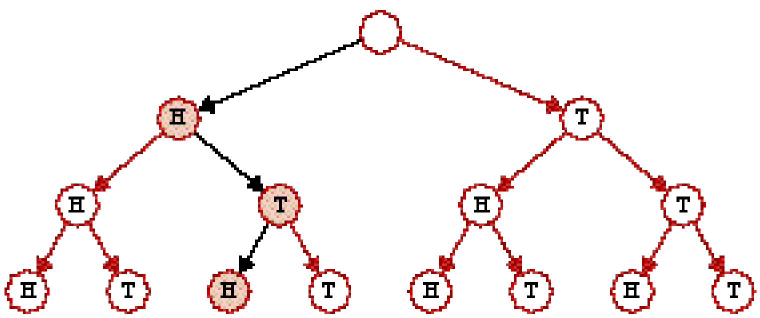


2. Examples

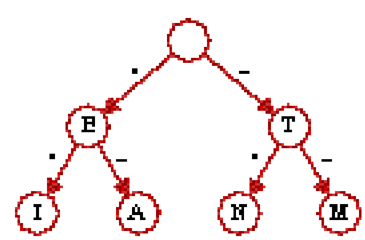
Game trees

Parse trees

Morse code trees



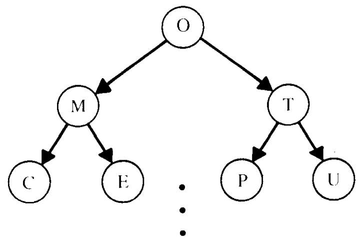
A	·–	M	––	Y	–·––
B	–···	N	–·	Z	––··
C	–·–·	O	–––	1	·––––
D	–··	P	·––·	2	··–––
E	·	Q	––·–	3	···––
F	··–·	R	·–·	4	····–
G	––·	S	···	5	·····
H	····	T	–	6	–····
I	··	U	··–	7	––···
J	·–––	V	···–	8	–––··
K	–·–	W	·––	9	––––·
L	·–··	X	–··–	0	–––––



3. Def: A **binary tree** is a tree in which ____

4. Array-Based Implementation:

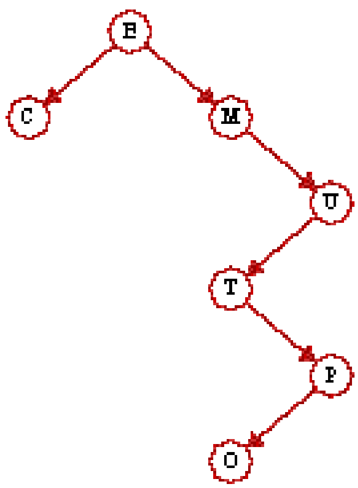
An array can be used to store some binary trees. In this scheme, we just number the nodes level by level, from left to right,



and store node #0 in array location 0, node #1 in array location 1, and so on:

i								. . .
$T[i]$								. . .

However, unless each level of the tree is full so there are no "dangling limbs," there can be much wasted space in the array. For example,

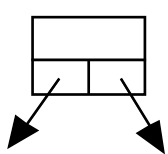


contains the same characters as before but requires 58 array positions for storage:

i	0	1	2	...		...		...		...	
$T[i]$	E	C	M	...	U	...	T	...	P	...	O

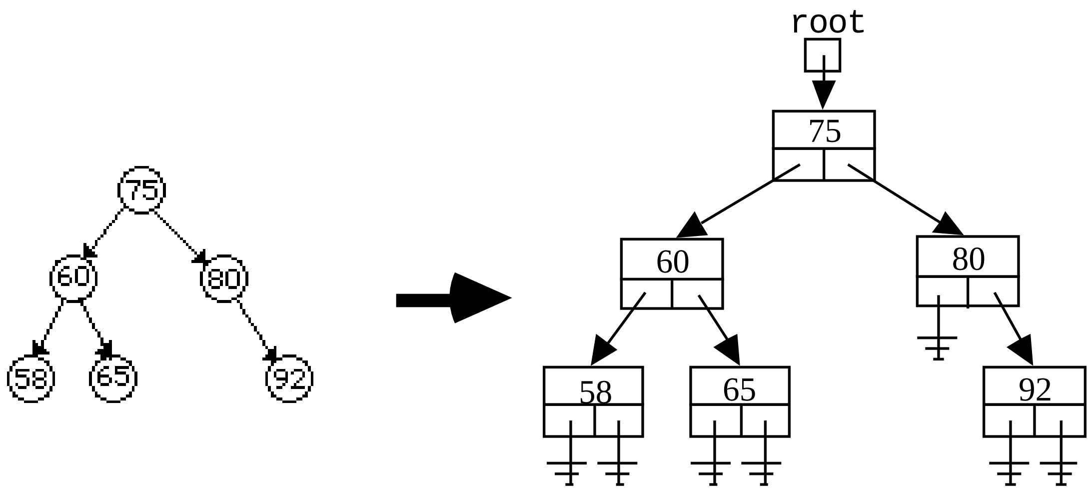
5. Linked Implementation:

Use nodes of the form



and maintain a pointer to the root.

Example:



b. C++ Implementation:

```
template <typename BinTreeElement>

class BinaryTree
{
 public:
  // ... BinaryTree function members

 private:
  class BinNode                   // a binary tree node
  {
  public:







     // ... Node member functions
    };

  typedef BinNode * BinNodePointer;  // an easy-to-read alias type

  // BinaryTree data members

  BinNodePointer root;              // pointer to the root node

};
```

5. Def. A **Binary Search Tree (BST)** is a binary tree in which the <u>value in each node is</u> ________________________

____

a. We can "binary search" a BST:

1. Set pointer locPtr = root.

2. Repeat the following:
   If locPtr is null

   ____

   If value < locPtr ->data

   ____

   Else if value > locPtr ->data

   ____

   Else

   ____

<u>Search time</u>: ____

b. What about traversing a binary tree?

This is most easily done recursively, viewing a binary tree as a *recursive data structure:*

<u>Recursive definition of a binary tree</u>:
A binary tree either:
i. is empty ← ¾¾¾¾¾¾¾¾¾¾¾¾ Anchor
or
ii. consists of a node called the root, which has \
pointers to two disjoint binary subtrees | ← ¾¾ Inductive step
called the *left subtree* and the *right subtree.* /

Now, for traverssal, consider the three operations:

V: Visit a node.

L: (Recursively) traverse the left subtree of a node.

R: (Recursively) traverse the right subtree of a node.

We can do these in six different orders: LVR, VLR, LRV, VRL, RVL, and RLV

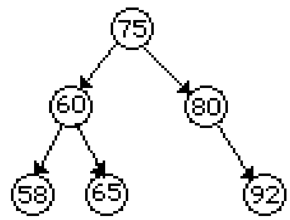
For example, LVR gives the following <u>traversal algorithm</u>:

If the binary tree is empty then // anchor
Do nothing.
Else do the following: // inductive step

<u>As a member function in a `BinaryTree` class:</u>

Rearranging the steps L, V, and R gives the other traversals.

Example:



LVR:

VLR:

LRV:

The first three orders, in which the left subtree is traversed before the right, are the most important of the six traversals and are commonly called by other names:

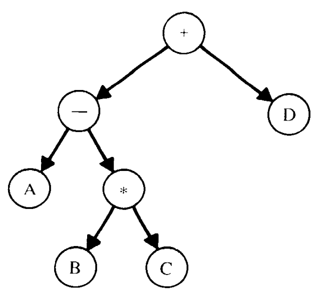
LVR ↔ _______________

VLR ↔ _______________

LRV ↔ ________________

Note: Inorder traversal of a BST visits the nodes ____.

To see why these names are appropriate, recall *expression trees*, binary trees used to represent the arithmetic expressions like A – B * C + D:



Inorder traversal → infix expression: ______________________________

Preorder traversal → prefix expression: ___________________________

Postorder traversal → postfix expression: __________________________

c. So how do we insert in a binary tree so it grows into a BST?

Modify the search algorithm so that a pointer *parentPtr* trails *locPtr* down the tree, keeping track of the parent of each node being checked:

1. Initialize pointers *locPtr* = *root, parentPtr* = NULL.
2. While *locPtr* ≠ NULL:
   a. *parentPtr* = *locPtr*
   b. If *value* < *locPtr->Data*
      locPtr = *locPtr->Left*
   Else if *value* > *locPtr->Data*
      locPtr = *locPtr->Right*
   Else
      value is already in the tree; return a found indicator.
3. Get a new node pointed to by *newPtr*, put the *value* in its *data* part,
   and set *left* and *right* to null.
4. if *parentPtr* = NULL // empty tree
   Set *root* = *newptr.*
   Else if *value* < *parentPtr->data*
   Set *parentPtr->left* = *newPtr.*
   Else
   Set *parentPtr->right* = *newPtr.*

Examples:

Insert in the order given:
M, O, T, H, E, R

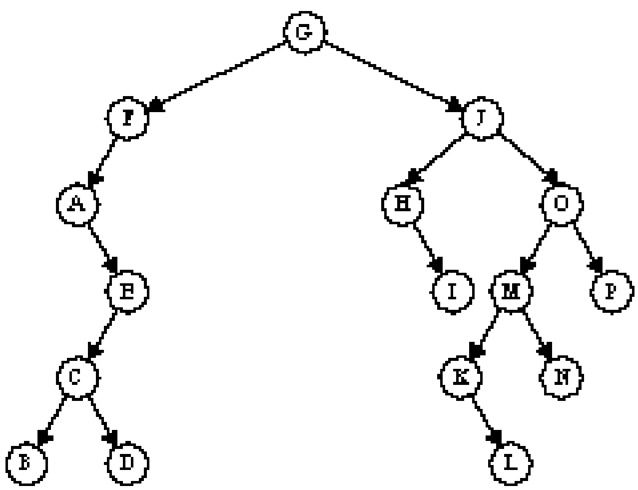
Insert in the order given:
T, H, E, R, M, O

Insert in the order given:
E, H, M, O, R, T

d. What about deleting a node a BST?

Case 1: A leaf, and Case 2: 1 child Easy — just reset link from parent

Case 3: 2 children: 1. Replace node with inorder successor X.
2. Delete X (which has 0 or 1 child)



Some Special Kinds of Trees:

- AVL Trees
- Threaded Binary Search Trees
- Tries
- B-Trees
- Huffman Code Trees (data compression)