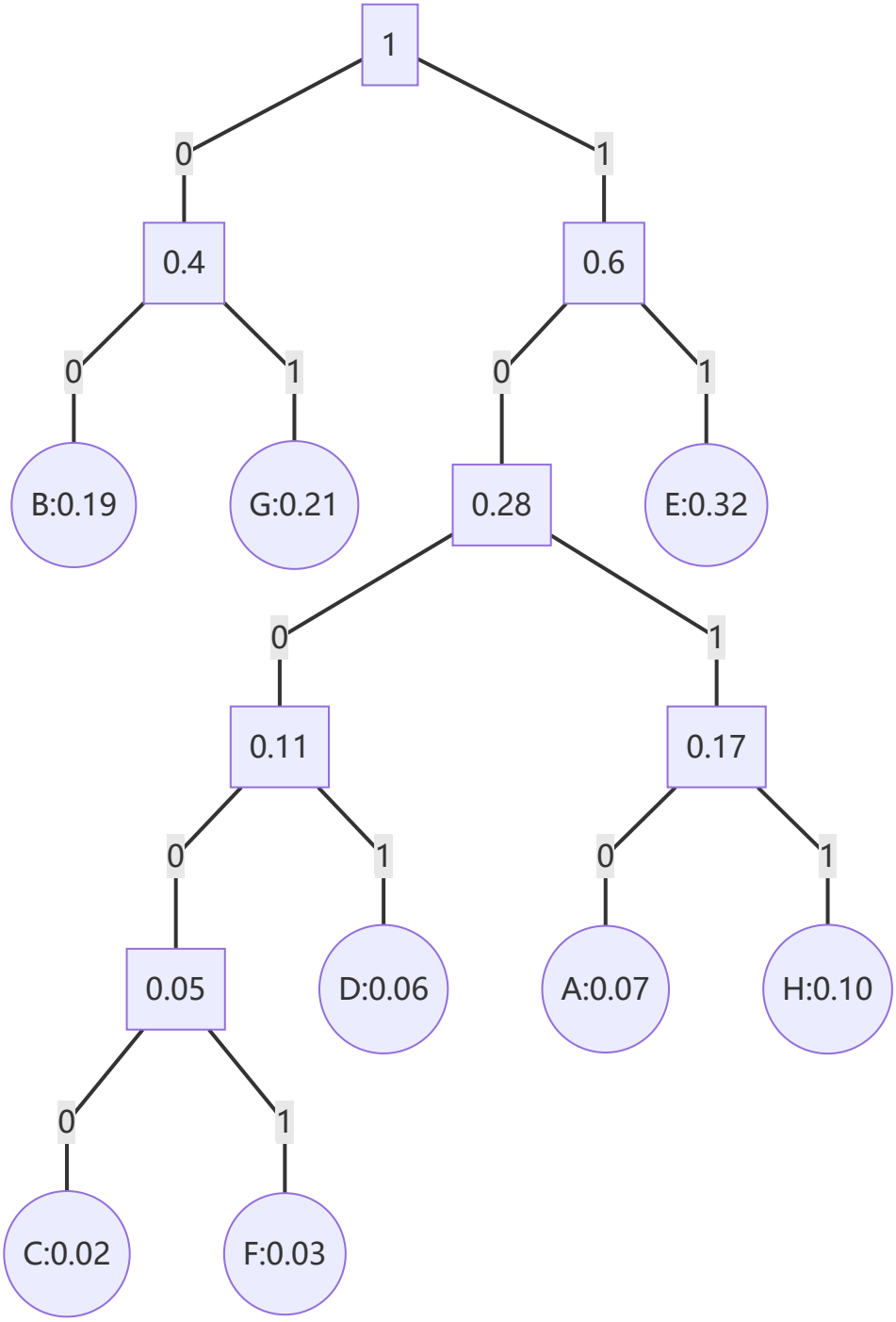
Homework9nd

应用题

(3)：假设用于通信的电文仅由8个字母组成，字母在电文中出现的频率分别为
$0.07，0.19，0.02，0.06，0.32，0.03，0.21，0.10$

1. 试为这8个字母设计哈夫曼编码



-

A	0.07	1010
B	0.19	00

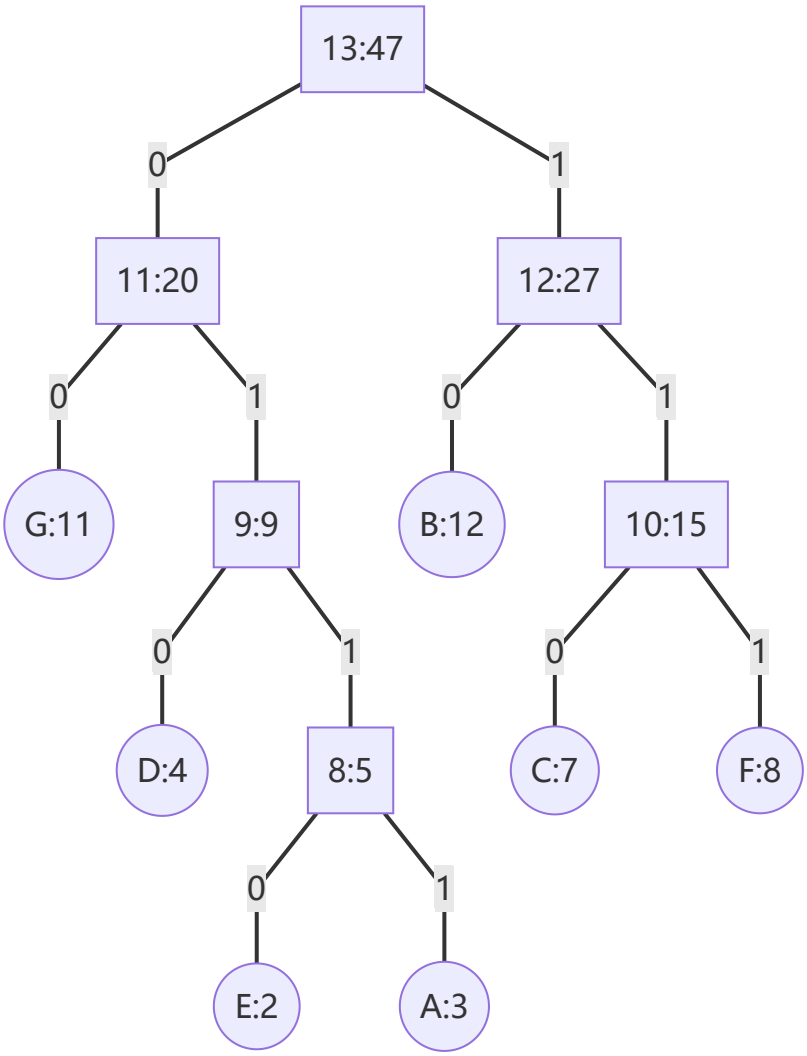
A	0.07	1010
C	0.02	10000
D	0.06	1001
E	0.32	11
F	0.03	10001
G	0.21	01
H	0.10	1011

2. 试设计另一种由二进制表示的等长编码方案

A	000
B	001
C	010
D	011
E	100
F	101
G	110
H	111

3. 对于上述实例，比较两种方案的优缺点
    - 哈夫曼编码的平均编码长度更小，但解码难度更大

(4)：已知下列字符A、B、C、D、E、F、G的权值分别为3、12、7、4、2、8、11，试填写出对应哈夫曼树HT存储结构的初态和终态



- 初态：

	weight	parent	lchild	rchild
A	3	0	0	0
B	12	0	0	0
C	7	0	0	0
D	4	0	0	0
E	2	0	0	0
F	8	0	0	0
G	11	0	0	0
8		0	0	0
9		0	0	0
10		0	0	0

	weight	parent	lchild	rchild
11		0	0	0
12		0	0	0
13		0	0	0

- 终态
  -

	weight	parent	lchild	rchild
A	3	8	0	0
B	12	12	0	0
C	7	10	0	0
D	4	9	0	0
E	2	8	0	0
F	8	10	0	0
G	11	11	0	0
8	5	9	E	A
9	9	11	D	8
10	15	12	C	F
11	20	13	G	9
12	27	13	B	10
13	47	0	11	12

算法设计题：以二叉链表作为二叉树的存储结构，编写以下算法

```
#include<bits/stdc++.h>
using namespace std;
template<class T>struct node{
    T val;
    node<T> *ls,*rs;
    node(){ls=rs=NULL;}
```

```
    node(T val):val(val),ls(NULL),rs(NULL){}
    bool operator ==(const node& _)const{
        return val==_.val;
    }
    bool operator !=(const node& _)const{
        return val!=_.val;
    }
};
template<class T>int Count(const node<T>* cur){
    return cur?1+Count(cur->ls)+Count(cur->rs):0;
}
template<class T>bool cmp(const node<T>* A,const node<T>* B){
    if(A&&B)return *A==*B&&cmp(A->ls,B->ls)&&cmp(A->rs,B->rs);
    return !A&&!B;
}
template<class T>void Swap(node<T>* cur){
    if(!cur)return;
    swap(cur->ls,cur->rs);
    Swap(cur->ls);
    Swap(cur->rs);
}
template<class T>void DoublePreorderTraversal(const node<T>* cur){
    if(!cur)return;
    cout<<cur->val<<' ';
    DoublePreorderTraversal(cur->ls);
    cout<<cur->val<<' ';
    DoublePreorderTraversal(cur->rs);
}
template<class T>int CountMaxDepth(const node<T>* cur,int depth=0)
{
    if(!cur)return depth;
    return ++depth,max(CountMaxDepth(cur-
>ls,depth),CountMaxDepth(cur->rs,depth));
}
template<class T>void CountMaxWidth(const node<T>* cur,int*
cnt,int depth=1){
    if(!cur)return;
    cnt[depth++]++;
    CountMaxWidth(cur->ls,cnt,depth);
    CountMaxWidth(cur->rs,cnt,depth);
}
template<class T>int MaxWidth(const node<T>* root){
    int maxDepth=CountMaxDepth(root);
    int* cnt=new int[maxDepth];
    for(int i=1;i<=maxDepth;i++)cnt[i]=0;
    CountMaxWidth(root,cnt);
```

```
    int mx=0;
    for(int i=1;i<=maxDepth;i++)
        mx=max(mx,cnt[i]);
    return mx;
}
template<class T>struct Queue{
    int maxSize;
    T* q;
    T *head,*tail;
    Queue(){q=head=tail=NULL;}
    Queue(int maxSize):maxSize(maxSize){
        q=new T[maxSize];
        head=tail=q;
    }
    T front()const{
        return *head;
    }
    void pop(){
        if(head==tail){
            cerr<<"The queue is empty!!!"<<endl;
            exit(114514);
        }
        head++;
    }
    void push(T elem){
        if(tail-q==maxSize){
            cerr<<"The queue is full!!!"<<endl;
            exit(998244353);
        }
        *(tail++)=elem;
    }
    bool empty()const{
        return head==tail;
    }
};
template<class T>int LevelOrderTraversal(const node<T>* root){
    Queue<const node<T>*>q(Count(root));
    q.push(root);
    int cnt=0;
    while(!q.empty()){
        const node<T>* cur=q.front();q.pop();
        if(!cur->ls&&!cur->rs)cnt++;
        if(cur->ls)q.push(cur->ls);
        if(cur->rs)q.push(cur->rs);
    }return cnt;
}
```

```
template<class T>void LongestPathSum(const node<T>* cur,int&
Depth,T& Sum,int depth=0,T sum=0){
    if(!cur){
        if(depth>Depth){
            Depth=depth;
            Sum=sum;
        }return;
    }
    depth++;
    sum+=cur->val;
    LongestPathSum(cur->ls,Depth,Sum,depth,sum);
    LongestPathSum(cur->rs,Depth,Sum,depth,sum);
}
template<class T>pair<int,T> LongestPathSum(const node<T>* root){
    T depth=0,sum=0;
    LongestPathSum(root,depth,sum);
    return make_pair(depth,sum);
}
template<class T>void PrintLeavesPath(const node<T>* cur,string
s=""){
    if(!cur)return;
    s+=to_string(cur->val)+" ";
    if(!cur->ls&&!cur->rs)cout<<s<<endl;
    PrintLeavesPath(cur->ls,s);
    PrintLeavesPath(cur->rs,s);
}
void Init(node<int>*& root){
    node<int>** nodes=new node<int>*[10];
    for(int i=0;i<10;i++)
        nodes[i]=new node<int>(i);
    nodes[0]->ls=nodes[1];
    nodes[1]->ls=nodes[2];
    nodes[1]->rs=nodes[3];
    nodes[2]->ls=nodes[4];
    nodes[2]->rs=nodes[5];
    nodes[4]->ls=nodes[6];
    nodes[5]->rs=nodes[7];
    nodes[3]->ls=nodes[8];
    nodes[7]->ls=nodes[9];
    root=nodes[0];
}
int main(){
    node<int>* root;
    node<int>* ROOT;
    Init(root);
    Init(ROOT);
```

```
    ROOT->ls->ls->ls=NULL;
    cout<<"Count:"<<Count(root)<<endl;
    cout<<"Cmp:"<<cmp(root,ROOT)<<","<<cmp(root,root)<<endl;
    DoublePreorderTraversal(root);puts("");
    Swap(root);
    DoublePreorderTraversal(root);puts("");
    Swap(root);
    cout<<"MaxWidth:"<<MaxWidth(root)<<endl;
    cout<<"LevelOrderTraversal:"<<LevelOrderTraversal(root)<<endl;
    pair<int,int>tmp=LongestPathSum(root);
    cout<<"LongestPathSum:"<<tmp.first<<','<<tmp.second<<endl;
    PrintLeavesPath(root);
    return 0;
}
```