6. 试编写算法求二叉树中双分支节点的个数。

```
template<class T>
int BiTree<T>::CountTwoBranch(BiNode<T>* p)
{
    if (p == nullptr)
        return 0;
    int left = CountTwoBranch(p->lchild);
    int right = CountTwoBranch(p->rchild);
    if (p->lchild && p->rchild)
        return 1 + left + right;
    else return 0 + left + right;
    }
template<class T>
int BiTree<T>::CountTwoBranch()
{
    return CountTwoBranch(root);
}
```

7. 试编写算法求二叉树中各个结点的平衡因子（左右子树高度之差）

```
int BiTree<T>::Height(BiNode<T>* p)
{
    if (p == nullptr)
        return 0;
    int left = Height(p->lchild);
    int right = Height(p->rchild);
    if (left > right)
        return left + 1;
    else return right + 1;
}
template<class T>
void BiTree<T>::BalanceFactor(BiNode<T>* p)
{
    if (p == nullptr)
        return;
    if (p->lchild || p->rchild)
    {//先序
        cout << Height(p->lchild) - Height(p->rchild) << " ";
        BalanceFactor(p->lchild);
        BalanceFactor(p->rchild);
    }
    else cout << "0 ";
}
template<class T>
```

```
void BiTree<T>::BalanceFactor_TraverseByPre()
{
    BalanceFactor(root);
}
```

8. 一棵二叉树以二叉链表来表示，求其指定的某一层 k(k>1)上的叶子结点的个数。

```
template<class T>
int BiTree<T>::CountLeafOnLevel(int level)
{
    int ilevel = 1;
    int cnt = 0;
    if (root == nullptr)
        return 0;
    queue<BiNode<T>*> Q;
    Q.push(root);
    BiNode<T>* endlevel=Q.back();//定义每层终止结点
    while (Q.size())
    {
        BiNode<T>* p = Q.front();
        if (ilevel == level)
            if (!(p->lchild || p->rchild))
                cnt++;
        Q.pop();
        if (p->lchild)
            Q.push(p->lchild);
        if (p->rchild)
            Q.push(p->rchild);
        if (Q.size() && p == endlevel)
        {//计算层数
            ilevel++;
            endlevel = Q.back();
        }
        if (ilevel > level)
            break;
    }
    return cnt;
}
```

9. 试编写算法输出一棵二叉树中根结点到各个叶子结点的路径。

```
template<class T>
void BiTree<T>::RootLeafPath(BiNode<T>* p, vector<T>& path)
{
    if (p == nullptr)
        return;
```

```
    path.push_back(p->data);
    if (!p->lchild && !p->rchild)
    {
        int len = path.size();
        for (int i = 0; i < len; i++)
            cout << path[i] << " ";
        cout << endl;
    }
    else
    {
        RootLeafPath(p->lchild, path);
        RootLeafPath(p->rchild, path);
    }
    path.pop_back();//注意此时递归弹出
}
template<class T>
void BiTree<T>::RootLeafPath_TraverseByPre()
{
    vector<T> path;
    RootLeafPath(root, path);
}
```

10. 设计一个算法，求二叉树中两个给定结点的最近公共祖先。

```
//仅适用于结点数值不同的情况
//先求从根到所给结点的路径再对比两路径
template<class T>
void BiTree<T>::RootAnyonePath(BiNode<T>* p,BiNode<T>* e, vector<T>& path, vector<T>&
result)
{
    if (p == nullptr)
        return;
    path.push_back(p->data);
    if (p == e)
    {
        int len = path.size();
        for (int i = 0; i < len; i++)
            result.push_back(path[i]);
//将path存到result，不然会pop掉，或者考虑强制跳出递归
    }
    else
    {
        RootAnyonePath(p->lchild, e, path, result);
        RootAnyonePath(p->rchild, e, path, result);
    }
```

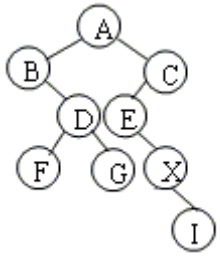
```
    path.pop_back();
}
template<class T>
BiNode<T>* BiTree<T>::NearestCommonAncestor(BiNode<T>* p, BiNode<T>* a, BiNode<T>* b)
{
    vector<T> path1, result1;
    vector<T> path2, result2;
    RootAnyonePath(root, a, path1, result1);
    RootAnyonePath(root, b, path2, result2);
    for (int i = result1.size() - 1; i >= 0; i--)
    {
        for (int j = result2.size() - 1; j >= 0; j--)
            if (result2[j] == result1[i])
                return Search(result1[j]);
//若要免去此处搜索可将前处找路径改为使用结点的vector,也可改变返回值为data而不是结点
    }
}
template<class T>
BiNode<T>* BiTree<T>::NearestCommonAncestor(T e1, T e2)
{
    return NearestCommonAncestor(root, Search(e1), Search(e2));
}
```

补充作业（选做题）：

11. 若一棵二叉树中没有数据域值相同的结点，试设计算法打印二叉树中数据域值为 x 的结点的所有祖先结点的数据域。如果根结点的数据域值为 x 或不存在数据域值为 x 的结点，则什么也不打印。例如对下图所示的二叉树，则打印结点序列为 A、C、E。

```
//与上文中 RootAnyonePath 原理一样，只消修改 path 直接打印
template<class T>
void BiTree<T>::AllAncestor(BiNode<T>* p, BiNode<T>* e, vector<T>& path)
{
    if (p == nullptr)
        return;
    path.push_back(p->data);
    if (p == e)
    {
        int len = path.size();
        for (int i = 0; i < len - 1; i++)
            cout << path[i] << " ";
        cout << endl;
    }
    else
    {
```

```
        AllAncestor(p->lchild, e, path);
        AllAncestor(p->rchild, e, path);
    }
    path.pop_back();
}
template<class T>
void BiTree<T>::AllAncestor(BiNode<T>* e)
{
    vector<T> path;
    AllAncestor(root, e, path);
}
```



12. 已知二叉树存于二叉链表中，试编写一个算法，判断给定二叉树是否为完全二叉树。

```
template<class T>
bool BiTree<T>::isCompleteTree()
{
    bool mark = false;
    if (root == nullptr)
        return false;
    queue<BiNode<T>*> Q;//层序
    Q.push(root);
    while (Q.size())
    {
        BiNode<T>* p = Q.front();
        Q.pop();
        if (mark && (p->lchild || p->rchild))
                return false;
        if(!mark)//判断结点是否非饱和
        {
            if (p->lchild && p->rchild)
            {
                Q.push(p->lchild);
                Q.push(p->rchild);
            }
            if (p->lchild && !p->rchild)
            {
```

```
                Q.push(p->lchild);
                mark = true;
            }
            if (!p->lchild && p->rchild)
                return false;
            if (!p->lchild && !p->rchild)
                mark = true;
        }
    }
    return true;
}
```

13. 已知二叉树存于二叉链表中，编写一个递归算法，利用叶结点中空的右链指针域rchild，将所有叶结点自左至右链接成一个单链表，算法返回最左叶结点的地址（链头）

```
template<class T>
void BiTree<T>::LeafList(BiNode<T>* p, BiNode<T>*& head, BiNode<T>*& tmp)
{
    if (p == nullptr)
        return;
    if (!p->lchild && !p->rchild)
    {
        if (tmp)
        {
            tmp->rchild = p;
            tmp = p;
        }
        else
        {
            tmp = p;
            head = p;
        }
    }
    LeafList(p->lchild, head, tmp);
    LeafList(p->rchild, head, tmp);
    return;
}
template<class T>
BiNode<T>* BiTree<T>::LeafList()
{
    BiNode<T>* head = nullptr;
    BiNode<T>* tmp = nullptr;
    LeafList(root, head, tmp);
    return head;
}
```

14. 已知二叉树存于二叉链表中，试编写一个算法计算二叉树的宽度，即同一层中结点数的最大值。

```
template<class T>
int BiTree<T>::Width()
{
    int ilevel = 1;
    vector<int> width;
    if (root == nullptr)
        return 0;
    queue<BiNode<T>*> Q;
    Q.push(root);
    BiNode<T>* endlevel = Q.back();//定义每层终止结点
    while (Q.size())
    {
        while (ilevel >width.size())
            width.push_back(0);
        BiNode<T>* p = Q.front();
        Q.pop();
        width[ilevel - 1]++;
        if (p->lchild)
            Q.push(p->lchild);
        if (p->rchild)
            Q.push(p->rchild);
        if (Q.size() && p == endlevel)
        {
            ilevel++;
            endlevel = Q.back();
        }
    }
    return *max_element(width.begin(), width.end());
}
```