数据结构作业第六章

庄震丰 22920182204393

Oct. 25^{th}, 2019

6-33

题目要求：假定用两个一维数组作为有 n 个结点的二叉树的存储结构，L[i] 和 R[i] 分别指示结点 i(i=1,2,...n) 的左孩子和右孩子，0 表示空，试写出一个算法判断 u 是否为 v 的子孙。

算法分析：对于给定的 u 和 v，从 v 开始向下递归搜索每个结点看是否为 u，并用全局变量进行标记，如果 v 的子孙没有则会全部遍历，若找到则直接输出判断结果。

时间复杂度为 O(N), 空间复杂度 O(N).

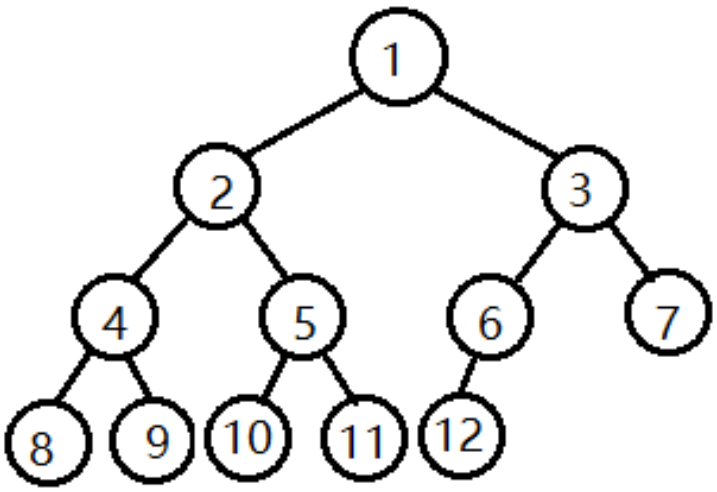
6-33.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
#define maxn 1000
int L[maxn+1]={0},R[maxn+1]={0};//建立L,数组储存节点左右儿子R
bool flag=false;
void getfa(int x,int target)//从开始往下递归搜素儿子节点v
{
    if (x==target) {flag=true;return;}
        if (L[x]!=0) getfa(L[x],target);
        if (R[x]!=0) getfa(R[x],target);
}
int main()
{
    int n;
    int v,u;
    cin>>n;
    for (int i=1;i<=n;i++)
        cin>>L[i]>>R[i];
    cin>>v>>u;
    getfa(v,u);
    cout<<flag;
    return 0;
}
```

样例输入 1

```
12
2 3
6 7
8 9
10 11
12 0
0 0
0 0
0 0
0 0
0 0
```

0 0
2 10
样例输出 1
1
说明



输入判断 2 是否为 10 的祖先，由图可知为真，输出 1。

6-62
题目要求：试编写算法，求一棵以孩子 -兄弟表示的树，编写其深度的算法。
算法分析：输入一棵双亲表示的树，运用机构提结点将其转换成孩子 -兄弟表示法，再将其进行深搜，每从左结点递归则层数加一，最后统计最大值即可。
时间复杂度 O(n), 空间复杂度 O(n)。
6-62.cpp

```
#include<bits/stdc++.h>
using namespace std;
#define maxn 1000
#define inf 0x3f3f3f3f
struct node
{
    char name;
    node * lson;
    node * bro;
};//孩子兄弟结构结点定义
struct parentnode
{
    int fa;
    char c;
    node * point;
};//双亲数组结构结点定义
parentnode A[maxn];
int n;
```

```
19  int cnt=1;
20  node* trans()//将双亲数组结构转变成孩子兄弟树形结构
21  {
22      node * hp,*p1,*p2;
23      hp=(node*)malloc(sizeof(node));
24      hp->name=A[1].c;
25      hp->bro=NULL;
26      A[1].point=hp;//根节点单独判断
27      for (int i=2;i<=n;i++)
28          {
29              if (A[i].fa!=A[i-1].fa)
30              {
31                  p1=(node*)malloc(sizeof(node));
32                  p1->bro=NULL;
33                  p1->lson=NULL;
34                  p1->name=A[i].c;
35                  A[i].point=p1;
36                  A[A[i].fa].point->lson=p1;
37                  //当前结点是新一层
38              }
39              else
40              {
41                  p2=(node *)malloc(sizeof(node));
42                  p2->bro=NULL;
43                  p2->lson=NULL;
44                  p2->name=A[i].c;
45                  A[i].point=p2;
46                  p1->bro=p2;
47                  p1=p2;//指针移动到兄弟结点
48                  //当前结点层数不变
49              }
50          }
51      return hp;
52  }
53  void finddeep(node * p,int deep)//按照孩子兄弟结构的指针进行遍历即可-
54  {
55      cout<<p->name<<" ";
56      if (p->bro==NULL&&p->lson==NULL)
57          {
58              cnt=max(cnt,deep);
59
60              return;
61          }
62      if (p->lson!=NULL) {finddeep(p->lson,deep+1);cout<<p->name<<" ";}
63      if (p->bro!=NULL) {finddeep(p->bro,deep);cout<<p->name<<" ";}
64      return;
65  }
66  bool cmp(parentnode a, parentnode b)
67  {
68      if (a.fa<b.fa) return true;
69      else return false;
70  }
```

```
71  int main()
72  {
73      node * Heap;
74      cin>>n;
75      A[0].fa=-inf;
76      for (int i=1;i<=n;i++)
77      {
78          scanf("%d %c",&A[i].fa,&A[i].c);//输入每个结点的父亲信息和该点的标号
79      }
80      sort(A+1,A+n+1,cmp);
81          Heap=trans();//返回根节点指针
82          finddeep(Heap,1);
83          cout<<endl<<cnt;
84      return 0;
85  }
```

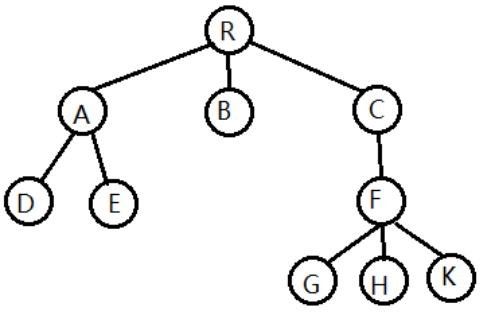
样例输入一

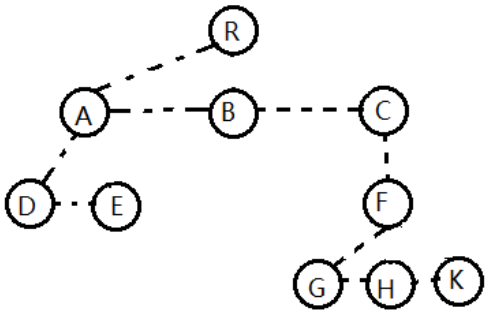
10
R -1
A 1
B 1
C 1
D 2
E 2
F 4
G 7
H 7
K 7

样例输出一

4

样例说明





样例描述的树和孩子 -兄弟表示法如图所示，层数为 4。

6-65

题目描述：已知一棵二叉树的前序和中序序列分别储存在两个一维数组中，试编写算法建立改树的二叉链表。

算法分析：先求树的结构，将其储存在一个数组中，在通过这个数组将其转变为链表形式，最后用后序遍历验证输出。

时间复杂度 O(n), 空间复杂度 O(n)。

6-65.cpp

```
#include<bits/stdc++.h>
using namespace std;
#define maxn 1000
#define inf 0x3f3f3f3f//设置无穷大
struct node
{
    char c;
    node *lson;
    node *rson;
};//二叉链表结构体数组
int n;
node * hp;
char Tree[maxn]={0};
void work(string ptree,string mtree,int cnt)//将二叉树两种遍历转变成数组存储
{
    char root=ptree[0];
    Tree[cnt]=root;
    if (ptree.length()!=1&&mtree.length()!=1)
    {
        int pos=mtree.find(root);
        work(ptree.substr(1,pos),mtree.substr(0,pos),cnt*2);
        work(ptree.substr(ptree.length()-pos,pos),mtree.substr(mtree.length()-pos,pos),cnt
            *2+1);
    }
    else
    {
        Tree[cnt]=root;
    }

}
void trans(int x,node * last,int size)//将数组变成二叉链表存储
{
    node *p1;
    if (x==1)
    {
        hp=(node *)malloc(sizeof(node));
        hp->lson=NULL;
        hp->rson=NULL;
        hp->c=Tree[x];
        if (Tree[x*2]) trans(x*2,hp,1);
        if (Tree[x*2+1]) trans(x*2+1,hp,2);
    }
    else
    {
        p1=(node *) malloc(sizeof(node));
```

```
45          p1->lson=NULL;
46          p1->rson=NULL;
47          p1->c=Tree[x];
48          if (Tree[x*2]) trans(x*2,p1,1);
49          if (Tree[x*2+1]) trans(x*2+1,p1,2);
50          if (size==1) last->lson=p1;
51          else last->rson=p1;
52      }
53  }
54  void printlast(node * p)//后序遍历输出
55  {
56
57      if (p->lson==NULL && p->rson==NULL) return;
58      if (p->lson!=NULL) printpre(p->lson);
59      if (p->rson!=NULL) printpre(p->rson);
60      cout<<p->c<<" ";
61      return;
62  }
63  int main()
64  {
65      cin>>n;
66      string pre,mid;
67      cin>>pre;
68      cin>>mid;
69      //初始化
70      work(pre,mid,1);
71      trans(1,NULL,0);
72      printlast(hp);
73  return 0;
74  }
```

样例输入 1

ABDECFG

DBEAFCG

样例输出 1

DEBFGCA

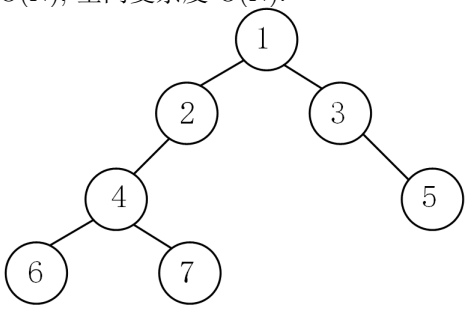
(标准的三层二叉树)

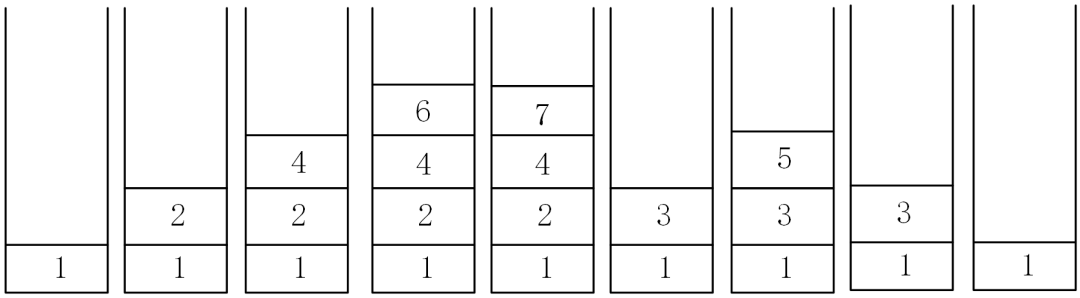
6-37,38

题目要求：用栈的基本操作写出先序遍历，后序遍历的非递归算法。

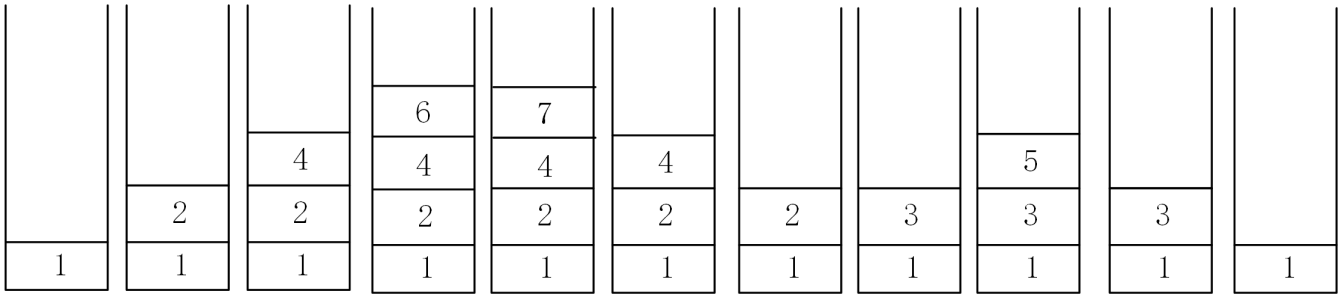
算法分析：先用数组存储树的结构，当栈不为空则一直进行操作，用 vis 进行标记某一节点是否遍历过，如果当前栈顶（根）节点没有左右儿子则弹出栈 pop，否则按照左右顺序依次入栈 push。

后序遍历类似，但先是左右儿子入栈，不标记当前节点遍历，当左右儿子都弹出 pop 后标记当前 top 节点并输出。时间复杂度 O(N), 空间复杂度 O(N).





先序：进栈遍历



后序：出栈遍历

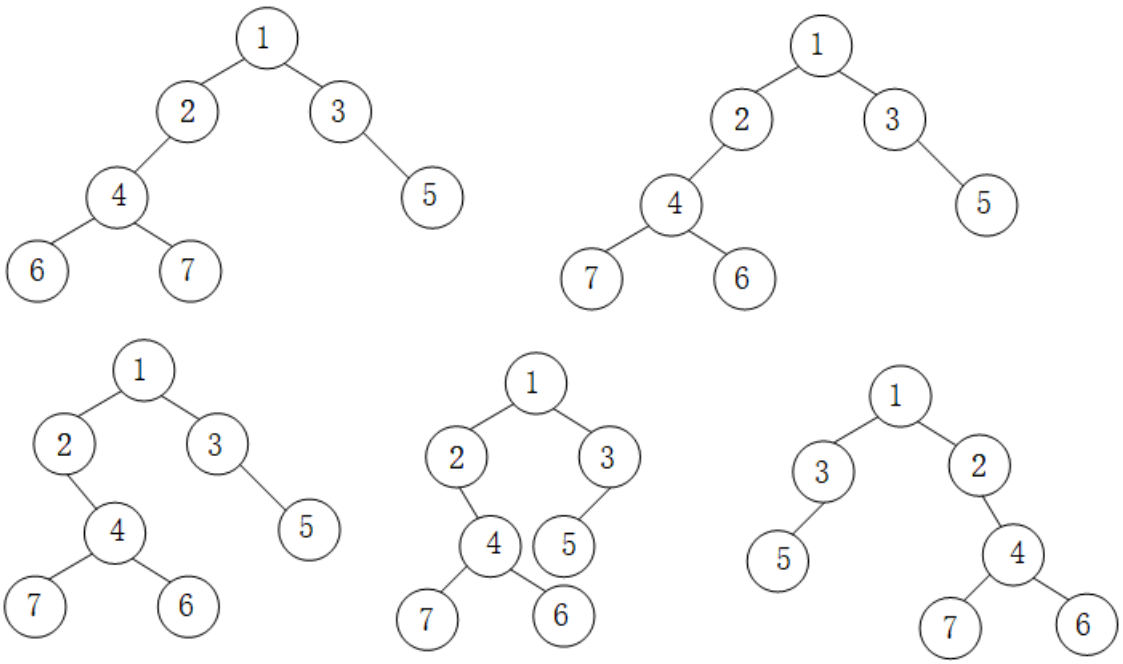
code

```
#include<bits/stdc++.h>
#define maxn 1000
using namespace std;
int n;
struct node
{
    int num;
    int lson;
    int rson;
    bool vis;
    bool pri;
}tree[maxn];
stack<node> s;
void pre_order()//先序遍历
{
    while(!s.empty())//栈不为空则运行
    {
```

```
18          if (!tree[s.top().num].vis)
19          {
20             cout<<s.top().num;
21             tree[s.top().num].vis=true;
22          }
23          if  (s.top().lson==-1&&s.top().rson==-1)  s.pop();
24          if (s.top().lson!=-1&&!tree[s.top().lson].vis) {//tree[s.top().lson].vis=true;
                                                                    s.push(tree[s.top().lson]);
              continue;}
25          if (s.top().rson!=-1&&!tree[s.top().rson].vis) {//tree[s.top().rson].vis=true;
26                                                  s.push(tree[s.top().rson]);continue;}
27          s.pop();
28       }
29    }
30    void mid_order()//中序遍历
31    {
32
33       while(!s.empty())
34       {
35          if (s.top().lson==-1&&s.top().rson==-1)  {tree[s.top().num].vis=true;cout<<s.top().
              num;s.pop();}
36          if (s.top().lson!=-1&&!tree[s.top().lson].vis) {s.push(tree[s.top().lson]);continue;}
37          if (!tree[s.top().num].vis)
38             {
39              cout<<s.top().num;
40              tree[s.top().num].vis=true;
41             }
42          if (s.top().rson!=-1&&!tree[s.top().rson].vis) {s.push(tree[s.top().rson]);continue;}
43          s.pop();
44       }
45    }
46    void last_order()//后序遍历
47    {
48
49       while(!s.empty())
50       {
51          if (s.top().lson==-1&&s.top().rson==-1)  {tree[s.top().num].vis=true;cout<<s.top().
              num;s.pop();}
52          if (s.top().lson!=-1&&!tree[s.top().lson].vis) {tree[s.top().lson].vis=true;s.push(
              tree[s.top().lson]);continue;}
53          if (s.top().rson!=-1&&!tree[s.top().rson].vis) {tree[s.top().rson].vis=true;s.push(
              tree[s.top().rson]);continue;}
54          cout<<s.top().num;
55          s.pop();
56       }
57    }
58
59    int main()
60    {
61       int F,S,root;
62       cin>>n;
63       for (int i=1;i<maxn;i++)
```

```
64      {
65          tree[i].lson=-1;
66          tree[i].rson=-1;
67          tree[i].vis=false;
68      }
69      for (int i=1;i<=n;i++)
70      {
71          int s_size;
72
73          scanf("%d%d%d",&s_size,&F,&S);
74          if (s_size==1)
75              tree[F].lson=S;
76          else tree[F].rson=S;
77          tree[F].num=F;
78          tree[S].num=S;
79      }
80      cin>>root;
81      while(!s.empty()) s.pop();
82      s.push(tree[root]);
83
84      cout<<"pre-order:";
85      pre_order();
86      cout<<endl;
87
88      for (int i=1;i<maxn;i++)
89          tree[i].vis=false;
90      tree[root].vis=true;
91      s.push(tree[root]);
92
93      cout<<"last-order:";
94      last_order();
95      cout<<endl;
96
97      for (int i=1;i<maxn;i++)
98      tree[i].vis=false;
99      s.push(tree[root]);
100     cout<<"mid-order:";
101     mid_order();
102 }
```

6-43 题目要求：编写递归算法，将二叉树的所有结点的左右子树互相交换。
算法分析：用数组结构体的形式表示二叉树，先根据输入信息建立数据关系，再递归从根节点到叶子节点进行翻转。
时间复杂度 O(N), 空间复杂度 O(N).



Code

```
#include<bits/stdc++.h>
using namespace std;
#define maxn 1000
#define inf 0x3f3f3f3f
int n,root,Mnode=-inf;
struct innode//树结点
{
    char name;
    innode* l;//结点左儿子
    innode* r;//结点右儿子
}A[maxn],*p=A;

void rev(innode x)//翻转左右儿子
{
    *innode pn;
    if (x.l!=NULL)  rev(*(x.l));
    if (x.r!=NULL)  rev(*(x.r));
    pn=x.l;
    x.l=x.r;
    x.r=pn;
}
int main()
{
    cin>>n;
    cin>>root;
    int a,ls,rs;
    char c;
    for (int i=0;i<maxn;i++)
        {
            A[i].l=NULL;
            A[i].r=NULL;
        }
```

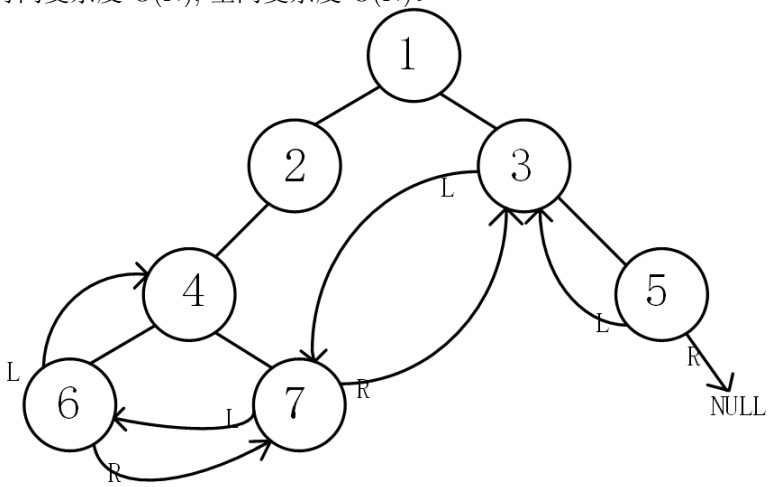
```
    for (int i=1;i<=n;i++)
    {
        scanf("%d %d %d %c",&a,&ls,&rs,&c);//输入每个结点的父亲信息和该点的标号
        A[a].name=c;
        A[a].l=(p+ls);
        A[a].r=(p+rs);
    }
    rev(root);
 return 0;
}
```

6-56

题目要求: 试写一个算法，再先序后继线索二叉树中，查找给定结点 *p 在先序序列中的后继。

算法分析：建立线索二叉树，若该节点没有右子树，则后继为 p->rson. 存在右子树，不存在左子树，则为 p-rson。若存在左子树不存在右子树，则为 p->lson。

时间复杂度 O(N), 空间复杂度 O(N)。



Code

```
#include<bits/stdc++.h>
using namespace std;
#define maxn 1000
#define inf 0x3f3f3f3f
int n,root;
struct node//树结点
{
    char name;
    node* ls;
    node* rs;
    int llog;
    int rlog;
    int num;
};
char A[maxn]={0};
int n;
node *heap=(node *)malloc(sizeof(node));//根节点
void build(node * x)//建立一棵树
{

```

```
21          node *p1,*p2;//左右子树指针
22          if (A[2*x->num])
23          {
24              x->llog=1;
25
26              p1=(node*)malloc(sizeof(node));
27              x->ls=p1;
28              p1->ls=p1->rs=NULL;
29              p1->name=A[2*x->num];
30              p1->llog=p1->rlog=0;
31              p1->num=2*x->num;
32              build(p1);//左子树
33          }
34
35          if (A[(2*x->num)+1])
36          {
37              x->rlog=1;
38              p2=(node*)malloc(sizeof(node));
39              x->rs=p2;
40              p2->ls=p2->rs=NULL;
41              p2->name=A[(2*x->num)+1];
42              p2->llog=p2->rlog=0;
43              p2->num=(2*x->num)+1;
44              build(p2);//右子树
45          }
46      }
47      node *P1,*P2;
48      void oline(node * pf)//先序遍历将其线索化
49      {
50          P1=P2;
51          P2=pf;
52          if (P1->rs==NULL) P1->rs=pf;
53          if (pf->ls==NULL) pf->ls=P1;
54          if (pf->ls!=NULL) oline(pf->ls);
55          if (pf->rs!=NULL) oline(pf->rs);
56      }
57      int main()
58      {
59          cin>>n;
60          for (int i=0;i<maxn;i++)
61              p[i]=NULL;
62          for (int i=1;i<n;i++)
63              cin>>A[i];//输入树节点信息
64          heap->name=A[1];
65          heap->llog=0;
66          heap->rlog=0;//根初始化条件
67          heap->num=1;
68          build(heap);
69          oline(heap);
70          return 0;
71      }
```