BFS

```
 void BFS()
    {
        queue<node *> q;
        q.push(root);
        while (!q.empty())
        {
            node *a = q.front();
            q.pop();
            int p = -1, l = -1, r = -1;
            if (a->left != 0)
            {
                l = a->left->id;
                q.push(a->left);
            }
            if (a->right != 0)
            {
                r = a->right->id;
                q.push(a->right);
            }
            if (a->parent != 0)
            {
                p = a->parent->id;
            }
            cout << "ID: " << a->id << " LEFT: " << l << " Right: " << r << " Parent: " << p << endl;
        }
    }
```

preorder:

```
 void DFS_Preorder(node *to)
    {
        if (to == 0)
        {
            return;
        }
        int p, l, r;
        p = l = r = -1;
        if (to->left != 0)
        {
            l = to->left->id;
        }
        if (to->right != 0)
        {
            r = to->right->id;
        }
        if (to->parent != 0)
        {
            p = to->parent->id;
        }
        cout << "----ID: " << to->id << " Left: " << l << " Right: " << r << " Parent: " << p << endl;
        DFS_Preorder(to->left);
        DFS_Preorder(to->right);
    }
```

inorder:

```
 void DFS_Inorder(node *to)
    {
        if (to == 0)
        {
            return;
        }
        DFS_Inorder(to->left);

        int p, l, r;
        p = l = r = -1;
        if (to->left != 0)
        {
            l = to->left->id;
        }
        if (to->right != 0)
        {
            r = to->right->id;
        }
        if (to->parent != 0)
        {
            p = to->parent->id;
        }
        cout << "----ID: " << to->id << " Left: " << l << " Right: " << r << " Parent: " << p << endl;
        DFS_Inorder(to->right);
    }
```

postorder:

```
 void DFS_PostOrder(node *to)
    {
        if (to == 0)
        {
            return;
        }
        DFS_PostOrder(to->left);
        DFS_PostOrder(to->right);
        int p, l, r;
        p = l = r = -1;
        if (to->left != 0)
        {
            l = to->left->id;
        }
        if (to->right != 0)
        {
            r = to->right->id;
        }
        if (to->parent != 0)
        {
            p = to->parent->id;
        }
        cout << "----ID: " << to->id << " Left: " << l << " Right: " << r << " Parent: " << p << endl;
    }
```

insertion:

```
    void insertion(int id)
    {
        node *newnode = newnodes(id);
        if (root == 0)
        {
            root = newnode;
            return;
        }
        queue<node *> q;
        q.push(root);
        while (!q.empty())
        {
            node *a = q.front();
            q.pop();
            if (a->left != 0)
            {
                q.push(a->left);
            }
            else
            {
                a->left = newnode;
                newnode->parent = a;
                return;
            }
            if (a->right != 0)
            {
                q.push(a->right);
            }
            else
            {
                a->right = newnode;
                newnode->parent = a;
                return;
            }
        }
    }
```

Search:

```
   bool search(int val)
    {
        if (root == 0)
        {
            return false;
        }
        queue<node *> q;
        q.push(root);
        while (!q.empty())
        {
            node *a = q.front();
            q.pop();
            if (a->val == val)
            {
                cout << "Found" << endl;
                return true;
            }
            if (a->left != NULL)
            {
                q.push(a->left);
            }
            if (a->right != 0)
            {
                q.push(a->right);
            }
        }
        cout << "Not Found" << endl;
        return false;
    }
    void search2(int val, node *a)
    {
        if (a == 0)
        {
            return;
        }
        if (a->val == val)
        {
            cout << "Found" << endl;
            return;
        }
        search2(val, a->left);
        search2(val, a->right);
    }
```

complete:

```
void complete()
    {
        if (root == 0)
        {
            cout << "Complete" << endl;
            return;
        }

        queue<node *> q;
        q.push(root);

        bool seenNull = false;

        while (!q.empty())
        {
            node *current = q.front();
            q.pop();

            // If a null node has been seen and current is not a null node, it’s not a complete binary tree
            if (seenNull && current != 0)
            {
                cout << "Not a complete BG" << endl;
                return;
            }

            if (current == 0)
                seenNull = true;
            else
            {
                q.push(current->left);
                q.push(current->right);
            }
        }
        cout << "complete" << endl;
    }
```

perfect:

```
void perfect()
    {
        if (root == 0)
        {
            cout << "Perfect" << endl;
            return;
        }
        queue<node *> q;
        q.push(root);
        while (!q.empty())
        {
            node *a = q.front();
            q.pop();
            if (a->left != 0 && a->right != 0)
            {
                q.push(a->left);
                q.push(a->right);
            }
            else if (a->left == 0 && a->right == 0)
            {
            }
            else
            {
                cout << "Not a perfect Tree" << endl;
                return;
            }
        }
        cout << "perfect" << endl;
    }
```

traversal:

```
 void bfs()
    {
        if (root == 0)
        {
            return;
        }
        queue<node *> q;
        q.push(root);
        while (!q.empty())
        {
            node *a = q.front();
            q.pop();
            cout << a->data << " ";
            if (a->left != NULL)
            {
                q.push(a->left);
            }
            if (a->right != NULL)
            {
                q.push(a->right);
            }
        }
    }
    void inorder(node *q)
    {
        if (q == 0)
        {
            return;
        }
        inorder(q->left);
        cout << q->data << " ";
        inorder(q->right);
    }
    void postorder(node *q)
    {
        if (q == 0)
        {
            return;
        }
        postorder(q->left);
        postorder(q->right);
        cout << q->data << " ";
    }
    void preorder(node *q)
    {
        if (q == 0)
        {
            return;
        }
        cout << q->data << " ";
        preorder(q->left);
        preorder(q->right);
    }
```

maximum:

```
 int maxi()
    {
        int ans = -1;
        if (root == 0)
        {
            return ans;
        }
        queue<node *> q;
        q.push(root);
        while (!q.empty())
        {
            node *a = q.front();
            q.pop();
            ans = max(ans, a->data);
            if (a->left != NULL)
            {
                q.push(a->left);
            }
            if (a->right != NULL)
            {
                q.push(a->right);
            }
        }
        return ans;
    }
```

deletetion:

```
void deletes(int val)
    {
        if (root == 0)
        {
            return;
        }
        if (root->data == val)
        {
            if (root->left == 0 && root->right == 0)
            {
                root = NULL;
                return;
            }
            if (root->left == 0)
            {
                root = root->right;
                return;
            }
            if (root->right == 0)
            {
                root = root->left;
                return;
            }
            node *tmp = root->right;
            while (tmp->left != 0)
            {
                tmp = tmp->left;
            }
            int x = tmp->data;
            deletes(x);
            root->data = x;
            return;
        }
        node *cur = root;
        node *prv = 0;
        while (cur != 0)
        {
            if (cur->data == val)
            {
                if (cur->left == 0 && cur->right == 0)
                {
                    if (prv->left != NULL && prv->left->data == val)
                    {
                        prv->left = 0;
                    }
                    else
                    {
                        prv->right = NULL;
                    }
                    delete cur;
                    return;
                }
                if (cur->left == 0)
                {

                    if (prv->left != NULL && prv->left->data == val)
                    {
                        prv->left = cur->right;
                    }
                    else
                    {
                        prv->right = cur->right;
                    }
                    delete cur;
                    return;
                }
                if (cur->right == 0)
                {

                    if (prv->left != NULL && prv->left->data == val)
                    {
                        prv->left = cur->left;
                    }
                    else
                    {
                        prv->right = cur->left;
                    }
                    delete cur;
                    return;
                }
                node *tm = cur->right;
                while (tm->left != 0)
                {
                    tm = tm->left;
                }
                int sav = tm->data;
                deletes(sav);
                cur->data = sav;
                return;
            }
            prv = cur;
            if (cur->data > val)
            {
                cur = cur->left;
            }
            else
            {
                cur = cur->right;
            }
        }
    }
```