一、填空（每空 1 分，共 15 分）

1．顺序　链式　　2.队列　　3.O(n)　　4.ABC-D*-EF/+　　5.152　　6.13,14

7.afcbdeg　　8. 邻接表　　9.有序数组（或顺序表）　10.11　11.n+1　　12.空间　13.栈顶

二、选择题（每小题 1 分，共 15 分）

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	D	A	A	B	B	C	D	A	B	B	C	C	A	C

1.对 2.　错　3. 对　　4. 错　　5. 错　　6. 错　7.　错　8. 错　9. 对　10. 对

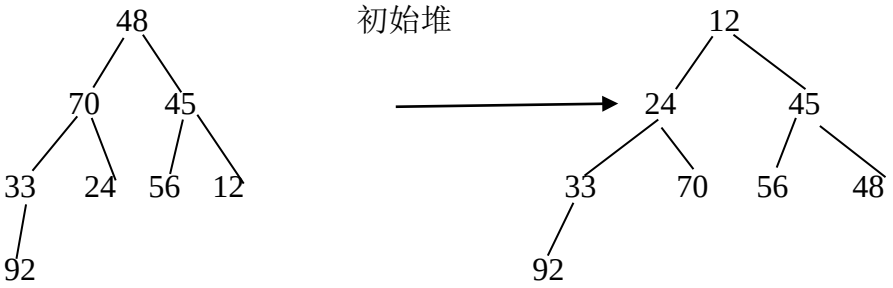
三、简答（每小题 5 分，共 40 分）

1．模式匹配 KMP 方法的基本思想是发生不匹配时，主串不回溯，子串少回溯。（2 分）

“abcaaababbb”的 next 值为-1，0，0，0，1，1，1，2，1，2，0

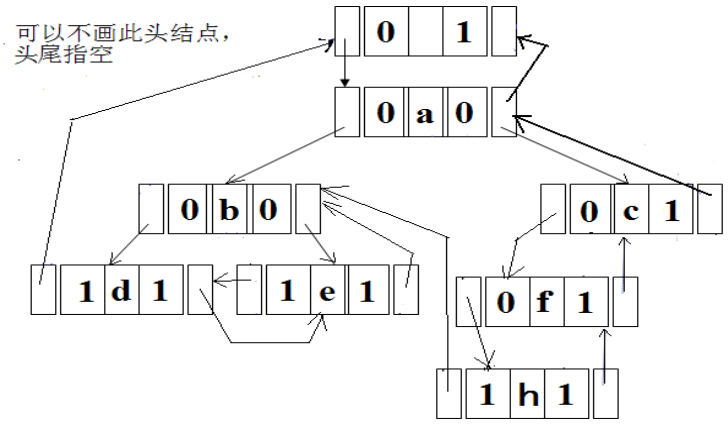
Next 值所有的均加 1 也可（3 分）。

2．要有中间调整的步骤,少一步骤扣 1 分

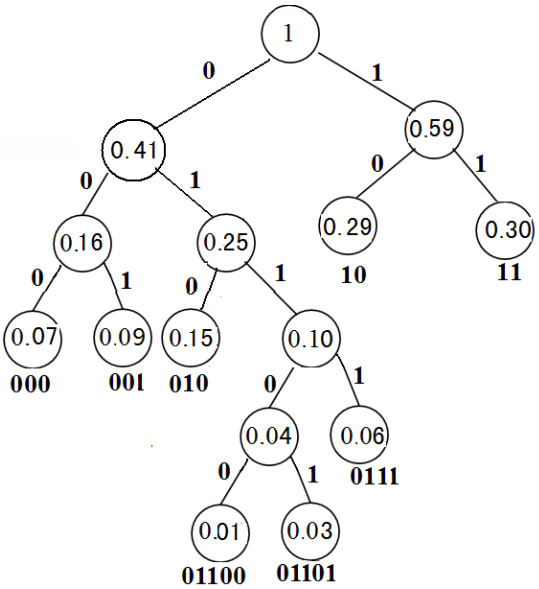


3.后序遍历序列 debhfca（2 分）

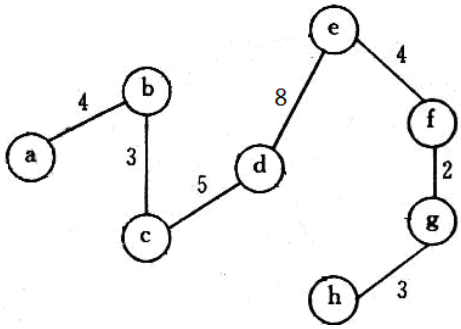
对应的后序线索二叉树（3 分）。简化方式也可以

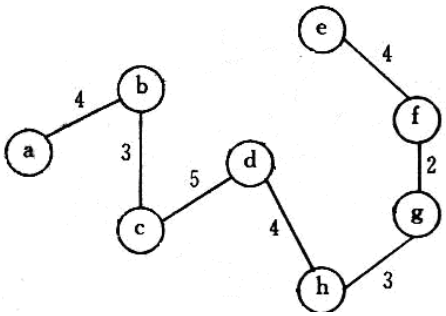


4. 哈夫曼树（3 分），为这 8 种颜色设计哈夫曼编码，对应叶子，长度一致即可（2 分）。



5. 从 a 开始的深度优先生成树（3 分），kruscal 算法最小生成树--cd 或 ch 均对（2 分）。





6．求顶点的最早最晚发生时间：（2 分）

顶点	1	2	3	4	5	6
VE	0	3	3	9	8	11
VL	0	3	5	9	10	11

求边的最早最晚发生时间：（2 分）

边	a1	a2	a3	a4	a5	a6	a7	a8	a9
AE	0	0	3	3	0	3	8	9	3
AL	0	2	5	3	6	5	10	9	6

关键路径是：a1 +a4 +a8=3+6+2=11　（1 分）

7. 哈希表的示意图（3 分）

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
	23	**34**	**56**	**24**	**12**	**49**	**6**	**52**	**75**			

等概率下的平均查找长度（2 分）。（1+2+3+3+1+5+2+1+2）/9=20/9

8．第 1 趟：11，1，14，19, 55, 68, 23, 82, 36,40

第 2 趟：1，11，14，19, 40,36, 23, 55, 82, 68

第 3 趟：1，11，14，19, 23, 36, 40, 55, 68, 82

第 4 趟：1，11，14，19, 23, 36, 40, 55, 68, 82 （可以不写这一趟）

或者第 1 趟：11，1，14，19, 55, 68, 23, 82, 36,40

第 2 趟：1，11，14，19, 40, 23,36, 55, 68, 82

第 3 趟：1，11，14，19, 36, 23, 40, 55, 68, 82

第 4 趟：1，11，14，19, 23, 36, 40, 55, 68, 82

四、编程题（共 30 分）只写出要求的内容，未要求的类和函数不必写。

1．单链表类（LinkList）的删除函数 Remove（5 分）

```
void LinkList:: Remove()
{ Node *p,*q,*r;
  p=first; //p 是当前结点，也可以是 p=first->link;
  while(p!=0)
  { q=p; //q 结点从 p 后开始循环到表尾，看有无重复的数据
    while( q->link!=0)
    { if(q->link->data=p->data) //删除 q->link 结点
        { r=q->link; q->link= q->link->link; delete r; }
      q= q->link;
    }
    p=p->link;
  }
}
```

2．给出二叉链式存储的二叉树类的定义；

```
struct   BiTNode      （2 分）
{      elemtype          data    ;        //结点数据类型
       BiTNode *lchild, *rchild; //定义左、右孩子为指针型
 };
class  BiTree   （3 分）
{    BiTNode   *root;
  public:  void create(BiTNode *root);
           int size(BiTNode *root);
};
```

定义成员函数 create，功能是读入完全前序序列建立二叉树（自选该函数,有无均可,不作计分项）；

```
void BiTree ::create(BiTNode *root )
 {   cin>>ch;
     if(ch==’#’ )   root=0;
     else  {      root=new   BiTNode  ;
                  if(!root)
                 { printf(“分配空间不成功!”) return 0;}
                  root->data=ch;
```

```
                createbitree(root->lchild);
                createbitree(root->rchild);
            }
}
```

定义成员函数 size，功能是计算二叉树中的结点个数。（5 分）

```
int BiTree ::size(BiTNode *root )
 {      if(root==0 )    return 0;
        else    return    1+size(root->lchild)+size(root->rchild);
}
```

定义主函数，在其中定义二叉树对象并调用这两个函数。（3 分）

```
int main()
{ BiTree    tree;
  tree. create(tree.getroot());   //用 getroot 函数或其他方法得到 root 都可以给分
  cout<<"结点个数为"<<tree.size(tree.getroot());
}
```

3.

定义成员函数 addedge，功能为插入给定的两个顶点（u,v）之间的一条边。 (5 分)

```
void    Graph::addedge(char u,char v )
{       int i,m=-1,n=-1;
        m= getVertexPos(u);
        n= getVertexPos(v);
        cin>>i;
         Edge[m][n]=i;
         Edge[n][m]=i;
          numEdges++;
         cout<<"成功"<<endl;    }
}
```