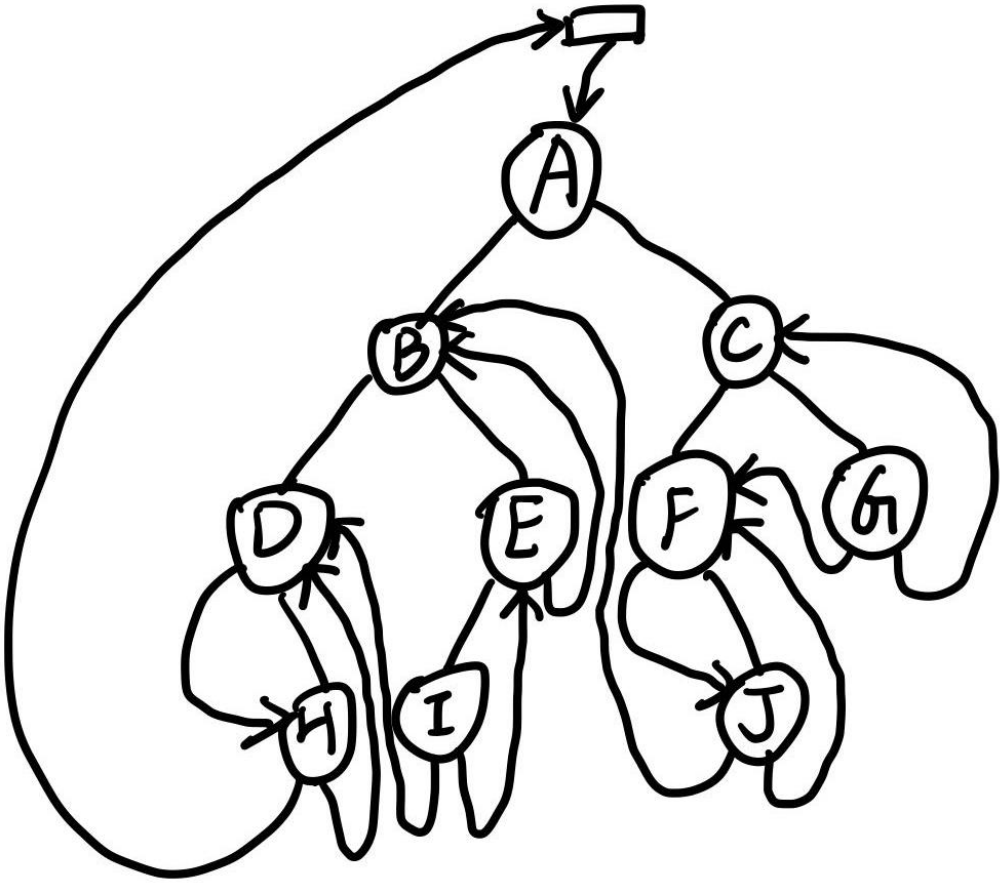
树和二叉树作业 2（提交版）

一、基础题

1、

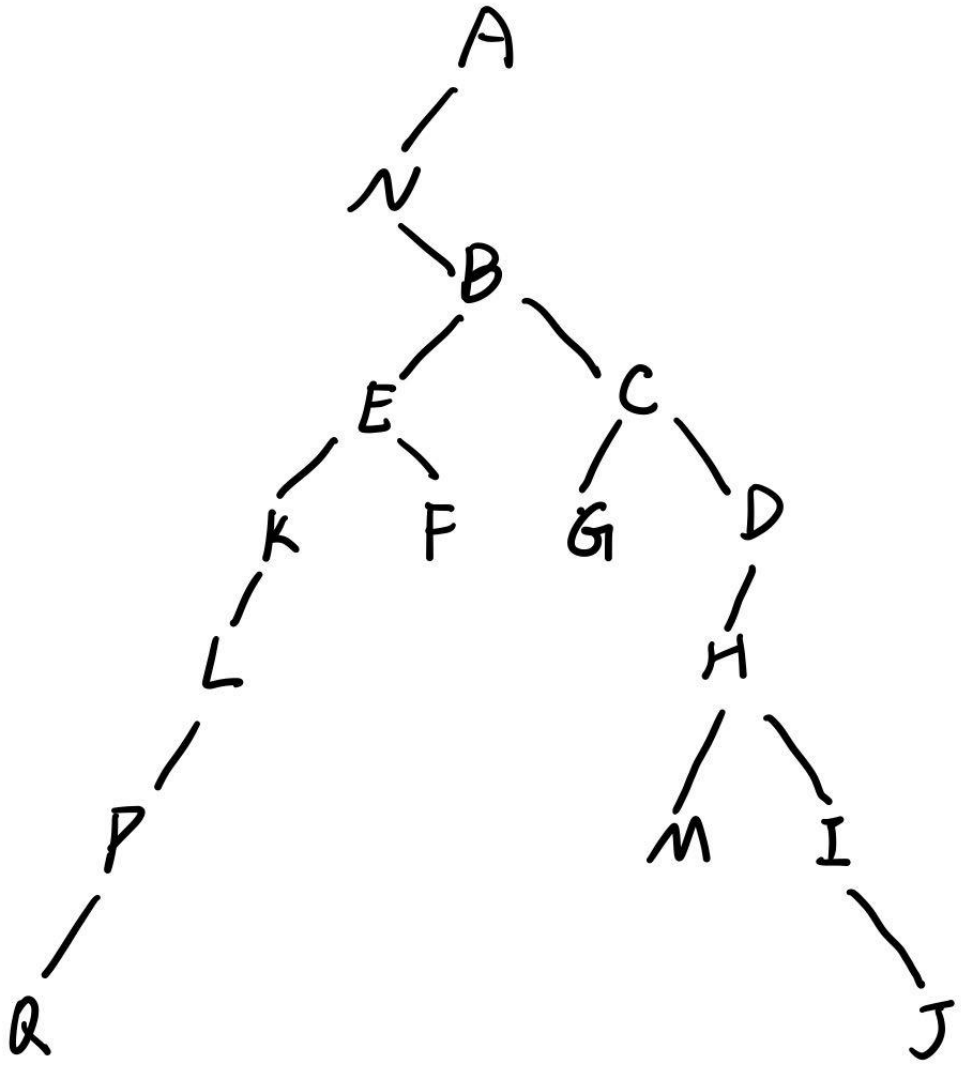


2、

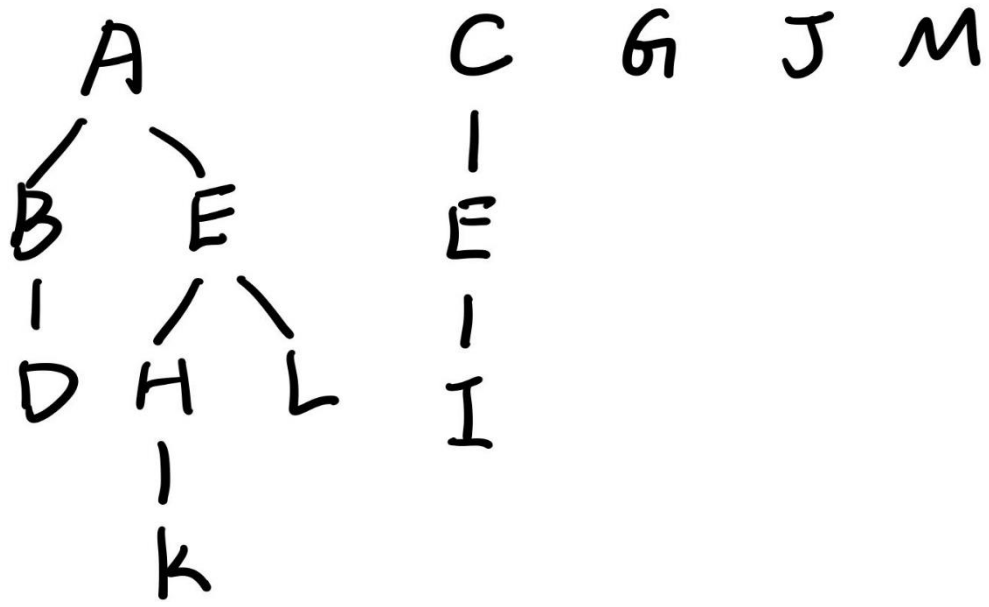
序号	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Info	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Ltag	0	0	1	1	0	1	0	1	1	11	0	1	1	1
Lchild	2	4	0	0	8	0	10	0	0	0	14	0	0	0
Rtag	0	0	0	0	1	0	0	1	0	0	1	1	1	1
Rchild	3	5	6	7	0	9	11	0	12	13	0	0	0	0

D 前驱 B，后继 G；F 前驱 C，后继 I；K 前驱 M，后继 N。

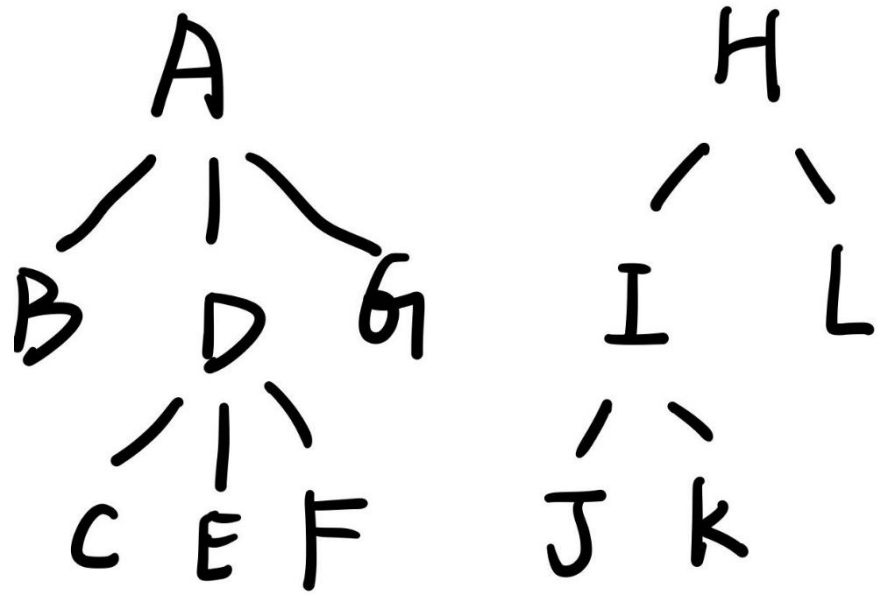
3、后根遍历：NKLPQEFBGCMHIJDA



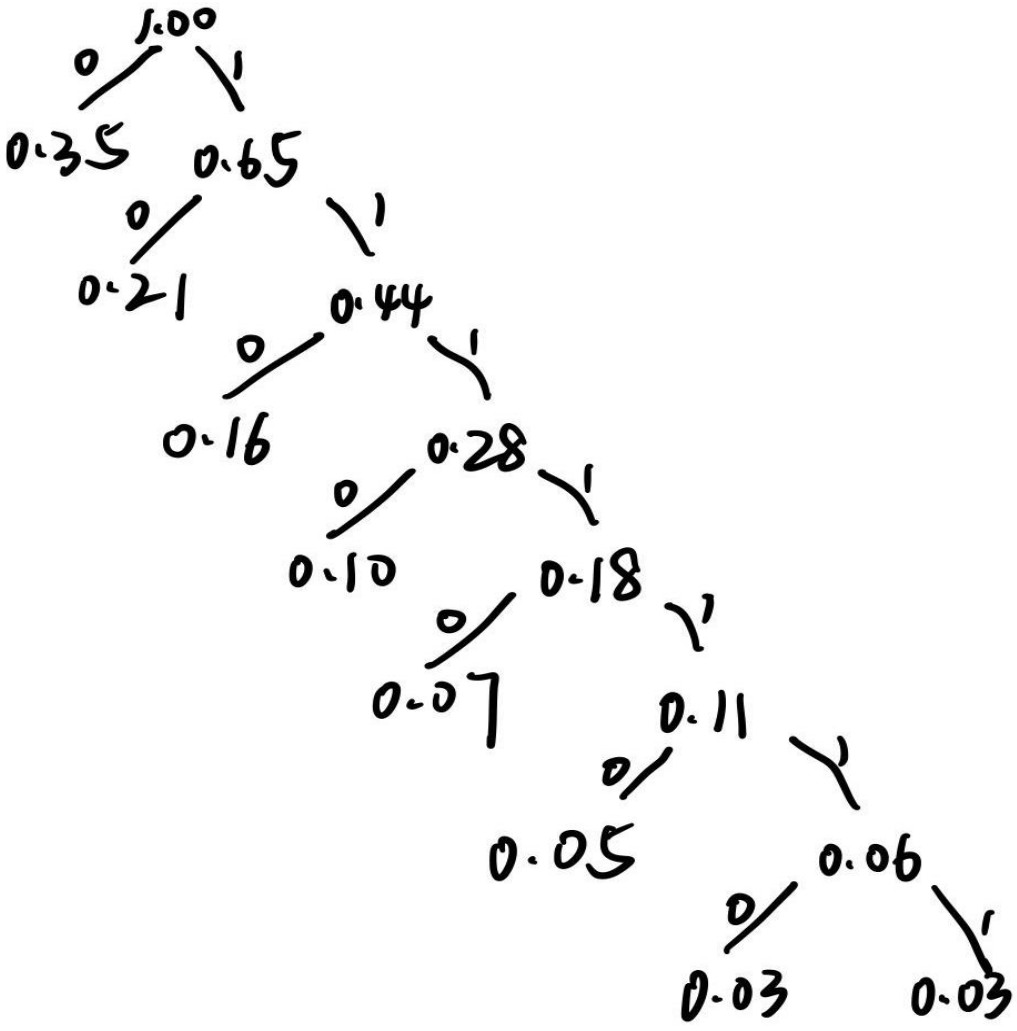
4、中序遍历：SBKHLEAIFCGJM



5、


A
H
B
D
G
I
L
C
E
F
J
K

6、哈夫曼树如下：


1.00
0
1
0.35
0.65
0
1
0.21
0.44
0
1
0.16
0.28
0
1
0.10
0.18
0
1
0.07
0.11
0
1
0.05
0.06
0
1
0.03
0.03

哈夫曼编码为：111110、110、1111110、11110、0、1111111、10、1110

权路径长度=0.35*1+0.21*2+0.16*3+0.10*4+0.07*5+0.05*6+（0.03+0.03）*7=2.72

定长编码：000、001、010、011、100、101、110、111

权路径长度=3*（0.35+0.21+0.16+0.10+0.07+0.05+0.03+0.03）=3

故哈夫曼编码优于定长编码。

二、算法设计题

1、

```
#include<stdio.h>
#include<iostream>
using namespace std;
typedef struct node{
    char ch;
    struct node *left;
    struct node *right;
}node, *nodePtr;
nodePtr create();//创建一棵二叉树
nodePtr find(nodePtr a, char p, char q);//找 a 树中 p，q 的最近祖先
void traverse(nodePtr a, char p, char q, int &k);//遍历 a 树，看有没有 p，q
void print(nodePtr a);//打印二叉树
void destroy(nodePtr a);//删除二叉树
int main()
{
    nodePtr a = create();//创建二叉树
    char p, q;
    cin>>p;
    cin>>q;
    nodePtr b = find(a, p, q);//找 a 树中 p，q 的最近祖先
    cout<<b->ch;
    destroy(a);//释放空间
    return 0;
}
nodePtr create()
{
    char ch;
```

```
    nodePtr a;
    scanf("%c",&ch);
    if(ch == ' ')
    {
        return NULL;
    }
    else
    {
        a = new node;
        a->ch = ch;
        a->left = create();
        a->right = create();
        return a;
    }
}
nodePtr find(nodePtr a, char p, char q)
{
    int left = 0, right = 0;
    traverse(a->left, p, q, left);//找到左子树中节点个数
    traverse(a->right, p, q, right);//找到右子树中节点个数
    cout<<"left:"<<left<<endl;
    cout<<"right:"<<right<<endl;
    if(a->ch == p || a->ch == q)//如果当前结点为其中一个
    {
        return a;//返回当前结点
    }
    else if(left == right)//如果当前结点左右子树各有一个节点，返回当前结点
    {
        return a;
    }
    else
    {
        //如果两个节点都在左子树中，那么在左子树中找，反之，在右子树中找
        if(left == 2)
        {
            find(a->left, p, q);
        }
        else
        {
            find(a->right, p, q);
        }
    }
}
```

```
71. void traverse(nodePtr a, char p, char q, int &k)
72. {
73.     if(a == NULL)
74.     {
75.         return;
76.     }
77.     else
78.     {
79.         if(a->ch == p || a->ch == q)
80.         {
81.             k++;
82.         }
83.         traverse(a->left, p, q, k);
84.         traverse(a->right, p, q, k);
85.     }
86. }
87. void print(nodePtr a)
88. {
89.     if(a == NULL)
90.     {
91.         cout<<" ";
92.         return ;
93.     }
94.     else
95.     {
96.         cout<<a->ch;
97.         print(a->left);
98.         print(a->right);
99.     }
100.  }
101.  void destroy(nodePtr a)
102.  {
103.      if(a == NULL)
104.      {
105.          return;
106.      }
107.      else
108.      {
109.          destroy(a->left);
110.          destroy(a->right);
111.          delete a;
112.      }
113.  }
```

2、

```
#include<stdio.h>
#include<stdlib.h>
#define ElemType int
typedef struct CSNode
{   ElemType data;
    struct CSNode* firstchild;//该结点的第一个孩子
    struct CSNode* nextbro;//该结点的下一个兄弟
}CSNode,*CSTree;
int CreateTree(CSTree* root)//种树
{   int data;
    scanf("%d",&data);
    if(data<=0)
    {   *root=NULL;
        return 0;
    }
    *root=(CSTree)malloc(sizeof(CSNode));
    if(!root){
        printf("failed\n");
    }
    if(data>0){
        (*root)->data=data;
        CreateTree(&((*root)->firstchild));
        CreateTree(&((*root)->nextbro));
    }
    return 0;
}
void PutOut(CSTree root)
{   if(root->firstchild==NULL)
        return;
    CSTree p=root->firstchild;
    while(p!=NULL)
    {   printf("(%d,%d)\n",root->data,p->data);
        PutOut(p);
        p=p->nextbro;
    }

}
int main()
{   CSTree root;
    CreateTree(&root);
    PutOut(root);
}
```

3、

```
int Depth(CSTree T)
{
    if(T == NULL)
        return 0;
    else
        return max(1 + Depth(T->firstchild), Depth(T->nextsibling));

}
```