.

作业：1-1, 7, 8　2-1，2, 4, 7, 9, 11, 13，19 3-2, 3，7, 8, 13, 14

4-3, 9, 13　5-1, 2, 6, 8　　5-1, 2, 6, 7，8，12, 14, 17

习题1　绪论

1-1 名词解释：数据结构。

数据结构：相互之间存在一定**关系**的数据元素的集合

1-2 数据结构的基本逻辑结构包括哪四种?

⑴ 集合：数据元素之间就是“属于同一个集合”
⑵ 线性结构：数据元素之间存在着一对一的线性关系
⑶ 树结构：数据元素之间存在着一对多的层次关系
⑷ 图结构：数据元素之间存在着多对多的任意关系

1-3 数据结构一般研究的内容不包括(　　)。

(A) 集合的基本运算
(B) 数据元素之间的逻辑关系
(C) 在计算机中实现对数据元素的操作
(D) 数据元素及其关系在计算机中的表示

选 D

数据的逻辑结构、数据的存储结构、数据的运算

1-4 算法包括哪五种特性?

2. 算法的五大特性：√
⑴ 输入：一个算法有零个或多个输入。
⑵ 输出：一个算法有一个或多个输出。
⑶ 有穷性：一个算法必须总是在执行有穷步之后结束，且每一步都在有穷时间内完成。
⑷ 确定性：算法中的每一条指令必须有确切的含义，对于相同的输入只能得到相同的输出。
⑸ 可行性：算法描述的操作可以通过已经实现的基本操作执行有限次来实现。

1-5 简述算法及其时间复杂度。

1.算法（Algorithm）:是对**特定问题**求解步骤的一种描述，是**指令**的**有限序列**。

.

.

算法复杂度(Algorithm Complexity)：算法占用机器资源的多少，主要有算法运行所需的机器时间和所占用的存储空间。
时间复杂度(Time Complexity)：算法运行所需要的执行时间，T(n)= O(f(n))。
空间复杂度(Space Complexity)：算法运行所需要的存储空间度量，S(n)= O(f(n))。

1-6 设数组 A 中只存放正数和负数。试设计算法，将 A 中的负数调整到前半区间，正数调整到后半区间。分析算法的时间复杂度。

```
A[n+1]
For(int i=n-1,j=0;i>j;i--)
{
    If(a[i]>0) continue;
    Else {
    A[n]=A[i];
    A[i]=A[j];
    A[j]=A[n];
    J++;
    }
}
```

时间复杂度为 O(n)

1-7 将上三角矩阵 A=(aij)n×n 的非0元素逐行存于 B[(n*(n+1)/2]中，使得 B[k]=aij 且 k=f1(i)+f2(j)+c (f1, f2不含常数项)，试推导函数 f1, f2和常数 c。

k+1=1+2+3+…+(i-1)+j
k=1/2*i*(i-1)+j-1;

1-8 描述下列递归函数的功能。

```
int F(int m, int n)
{
        if (n>m) return F(n, m);
        else if (n==0) return m;
                else
        {
            r=m%n;
            return F(n, r);
        }
}
```

求 m 与 n 的最大公约数

1-9 编写递归算法：

$$g(m,n)=\begin{cases}0, & m=0\text{且 } n\geq 0\\ g(m-1,2n)+n, & m>0\text{且 } n\geq 0\end{cases}$$

.

.

```
double g(double    m,double    n)
{
      If(m==0&&n>=0)
            return 0;
      else
            return g(m-1,2*n)+n;
}
```

1-10 将下列递归过程改写为非递归过程。

```
void test(int &s)
{
            int x;
            scanf (“%d”, &x);
            if (x==0) s=0;
            else
      {
            test(s);
            s+=x;
      }
}
```

习题2　表

2-1 如果长度为 n 的线性表采用顺序存储结构存储，则在第 i (1≤i≤n+1)个位置插入一个新元素的算法的时间复杂度为(　　)。

(A) O(1)　　(B) O(n)　　(C) O(nlog2n)　　(D) O(n2)

B

需要让线性表移动 n+1-i 个

2-2 在一个有127个元素的顺序表中插入一个新元素，要求保持顺序表元素的原有(相对)顺序不变，则平均要移动(　　)个元素。

(A) 7　　(B) 32　　(C) 64　　(D) 127

C　　n/2+1

2-3 将关键字2，4，6，8，10，12，14，16依次存放于一维数组 A[0...7]中，如果采用折半查找方法查找关键字，在等概率情况下查找成功时的平均查找长度为(　　)。

(A) 21/8　　(B) 7/2　　(C) 4　　(D) 9/2

A

3,2,3,1,3,2,3,4

公式法 1*2^0+2*2^1+3*2^2+…+i*2^(n-1);

.

.

2-4 已知顺序表 L 递增有序。设计一个算法，将 a 和 b 插入 L 中，要求保持 L 递增有序且以较高的效率实现。

先用折半查找法查询位置，然后移动

```
void insert(int L[],int a,int b)//a<b
{
    int i=0,p,q;
    n= length(L);//L 现有长度
    //查找确定 a、b 的位置
    for(;i<n;i++)
    {
        if( L[i]<=a&&(a<L[i+1]||i==n-1) )
        {
            p = i+1; //a 的最终位置
            n++;
            break;
        }
    }
    for(;i<n;i++)
    {
        if( L[i]<=b&&(b<L[i+1]||i==n-1) )
        {
            q = i+2; //b 的最终位置
            n++;
            break;
        }
    }
    //移动元素，插入 a，b
    for(i=n+1;i>q;i--)
      L[i] = L[i-2];
    L[q] = b;//插入 b
    for(i=q-1;i>p;i--)
          L[i] = L[i-1];
    L[p] = a;//插入 a
}
```

2-5 简单描述静态查找和动态查找的区别。

A 静态查找表只查找
B、静态查找表不改变数据元素集合内的数据元素
C、动态查找表不只查找
D、动态查找表还插入或删除集合内的数据元素

2-6 综合比较顺序表和链表。

(1)存储空间利用率高——只存储元素值。

.

.

(2)随机存取——可以通过计算来确定顺序表中第 i 个数据元素的存储地址 Li = L0+(i-1)*m,其中，L0为第一个数据元素的存储地址，
m 为每个数据元素所占用的存储单元数。
(3)插入和删除数据元素会引起大量结点移动.

顺序表：内存中地址连续
长度不可变更
支持随机查找 可以在 0(1)内查找元素
适用于需要大量访问元素的 而少量增添/删除元素的程序
链表 ：内存中地址非连续
长度可以实时变化
不支持随机查找 查找元素时间复杂度 0（n）
适用于需要进行大量增添/删除元素操作 而对访问元素无要求的程序

2-7 解释链表的“头指针、头结点和首元素结点”三个概念。
“头指针”是指向头结点的指针。
"头结点"是为了操作的统一、方便而设立的，放在首元素结点之前，其数据域一般无意义（当然有些情况下也可存放链表的长度、用做监视哨等等）。
“首元结点”也就是第一元素结点，它是头结点后边的第一个结点。
2-8 描述下列算法的主要功能是(　　)。
① 构造头结点 L，取 q=L;
② 产生1个结点 p;
③ q->next=p;
④ 输入 p->data 的值;
⑤ 取 q=p;
⑥ 重复执行②至⑤n 次;
⑦ p->next=NULL;

(A) 通过输入 n 个数据元素构建链表 L
(B) 采用前插法，在链表 L 中输入 n 个数据元素
(C) 通过产生 n 个结点构建链栈 L，q 为栈顶指针
(D) 在链队列 L 中输入 n 个数据元素，q 为队尾指针

A
2-9 设 L 是不带头结点的单链表的头指针，k 是一个正整数，则下列算法的主要功能是(　　)。

```
LinkSearch (LinkList L, int k)
{
    k0=0;
    p=L->next;   // next 为单链表的指针域
    q=p;
    while ( p )
```

.

.

```
    {
        if (k0<=k) k0++;
        else q=q->next;
        p=p->next;
    }
    q->next=0;
}
```

(A) 计算单链表 L 的长度
(B) 查找单链表 L 中倒数第 k 个结点
(C) 删除单链表 L 中最后面的 k 个结点
(D) 将单链表 L 中第 k 个结点 q 的指针置0

只遍历一次的高效算法
（排除法）
B

2-10 设链表 L 不带头结点，试分析算法的功能。

```
A(Linklist &L)
{
    if (L && L->next)
    {
        Q=L;
        L=L->next;
        P=L;
        while (P->next) P=P->next;
        P->next=Q;
        Q->next=NULL;
    }
} //A 算法结束
```

将链表的第一个结点接到最后一个结点后面

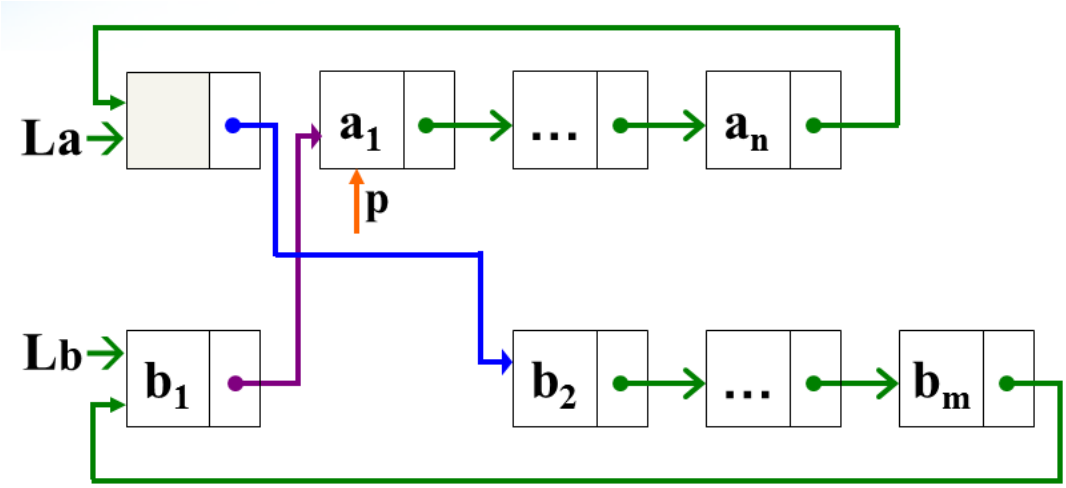
2-11 设两个循环链表的长度分别为 n 和 m，则将这两个循环链表连接成一个循环链表，最好的时间复杂度为(　　)。

(A) O(1)　　(B) O(n)　　(C) O(m)　　(D) O(min(n，m))

A
首先取一个指针 p 指向 la 的第一个节点（不包括头节点，头节点是空），然后让 la 头指针指向 lb 的第二个节点，接着用 lb 的第一个节点填充 lb 的头节点，最后将 la 头节点 next 指向 p
如下图：

.

.



还是不明白请自己看 ppt 第二章 P65

2-12 设有6个数据元素 A，B，C，D，E，F 依次进栈。如果6个数据元素的出栈顺序为 B，C，D，F，E，A，则该栈的最小容量为(　　)。

(A) 2　　(B) 3　　(C) 4　　(D) 5

B

操作	栈内元素	出栈顺序
A，B 入栈	A,B	
B 出栈	A	B
C 入栈	A,C	
C 出栈	A	B,C
D 入栈	A,D	
D 出栈	A	B,C,D
E,F 入栈	A,E,F	
F 出栈	A,E	B,C,D,F
E 出栈	A	B,C,D,F,E
A 出栈		B,C,D,F,E,A

因此栈的最小容量只需3

2-13 设进栈序列为123，试给出所有可能的出栈序列。

所有可能的出栈序列为：

1, 2, 3 （1入栈，1出栈，2入栈，2出栈，3入栈，3出栈）
1, 3, 2 （1入栈，1出栈，2, 3, 入栈，3出栈，2出栈）
2, 1, 3 （1, 2入栈，2出栈，1出栈，3入栈，3出栈）
2, 3, 1 （1, 2入栈，2出栈，3入栈，3出栈，1出栈）
3, 2, 1 （1, 2, 3入栈，3出栈，2出栈，1出栈）

注意：唯一只有3,1,2不可能出现，因为3要先出栈，前面1,2,3要按顺序一起入栈，因此不可能出现1在2的前面，后面的题目也是一样。

原则就是只要后入栈的先出栈，那么这个元素前面入栈的元素一定按照入栈顺序的反序排列

2-14 如果进栈序列为123456，能否得到出栈序列435612和135426?

435612 不能得到 6的后面不可能出现1,2的出栈顺序

135426 能够得到

.

.

2-15 简述算法的功能(设数据元素类型为 int)：

```
void proc(LinkQueue *Q)
{
    LinkStack S;
    InitStack(S);
    while(!EmptyQueue(Q) )
    {
        DeleteQueue(Q, d);
        Push(S,d);
    }
    while(!EmptyStack(S) )
    {
        Pop(S, d);
        InsertQueue(Q, d);
    }
}
```

将链队列中的顺序倒过来

如原来 ABCDEF，执行完这个算法之后是 FEDCBA

2-16 按照格式要求给出调用 F(3,'A','B','C')的运行结果：

```
void F(int n, char x, char y, char z)
{
    if (n==1) printf("1    %c → %c\n", x, z);
    else
    {
        F(n-1, x, z, y);
        printf("%d    %c → %c\n", n, x, z);
        F(n-1, y, x, z);
    }
}
```

自己去计算，类似汉诺塔

1 A->C
2 A->B
1 C->B
3 A->C
1 B->A
2 B->C
1 A->C

2-17 已知一维数组 B[0..20]用于存储一个下三角矩阵 A 元素。设 A 中第一个元素的下标为1，1，则数组元素 B[10]对应 A 中下标为(　　)的元素。

(A) 2，5　　(B) 4，3　　(C) 5，1　　(D) 6，1

C

.

.

$$\mathbf{L}=\begin{bmatrix} l_{1,1} & \cdots & & 0 \\ l_{2,1} & l_{2,2} & (0) & \\ l_{3,1} & l_{3,2} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n-1} \quad l_{n,n} \end{bmatrix}$$

下三角矩阵

因此 B 中第10个元素，也就是 B[9]对应 A[4][4]

[B[10]对应 A 中是 A[5][1]

2-18 已知 An×n 为稀疏矩阵。试从时间和空间角度比较，采用二维数组和三元组顺序表两种存储结构计算Σaij 的优缺点。

稀疏矩阵为 n 行 n 列.

1) 采用二维数组存储，其空间复杂度为O(n×n)；因为要将所有的矩阵元素累加起来，所以，需要用一个两层的嵌套循环，其时间复杂度亦为O(n×n)。
2) 采用三元组顺序表进行压缩存储，假设矩阵中有 t 个非零元素，其空间复杂度为O(t)，将所有的矩阵元素累加起来只需将三元组顺序表扫描一遍，其时间复杂度亦为O(t)。当 t << n×n 时，采用三元组顺序表存储可获得较好的时、空性能。

2-19 链地址法是 Hash 表的一种处理冲突的方法，它是将所有哈希地址相同的数据元素都存放在同一个链表中。关于链地址法的叙述，不正确的是( )。

(A) 平均查找长度较短 pptP165上面有表述
(B) 相关查找算法易于实现 查找的时候只需找到哈希地址的那条链，再顺着那条链遍历下去就可以实现。
(C) 链表的个数不能少于数据元素的个数 可以少于，很明显
(D) 更适合于构造表前无法确定表长的情况 链表的特点之一‘

C

2-20 设哈希(Hash)函数 H(k)=(3k)%11，用线性探测再散列法处理冲突，di=i。已知为关键字序列22，41，53，46，30，13，01，67构造哈希表如下：

哈希地址	0	1	2	3	4	5	6	7	8	9	10
关键字	22		41	30	01	53	46	13	67		

则在等概率情况下查找成功时的平均查找长度是( )。

(A) 2 (B) 24/11 (C) 3 (D) 3.5

有公式 $\frac{1}{2}(1+\frac{1}{1-\alpha})$ ASL=1/2(1+1/(1-a)) = 1/2(1+1/(1+11/3))=7/3 其中 a = 8/11（实际填入的数量/线性表的大小）

2-21 有100个不同的关键字拟存放在哈希表 L 中。处理冲突的方法为线性探测再散列法，其平均查找长度为 $\frac{1}{2}(1+\frac{1}{1-\alpha})$。试计算 L 的长度(一个素数)，要求在等概率情况下，查找成功时的平均查找长度不超过3。

素数表：101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167。

.

.

由于要求平均查找长度≤3，则

$$ASL=\frac{1}{2}(1+\frac{1}{1-\alpha})\leqslant 3 \rightarrow \alpha\leq 0.8$$

设线性表 L 长度 ln,有:
α =100/ln<=0.8　求出 ln>=125,即由题意选择127这个素数

习题3　树

3-1 若深度为5的完全二叉树第5层有3个叶子结点，则该二叉树一共有(　　)个结点。

(A) 8　　(B) 13　　(C) 15　　(D) 18

D
利用公式　前四层有2^5-1 = 15个节点，总共为15+3=18个。

3-2 设树 T 的度为4，其中度为1，2，3和4的结点个数分别为4，2，1，1　则 T 中的叶子数为(　　)。

(A) 5　　(B) 6　　(C) 7　　(D) 8

一共有1*4+2*2+3+4=15个度，4+2+1+1=8个节点

叶子为15-8+1=8

解析：节点数=度数+1

此题也可用画图法（图略）

3-3 已知二叉树 T 中有50个叶子结点，试计算 T 的总结点数的最小值。

由于是二叉树，那么可知欲使节点数最小，则该二叉树需满足至多存在一个结点的入度为1（即——使每两个结点都有一个父节点）。50/2=25,　（25-1）/2=12 12/2=6 6/2=3 (3+1)/2=2 2/2=1　红色部分+1为之前25个结点归根时剩下的1个。那么总共有50+25+12+6+3+2+1=99个结点

节点数/2+1 =叶子数

3-4 已知一棵度为 k 的树中，有 n1个度为1的结点，n2个度为2的结点，…，nk 个度为 k 的结点。试计算该树的叶子结点数。

解析：节点数=度数+1

.

.

叶子结点为 $\sum_{0}^{k} mn_k - \sum_{0}^{k} n_k + 1$

3-5 对于任意非空二叉树，要设计出其后序遍历的非递归算法而不使用栈结构，最适合的方法是对该二叉树采用(　　)存储结构。

(A) 二叉链表　　(B) 三叉链表　　(C) 索引　　(D) 顺序

B

解析：三叉链表比二叉链表多一个指向父结点的指针

3-6 一棵二叉树的叶子结点在其先序、中序和后序序列中的相对位置(　　)。

(A) 肯定发生变化　(B) 可能发生变化　(C) 不会发生变化　(D) 无法确定

B

解析：举例子说明即可

3-7 设二叉树 T 按照二叉链表存储，试分析下列递归算法的主要功能。

```
int F(BiTree T)
{
    if (!T) return 0;
    if (!T->Lchild && !T->Rchild) return 1;
    return F(T->Lchild)+F(T->Rchild);
}
```

求二叉树 T 的叶子结点数

```
int F(BiTree T)
{
    if (!T) return 0;
    if (!T->Lchild && !T->Rchild) return 1;
    return F(T->Lchild)+F(T->Rchild)+1;
}
```

求二叉树 T 的结点数

3-8 已知二叉树 T 的先序序列为 ABCDEF，中序序列为 CBAEDF, 则 T 的后序序列为(　　)。

(A) CBEFDA　　(B) FEDCBA　　(C) CBEDFA　　(D) 不确定

A

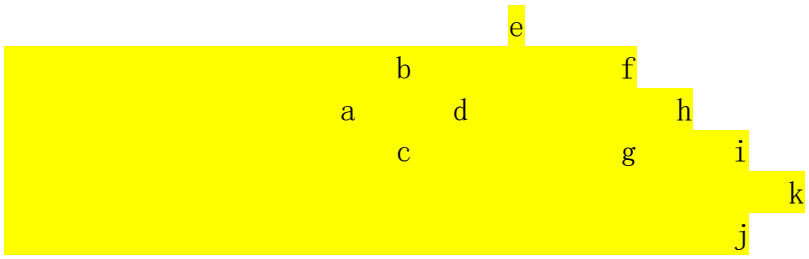
3-9 简述由先序序列和中序序列构造二叉树的基本操作方法。

.

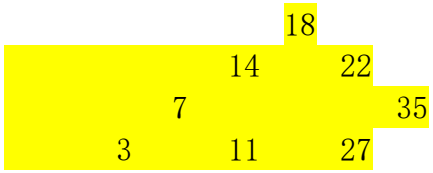
.

1）取先序遍历序列的第一个值，用该值构造根结点，，然后在中序遍历序列中查找与该元素相等的值，这样就可以把序列分为三部分：左子树（如果有）、根结点和右子树（如果有）。
2）将两个序列都分成三部分，这样就分别形成了根结点的左子树和右子树的先序遍历和后序遍历的序列。
3）重复 1）和 2）步骤，直至所有结点都处理完就可以完整构成一颗二叉树了。

3-10 已知二叉树的先序序列为 ebadcfhgikj，中序序列为 abcdefghijk，试画出该二叉树。



3-11 已知二叉树 T 的中序序列和后序序列分别为
(中序) 3, 7, 11, 14, 18, 22, 27, 35
(后序) 3, 11, 7, 14, 27, 35, 22, 18
试画出二叉树 T。



3-12 已知二叉树 T 按照二叉链表存储，设计算法，计算 T 中叶子结点的数目。

```
int F(BiTree T)
{
    if (!T) return 0;
    if (!T->Lchild && !T->Rchild) return 1;
    return F(T->Lchild)+F(T->Rchild);
}
```

3-13 已知二叉树 T 按照二叉链表存储，设计算法，交换 T 的左子树和右子树。
递归：

```
Int  ExchangeBTree(BiTree T)
{
    temp=(BiTree) malloc(sizeof(BiTNode));
    if(!T) return;
    if(!T->lchild&&!T->rchild) return;
    temp=T->lchild;
    T->lchild=T->rchild;
    T->rchild=temp;
    ExchangeBTree(T->lchild);
    ExchangeBTree(T->rchild);
```

.

.

}

3-14 先序后继线索化算法是根据二叉链表建立先序后继线索二叉链表，其基本原则是在前驱空指针域中写入后继线索，即将右子树的(　　)指针写入左子树的最后一个叶子结点右指针域。

(A) 线索　　(B) 根结点　　(C) 前驱结点　　(D) 后继结点

C

3-15 设计算法，在先序线索二叉树中，查找给定结点 p 在先序序列中的后继。

线索二叉树：根据某次遍历，　在二叉树中的相关空指针域都写入线索(后继线索或前驱线索)，即成为线索二叉树。
线索二叉树可理解为已经线索化的二叉树。
先序后继：先序遍历中得到的后继(先序前驱，　中序后继，　中序前驱，　后序后继，　后序前驱)。
线索二叉树的存储结构

```
typedef struct Node {
    Type  data; //数据元素域
    struct Node *Lchild; //左孩子域
    struct Node *Rchild; //右孩子域
    int Tag; //0: 分支结点，  1: 叶子结点
} BiTNode, *BiTree;
findBirthNode(BiTNode p)
{
    If(p->tag==0)//p 的左子树非空,指向左子树
        Return p->Lchild;
    Else //p 为空，后驱则是右子树
        Return p->Rchild;
}
```

3-16 设计一种二进制编码，使传送数据 a，act，case，cat，ease，sea，seat，state，tea 的二进制编码长度最短。
要求描述：
(1)数据对象；a, c, t, s, e
(2)权值集；8, 3, 5, 5, 6
(3)哈夫曼树；略
(4)哈夫曼编码。00, 010, 011, 10, 11

3-17 按照“逐点插入方法”建立一个二叉排序树，树的形状取决于(　　)。
(A) 数据序列的存储结构　　(B) 数据元素的输入次序

.

.

(C) 序列中的数据元素的取值范围　　(D) 使用的计算机的软、硬件条件

B

显然 D 错误，A 也错误因为大小是相对的，对于 C，1，3,5 和1，10,100相对位置一样，假若插入次序也一样的话，树的形状也不会变得，所以选 B

3-18 用利用逐点插入法建立序列(50, 72, 43, 85, 75, 20, 35, 45, 65, 30)对应的二叉排序树以后，查找元素35要在元素间进行(　　)次比较。

(A) 3　　(B) 4　　(C) 5　　(D) 8

B

1　　50

2　　43　　72

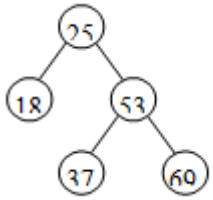
3　20　　45　　65　　85

4　　35　　75

　　30

3-19 在平衡二叉树中，插入关键字46后得到一颗新的平衡二叉树。在新的平衡二叉树中，关键字37所在结点的左、右孩子结点中保存的关键字是(　　)。

(A) 18，46　　(B) 25，46　　(C) 25，53　　(D) 25，69



C

3-20 用依次插入关键字的方法，为序列{ 5, 4, 2, 8, 6, 9 }构造一棵平衡二叉树(要求分别画出构造过程中的各棵不平衡二叉树)。

每次都将平衡树平衡

3-21 给定 n 个整数，设计算法实现：

(1) 构造一棵二叉排序树；

(2) 从小到大输出这 n 个数。

```
int SearchBST(BiTree T, int key, BiTree &p)
{
    //在 T 中递归查找关键字值=key 的数据元素
    if (!T) return 0;     //查找不成功
    if (key==T->key) return 1;    //查找成功
    p=T;    //p 记录双亲结点
    if (key<T->key) return SearchBST(T->lchild, key, p);
```

.

```
        return SearchBST(T->rchild, key, p);
} //SearchBST

//  二叉排序树的插入算法
void InsertBST(BiTree &T, int a[n]) //数组保存 n 个整数
{
        BiTree p=T;    //p 指向双亲结点
Int i;
        For(i=0;i<n;i++)
        {
        if (SearchBST(T->lchild, a[i], p)) return 0;    //查找成功
        BiTree s=(BiTree) malloc(sizeof (BiTnode));    //申请结点
        s->key=a[i];
        s->lchild=s->rchild=NULL;
        if (!T->lchild) T->lchild=s;    //s 为根结点
        else if (a[i]<p->key) p->lchild=s;    //s 为 p 的左孩子
        else p->rchild=s;    //s 为 p 的右孩子
        }
} //InsertBST
  //用中序遍历即可从大到小输出
```

习题4　排序

4-1 在下列排序算法中，时间复杂度最好的是(　　)。

(A) 堆排序　　(B) 插入排序　　(C) 冒泡排序　　(D) 选择排序

A

4-2 如果顺序表中的大部分数据元素按关键字值递增有序，则采用(　　)算法进行升序排序，比较次数最少。

(A) 快速排序　　(B) 归并排序　　(C) 选择排序　　(D) 插入排序

D

4-3 对关键字序列 56, 23, 78, 92, 88, 67, 19, 34 进行增量为3的一趟希尔排序(Shell 升序或降序)的结果为(　　)。

(A) 19, 23, 78, 56, 34, 67, 92, 88　　(B) 23, 56, 78, 67, 88, 92, 19, 34

(C) 19, 23, 34, 56, 67, 78, 88, 92　　(D) 92, 88, 78, 56, 34, 67, 19, 23

D

4-4 若一组记录的关键码为46, 79, 56, 38, 40, 84，则利用快速排序方法且以第一个记录为基准，得到的一次划分结果为(　　)。

(A) 38，40，46，56，79，84　　(B) 40，38，46，79，56，84

(C) 40，38，46，56，79，84　　(D) 40，38，46，84，56，79

C

4-5 对数据84，47，25，15，21进行排序。如果前三趟的排序结果如下：

第1趟：15，47，25，84，21

第2趟：15，21，25，84，47

第3趟：15，21，25，47，84

.

则采用的排序方法是(　　)。

(A) 冒泡排序　　(B) 快速排序　　(C) 选择排序　　(D) 插入排序

C

4-6 对数据36，12，57，86，9，25进行排序，如果前三趟的排序结果如下：

第1趟：12，36，57，9，25，86

第2趟：12，36，9，25，57，86

第3趟：12，9，25，36，57，86

则采用的排序方法是(　　)。

(A) 希尔排序　　(B) 起泡排序　　(C) 归并排序　　(D) 基数排序

B

4-7 根据建堆算法，将关键字序列5，7，10，8，6，4调整成一个大顶堆，最少的交换次数为(　　)。

(A) 1　　(B) 2　　(C) 3　　(D) 4

B

4-8 在链式基数排序中，对关键字序列369, 367, 167, 239, 237, 138, 230, 139进行第1趟分配和收集后，得到的结果是(　　)。

(A) 167, 138, 139, 239, 237, 230, 369, 367

(B) 239, 237, 138, 230, 139, 369, 367, 167

(C) 230, 367, 167, 237, 138, 369, 239, 139

(D) 138, 139, 167, 230, 237, 239, 367, 369

C

4-9 设 int r[7]={5，2，6，4，1，7，3}; 则执行 for (i=0; i<7; i++) r[r[i]-1]=r[i]; 命令之后，数组 r[7]中的数据元素存放顺序是(　　)。

(A) 5，2，7，4，1，6，3　　(B) 3，2，1，4，5，7，6

(C) 1，2，3，4，5，6，7　　(D) A、B、C 都不对

这是一个计数排序，需要去好好看 ppt

D

4-10 设计一种排序算法，对1000个[0, 10000]之间的各不相同的整数进行排序，要求比较次数和移动次数 尽可能少。

采用计数排序

```
int a[10000], j=0;  //a: Hash表，  j: 计数

for(i=0; i<1000; i++) a[r[i]-1]=r[i];

for(i=0; i<10000; i++) if(a[i]>0) r[j++]=a[i];
```

4-11 给定序列 r[11]={30，25，12，16，46，21，27，38，9，18，31}，试分别给出一趟希尔排序(增量 m=3)和一趟快速排序(枢轴为 r[0])之后的序列 r[11]。

希尔排序：r[11]={16, 25, 9, 18, 31, 12, 27, 38, 21, 30, 46}

快速排序：r[11]={18, 25, 12, 16, 9, 21, 27, 30, 38, 46, 31}

4-12 对关键字序列503, 87, 512, 61, 908, 170, 897, 275, 653, 426分别执行下列排序算法，写出第1趟排序后的关键字序列：

(1)快速排序; {426,87,275,61,170,503,897,908,653,512}

(2)堆排序;　　{512,87,512,61,426,170,897,275,653,908}

.

.

(3)链式基数排序。{170, 61, 512, 503, 653, 275, 426, 87, 897, 908}

4-13 设顺序表的结点结构为(Type Key; int Next)，其中，Key 为关键字，Next 为链表指针。试设计静态链表排序算法。

根据静态链表调整记录序列(排序)：

```
Arrange(RecordType *r, int n) {
    for (i=1; i<n; ++i) {
        p=r[0].next;  r[p]⇔r[i]; //交换记录位置
        //将r[0].next指向第i个元素
        if (r[i].next!=i)  r[0].next=r[i].next;
        for (j=i+1; j<=n; j++) {
            if (r[j].next!=i) continue;
            r[j].next=p; break;} //修改指针
    }
} //算法时间复杂度为O(n²)
```

4-14 假设 n 个部门名称的基本数据存储在字符数组 name[N][31]中，其中 0≤n≤N≤90。试设计一个冒泡排序算法，将 n 个部门名称按字典序重新排列顺序。

```
void NameSort(RecordType **Name,int n)
{
    while(n>1)//一趟没有交换记录就弹出
    {
        i=1;
        for(j=1;j<n;j++)
            if(getNameSize(Name,j)>getNameSize(Name,j+1))
            {
                Name[j]<->Name[j+1];//交换
                i=j;
            }
        n=i;
    }
}
//计算每个部门名称的大小
int getNameSize(RecordType **Name,int j)
{
    int sum=0;
    for(k=0;k<=30;k++)
        sum = sum + Name[j][k];
    return sum;
```

.

.

```
}
```

4-15 设计基于顺序表存储结构的树形选择排序算法。

```
//基于顺序表存储结构的完全二叉树的选择排序
void Sorting(int L[],int n)
{
    for (int i=1; i<=n; i++)
    {
        printf("%5d",L[1]);    //输出根结点

        //将底层结点设置成“∞”
        int j=1;
        while(L[2*j]==L[1] || L[2*j+1]==L[1])
        {
            j*=2;
            if (L[j]!=L[1]) j++;
        }
        L[j]=X;

        //将第 i+1小的数据元素调整到根结点
        for (int k=j; k>0; k/=2)
        {
            if (k%2) j=L[k-1];
            else j=L[k+1];
            if (j<L[k]) L[k/2]=j;
            else L[k/2]=L[k];
        }
    }
}//Sorting
```

4-16 假设采用链表存储类型：

```
typedef struct RNode
{
    int key;    //数据域(也是关键字域)
    struct RNode *next;    //指针域
} RNode, *RList;
typedef RList R[N];     //链表类型，常变量 N≥n
```

又设 R[1..n]是[10, 999]之间的随机整数。试设计一个链表基数排序算法，将 R[n]中的数从小到大排序。排序结果仍存放在 R[n]中。

习题5　图

5-1 设某个非连通无向图有25条边，问该图至少有(　　)个顶点。

.

(A) 7　　(B) 8　　(C) 9　　(D) 10

C

5-2 设某无向图中有 n 个顶点 e 条边，则建立该图邻接表的时间复杂度为(　　)。

(A) O(n+e)　　(B) O(n^2)　　(C) O(ne)　　(D) O(n^3)

A

5-3 带权有向图 G 用邻接矩阵 R 存储，则顶点 i 的入度等于 R 中(　　)。

(A) 第 i 行非∞(或非0)的元素之和　　(B) 第 i 列非∞(或非0)的元素之和

(C) 第 i 行非∞(或非0)的元素个数　　(D) 第 i 列非∞(或非0)的元素个数

D

邻接矩阵　横坐标 头　纵坐标 尾

5-4 在关于图的遍历描述中，不正确的是(　　)。

(A) 图的深度优先搜索遍历不适用于有向图

(B) 图的深度优先搜索遍历是一个递归过程

(C) 图的遍历是从给定的源点出发访问图中每一个顶点一次

(D) 遍历的基本算法有两种：深度优先搜索遍历和广度优先搜索遍历

A

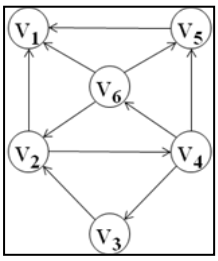
5-5 设计算法，由依次输入的顶点数目、狐的数目、各个顶点元素信息和各条狐信息建立有向图的邻接表。

5-6 请给出有向图的

(1) 每个顶点的入度和出度；

(2) 邻接矩阵；

(3) 邻接表。



5-7 设无向图 G=(V，E)，V={a，b，c，d，e，f}，E={(a，b)，(a，e)，(a，c)，(b，e)，(c，f)，(f，d)，(e，d)}。从顶点 a 出发对图 G 进行深度优先搜索遍历，得到的顶点序列是(　　)。

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

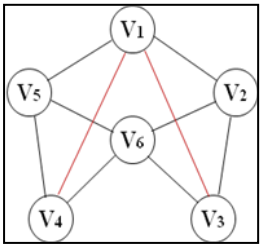
D

5-8 给出无向图的

(1)深度优先遍历的顶点序列和边序列；

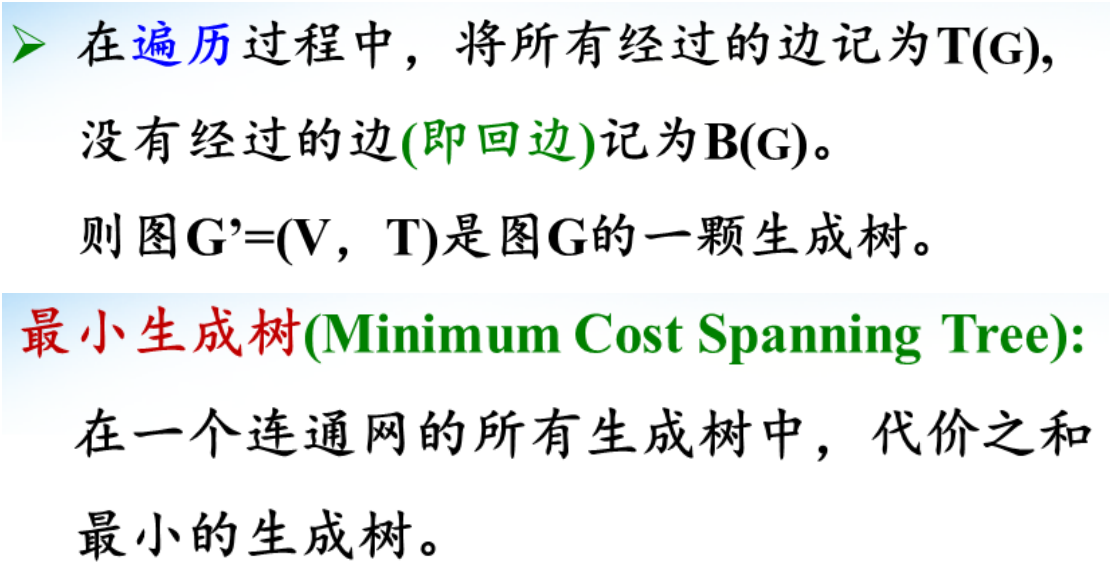
(2)广度优先遍历的顶点序列和边序列。

.



5-9 概念解释：最小生成树。

设 N=(V, E)是一连通网，U 是顶点集 V 的一个非空子集。若(u, v)是一条具有最小权值的边，其中 u U，v V-U，则存在一棵包含边(u, v)的最小生成树。



5-10 设无向图 G=(V, E)，V={a, b, c, d, e}，E={<a, b>, <a, c>, <a, d>, <b, c>, <c, e>, <d, e>}，G1=(V, E1)。如果 G1是 G 的生成树，则错误的是( )。

(A) E1={<a, b>，<a, c>，<a, d>，<c, e>}
(B) E1={<a, b>，<a, c>，<c, e>，<d, e>}
(C) E1={<a, c>，<b, c>，<c, e>，<d, e>}
(D) E1={<a, d>，<b, c>，<c, d>，<d, e>}

D

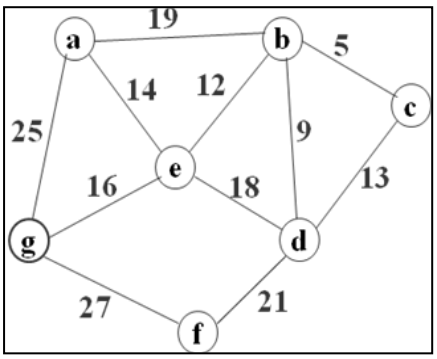
5-11 判断一个有向图是否存在回路，除了可以利用深度优先遍历算法外，还可以利用( )。

(A) 广度优先遍历算法　　(B) 求最短路径的方法
(C) 拓扑排序方法　　(D) 求关键路径的方法

C
或者深度优先排序

5-12 试绘出无向网 G 的最小生成树，并简要描述所依据的算法(或遍历方法)。

.

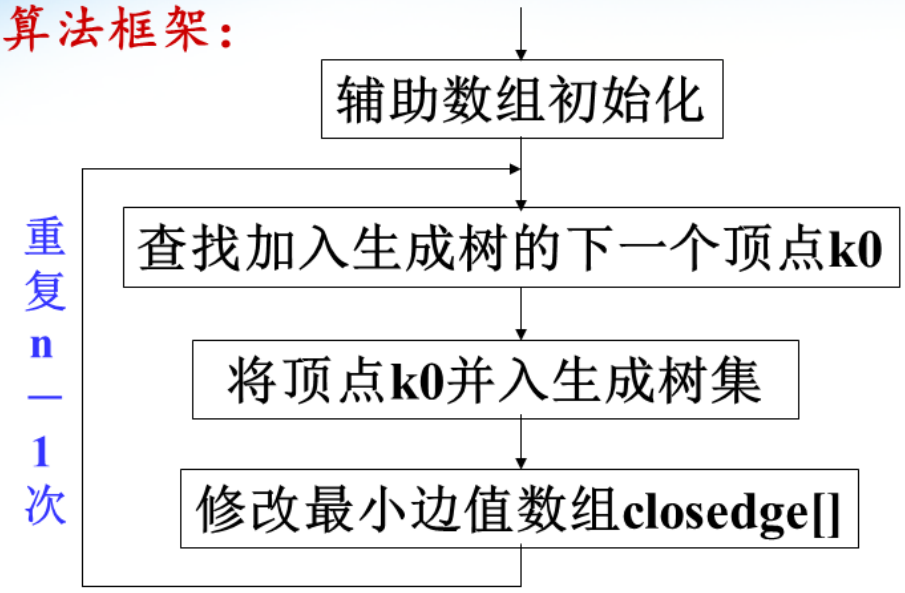


☞基本思想：任取一个顶点v作为生成树的根，然后往生成树上添加新顶点w：

➢添加的顶点w和已经在生成树上的顶点之间存在一条边，并且在所有与生成树的连通边中，该边的权值最小；

➢继续往生成树上添加顶点，直至生成树上含有n个顶点为止。

☞算法框架：



5-13 设带权无向图 G =(V, E)含有 n 个顶点 e 条边。试描述从顶点 u 出发，构造图 G 的最小生成树的克鲁斯卡尔(Kruskal)算法。

☞基本思路：为使生成树上边的权值之和达到最小，应使生成树中每一条边的权值尽可能地小(都能最小则最佳)。

具体做法：先构造一个只含n个顶点的子图SG，然后从权值最小的边开始，若它的添加不使SG中产生回路，则在SG上加上这条边。如此重复，直至加上n-1条边为止。

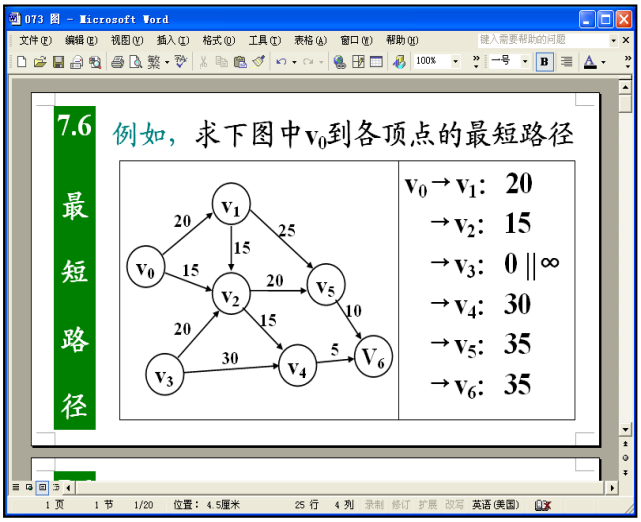
克鲁斯卡尔算法描述—构造最小生成树：

```
Kruskal(edge E[ ], int En) {
    link[Vn]={0};  n=0;  //n:已添加的边数
    for( i=0; i<En && n<Vn-1; i++) {
        v1=Find(link, E[i].beginV);
        v2=Find(link, E[i].endV);
        if (v1!=v2) { link[v1]=v2; n++; }
    }
}
```

```
int Find(int link[], int k) {
    while(link[k]>0)  k=link[k];
    return  k;
} // 找连通分量的尾部
```

5-14 在有向图中，路径(　　)是从 v0出发的一条最短路径。

.



(A) $v_0 \rightarrow v_1 \rightarrow v_5$　　(B) $v_0 \rightarrow v_2 \rightarrow v_3$　　(C) $v_0 \rightarrow v_2 \rightarrow v_4$　　(D) $v_0 \rightarrow v_2 \rightarrow v_5 \rightarrow v_6$

C

5-15 采用邻接表存储结构，设计一个算法，判别无向图 G 中指定的两个顶点之间是否存在一条长度为 k 的简单路径。

☞注：简单路径是指顶点序列中不含有重复的顶点。

```
//采用邻接表存储结构
int EXL(Graph G,int i,int j,int k)//顶点i到j长度为k
{
    if(i==j&&k==0)
        return 1;//找到一条路径，且长度符合要求
    else if(k>0)
    {
        visited[i] = 1;//将这个点标志为已走
        for(p=G.vexs[i].firstarc;p;p=p->next)//这个点邻接的第一条边
        {
            l=p->Adjx;
            if(!visited[l])
                if(EXL(G,l,j,k-1))
                    return 1;//剩余路径长度减1，直到减到0表示找到长度为k的路径
        }
    }
}
```

5-16 设带权有向图 G =(V, E)含有 n 个顶点 e 条边，采用邻接矩阵 Graph[n][n]表示。试设计算法 Dijkstra(int V_0，int n)，用于计算从源点 V_0到其它各顶点的最短路径。

.

☞迪杰斯特拉(Dijkstra)算法基本思想：

↳把图中所有顶点分成两组，第一组S包含已确定最短路径的顶点，初始值只含源点u；
第二组V-S包含尚未确定最短路径的顶点，
初始值含有图中除源点u之外的其它顶点。

↳按路径长度递增的顺序计算源点u到其它各顶点的最短路径，逐个把第二组中的顶点加到第一组中去，直至S=V为止。

算法框架：

(1) 选择v_k，使得D[k]=min{D[i] | v_i∈V-S}，
即求n-1条最短路径的最小值；

(2) 修改各顶点的D[i]值：

若D[k]+acrs[k][i]<D[i]，

则D[i]=D[k]+acrs[k][i]；

(3) 重复操作(1)和(2)操作n-1次。

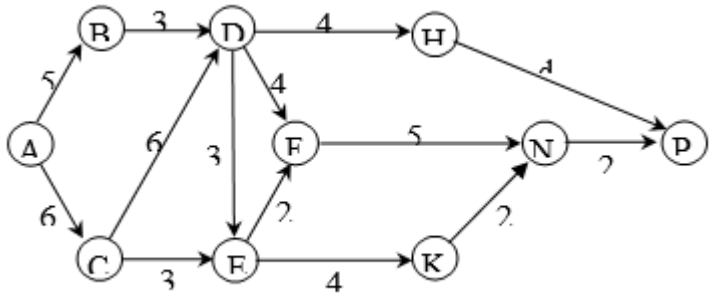
```
Dijkstra(int u) {
    for (j=1; j<n; j++) {
        //①寻找当前最小路径值:
        Min=∞; k=0; //最小值对应的顶点编号
        for (i=0; i<n; i++)
            if (Visited[i]=0 & D[i]<Min)
            { k=i; Min=D[i]; }
        Visited[k]=1;

        //②重新计算当前最短路径:
        for (i=0; i<n; i++) {
            sum=D[k]+acrs[k][i];
            if (Visited[i]=0 & sum<D[i])
            { D[i]=sum; path[i]=k; }
        }
    }
} //Dijkstra算法时间复杂度为O(n²)
```

5-17 设软件工程专业开设的主要课程如表所示：

代码	课程名称	先修课程
S1	高等数学	无
S2	程序设计基础	无
S3	离散数学	S1
S4	计算机组成原理	S2
S5	数据结构与算法	S2, S3
S6	操作系统	S4, S5
S7	数据库系统	S5
S8	编译技术	S5
S9	算法分析	S1, S5
S10	软件工程导论	S5
S11	计算机网络	S4, S6

试根据先修课程要求绘制课程体系拓扑结构图(结点用课程代码表示)。

5-18 描述关键路径基本概念，并给出有向图的关键路径。



关键路径基本概念：从源点到汇点的最长路径

本图的关键路径是：A →C→D→E→F→N→P