

一、 选择题

- 1、二叉树的中序遍历序列之中，结点 **a** 排在 **b** 之前的条件是 **B**。
A) **a** 在 **b** 的右方 **B) a 在 b 的左方** C) **a** 是 **b** 的祖先 D) **a** 是 **b** 的子孙
- 2、在数据结构中，构成数据元素的最小单位称为 **B**。
A) 字符 **B) 数据项** C) 数据元素 D) 关键字

注：数据元素是数据的基本单位

数据项有称字段或域

- 3、已知含有 12 个结点的二叉排序树是一棵完全二叉树，则该二叉树排序树在等概率情况下，查找成功的平均查找长度等于 **C**。

$$1/12(1*1+2*2+3*4+4*5) = 3.1$$

- A) 1.0 B) 2.9 **C) 3.1** D) 5.2
- 4、在一棵二叉树排序树中，关键字值最小的结点 **B**。
A) 右孩子指针一定为空 **B) 左孩子指针一定为空**
C) 左右孩子指针均为空 D) 左右孩子指针都不为空

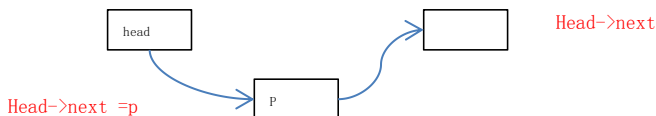
注：（1）若左子树不空，则左子树上所有结点的值均小于或等于它的根结点的值；

（2）若右子树不空，则右子树上所有结点的值均大于或等于它的根结点的值；

（3）左、右子树也分别为二叉排序树；

- 5、将 **P** 所指的结点插入到带头结点的单链表 **head** 中作为首元结点的操作序列为 **B**。

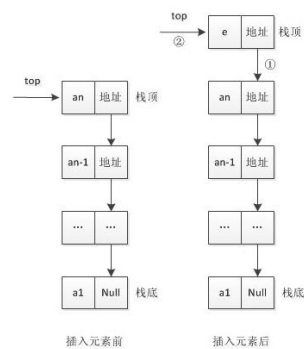
- A) **p->next=head;head=p;**
- B) **p->next=head->next; head->next=p;**
- C) **p->next=head;**
- D) **head->next=p->next;p->next=head->next;**



注：一定是先 把 head->next 先赋给 p->next 即 p->next= head->next 要不然链表就断了！

- 6、在一个栈顶指针为 **top** 的链栈中，将 **s** 所指结点入栈的操作序列为 **A**。

- A) **s->next=top;top=s;**
- B) **s->next=top->next; top->next=s;**
- C) **top->next=s;**
- D) **s->next=top;top=top->next;**



- 7、设进 **ABCDEF**， **A** 是不可能得到的出栈序列。

- A) **CDABFE**
- B) **BCDEFA**
- C) **ABCDEF**
- D) **EFDCBA**

注：

A: CBA -> DBA -> BA -> A -> EF -> E ->
C CD CDB CDBA CDBAF CDBAFE
B: BA -> CA -> DA -> EA -> FA -> A ->
B BC BCD BCDE BCDEF BCDEFA

C: A → B → C → D → E → F →
 A AB ABC ABCD ABCDE ABCDEF
 D: EDCBA → FDCBA → DCBA → CBA → BA → A →
 E EF EFD EFDC EFDCB EFDCBA

8、栈的插入和删除操作是在 D 进行中。

A) 指定位置 B) 栈底 C) 任意位置 **D) 栈顶**

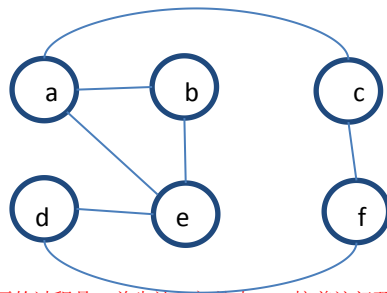
9、在单链表中删除指针 p 指向的结点的后继结点(不考虑释放结点), 正确的操作为 D。

A) p->next=p; B) p=p->next=s; C) p=p->next->next; **D) p->next=p->next->next;**

10、无向图 $G=(V,E)$, 其中: $V=\{a,b,c,d,e,f\}$, $E=\{(a,b),(a,e),(a,c),(b,e),(c,f),(f,d),(e,d)\}$, 从 a 顶点开始对该图进行广度优先遍历, 得到的顶点序列正确的是 A。

A) a b e c d f B) a c f e b d C) a e b c f d **D) a d e f c b**

注:



广度优先遍历的过程是: 首先访问初始点 a 接着访问顶点 a 的所有未被访问过的邻接点 b c e (亦可是 b e c / b c e / e b c / e c b / c e b / c b e) 然后再按照 b c e (对应是 b e c / b c e / e b c / e c b / c e b / c b e) 的一次访问顺序访问每一个顶点的所有未被访问过的邻接点为止。则可得如下结果:

a b c e f d 、 a b e c d f 、
 a c e b d f 、 a c b e f d 、
 a e b c d f 、 a e c b d f

11、对于任何一棵二叉树 T, 如果其叶子结点数为 n_0 , 度为 2 的结点数为 n_2 , 则 D。

A) $n_2=n_0+1$ B) $n_0=2n_2+1$ C) $n_2=2n_0+1$ **D) $n_2=n_0-1$**

二叉树性质 1: 非空二叉树上的叶子结点等于双分支节点数加 1.

12、对有 11 个元素的有序表作折半查找, 则查找 A[4] 的比较序列的下标依次为 B。

A) 1, 3, 4 **B) 6, 3, 4** C) 6, 5, 4 **D) 8, 5, 2, 4**

注: 1 2 3 4 5 6 7 8 9 10 11

取有序数列的中间位置作为比较对象 $(1+11)/2=6$ 则以下标为 6 的元素作为比较对象。

如果要查找的元素小于中间的元素 则将待查序列缩小为左半部分, 否则为右半部分。即 $4 < 6$ 则取右半部分 (1 2 3 4 5)

重复 1 得 $(1+5)/2=3$ 以 3 作为比较对象 重复 2: $3 < 4$ 得 序列 (4 5)

再重复 1 得 $(4+5)/2=4$ 找到了。

13、如果待排序的数据已经有序, 那么使用 B 排序算法最快。

A) 直接插入 **B) 冒泡** C) 直接选择 **D) 快速**

14、一个有 n 个顶点的无向图最多有 C 条边。

A) n^2 B) $n^2/2$ **C) $n(n-1)/2$** **D) $n(n+1)$**

15、在二叉树中, 如果度为 2 的结点数位 m, 则叶子结点的个数为 B。

A) 2m **B) m+1** C) m **D) 不确定**

16、下面程序段的时间复杂度为 C。

```

for(i=0;i<m;i++)
for(j=0;j<n;j++)
a[i][j]=i*j;

```

- A) $O(m^2)$ B) $O(m+n)$ **C) $O(m \times n)$** D) $O(n^2)$

17、在有向图中，所有顶点的入度之和等于所有顶点的出度之和的 C 倍。

- A) 1/2 B) 2 **C) 1** D) 4

18、某算法的时间复杂度是 $O(n^3)$ ，表明该算法 C。

- A) 题规模是 n^3 B) 问题的规模与 n^3 成正比 **C) 执行时间与 n^3 成正比** D) 执行时间等于 n^3

19、在一个单链表中，若要在 P 指向的结点之后插入一个新结点，则需要修改 2 个结点的指针。

- A) 4 B) 3 **C) 2** D) 1

20、若要对元素比较多的表进行排序，要求既快速又稳定，则最好采用 D 方法。

- A) 直接插入排序 B) 快速排序 C) 堆排序 **D) 归并排序**

各种常用排序算法						
类别	排序方法	时间复杂度			空间复杂度	稳定性
		平均情况	最好情况	最坏情况	辅助存储	
插入排序	直接插入	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	shell排序	$O(n^{1.3})$	$O(n)$	$O(n^2)$	$O(1)$	不稳定
选择排序	直接选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
	堆排序	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(1)$	不稳定
交换排序	冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	快速排序	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n^2)$	$O(n \log_2 n)$	不稳定
归并排序		$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(1)$	稳定
基数排序		$O(d(r+n))$	$O(d(n+rd))$	$O(d(r+n))$	$O(rd+n)$	稳定

注：基数排序的复杂度中，r代表关键字的基数，d代表长度，n代表关键字的个数

二、填空题

1、带头结点的单链表 L 为空的条件是 L->next==Null

2、非空单循环链表 L 中 P 是尾结点的条件是 P->next==L

3、假定一个链队列的队首和队尾指针分别为 front 和 rear，则判断队空的条件是 front==rear

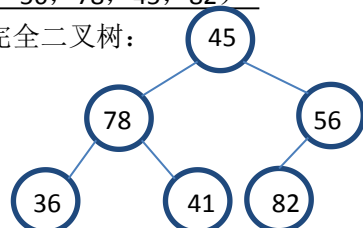
4、在一个完全二叉树的顺序存储结构中，若一个元素的下标为 i，且其有右孩子，那么它的右孩子的下标为 2i+1

5、一个含 n 个结点的二叉树的最小高度为 $\lceil \log_2 n \rceil + 1$

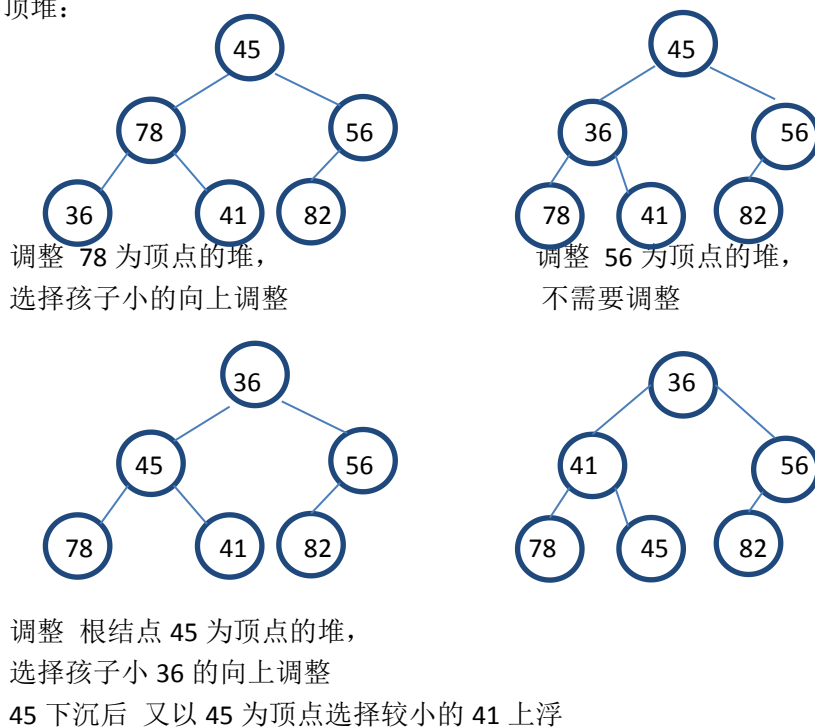
6、对于一棵有 n 个结点的树，其所有结点的度数之和为 n-1

7、给定一组关键字序列 (45, 78, 56, 36, 41, 82)，利用堆排序的方法建立的初始小顶堆为 (36, 41, 56, 78, 45, 82)

注： 1.先构造完全二叉树：



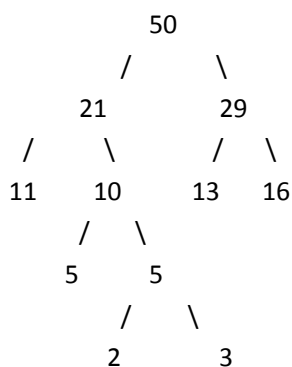
2.建立小顶堆:



8、判定一个带结点的单链表不为空的条件为 L->next!=Null

9、已知字符集{A,B,C,D,E,F}各字符的哈夫曼编码依次是 110, 010, 10, 001, 11, 000, 那么, 对编码序列“01011011000111011001”的译码结果是 BEAFECED

10、有权值序列: 2、11、5、13、3、16 构造哈夫曼树, 则其带权路径长度为 115



带权路径长度为:

$$11*2 + 5*3 + 2*4 + 3*4 + 13*2 + 16*2 = 115$$

11、采用邻接表存储的图的广度优先搜索遍历算法类似于二叉树的 按层次 遍历。

12、一棵二叉树的先序遍历序列为 ABDGCEF, 中序遍历序列为 DGBAECF, 则该棵树后序遍历序列为 GDBEFCA

注: 根据先序, 中序反推二叉树的思路是:

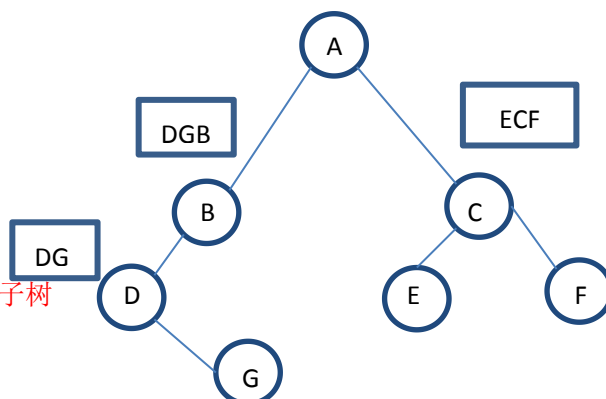
1. 先根据先序确定 根, 再根据中序确定左右子树;
2. 然后根据先序确定根, 再跟新中序确定的左右子树 反复 1,2.

1.根据先序确定根

2.根据中序确定左右子树

3.根据先序确定新的根

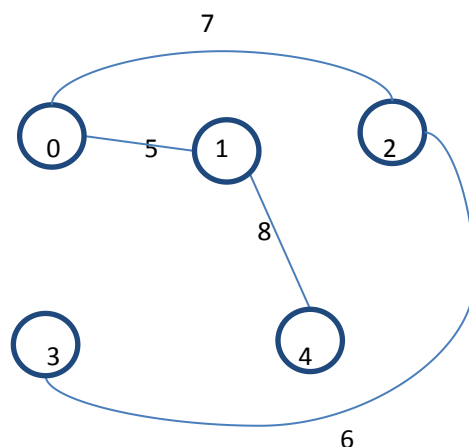
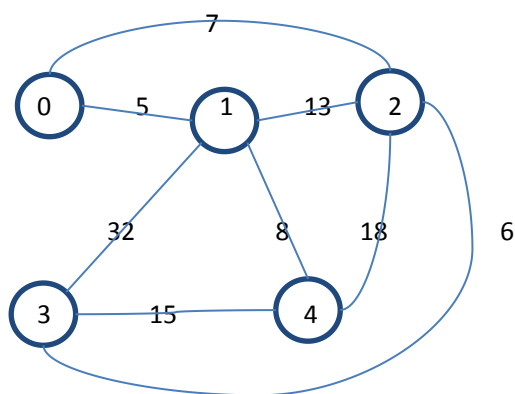
4.根据中序确定新的左右子树



13、队列的插入操作是在队尾端进和的。

14、由树转换成二叉树时，其根结点的右子树总是空的。

15、假定一个图的边集为 $\{(0,1)5,(0,2)7,(1,2)13,(1,3)32,(1,4)8,(2,3)6,(2,4)18,(3,4)15\}$ ，则其最小生成树的权为26



$$5 + 8 + 7 + 6 = 26$$

注：按权最小算起，联通所有点，不能形成闭合。

16、对具有 n 个顶点的图，其生成树有且仅有 $n-1$ 条边。

17、若经常需要对线性表进行插入和删除操作，则最好采用链表存储结构。

18、使用双向链表结构来存储线性表，可以在表中两个方向上访问到所有结点。

19、排序的目的是为了提高对数据查找操作的效率。

20、具有 3 个结点的二叉树5种不同的形态。

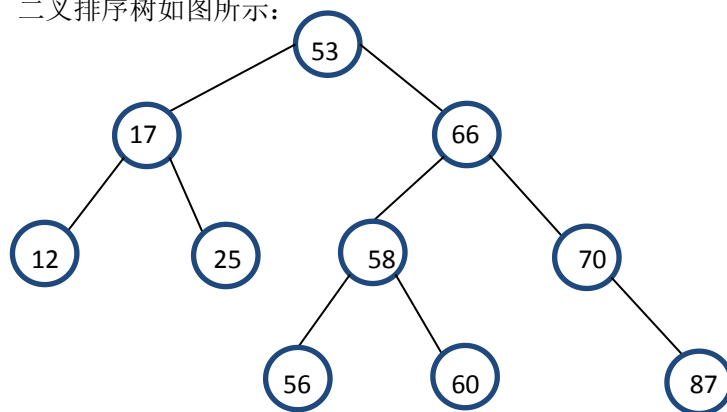
三、应用设计题

1、输入一个正整数序列（53，17，12，66，58，70，87，25，56，60），试完成：

（1）按次序构造一棵二叉排序树；

（2）中序遍历该二叉排序树的序列；

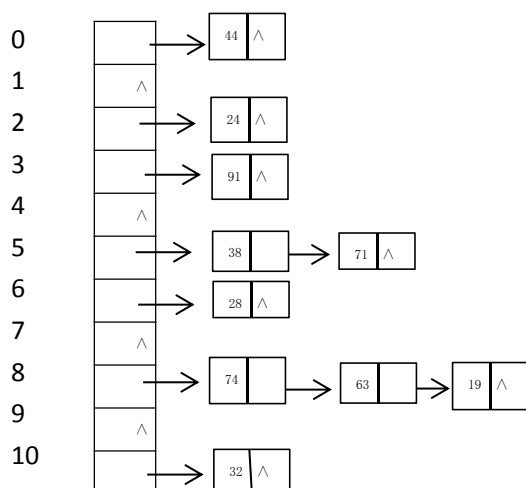
解：（1）：二叉排序树如图所示：



（2）：中序遍历：12 17 25 53 56 58 60 66 70 87

2、有一组数据集{38, 74, 28, 63, 44, 91, 24, 32, 19, 71}, 哈希表的地址空间是 HT[11], 若哈希函数 $H(key)=key\%11$, 采用链地址法处理冲突, 画出最后得到的哈希表, 并求其平均查找长度。

解：

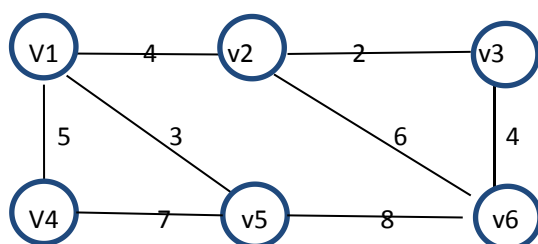


平均查找长度：ASL= $1/10(7*1+2*2+3*1)=14/10=1.4$

3、请对下面的带权无向连通图，试完成：

（1）写出它的邻接矩阵；

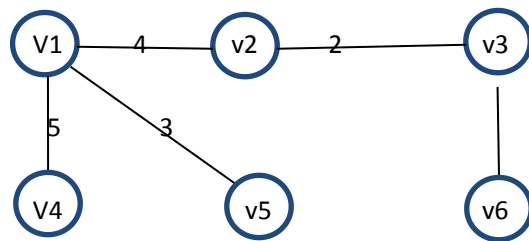
（2）按 Prim 算法构造出它的最小生成树；



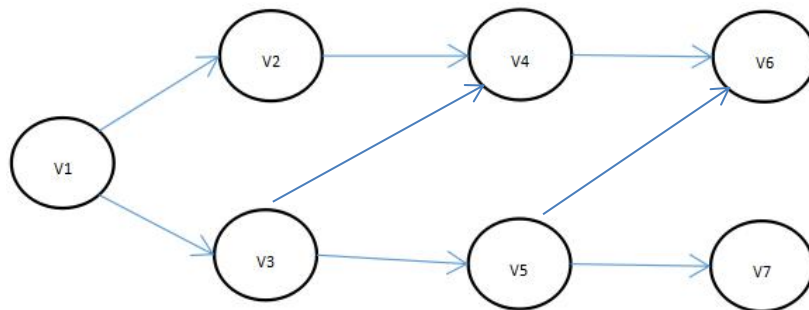
解：（1）所求邻接矩阵如下：

	V1	V2	V3	V4	V5	V6
V1	0	4	∞	5	3	∞
V2	4	0	2	∞	∞	6
V3	∞	2	0	∞	∞	4
V4	5	∞	∞	0	7	∞
V5	3	∞	∞	7	0	8
V6	∞	6	4	∞	8	0

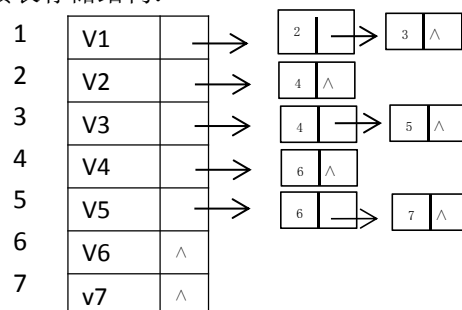
（2）所求最小生成树如下：



4、请对下面的有向图给出它的邻接表存储结构；并给出 4 个拓扑序列。



解：邻接表存储结构：



拓扑序列：

V1 V2 V3 V4 V5 V6 V7
 V1 V2 V4 V3 V5 V6 V7

V1 V3 V2 V4 V5 V6 V7
V1 V3 V5 V2 V4 V6 V7

拓扑排序遵循的方法：

- 1.从有向图中选择一个没有前驱（即入度为0）的顶点并且输出它。
- 2.从图中删除该顶点，并且删去从该店出发的全部有向边。
- 3.重复上述两步，直到剩余的图中不在没有前驱的顶点为止。


5、给出一组关键字：29，18，25，47，58，12，51，10，写出按快速排序和直接插入排序两趟的排序结果。


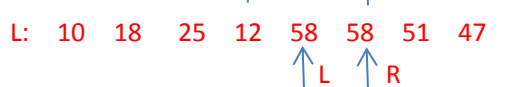
解：快速排序：

第一趟： (10 18 25 12) 29 (58 51 47)

第二趟： 10 (18 25 12) 29 (47 51) 58

注：第一趟：

初始： 29 18 25 47 58 12 51 10 基准值取第一个 29

 R: 10 18 25 47 58 12 51 29 L = R

L: 10 18 25 47 58 12 51 47 R = L

 R: 10 18 25 12 58 12 51 47 L = R

 L: 10 18 25 12 58 58 51 47 R = L

下一步的时候 R 移动到 L 的位置 R==L 赋值 基准值 29 即如下

R: 10 18 25 12 29 58 51 47

第二趟：分两部分 (10 18 25 12) 29 (58 51 47)

前半部分以 10 作为基准发现已经是 排好序的 则为 10 (18 25 12)

后半部分 58 作为基准值

R: 47 51 47 L = R


L: 47 51 47

 没有找到 比基准值大的 但 L==R 此时赋值 58
 即 47 51 58

最终为 10 (18 25 12) 29 (47 51) 58

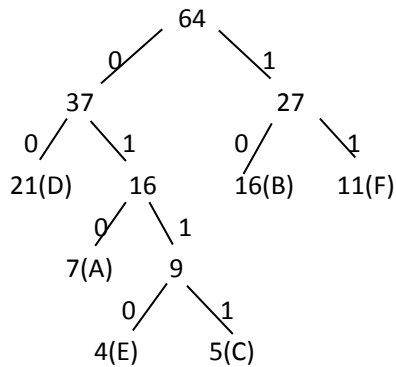
直接插入排序：

第一趟： 18 29 25 47 58 12 51 10

第二趟： 18 25 29 47 58 12 51 10

6、已知字符 A、B、C、D、E、F 的权值为 7、16、5、21、4、11，请写出构造哈夫曼树的过程，并为这些字符设计哈夫曼编码。

解：根据权值构建哈夫曼树：

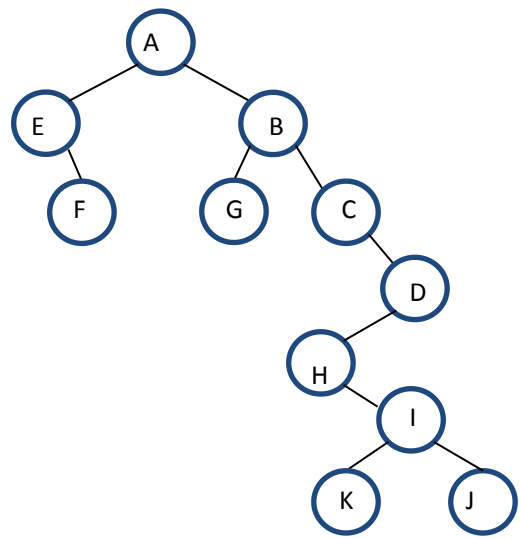


由上图可得：

A: 010 B: 10 C: 0111 D: 00 E: 0110 F: 11

7、已知二叉树的先序遍历序列是 AEFBGCDHIKJ，中序遍历序列是 EFAGBCHKIJD，画出此二叉树。并给出它的后序遍历序列。

解： 由题可得该二叉树为：



第一步 根据先序 AEFBGCDHIKJ	确定	根为 : A
第二步 根据中序 EFAGBCHKIJD	确定	左子树 : EF 右子树: GBCHKIJD
第三步 根据先序 EF	确定	根为 : E
BGCDHIKJ	确定	根为 : B
第四步 根据中序 EF	确定	没有左子树 右子树: F
GBCHKIJD	确定	左子树 : G 右子树: CHKIJD
第五步 根据先序 CDHIKJ	确定	根为 : C
第六步 根据中序 CHKIJD	确定	没有左子树 右子树: HKIJD
第七步 根据先序 DHIKJ	确定	根为 : D
第八步 根据中序 HKIJD	确定	没有右子树 左子树: HKIJ

第九步 根据先序 HKIJ	确定	根为 : H
第十步 根据中序 HKIJ	确定	没有左子树 右子树: KIJ
第十一步 根据先序 IKJ	确定	根为 : I
第十二步 根据中序 KIJ	确定	左子树 : K 右子树: J

后续遍历序列为: FEGKJIDCBA

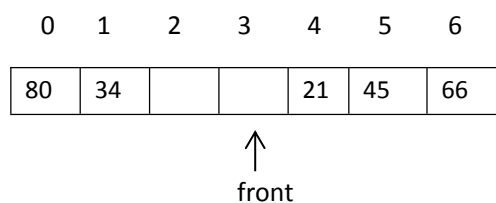
先序遍历: (1) 访问根结点;
 (2) 先序遍历左子树;
 (3) 先序遍历右子树;

中序遍历:
 (1) 中序遍历左子树;
 (2) 访问根结点;
 (3) 中序遍历右子树;

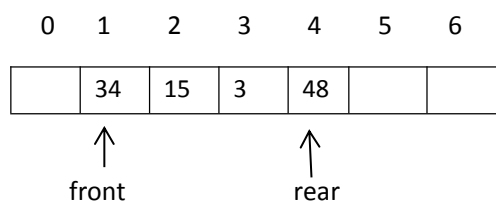
后序遍历:
 (1) 后序遍历左子树;
 (2) 后序遍历右子树;
 (3) 访问根结点;

8、假定用一个数组 `que[7]` 顺序存储一个循环队列，队首和队尾指针分别用 `front` 和 `rear` 表示，当前队列中已有 5 个元素: 21, 45, 66, 80, 34。其中 21 为队首元素，`front` 的值为 3，请画出对应的存储状态；当连续做 4 次出队运算后，再让 15, 3, 48 元素依次进队，请再次画出对应的存储状态。

解: 当 21 为队首元素，`front` 值为 3 时的存储状态如下:



当连续做 4 次出队运算后，15, 3, 48 进队后的存储状态如下:



四、算法填空题

1、下面的是实现在二叉排序树中插入一个关键字的值为 `kx` 的结点的算法，先对 `kx` 进行检索，若查找成功，则不插入，若失败，则插入相应位置，保持其二叉树的特性。试在算法中的___处填上正确的内容，完成该算法。

```
typedef struct node{
```

```
    datatype key;
```

```

    struct node* lchild,*rchild;
}bstnode,*bstree;
Void InsertBstree(bstree*t,datatype kx)
{
    bstree f,p=*t;
    while( P!=Null )
    {
        if(kx==p->key)
            return;
        f=p;
        if(kx < p->key)
            p= p->lchild ;
        else
            p=p->rchild;
    }
    p=(bstnode*)malloc(sizeof(bstnode));
    P->key= kx ;
    p->lchild=NULL;
    p->rchild= Null ;
    if(!(*t))
        *t=p;
    else if(kx < f->key)
        f->lchild = p;
    else
        f->rchild = p;
}

```

2、下面程序段的功能是实现快速排序的一次划分，请在____处填上正确的语句。

```

struct record
{
    int key;
    datatype others;
}Recordtype;
int quickpass(Recordtype r[],int s,int t)/*对 r[s..t]进行排序*/
{
    int i,j;struct record x;
    i=s;j=t;   x= r[s] ;
    while(i<j)
    {
        while(i<j&& r[j].key>=x.key) j-- ;
        if(i<j){ r[i]=r[j] ; i=i+1;}
        while( i<j && r[i].key<=x.key) i=i+1;
        if(i<j)
        {

```

```

        r[j]=r[i];
        j=j+1;
    }
}
r[i]=x;
return;
}

```

五、算法设计题

1、写一算法，求带有头结点的单链表的表长。

```

typedef struct node
{
    int data;
    struct node*next;
}LNode,*LinkList;
int Length_Lklist(Linklist head)
{
    LNode*p=head->next;
    int num=0;
    while(p!=NULL)
    {
        num++;
        p=p->next;
    }
    return num;
}

```

2、试编写算法求二叉树中叶子节点的个数（采用二叉链表存储）

```

typedef struct BiTNode
{
    int data;
    struct BiTNode*lchild,*rchild;
}BiTNode,*BiTree;
void leaf(BiTree bt; int *count);
/*递归计算二叉树 bt 中叶子节点的数目*/
{
    if(bt)
    {
        if(!bt->lchild&&!(bt->rchild))
            (*count)++;
        else
        {
            leaf(bt->lchild,count);

```

```

        eaf(bt->rchild,count);
    }
}
}

```

3、请编写一个算法，统计出带头结点的单链表（头指针为 head）中值为 x 的结点的个数。

```

typedef struct node{
    datatype data;
    struct node*next;
}LNode;
int NunberX(LNode *head,datatype x)
{
    LNode *p=head->next;
    int num=0;
    while(p!=NULL)
    {
        if(p->data==x)
            num++;
        p=p->next;
    }
    return num;
}

```

4、请编写算法，在一棵已建好的二叉排序树中查找具有最大值的结点，要求返回指向该结点的指针。

```

typedef struct{
    datatype data;
    struct bnode*lchild,*rchild;
}BNode,*BiTree;
BNode*BstMax(BiTree bst)
{
    BNode*p,*q;
    p=bst;
    while(p!=NULL)
    {
        q=p;
        p=p->rchild;
    }
    return q;
}

```