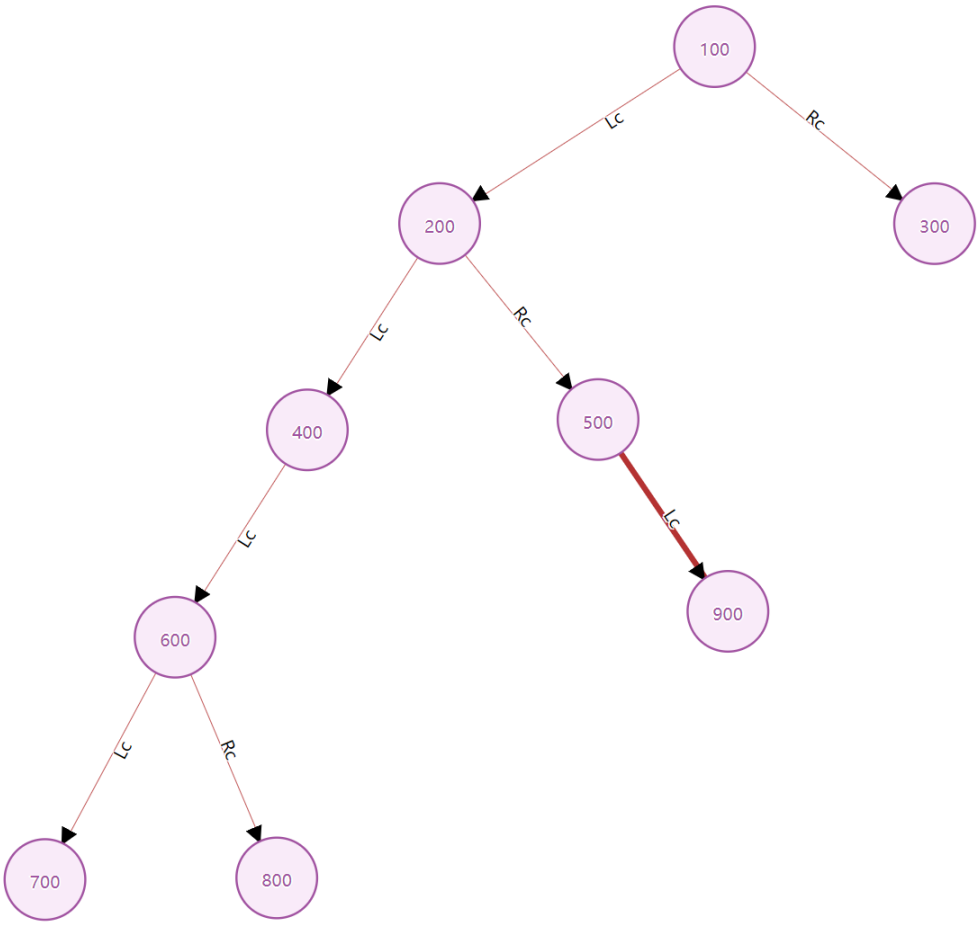
lab5

删除value=1000后的结果：



求id=4和id=9的最近公共祖先，最大最小差值，id=8的节点的路径：

```
测试二叉树生成 edge :测试删除value = x的结点及其子树:
请输入x:
1000
成功删除value = x的节点及其子树
再次遍历该二叉树:

*(addr = 671470):(id =  0) =   100,     其父节点id为0
*(addr = 6714c0):(id =  1) =   200,     其父节点id为671470
*(addr = 671560):(id =  3) =   400,     其父节点id为6714c0
*(addr = 671600):(id =  5) =   600,     其父节点id为671560
*(addr = 671650):(id =  6) =   700,     其父节点id为671600
*(addr = 6716a0):(id =  7) =   800,     其父节点id为671600
*(addr = 6715b0):(id =  4) =   500,     其父节点id为6714c0
*(addr = 6716f0):(id =  8) =   900,     其父节点id为6715b0
*(addr = 671510):(id =  2) =   300,     其父节点id为671470
测试寻找最近公共祖先:
请输入id1和id2:
3 8
结点id1和id2的最近公共祖先的id为:1

求二叉树中最大value与最小value之差:
最大值与最小值之差为: 800
求根节点到给定id值的路径:
请输入id值:
7
从根节点到id值结点的路径为:
左左左右
请按任意键继续. . .
```

注：老师的生成树的代码id是从0开始计数的，我尝试着修改了对应的代码，但是在修改后编译却在老师给的showgt.h里面报错了，关于这部分的代码我不知道该怎么改，所以我这里的id也是从0开始的。所以样例里要求4和9的最近公共祖先，我求的是3和8的，得到的答案是1，在样例里对应的就是2，后面输出路径我也是求的是id=7的节点的路径。

如何查询最近公共祖先： 在这里我首先修改了数据结构，添加了双亲指针,为此我写了一个AddParent函数，采用递归的方式为每一个节点添加双亲，并且在生成树的时候将每个结点的双亲都赋值为NULL，最后就只有根节点的双亲指针为空。（在这里也可以不添加双亲指针，采用老师代码中的parent函数来求双亲，我在代码中的相应部分都有注释。但是每次求双亲都会从根节点开始往下遍历，我的代码中又反复用到双亲，所以我就添加了双亲指针）。 然后我又写了一个函数isParent来判断t1是否为t2的祖先（默认两个结点不同）。 最后在求最近公共祖先之前，我先判断了输入的两个id是否合法，在这里我先写了一个getLength函数来求树中结点的数目length，也是通过递归的方式，输入的id应该在0-length-1之间才是合法的，如果不合法就直接输出下标不合法。

最后我在求最近公共祖先的SearchSameAncestor函数中首先判断了两个结点是否有根节点，如果有一个是根节点，那么根结点无祖先，直接返回-1表示错误（因为树的id最小为0，不会出现负值）。如果两个结点都不是根节点，再调用老师写的locateByID函数寻找两个id对应的结点，用t1,t2表示，然后我用了一个循环，循环中首先判断t1是否为t2的祖先，这里调用isParent函数，如果是，直接返回t1的id，如果不是，就将t1 = t1->parent，然后再判断t1是否为t2的祖先，直至t1为空，因为我在为结点赋双亲结点的时候，根节点的双亲指针没有赋值，是初始化时生成的空指针。但其实理论上来说不会到t1为空的时候，因为t1变成根节点的时候它一定是t2的祖先（在前面已经确定t2不是根节点的前提下）

下面是我的代码：

求最近公共祖先部分的函数：

```
//求二叉树的结点数目
int getLength(bNode *root)
{
    if(!root) return 0;
    else
        return 1 + getLength(root->lchild) + getLength(root->rchild);
}

void AddParent(bNode *root)
{
    if(root)
    {
        if(root->lchild){
            root->lchild->parent = root;
            AddParent(root->lchild);
        }
        if(root->rchild){
            root->rchild->parent = root;
            AddParent(root->rchild);
        }
    }
}
bNode *locateByID(bNode *root, int id){
    if (!root) return NULL;
    if (root->data.id==id) return root;
    bNode *t = locateByID(root->lchild,id);
    if (t) return t;
    return locateByID(root->rchild,id);
}
//判断t1是否为t2的祖先（默认两个结点不同）
int isParent(bNode *root,bNode *t1,bNode *t2)
{
    bNode *p = (bNode *)malloc(sizeof(bNode));
    if(!p)
    {
        printf("ERROR!");
        exit(0);
    }
    if(t1 == root)
    {
```

```
            return 1;
        }
        else
        {
            //p = parent(root,t2);
            p = t2->parent;
            while(p != root)
            {
                if(p == t1)
                    return 1;
                p = p->parent;
            }
        }
        return 0;
    }

    //寻找两个结点最近的公共祖先
    int SearchSameAncestor(bNode *root,int id1,int id2)
    {
        //如果id1，id2中有一个为根，那么它们无公共祖先，否则必有公共祖先（因为根即为它们的一个公共祖先）
        if(!id1 || !id2){
            printf("无公共祖先!\n");
            return -1;
        }
        bNode *t1 = (bNode *)malloc(sizeof(bNode));
        bNode *t2 = (bNode *)malloc(sizeof(bNode));
        if(!t1 || !t2){
            printf("ERROR!\n");
            return -1;//一个数的结点的id最小为根节点的0，故返回-1表示错误
        }
        //先查找id为id1，id2的两个结点
        t1 = locateByID(root,id1);
        t2 = locateByID(root,id2);
        //再查找这两个节点的父节点，若二者的父节点相同，则该父节点为最近的公共祖先，否则比较这两个父节点的父节点，以此类推
        while(t1)
        {
            //t1 = parent(root,t1);
            //t2 = parent(root,t2);
            if(isParent(root,t1,t2))
            {
                return t1->data.id;
            }
            t1 = t1->parent;
            //t2 = t2->parent;
        }
        return -1;
    }
```

其余部分：

```
//求二叉树节点中最大value与最小value
void SearchMaxandMin(bNode *root,int *max,int *min)
{
    if(root)
    {
        *max = (root->data.value > *max) ? root->data.value : *max;
        *min = (root->data.value < *min) ? root->data.value : *min;
        SearchMaxandMin(root->lchild,max,min);
        SearchMaxandMin(root->rchild,max,min);
    }
}

//求二叉树节点中最大value与最小value之差
int MaxDifferenceValue(bNode *root)
{
    if(!root)
    {
        printf("该二叉树为空树!\n");
        return -1;
    }
    int max = MIN_VALUE,min = MAX_VALUE;//max和min分别初始化为整型变量value所能达到的
最小值和最大值
    SearchMaxandMin(root,&max,&min);
    return (max - min);
}

void SearchPath(bNode *root,int id)
{
    if(!root)
    {
        printf("该二叉树为空树!\n");
        return;
    }
    else if(root->data.id == id)
    {
        printf("%d为根节点!\n");
        return;
    }
    int height = deepth(root), i = 0;
    int *path = (int *)malloc(sizeof(int)*height);//定义一个用于存放路径的“数组”
    /*for(i = 0;i < height;i++)
        path[i] = -1;
    i = 0;*/
    bNode *t = (bNode *)malloc(sizeof(bNode));
    bNode *p = (bNode *)malloc(sizeof(bNode));
    p = locateByID(root,id);
    t = p;
    while(t != root)
    {
        //t = parent(root,p);
        t = p->parent;
        if(p == t->lchild)
            path[i++] = 0;
```

```
            else
                path[i++] = 1;
            p = t;
        }
        printf("从根节点到id值结点的路径为:\n");
        for(i--;i >= 0;i--)
        {
            if(path[i] == 0)
                printf("左");
            else
                printf("右");
        }
        printf("\n");
    }

    bNode *LocateByValue(bNode *root,int value)
    {
        if(!root)   return NULL;
        if(root->data.value == value)   return root;
        bNode *t = (bNode *)malloc(sizeof(bNode));
        t = LocateByValue(root->lchild,value);
        if(t)   return t;
        return LocateByValue(root->rchild,value);
    }

    //删除value = x的节点及其子树
    bNode *DeleteByValue(bNode *root,int x)
    {
        bNode *t = (bNode *)malloc(sizeof(bNode));
        bNode *p = (bNode *)malloc(sizeof(bNode));
        int flag = 0;
        if(!t || !p){
            printf("ERROR!\n");
            system("pause");
            exit(0);
        }
        if(!LocateByValue(root,x)){
                printf("树中无value = %d的节点!\n",x);
                return root;
            }
        while(t = LocateByValue(root,x))
        {
            if(t == root){
                DestroyBTree(root);
                root = NULL;
                break;
            }
            //p = parent(root,t);
            p = t->parent;
            if(t == p->rchild)
                flag = 1;
            DestroyBTree(t);
            if(flag)
                p->rchild = NULL;
```

```
            else
                p->lchild = NULL;
        }
        printf("成功删除value = x的节点及其子树\n");
        return root;
    }
```