General Trees

Shiv Shankar Dayal

Theory

A *tree* is finite nonempty set of elements in which on element is called the *root* and the remaining elements are partitioned into $m > 0$ disjoint subsets, each of which is itself a tree. Each element in a tree is called a *node* of a tree. Each node may be the root of a tree with zero of more subtrees. A node with no subtrees is a *leaf*. We use the terms *father, son, brother, ancestor, descendant, level* and *depth* in the same sense as binary trees. **Degree** of a node in a tree is the no. of its sons. An **ordered tree** is defined as a tree in which the subtrees of each node form an ordered set. The first son of a node in an ordered tree is often called the **oldest** son of that node, and the last son is called the **youngest**. A **forest** is an ordered set of ordered trees.

Every binary tree except for the empty binary tree is indeed a tree. However, not every tree is binary. A tree node may have more than two sons, whereas a binary tree node may not. Even a tree whose nodes have at most two sons is not necessarily a binary tree. This is because an only son in a general tree is not designated as being a "leaf" or a "right" son, whereas in a binary tree, every son must be either a left son or a right son.

```
#define MAXSONS 20

struct treenode {
  int info;
  struct treenode father;
  struct treenode *sons[MXSONS].
}
// linking all nodes in a linked list
#define MAXNOOES 500

struct treenode {
  int info;
  int father;
  int son; // points to oldest son
  int next;
};

struct treenode node[MAXNODES];
```

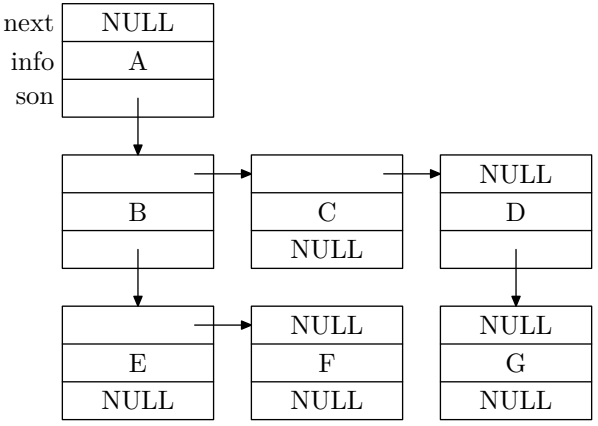
Theory contd.

```
struct treenode {
  int info;
  struct treenode *father.
  struct treenode *son;
  struct treenode *next.
};

typedef struct treenode *NODEPTR;
```

Like binary trees if we traverse from root to sons then father field is not needed. Given below is the code for inorder traversal of such a tree.

```
void inorder_trav(NODEPTR root) {
  if(root != NULL) {
    inorder_trav(root->son);
    printf("%d\n", root->info);
    inorder_trav(root->next);
  }
}
```



Similarly we can draw inspiration from BST traversal and implement preorder and postorder traversal for such trees.

Constructing a Tree

We assume that father node is not needed.

```
// p is a pointer to a node and list is a linear list of nodes linked through the next fields.
void set_sons(NODEPTR p, NODEPTR list) {
  if(p == NULL) {
    printf("Invalid ndoe\n");
    exit(1);
  }

  if(p->son != NULL) {
    printf("p must have a son.\n");
    exit(2);
  }

  p->son = list;
}
```

Constructing a Tree

Now we will add a son to a list as youngest son.

```
void addson(NODEPTR p, int x) {
  NODEPTR q;

  if(p == NULL) {
    printf("Invalid ndoe\n");
    exit(1);
  }

  NODEPTR r = NULL;
  q = p->son;

  while(q != NULL) {
    r = q;
    q = q->next;
  }

  q = (void*)malloc(sizeof(NODEPTR));
  q->info = x;
  q->next = NULL

  if(r == NULL)
    p->son = q;
  else
    r->next = q;
}
```