TREES

Contents:
Binary tree representations
   Threaded binary trees
   Traversal using a father field
   Heterogeneous binary trees
Representing lists as binary trees
   Finding the k\textsuperscript{th} element
   Deleting an element
   Implementing tree-represented lists in C
   Constructing a tree-represented list
Trees and their applications
   C representations of trees
   Tree traversals
   General expressions as trees
   Evaluating an expression tree
   Constructing a tree

In tree traversals, stack is needed. Instead, replace NULL pointer in right field, with the pointer to the node that would be on top of the stack at that time. A pointer to its inorder successor can be attached. This pointer is called a thread and must be differentiable from a tree pointer. Note that the rightmost node in each tree still has a NULL right pointer, since it has no inorder successor. In this way, we can achieve a \textit{Right in-threaded binary trees}. Similarly, \textit{left in-threaded binary tree}, each NULL left pointer is altered to node’s inorder predecessor.

\textit{In-threaded binary tree}, has both left in-threaded and right in-threaded. Also, right and left \textit{pre-threaded binary trees}, in which NULL right and left pointers of nodes are replaced by their preorder successors and predecessors respectively.
**Traversal using a father field**

If each tree node contains a father field, neither a stack nor threads are necessary for nonrecursive traversal. Make corresponding changes in traversal algorithms.

**Heterogeneous binary trees**

A binary tree whose leaves contain numbers, but whose nonleaf nodes contain characters representing operators. Some examples are considered to represent the mathematical expressions in heterogeneous binary trees.

**Representing lists as binary trees**

Elements of the original list are represented by leaves of the tree, whereas nonleaf nodes of the tree are present as part of the internal tree structure. Each leaf nodes will have the contents of the corresponding list element. Each nonleaf nodes – contains a count representing the number of leaves in the node’s left subtree. Note that several binary trees can represent the same list.

**Finding the kth element**

It is possible to find specific element of a list in the tree, as per the following routine.

```
    r=k;
    p=tree;
    while(p is not a leaf node)
    if(r<=lcount(p))
       p=left(p));
    else{
       r=r-lcount(p);
       p=right(p);
    }
    find=p;
```

The number of tree nodes examined in finding the kth list element is less than or equal to 1 more than the depth of the tree.

**Deleting an element**

Deleting an element should make sure that the corresponding non leaf entries are updated. The deletion algorithm with corresponding illustration has been discussed.
Implementing tree-represented lists in C

Learn about the C routines like `findelement(int k)` and `delete(int p)`.

Constructing a tree-represented list

To construct a tree from a list of input data, we have the C routine `buildtree(int n)`

Tree and their applications

We understood different terminologies involved in trees such as: Tree, root, node, father, son, brother, ancestor, descendent, level, depth, degree of a node. An ordered tree is defined as a tree in which the sub trees of each node form an ordered set. Oldest son, youngest son also exists. A forest is an ordered set of ordered trees. These terminologies are extended for other forms of trees also. Degree of a node is the number of its sons. Multiple trees may be equivalent to each other. An ordered tree may involve first, second or last son of a particular node.

C representations of trees

The C representation for ordered tree is:

```c
#define MAXSONS 20
struct treenode{
    int info;
    struct treenode *father;
    struct treenode *sons[MAXSONS];
};
```

Its representation in lists is possible. When the number of sons increases more than 20, then, preferably list representation is used. A binary tree may be used to represent an entire forest, since the next pointer in the root of a tree can be used to point to the next tree of the forest.

Tree traversals

- **Preorder**
  - Visit the root of the first tree in the forest.
  - Traverse in preorder the forest formed by the subtrees of the first tree, if any.
  - Traverse in preorder the forest formed by the remaining trees in the forest, if any.

- **Inorder**
Traverse in inorder the forest formed by the subtrees of the first tree in the forest, if any.
Visit the root of the first tree.
Traverse in inorder the forest formed by the remaining trees in the forest, if any.

• **Postorder**
  Traverse in postorder the forest formed by the subtrees of the first tree in the forest, if any.
  Traverse in postorder the forest formed by the remaining trees in the forest, if any.
  Visit the root of the first tree in the forest.

**General expressions as trees**
Tree representation of an arithmetic expression, can be seen as an example. Preorder, inorder and postorder tree traversals are been illustrated with the tree representations.

**Evaluating an expression tree**
Representing an expression tree and C routines for `replace(NODEPTR p)` and `evaltree(NODEPTR p)` are considered.

**Constructing a tree**
The operation `setsons` accepts a pointer to a tree node with no sons and a linear list of nodes linked together through the `next` filed. `setsons` establishes the nodes in the list as the node in the tree. Another common operation is `addson(p,x)`, in which `p` points to a node in a tree and it is desired to add a node containing `x` as the youngest son of `node(p)`.

**Exercises**

*****