/\*树和森林\*/

//双亲表示法

#define MAX\_TREE\_SIZE 100

typedef struct {

ElemType data;

int parent;

} PTNode;

typedef struct {

PTNode nodes[MAX\_TREE\_SIZE];

int n;

} PTree;

//孩子兄弟表示法

typedef struct CSNode {

ElemType data;

struct CSNode \*firstchild, \*nextsibling;

} CSNode, \*CSTree;

//并查集

#define SIZE 100

int UFSets[SIZE];

//并查集的初始化操作（s即为并查集）

void Initial(int S[]) {

for (int i = 0; i < size; ++i) {

s[i] = -1;

}

}

//Find操作（函数在并查集S中查找并返回包含元素x的树的节点

int Find(int S[], int x) {

while (S[x] >= 0)

x = S[x];

return x;

}

//Union操作（函数要求两个不相交的子集）

void Union(int S[], int Root1, int Root2) {

//要求Root1和Root2是不同的，且表示子集的名字

S[Root2] = Root1;

}

//求以孩子兄弟表示法存储的树的高度

int Height(CSTree bt) {

int hc, hs;

if (bt == NULL)

return 0;

else {

hc = Height(bt->firstchild);

hs = Height(bt->nextsibling);

if (hc + 1 > hs)

return hc + 1;

else

return hs;

}

}

//求以孩子兄弟表示法存储的森林的叶子结点数

//以孩子兄弟表示法存储时，若结点没有左子树，则为叶结点

typedef struct node {

ElemType data; //数据域

struct node \*fch, \*nsilb;

} \*Tree;

int Leaves(Tree t) {

if (t == NULL)

return 0;

if (t->fch == NULL)

return 1 + Leaves(t->nsilb);

else

return Leaves(t->fch) + Leaves(t->nsilb);

}

//已知一棵树的层次序列以及每个结点的度，构造此树的孩子-兄弟链表

//pointer[]，存储新建树的各结点的地址

typedef struct CSNode {

ElemType data;

struct CSNode \*firstchild, \*nextsibling;

} CSNode, \*CSTree;

#define maxNodes 15

void createCSTree\_Degree(CSTree &t, DataType e[], int degree[], int n) {

//根据树的层次遍历序列e[]和各个结点的度degree[]构造，n是结点个数

CSNode \*pointer = new CSNode[maxNodes];

int i, j, d, k = 0;

for (i = 0; i < n; ++i) {

pointer[i] = new CSNode;

pointer[i].data = e[i];

pointer[i].firstchild = pointer[i].nextsibling = NULL;

}

for (i = 0; i < n; ++i) {

d = degree[i];

if (d) {

k++; //k为子女结点序号

pointer[i].firstchild = pointer[k];//建立i与子女k的链接

for (j = 2; j <= d; ++j)

pointer[k - 1].nextsibling = pointer[k];

}

}

T = pointer[0];

delete[] pointer;

}