第一章作业

常用的逻辑结构包括哪几类？ 集合结构、线性结构、树状结构和图结构

分析下列算法的时间复杂度。

```
（1）for (int i = n; i > 1; i--) {
    for (int j = i - 1; j > 1; j--)
        x++;
    }
}
(2)  int i=1;
       while (i <= sum) {
            i = i * 2;
        }
（3）for (int i = 1; i < n; i++) {
        while (i <= n) {
            i = i * 2;
        }
    }
```

(1) O(n^2)　(2) O(log(n))　(3) O(nlog(n))

调用 mergesort(1,n),下列函数的时间复杂度为多少？并说明。

```
void mergesort(int i,int j)
{
     int mid;
     if (left!=right)
     {
          mid=(left+right)/2;
          mergesort(left,mid);
          mergesort(mid,right);
          merge(left,right,mid);    //该函数的时间复杂度为O(n)
       }
}
```

Nlogn

给出两个正整数 x 和 y，最大公因子(GCD)是能够同时被两个正整数整除的最大数。要求使用两种不同时间/空间复杂度的方法设计函数，求出 x 和 y 的最大公因子。

方法一：基于最通用的方法，测试每一种可能性。

```
int GCD（int x,int y）
{int g;
g=x;
while(x%g!=0||y%g!=0){
g--;
}
return g;
}
```

方法二：欧几里德算法描述如下：

(1) 取 x 除以 y 的余数，称余数为 r。
(2) 如果 r 是 0，过程完成，答案是 y。
(3) 如果 r 非 0，设 x 等于原来 y 的值，y 等于 r，重复整个过程。

```
int GCD（int x,int y）
{int r;
while(1){
r=x%y;
if(r==0)break;
x=y;
y=r;
}
return y;
}
```

编写一个程序，求出 n 个数据中的最大值，分析说明该算法的时间复杂度。

正确答案：

```
void main()
{
  int i, n;
  int List[MAXN];
  printf("请输入需要随机生成的数目,N=");
  scanf_s("%d", &n);
  if (n<0 || n>MAXN) {
      fprintf(stderr,"数字错误！\n");
     exit(1);
   }
   printf("\n 排序前的数组为：");
   for (i = 0; i < n; i++) {
      List[i] = rand() % 1000;
      printf("%d    ",List[i]);
   }
   int max=0;
   for (i = 0; i < n; i++) {
     if List[i]>=max
      {    max=List[i];}
}
```

时间复杂度 O(n)

第三章作业

栈和队列具有相同的（　）。
A、抽象数据类型　B、运算　C、存储结构　D、逻辑结构

设 h 为不带头结点的单向链表。在 h 的头上插入一个新结点 t 的语句是：( )。
A、h=t; t->next=h->next;　B、t->next=h->next; h=t;
C、h=t; t->next=h;　D、t->next=h; h=t;

如果循环队列用大小为 m 的数组表示，队头位置为 front、队列元素个数为 size，那么队尾元素位置 rear 为：
A、front+size B、front+size-1 C、(front+size)%m　D、(front+size-1)%m

假设有 5 个整数以 1、2、3、4、5 的顺序被压入堆栈，且出栈顺序为 3、5、4、2、1，那么为了获得这样的输出，堆栈大小至少为：
A、2　B、3　C、4　D、5

有六个元素以 6、5、4、3、2、1 的顺序进栈，问哪个不是合法的出栈序列？
A、2 3 4 1 5 6　B、3 4 6 5 2 1　C、5 4 3 6 1 2　D、4 5 3 1 2 6

若一个栈的入栈序列为 1、2、3、…、N，其输出序列为 p1、p2、p3、…、pN。若 p1=N，则 pi 为：
A、i　B、n-I　C、n-i+1　D、不确定

令 P 代表入栈，O 代表出栈。当利用堆栈求解后缀表达式 1 2 3 + * 4 - 时，堆栈操作序列是：
A、PPPOOPOO　B、PPOOPPOOPPOO　C、PPPOOPOOPPOO　D、PPPOOPOOPPOOPO

若采用带头、尾指针的单向链表表示一个堆栈，那么该堆栈的栈顶指针 top 应该如何设置？
A、将链表头设为 top　B、将链表尾设为 top
C、随便哪端作为 top 都可以　D、链表头、尾都不适合作为 top

利用大小为 n 的数组（下标从 0 到 n-1）存储一个栈时，假定栈从数组另一头开始且 top==n 表示栈空，则向这个栈插入一个元素时，修改 top 指针应当执行：
A、top=0　B、top++　C、top--　D、top 不变

表达式 a*(b+c)-d 的后缀表达式是：
A、a b c + * d -　B、a b c d * + -　C、a b c * + d -　D、- + * a b c d

已知 L 是有表头结点的非空循环单链表，删除 P 结点之后的结点，语句序列是：（ Q=P->next ）、（ P->next=Q->next）、free(Q);。

判断循环队列(容量为 MaxSize)满的条件为:(　　)，空的条件为:(　　)，元素个数=(　　)。
第一空：（rear+1）%MaxSize=front;
第二空：rear=front;　第三空：(Rear-Front+ MaxSize)% MaxSize

后缀表达式的 1 2 3 4 - / * 5 +的值（ 3 ）。

栈结构的特点是（ 先进后出 或者 后进先出 ）。

队列的特点是（ 先进先出 或者 后进后 ）。

采用递归方式求不带头结点的单链表的长度。
其中，单链表结点类型定义如下:

```
typedef struct LNode {
    int data;
    struct LNode* next;
} LinkNode;
//求单链表长度
int getLength(LinkNode* L) {
    if (L == NULL)
       return 0;
    return    （  getLength(L->next)+1  ）  ;//填空

}
```

为解决计算机主机与打印机之间速度不匹配问题，通常设置一个打印数据缓冲区，主机将要输出的数据依次写入该缓冲区，而打印机则依次从该缓冲区中取出数据。该缓冲区的逻辑结构应该是（ 队列 ）。

在具有 N 个结点的单链表中，访问结点和增加结点的时间复杂度分别对应为 O(1)和 O(N)。
×

在线性表的顺序存储结构中可实现快速的随机存取，而在链式存储结构中则只能进行顺序存取。
√

若一个栈的输入序列为 1，2，3，…，N，输出序列的第一个元素是 i，则第 j 个输出元素是 j-i-1。
×

栈是插入和删除只能在一端进行的线性表；队列是插入在一端进行，删除在另一端进行的线性表。
√

在 n 个元素连续进栈以后，它们的出栈顺序和进栈顺序一定正好相反。 √

n 个元素进队的顺序和出队的顺序总是一致的。 √

循环队列中有多少个元素可以根据队首指针和队尾指针的值来计算。 √

在用数组表示的循环队列中，front 值一定小于等于 rear 值。 ×

循环队列也存在着空间溢出问题。 √

循环链表可以做到从任一结点出发，访问到链表的全部结点。 √

线性表可用顺序表或链表存储。如果有 n 个表同时并存，并且在处理过程中个表的长度会动态发生变化，表的总数也可以自动改变，在此情况下，应选哪种存储表示？为什么？

采用链表。如果采用顺序表，在多个表并存的情况下，在问题求解的过程中，一旦发现某个表存满并溢出的情况，可能需要移动其他表以腾出位置为其扩充空间，导致不断地把大片数据移来移去，不但时间耗费多，且操作复杂。

设计一个算法，从给定的顺序表 L 中删除下标 i 到 j 的所有元素，假定 i 和 j 都是合法的位置。

```
typedef  struct LNode *PtrToLNode;
struct LNode{
    ElementType  Data[MAXSIZE];
    Position  Last;      /*当前最后一个元素的下标*/
};
```

```
typedef  PtrToLNode List;
List L;
void delete(List L,int i,int j)
{ int k,del;
   del=j-i+1; //删除的共 del 个数
   for(k=j+1;k<=L->Last;k++)  //要移位的是从下标 j+1 开始到 Last 的数
        L->Data[k-del]=L->data[k]; //从下标为 i 开始把后面的数拿过来覆盖
    L->Last-=del;// 改变顺序表的长度
}
```

有一个带头结点的单链表 L，设计一个函数使其元素递减有序。其中单链表结构定义如下：

```
typedef  struct LNode *PtrToLNode;
struct LNode{
       ElementType  Data;
       PtrToLNode Next;
};
typedef  PtrToLNode Position;
typedef  PtrToLNode List;
List L;
```

答案自己看：四、3

什么是队列的上溢现象和“假溢出”现象？解决它们有哪些方法？

在队列的顺序结构中,设队头指针为 front,队尾指针为 rear,队中元素的容量为 MaxSize。当有元素加入到队列时，若有 rear=MaxSize-1（初始时 rear=-1），则发生队列的上溢现象，即用于队列的存储空间已全部被队列元素所占用,欲加入到队列的元素不能入队。特别要注意的是队列的假溢出现象：

队列的存储空间还有剩余但元素却不能入队，造成这种现象的原因是由于队列的操作方法所致。
解决队列上溢的方法有以下：
1．建立一个足够大的存储空间，但这样做会造成空间的使用效率降低。
2．当出现假溢出时可采用以下三种方法：
法一：采用平移元素的方法，每当队列中加入一个元素时，队列中已有的元素向队头移动个位置（假溢出时必然有空闲的空间可供移动）；
法二：每当删除一个队头元素时，则依次移动队中的元素，使 front 指针固定不变且始终指向队列中的第一个元素位置；
法三：采用循环队列方式：把队列看成一个首尾相接的循环队列，在循环队列上进行插入或者删除，运算时仍遵循“先进先出”的原则。

单链表逆转。本题要求实现一个函数，将给定的单链表逆转。
函数接口定义：
```
List Reverse( List L );
```
其中 List 结构定义如下：
```
typedef struct Node *PtrToNode;
struct Node {
    ElementType Data; /* 存储结点数据 */
    PtrToNode   Next; /* 指向下一个结点的指针 */
};
typedef PtrToNode List; /* 定义单链表类型 */
```
L 是给定单链表，函数 Reverse 要返回被逆转后的链表。
```
List Reverse( List L )
{
  List p,q;
  p=L;
  L=NULL;
  while(p)
  {
    q=p;
    p=p->Next;
    q->Next=L;
    L=q;
  }
  return L;
}
```

第四章作业

在下述结论中，正确的是：（①④）
① 只有 2 个结点的树的度为 1；
② 二叉树的度为 2；
③ 二叉树的左右子树可任意交换；
④ 在最大堆（大顶堆）中，从根到任意其它结点的路径上的键值一定是按非递增有序排列的。

已知一棵完全二叉树的第 6 层（设根为第 1 层）有 8 个叶结点，则该完全二叉树的结点个数最多是：（　　）
A、39　　B、52　　C、111　　D、119

将{28, 15, 42, 18, 22, 5, 40}依次插入初始为空的二叉搜索树。则该树的后序遍历结果是：（　　）
A、5, 15, 18, 22, 40, 42, 28　　B、5, 22, 15, 40, 18, 42, 28
C、28, 22, 18, 42, 40, 15, 5　　D、5, 22, 18, 15, 40, 42, 28

二叉树的中序遍历也可以循环地完成。给定循环中堆栈的操作序列如下（其中 push 为入栈，pop 为出栈）：
push(1), push(2), push(3), pop, push(4), pop, pop, push(5), pop, pop, push(6), pop
以下哪句是对的？（　　）
A、6 是根结点　B、2 是 4 的父结点　C、2 和 6 是兄弟结点　D、以上全不对

下列叙述正确的是（　　）。
A、在任意一棵非空二叉搜索树，删除某结点后又将其插入，则所得二叉搜索树与删除前原二叉搜索树相同。
B、二叉树中除叶结点外，任一结点 X，其左子树根结点的值小于该结点（X）的值；其右子树根结点的值≥该结点（X）的值，则此二叉树一定是二叉搜索树。
C、虽然给出关键字序列的顺序不一样，但依次生成的二叉搜索树却是一样的。
D、在二叉搜索树中插入一个新结点，总是插入到最下层，作为新的叶子结点。

将一系列数字顺序一个个插入一棵初始为空的 AVL 树。下面哪个系列的第一次旋转是“右-左”双旋？（　　）
A、1,2,3,4,5,6　B、6,5,4,3,2,1　C、4,2,5,6,3,1　D、3,1,4,6,5,2

对最小堆（小顶堆）{1,3,2,12,6,4,8,15,14,9,7,5,11,13,10} 进行三次删除最小元的操作后，结果序列为：（　　）
A、4,5,6,7,8,9,10,11,12,13,14,15
B、4,6,5,12,7,10,8,15,14,9,13,11
C、4,6,5,13,7,10,8,15,14,12,9,11
D、4,5,6,12,7,10,8,15,14,13,9,11

将 {28, 15, 42, 18, 22, 5, 40} 逐个按顺序插入到初始为空的最小堆（小根堆）中。则该树的前序遍历结果为：（　　）
A、5, 18, 15, 28, 22, 42, 40　　B、5, 15, 18, 22, 28, 42, 40
C、5, 18, 28, 22, 15, 42, 40　　D、5, 15, 28, 18, 22, 42, 40

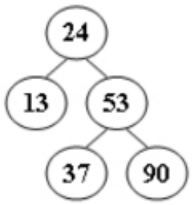
对二叉搜索树进行（　　）可以得到从小到大的排序序列。
A、前序遍历　　B、后序遍历　　C、中序遍历　　D、层次遍历

在并查集问题中，已知集合元素 0~8 所以对应的父结点编号值分别是{ 1, -4, 1, 1, -3, 4, 4, 8, -2 }（注：-n 表示树根且对应集合大小为 n），那么将元素 6 和 8 所在的集合合并（要求必须将小集合并到大集合）后，该集合对应的树根和父结点编号值分别是多少？
A、8 和-5　　B、8 和-6　　C、4 和-5　　D、1 和-6

一棵二叉树的第 i（i>=1）层最多有 2^{i-1} 个结点；一棵有 n（n>0）个结点的满二叉树共有 （n+1）/2 个叶子结点和 （n-1）/2 个非叶子结点。

在具有 n 个结点的二叉链表里，共有 2n 个指针域，其中 n-1 个指针域用于指向其左右孩子，剩下的 n+1 个指针域则是空的。

在下列所示的平衡二叉树中，插入关键字 48 后得到一棵新平衡二叉树。在新平衡二叉树中，关键字 37 所在结点的左、右子结点中保存的关键字分别是：
24 和 53。



将 28, 23, 54, 61, 98, 37 插入一棵初始为空的平衡二叉树（AVL 树），然后马上插入下列选项(10,30,60,70)中的一个键值。问:插入键值 70 将引起 RL 旋转？

将 1, 2, 3, 6, 5, 4 顺序一个个插入一棵初始为空的 AVL 树，会经历 一个“右-右”旋和两个“右-左”旋 这些旋转。

下列代码的功能是将小顶堆 H 中指定位置 P 上的元素的整数键值下调 D 个单位，然后继续将 H 调整为小顶堆。

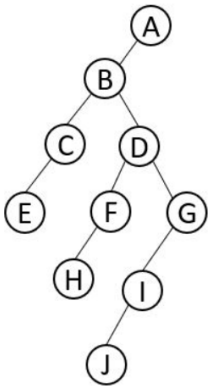
```
void DecreaseKey( int P, int D, MinHeap H )
{
   int i, key;
   key = H->Elements[P] - D;
   for ( i =   P    ; H->Elements[i/2] > key; i/=2 )
       H->Elements[i]=H->Elements[i/2];
   H->Elements[i] = key;
}
```

设二叉树的存储结构如下：

	1	2	3	4	5	6	7	8	9	10
Lchild	0	0	2	3	7	5	8	0	10	1
data	J	H	F	D	B	A	C	E	G	I
Rchild	0	0	0	9	4	0	0	0	0	0

其中根结点的针值为 6（实际为所在数组位置的下标），Lchild，Rchild 分别为结点的左、右孩子指针域，data 为数据域。

（1） 画出二叉树的逻辑结构 （2） 写出该树的前序、中序和后序遍历的序列。

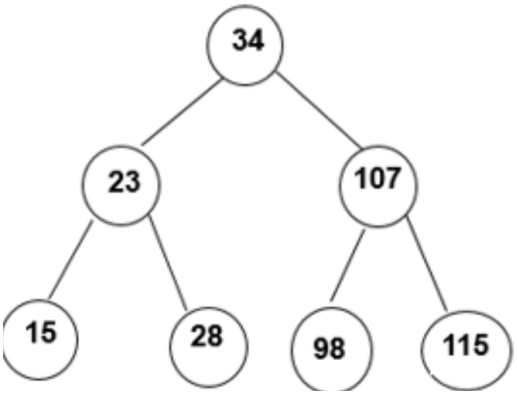


先序遍历序列: A B C E D F H G I J
中序遍历序列: E C B H F D J I G A
后序遍历序列: E C H F J I G D B A

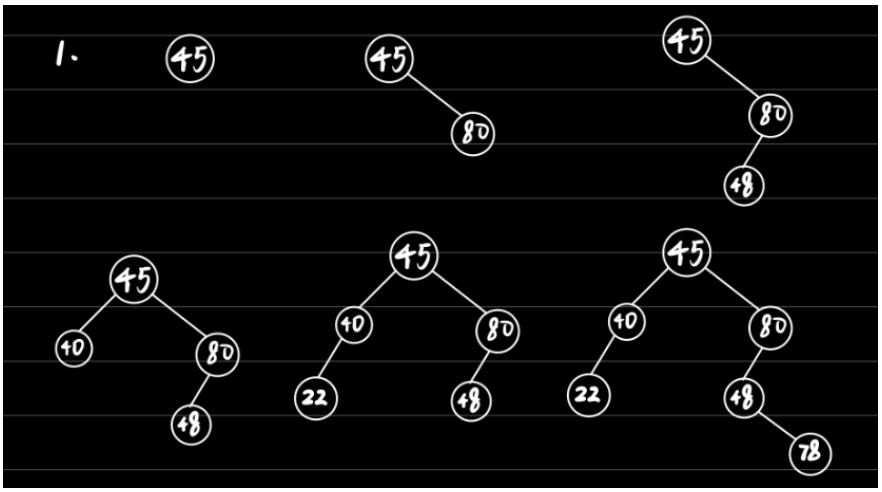
找出分别满足下面条件的所有二叉树：（1）先序序列和中序序列相同；（2）中序序列和后序序列相同；（3）先序序列和后序序列相同；（4）先序、中序和后序序列均相同；（5）先序和后序遍历序列正好相反。

（1）先序序列和中序序列相同:空二叉树或没有左子树的二叉树(右单支树);
（2）中序序列和后序序列相同:空二叉树或没有右子树的二叉树(左单支树);
（3）先序序列和后序序列相同:空二叉树或只有根结点的二叉树;
（4）先序、中序和后序序列均相同:空二叉树或只有根结点的二叉树;
（5）先序和后序遍历序列正好相反:高度等于其结点数的二叉树;

依次把结点（34，23，15，98，115，28，107）插入初始状态为空的平衡二叉排序树，使得在每次插入后保持该树依然是平衡二叉树，请依次画出每次插入后形成的平衡二叉树。



设有一组序列关键字为（45,80,48,40,22,78），构造一棵二叉排序树并给出构造过程。然后求等概率下，查找成功的平均查找长度。

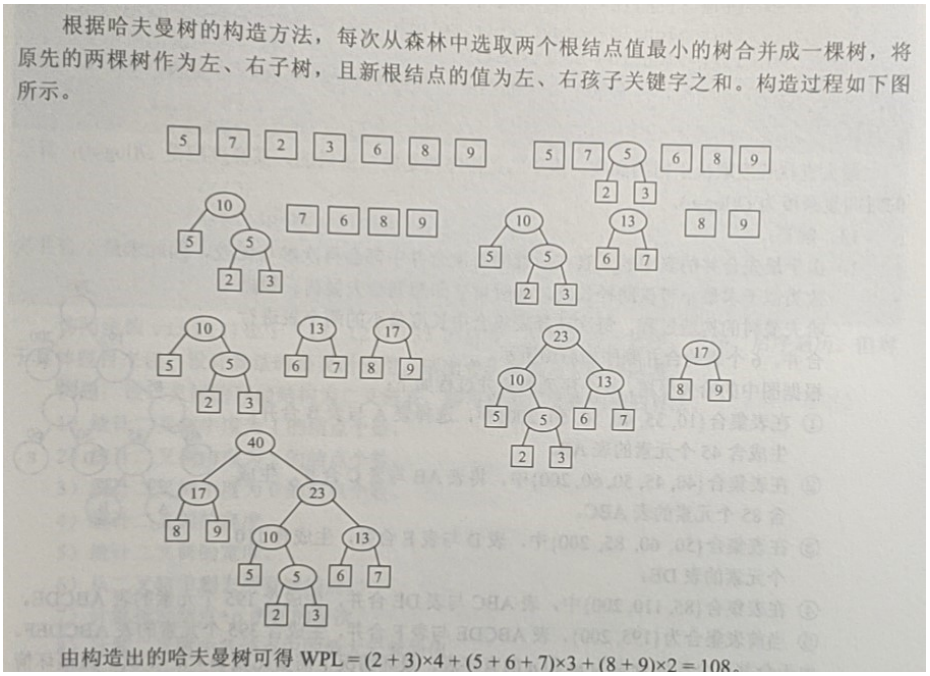


平均查找长度 ASL=（1+2*2+2*3+1*4）/6=2.5

假设二叉树次用二叉链表方式存储，编写一个程序，输出先序遍历中第 k 个结点的值（假设 k 不大于总的结点数）。
在先序遍历算法中添加计数变量。

```
void PreOrderTraversal( BinTree BT,int k )
{
if( BT ) {
    ++n;
    if(k==n)
            {printf( “%d”, BT->Data);
                 return ;}
        PreOrderTraversal( BT->Left );
        PreOrderTraversal( BT->Right );
    }
}
```

给定权值 w={5,7,2,3,6,8,9},试构造一棵哈夫曼树，并求其加权路径长度 WPL。



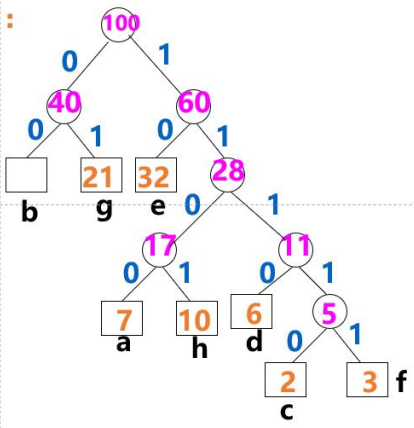
假设用于通信的电文由字符集{a，b，c，d，e，f，g，h}中的字母构成，这 8 个字母在电文中出现的概率分别为{0.07，0.19，0.02，0.06，0.32，0.03，0.21，0.10}

（1）试为这 8 个字母设计哈夫曼编码。

（2）如果用 3 位二进制数对这 8 个字母进行等长编码，则哈夫曼编码的平均码长是等长编码的百分之几？它使电文总长平均压缩多少？

对应的哈夫曼编码（左0右1）：

符	编码	频率
a	1100	0.07
b	00	0.19
c	11110	0.02
d	1110	0.06
e	10	0.32
f	11111	0.03
g	01	0.21
h	1101	0.10



第五章作业

静态查找与动态查找的根本区别在于（）。
A、它们的逻辑结构不一样　　B、施加在其上的操作不同
C、所包含的数据元素的类型不一样　　D、存储实现不一样

在下列查找的方法中，平均查找长度与结点个数无关的查找方法是：（　　）。
A、顺序查找　B、二分法　　C、利用哈希（散列）表　　D、利用二叉搜索树

若结点的存储地址与其关键字之间存在某种映射关系，则称这种存储结构为( )。
A、顺序存储结构　B、链式存储结构　C、索引存储结构　D、散列存储结构

在散列表中，所谓同义词就是：（　）。
A、两个意义相近的单词　　B、被不同散列函数映射到同一地址的两个元素
C、具有相同散列地址的两个元素　D、被映射到不同散列地址的一个元素

哈希表中的冲突指的是（ ）。
A、两个元素具有相同的序号
B、两个元素关键字值不同，而其他属性相同
C、数据元素过多
D、不同关键字值元素对应于相同的存储地址

在哈希函数 H（k）=k MOD m 中，一般来讲，m 应该取（ ）。
A、素数　　B、很大的数　　C、偶数　　D、奇数

一个哈希函数被认为是“好的”，如果它满足条件（）。
A、哈希地址分布均匀　　B、哈希地址分布均匀，并且保证不产生冲突。
C、保证不产生冲突　　D、所有哈希地址在表长范围内

设散列表的地址区间为[0,16]，散列函数为H(Key)=Key%17。采用线性探测法处理冲突，并将关键字序列{ 26，25，72，38，8，18，59 }依次存储到散列表中。元素 59 存放在散列表中的地址是：（　　）。
A、8　　B、9　　C、10　　D、11

将元素序列{18，23，11，20，2，7，27，33，42，15}按顺序插入一个初始为空的、大小为 11 的散列表中。散列函数为：H(Key)=Key%11，采用线性探测法处理冲突。问：当第一次发现有冲突时，散列表的装填因子大约是多少？
A、0.27　　B、0.45　　C、0.64　　D、0.73

给定散列表大小为 11，散列函数为 H(Key)=Key%11。按照线性探测冲突解决策略连续插入散列值相同的 4 个元素。问：此时该散列表的平均不成功查找次数是多少？
A、1　　B、4/11　　C、21/11　　D、不确定

下列哪个是 HASH 查找的冲突处理方法？

A、求余法　　B、平方取中法　　C、二分法　　D、开放定址法

下面关于哈希查找的说法，不正确的是（ ）。
A、采用链地址法处理冲突时，查找一个元素的时间是相同的
B、采用链地址法处理冲突时，若插入规定总是在链首， 则插入任一个元素的时间是相同的
C、用链地址法处理冲突，不会引起二次聚集现象
D、用链地址法处理冲突，适合表长不确定的情况

给定输入序列 {4371, 1323, 6173, 4199, 4344, 9679, 1989} 以及散列函数 H(X)=X%10。如果用大小为 10 的散列表，并且用分离链接法解决冲突，则输入各项经散列后在表中的下标为：（-1 表示相应的插入无法成功）
A、1, 3, 3, 9, 4, 9, 9　　B、1, 3, 4, 9, 7, 5, -1
C、1, 3, 4, 9, 5, 0, 8　　D、1, 3, 4, 9, 5, 0, 2

给定输入序列 {4371, 1323, 6173, 4199, 4344, 9679, 1989} 以及散列函数 h(X)=X%10。如果用大小为 10 的散列表，并且用线性探测解决冲突，则输入各项经散列后在表中的下标为：（-1 表示相应的插入无法成功）
A、1, 3, 3, 9, 4, 9, 9　　B、1, 3, 4, 9, 7, 5, -1
C、1, 3, 4, 9, 5, 0, 8　　D、1, 3, 4, 9, 5, 0, 2

给定输入序列 {4371, 1323, 6173, 4199, 4344, 9679, 1989} 以及散列函数 h(X)=X%10。如果用大小为 10 的散列表，并且用平方探测解决冲突，则输入各项经散列后在表中的下标为：（-1 表示相应的插入无法成功）
A、1, 3, 3, 9, 4, 9, 9　　B、1, 3, 4, 9, 7, 5, -1
C、1, 3, 4, 9, 5, 0, 8　　D、1, 3, 4, 9, 5, 0, 2

给定输入序列 {4371, 1323, 6173, 4199, 4344, 9679, 1989} 以及散列函数 H(X)=X%10。如果用大小为 10 的散列表，并且用开放定址法以及一个二次散列函数 h2（X）=7-（X%7）解决冲突，则输入各项经散列后在表中的下标为：（-1 表示相应的插入无法成功）
A、1, 3, 3, 9, 4, 9, 9　　B、1, 3, 4, 9, 7, 5, -1
C、1, 3, 4, 9, 5, 0, 8　　D、1, 3, 4, 9, 5, 0, 2

现有长度为 7、初始为空的散列表 HT，散列函数 H(k)=k%7，用线性探测法解决冲突。将关键字 22, 43, 15 依次插入到 HT 后，查找成功的平均查找长度是：
A、1.5　　B、1.6　　C、2　　D、3

现有长度为 11 且初始为空的散列表 HT，散列函数是 H(key)=key%7，采用线性探测法解决冲突。将关键字序列 87,40,30,6,11,22,98,20 依次插入到 HT 后，HT 查找失败的平均查找长度是：
A、4　　B、5.25　　C、6　　D、6.29

散列查找一般适用于（ ）情况下的查找。
A、查找表为链表　　B、关键字集合与地址结合之间存在对应关系

C、查找表为有序表　　D、关键字集合比地址集合大得多

下列关于散列表的说法中，正确的是（　）。
A、若在散列表中删除一个元素，不能简单地将该元素删除
B、若散列表的装填因子小于 1，则可以避免碰撞的产生
C、散列查找中不需要任何关键字的比较
D、散列表在查找成功时平均查找长度与表长有关

对于长度为 n 的线性表，如果进行顺序查找，则时间复杂度为（ O(n) ）;如果进行二分查找，则时间复杂度为（ O(logn) ）。

二叉搜索树中查询的最优时间复杂度是（ O(logn) ），即在满二叉树的情况下；最坏时间复杂度是（ O(n) ），即除叶子结点外每个结点只有一个子结点。

AVL 树是最先发明的自平衡二叉查找树。在 AVL 树中任何结点的两个子树的高度最大差别为一，所以它也被称为高度平衡树。查找、插入和删除在平均和最坏情况下都是（ O(log n) ）。

在哈希表中，查询、插入和删除一个元素，因为其操作类似所以时间复杂度应该是类似的。时间复杂度在平均情况下，查询、插入、删除都是（ O(1) ）；但在最差情况下，会退化成 O(n)。

在哈希表中，装填因子 α 值越大，存取元素发生冲突的可能性就（ 大 ）。当装填因子 α 值越小，存取元素发生冲突的可能性就（ 小 ）。

将 M 个元素存入用长度为 S 的数组表示的散列表，则该表的装填因子为（ M/S ）。

假定有 K 个关键字互为同义词，若用线性探测法把这 K 个关键字存入散列表中，至少要进行多少次探测？　　K(K+1)/2

在散列表中，“插入”和“查找”具有同样的时间复杂度。　　√

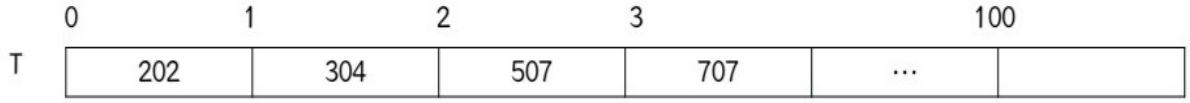
采用平方探测冲突解决策略（hi(k)=(H(k)+i2)%11，注意：不是±i2），将一批散列值均等于 2 的对象连续插入一个大小为 11 的散列表中，那么第 4 个对象一定位于下标为 0 的位置。
　　√

用平方探测法解决冲突，则插入新元素时，若散列表容量为质数，插入就一定可以成功。
　　×

即使把 2 个元素散列到有 100 个单元的表中，仍然有可能发生冲突。　　√

已知一组记录的关键字序列为{18，34，58，26，75，67，48，93，81}，TableSize=11，哈希函数为 H（k）=k MOD 11。如果采用线性探测法解决冲突，则平均成功查找长度为 16/9；如果采用链地址法解决冲突，则平均查找成功长度为 13/9。　√

设哈希函数为 H（k）= k MOD 101，解决冲突的方法采用线性探测再散列法，表中用"-1"表示空单元，哈希表 T 如下图所示。



（1）如果删除 T 中的 304（即令 T[1]= -1）之后，在 T 中查找 707 将会发生什么?
（2）如果将删除的表项标记为"-2"，查找时探测到 -2 继续向前搜索，探测到 -1 时终止搜索，则用这种方法删除 304 后能否正确地查找到 707?
（1）查找 707 时，首先根据哈希函数计算得出该元素应该在哈希表中的 0 单元，但是在 0 单元没有找到，因此将向下一单元探测，结果发现该单元是-1（为空单元），所以结束查找，这将导致 707 无法找到。
（2）如果改用“-2”作为删除标记，则可以正确找到 707 所在的结点。

假设把关键字 key 散列到有 n 个表项的散列表中，对于下面的每个函数 H（key），这些函数能当做散列函数吗？若能，它是一个好的散列函数吗？说明理由。
（1）H(key)=key/n
（2）H(key)=1
（3）H(key)=(key+random(n))%n
（4）H(key)=key%p(n),其中 p(n)是不大于 n 的最大素数
H(key)=key/n，不能，如果 key/n 结果大于 n，无法找到合适的存储位置
H(key)=1，能，不是好的散列函数，造成大量冲突
H(key)=(key+random(n))%n，不能，返回值不确定，无法进行正常的查找
H(key)=key%p(n),其中 p(n)是不大于 n 的最大素数，能，是一个好的散列函数。

将关键字序列{7,8,30,11,18,9,14}散列存储到散列表中，散列表为一个下标从 0 开始的一维数组，散列函数为 H(key)=（key*3）MOD 7，处理冲突采用线性探测法，要求装填因子为 0.7。
（1）画出所构造的散列表；
（2）在等概率的情况下，查找成功和查找不成功的平均查找长度。
根据装填因子 0.7，数据总数 7，得到一维数组的长度为 7/0.7=10,数组下标为 0-9。所构造的散列函数数值如下所示：

下标	0	1	2	3	4	5	6	7	8	9
关键字	7	14		8		11	30	18	9	

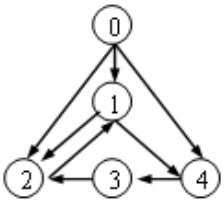
(2)查找成功的平均查找长度：ASL 成功=(1+1+1+1+2+3+3)/7=12/7。
查找不成功的平均查找长度：AsL 不成功=(3+2+1+2+1+5+4)/7=18/7。

第六章作业

若无向图 G =（V，E）中含 7 个顶点，要保证图 G 在任何情况下都是连通的，则需要的边数最少是：

A、16　　B、21　　C、15　　D、6

下面给出的有向图中，有_____个强连通分量。



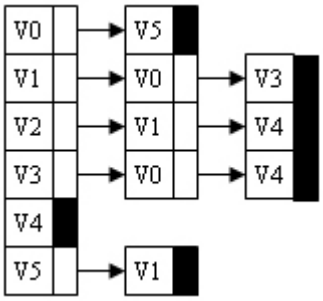
A、5 个({0}，{1}，{2}，{3}，{4})

B、2 个({1,2,3,4}，{0})

C、1 个 ({1,2,3,4})

D、1 个 ({0,1,2,3,4})

给定一个有向图的邻接表如下图，则该图有_____个强连通分量。



A、1 个 {0，5，1，3}

B、3 个 {{2}，{4}，{0，1，3，5}}

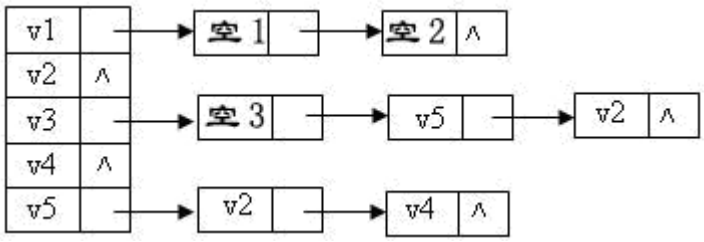
C、1 个 {0，1，2，3，4，5

D、4 个 {{0，1，5}，{2}，{3}，{4}}

如果 G 是一个有 28 条边的非连通无向图，那么该图顶点个数最少为多少？

A、9　　B、8　　C、10　　D、7

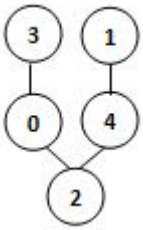
给定一有向图的邻接表如下。若从 v1 开始利用此邻接表做广度优先搜索得到的顶点序列为：{v1，v3，v2，v4，v5...}，则该邻接表中顺序填空的结果应为：

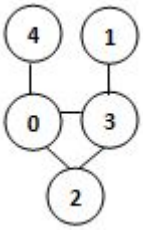


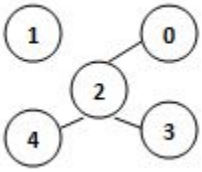
A、v3，v2，v4　　B、v4，v3，v2　　C、v3，v4，v2　　D、v2，v3，v4

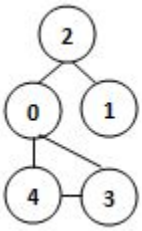
若某图的深度优先搜索序列是{V2，V0，V4，V3，V1}，则下列哪个图不可能对应该序列？

A、　　B、　　C、　　D、

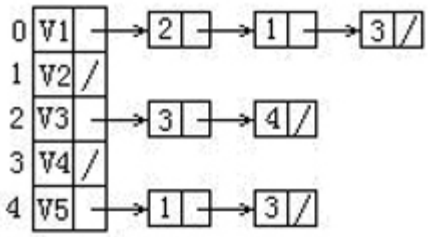








给定一有向图的邻接表如下。从顶点 V1 出发按深度优先搜索法进行遍历，则得到的一种顶点序列为：



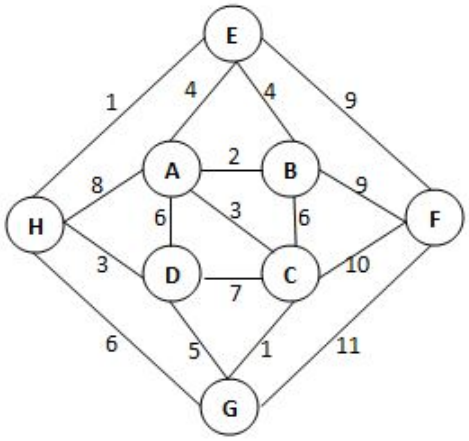
A、V1,V2,V4,V5,V3
B、V1,V4,V3,V5,V2
C、V1,V3,V4,V5,V2
D、V1,V2,V3,V5,V4

给定有权无向图的邻接矩阵如下，其最小生成树的总权重是：

$$\begin{bmatrix} 0 & 4 & 10 & 3 & 2 \\ 4 & 0 & 9 & 5 & 6 \\ 10 & 9 & 0 & 8 & 7 \\ 3 & 5 & 8 & 0 & 1 \\ 2 & 6 & 7 & 1 & 0 \end{bmatrix}$$

A、10　　B、11　　C、1　　D、14

给定有权无向图如下。关于其最小生成树，下列哪句是对的？



A、边(B, F)一定在树中，树的总权重为 23
B、边(H, G)一定在树中，树的总权重为 20
C、最小生成树唯一，其总权重为 20

D、最小生成树不唯一，其总权重为 23

下列关于最小生成树的叙述中，正确的是：

Ⅰ．最小生成树的代价唯一

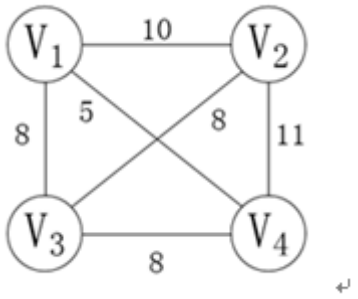
Ⅱ．所有权值最小的边一定会出现在所有的最小生成树中

Ⅲ．使用普里姆（Prim）算法从不同顶点开始得到的最小生成树一定相同

Ⅳ．使用普里姆算法和克鲁斯卡尔（Kruskal）算法得到的最小生成树总不相同

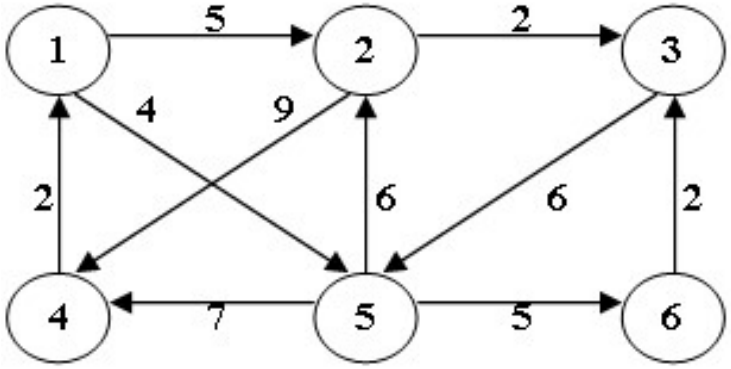
A、仅Ⅰ　　B、仅Ⅱ　　C、仅Ⅰ、Ⅲ　　D、仅Ⅱ、Ⅳ

求下面带权图的最小（代价）生成树时，可能是克鲁斯卡（Kruskal）算法第 2 次选中但不是普里姆（Prim）算法（从 V4 开始）第 2 次选中的边是：



A、(V1,V3)　　B、(V1,V4)　　C、(V2,V3)　　D、(V3,V4)

使用迪杰斯特拉（Dijkstra）算法求下图中从顶点 1 到其他各顶点的最短路径，依次得到的各最短路径的目标顶点是：



A、5, 2, 6, 3, 4 B、5, 2, 3, 6, 4 C、5, 2, 4, 3, 6 D、5, 2, 3, 4, 6

在一个有权无向图中，如果顶点 b 到顶点 a 的最短路径长度是 10，顶点 c 与顶点 b 之间存在一条长度为 3 的边。那么下列说法中有几句是正确的？

（1）c 与 a 的最短路径长度就是 13

（2）c 与 a 的最短路径长度就是 7

（3）c 与 a 的最短路径长度不超过 13

（4）c 与 a 的最短路径不小于 7

A、4 句　　B、3 句　　C、2 句　　D、1 句

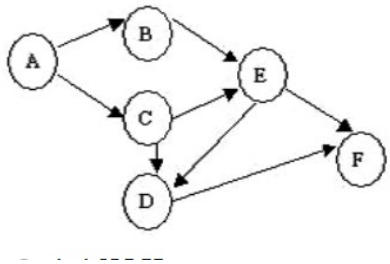
在 AOE 网中，什么是关键路径？

A、最短回路　　B、从第一个事件到最后一个事件的最短路径

C、最长回路　　D、从第一个事件到最后一个事件的最长路径

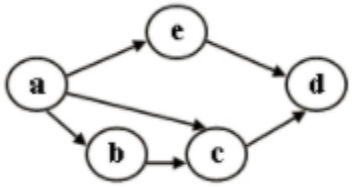
在拓扑排序算法中用堆栈和用队列产生的结果会不同吗？
A、是的肯定不同　B、肯定是相同的　C、有可能会不同　D、其它选项全不对

下图为一个 AOV 图，其可能的拓扑有序序列为：



A、ACBDEF　B、ABCEFD　C、ABCDFE　D、ABCEDF

对下图进行拓扑排序，可以得到不同的拓扑序列的个数是：



A、4　B、3　C、2　D、1

在 N 个顶点的无向图中，所有顶点的度之和不会超过顶点数的多少倍？

A、N-1　B、(N-1)/2　C、2　D、1

数据结构中 Dijkstra 算法用来解决哪个问题？
A、关键路径　B、最短路径　C、拓扑排序　D、最小生成树

无向连通图边数一定大于顶点个数减 1。　×

图的深度优先遍历非递归算法通常采用队列实现，广度优先遍历非递归算法通常采用堆栈实现。
×

用邻接矩阵存储图，占用的存储空间只与图中结点个数有关，而与边数无关；用邻接表存储图，占用的存储空间与图中结点个数和边数都有关。　√

无向图的邻接矩阵一定是对称的；有向图的邻接矩阵可以是对称的，也可以是不对称的。
√

最小生成树是指边数最少的生成树。　×

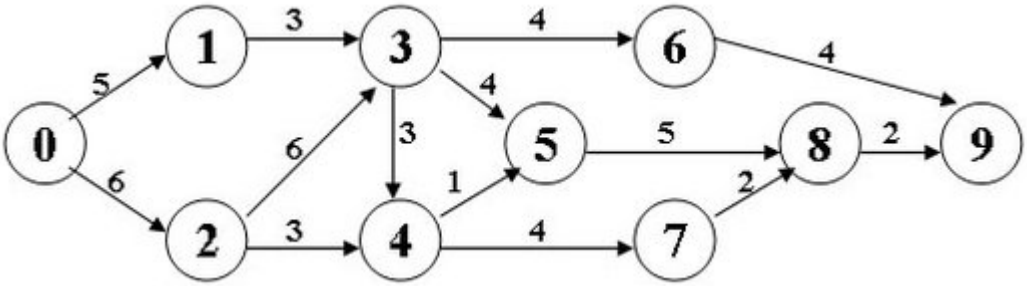
Kruskal 算法是通过每步添加一条边及其相连的顶点到一棵树，从而逐步生成最小生成树。
×

一个带权无向连通图的最小生成树有可能不唯一。　√

对于带权无向图 G = (V, E)，M 是 G 的最小生成树，则 M 中任意两点 V1 到 V2 的路径一定是它们之间的最短路径。 ×

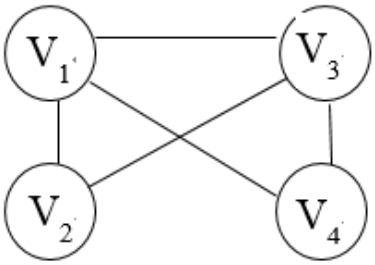
如果从有向图 G 的每一点均能通过深度优先搜索遍历到所有其它顶点，那么该图一定不存在拓扑序列。 √

下图给定了一个项目的 AOE，整个项目最早完工需要的时间是 23。如果<0,2>组能加快进度，整个项目就能提前完工。 √



教材 P259-6.3

(1)



(2)

V1 度为：3，V2 度为：2，V3 度为：3，V4 度为：2

(3)

	V1	V2	V3	V4
V1	0	1	1	1
V2	1	0	1	0
V3	1	1	0	1
V4	1	0	1	0

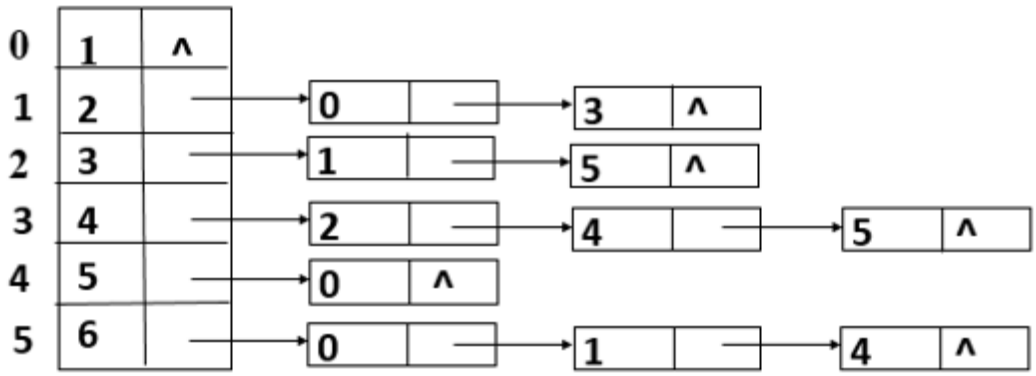
教材 P259-6.4

(1) 顶点 1 入\出度：3/0
顶点 2 入\出度：2/2
顶点 3 入\出度：1/2
顶点 4 入\出度：1/3
顶点 5 入\出度：2/1
顶点 6 入\出度：2/3

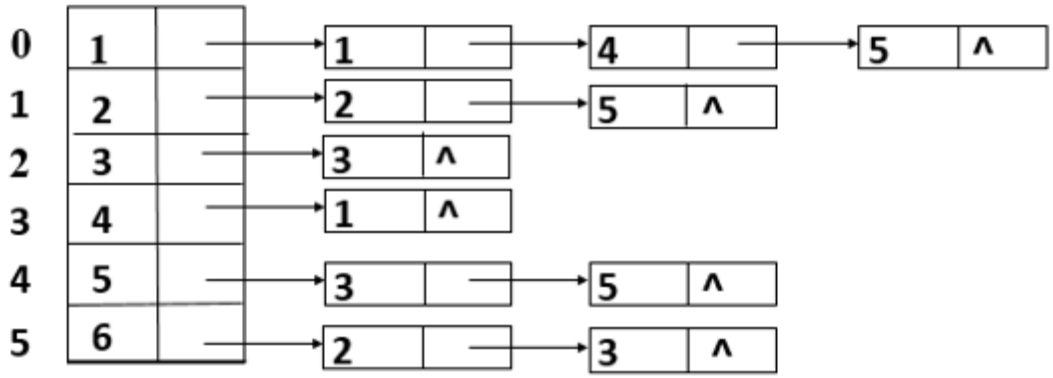
(2)

	1	2	3	4	5	6
1	0	0	0	0	0	0
2	1	0	0	1	0	0
3	0	1	0	0	0	1
4	0	0	1	0	1	1
5	1	0	0	0	0	0
6	1	1	0	0	1	0

(3)



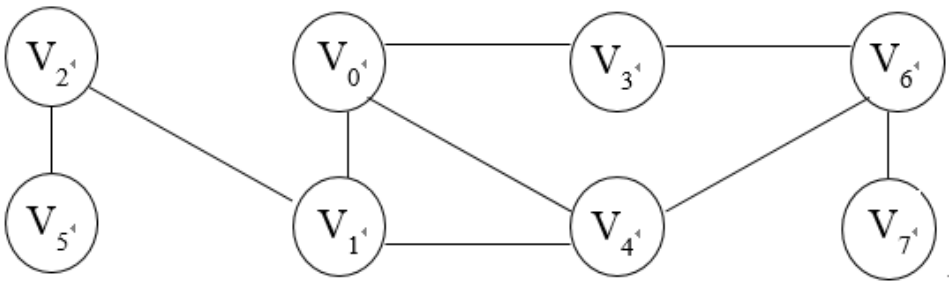
(4)



(5)三个强连通分量：{1}，{5}，{2，3，4，6}

教材 P259-6.5

（1）



（2）

V0 出发深度优先遍历序：V0 V1 V2 V5 V4 V6 V3 V7

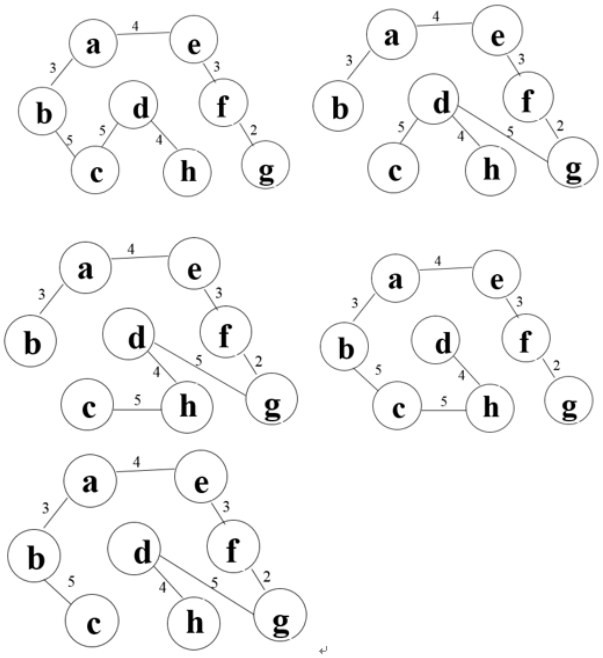
V0 出发广度优先遍历序：V0 V1 V3 V4 V2 V6 V5 V7

教材 P260-6.7

(1)

	a	b	c	d	e	f	g	h
a	∞	3	∞	∞	4	∞	∞	∞
b	3	∞	5	8	9	∞	∞	∞
c	∞	5	∞	5	∞	∞	∞	5
d	∞	8	5	∞	7	6	5	4
e	4	9	∞	7	∞	3	∞	∞
f	∞	∞	∞	6	3	∞	2	∞
g	∞	∞	∞	5	∞	2	∞	6
h	∞	∞	5	4	∞	∞	6	∞

(2)五棵最小生成树，画出其一即可。



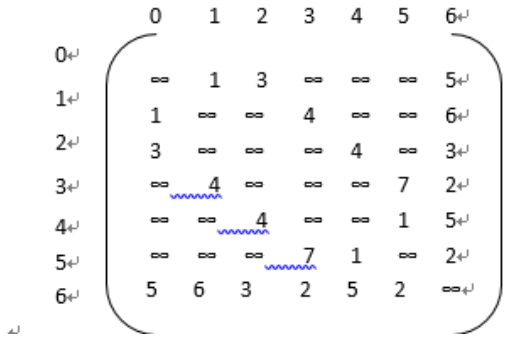
（3）abcdefgh

教材 P260-6.9

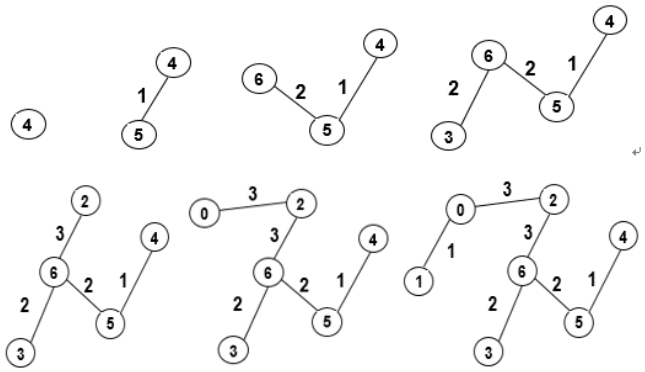
过程 / 终点	=> 从 A 到各终点的 D 值和最短路径父顶点 P 的变化过程 =>													
	初始 (第 0 步)		选 C (第 1 步)		选 F (第 2 步)		选 E (第 3 步)		选 D (第 4 步)		选 B (第 5 步)		选 G (第 6 步)	
	D	P	D	P	D	P	D	P	D	P	D	P	D	P
A	0	-1	0	-1	0	-1	0	-1	0	-1	0	-1	0	-1
B	15	0	15	0	15	0	15	0	15	0	15	0	15	0
C	2	0	2	0	2	0	2	0	2	0	2	0	2	0
D	12	0	12	0	12	0	12	0	12	0	12	0	12	0
E	∞	-1	10	2	10	2	10	2	10	2	10	2	10	2
F	∞	-1	6	2	6	2	6	2	6	2	6	2	6	2
G	∞	-1	∞	-1	16	5	16	5	15	3	15	3	15	3

教材 P261-6.10

(1)



		0	1	2	3	4	5	6
初始	dist	∞	∞	4	∞	0	1	5
	Parent	4	4	4	4	-1	4	4
选5	dist	∞	∞	4	7	0	0	2
	Parent	4	4	4	5	-1	4	5
选6	dist	5	6	3	2	0	0	0
	Parent	6	6	6	6	-1	4	5
选3	dist	5	4	3	0	0	0	0
	Parent	6	3	6	6	-1	4	5
选2	dist	3	4	0	0	0	0	0
	Parent	2	3	6	6	-1	4	5
选0	dist	0	1	0	0	0	0	0
	Parent	2	0	6	6	-1	4	5
选1	dist	0	0	0	0	0	0	0
	Parent	2	0	6	6	-1	4	5



教材 P261-6.11

最短路径长度矩阵序列 D

$D^{(-1)}$
$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 3 & 4 & 2 \\ 8 & 0 & \infty & \infty \\ 6 & 2 & 0 & \infty \\ \infty & \infty & 3 & 0 \end{bmatrix} \end{array}$$

$D^{(0)}$
$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 3 & 4 & 2 \\ 8 & 0 & 12 & 10 \\ 6 & 2 & 0 & 8 \\ \infty & \infty & 3 & 0 \end{bmatrix} \end{array}$$

$D^{(1)}$
$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 3 & 4 & 2 \\ 8 & 0 & 12 & 10 \\ 6 & 2 & 0 & 8 \\ \infty & \infty & 3 & 0 \end{bmatrix} \end{array}$$

$D^{(2)}$
$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 3 & 4 & 2 \\ 8 & 0 & 12 & 10 \\ 6 & 2 & 0 & 8 \\ 9 & 5 & 3 & 0 \end{bmatrix} \end{array}$$

$D^{(3)}$
$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 3 & 4 & 2 \\ 8 & 0 & 12 & 10 \\ 6 & 2 & 0 & 8 \\ 9 & 5 & 3 & 0 \end{bmatrix} \end{array}$$

最短路径矩阵序列 P

$P^{(-1)}$
$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \end{bmatrix} \end{array}$$

$P^{(0)}$
$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 \\ -1 & -1 & 0 & 0 \\ -1 & -1 & -1 & 0 \\ -1 & -1 & -1 & -1 \end{bmatrix} \end{array}$$

$P^{(1)}$
$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 \\ -1 & -1 & 0 & 0 \\ -1 & -1 & -1 & 0 \\ -1 & -1 & -1 & -1 \end{bmatrix} \end{array}$$

$P^{(2)}$
$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 \\ -1 & -1 & 0 & 0 \\ -1 & -1 & -1 & 0 \\ 2 & 2 & -1 & -1 \end{bmatrix} \end{array}$$

$P^{(3)}$
$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 \\ -1 & -1 & 0 & 0 \\ -1 & -1 & -1 & 0 \\ 2 & 2 & -1 & -1 \end{bmatrix} \end{array}$$

教材 P261-6.12

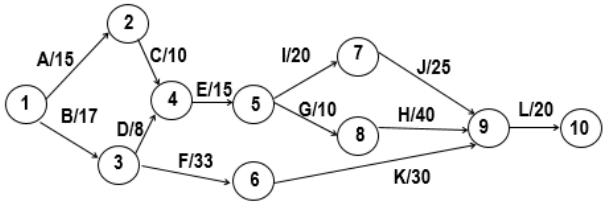
不同的拓扑排序序列有四种：

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

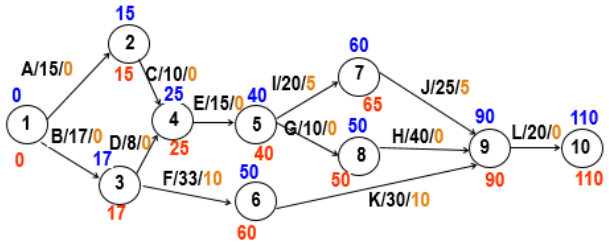
按字母序从小到大的拓扑序列：a b e d f c

教材 P261-6.13

（1）AOE↵



（2）↵



（3）↵

完成该项目所需时间：110。↵

关键活动：A， B， C， D， E， G， H， L ↵

第七章作业

下列排序算法中，哪种算法可能出现：在最后一趟开始之前，所有的元素都不在其最终的位置上？（设待排元素个数 N>2）

A、冒泡排序　　B、插入排序　　C、堆排序　　D、快速排序

对初始数据序列{ 8, 3, 9, 11, 2, 1, 4, 7, 5, 10, 6 }进行希尔排序。若第一趟排序结果为（ 1, 3, 7, 5, 2, 6, 4, 9, 11, 10, 8 ），第二趟排序结果为（ 1, 2, 6, 4, 3, 7, 5, 8, 11, 10, 9 ），则两趟排序采用的增量（间隔）依次是：

A、3, 1　　B、3, 2　　C、5, 2　　D、5, 3

若数据元素序列{ 11，12，13，7，8，9，23，4，5 }是采用下列排序方法之一得到的第二趟排序后的结果，则该排序算法只能是：

A、冒泡排序　　B、选择排序　　C、插入排序　　D、快速排序

对 N 个记录进行堆排序，需要的额外空间为：

A、O(1)　　B、O(logN)　　C、O(N)　　D、O(NlogN)

对一组数据{ 2，12，16，88，5，10 }进行排序，若前三趟排序结果如下： 第一趟排序结果：2，12，16，5，10，88 第二趟排序结果：2，12，5，10，16，88 第三趟排序结果：2，5，10，12，16，88 则采用的排序方法可能是：

A、冒泡排序　　B、希尔排序　　C、归并排序　　D、堆排序

对 N 个记录进行快速排序，在最坏的情况下，其时间复杂度是：

A、O(N)　　B、O(NlogN)　　C、O(N2)　　D、O(N2logN)

在快速排序的一趟划分过程中，当遇到与基准数相等的元素时，如果左右指针都会停止移动，那么当所有元素都相等时，算法的时间复杂度是多少？

A、O(logN)　　B、O(N)　　C、O(NlogN)　　D、O(N2)

在快速排序的一趟划分过程中，当遇到与基准数相等的元素时，如果左右指针都不停止移动，那么当所有元素都相等时，算法的时间复杂度是多少？

A、O(logN)　　B、O(N)　　C、O(NlogN)　　D、O(N2)

有一组数据{84，25，21，47，15，27，68，20，35}，用快速排序的划分方法进行一趟划分后数据的排序为（ ）。

A、{15，25，21，27，20，35，68，47，84}

B、{35，25，21，47，15，27，68，20，84}

C、{15，20，27，25，21，35，47，68，　84}

D、{15，20，21，25，27，35，47，68，　84}

数据序列{ 3, 1, 4, 11, 9, 16, 7, 28 }只能是下列哪种排序算法的两趟排序结果？

A、冒泡排序　　B、快速排序　　C、插入排序　　D、堆排序

对大部分元素已有序的数组进行排序时，直接插入排序比简单选择排序效率更高，其原因是：

(I). 直接插入排序过程中元素之间的比较次数更少

(II). 直接插入排序过程中所需要的辅助空间更少

(III). 直接插入排序过程中元素的移动次数更少
A、仅 I　　B、仅 III　　C、仅 I、II　　D、I、II 和 III

下列排序算法中()不能保证每趟排序至少能将一个元素放到其最终的位置上。
A、快速排序　B、shell 排序　C、堆排序　D、冒泡排序

对序列{15,9,7,8,20,-1,4}进行排序,进行一趟后数据的排列变为{4,9,-1,8,20,7,15};则采用的是()排序。
A、选择　B、快速　C、希尔　D、冒泡

下列排序方法中,哪一个是稳定的排序方法?()
A、直接选择排序　B、二分法插入排序　C、希尔排序　D、快速排序

排序算法的稳定性是指()。
A、经过排序之后,能使值相同的数据保持原顺序中的相对位置不变。
B、经过排序之后,能使值相同的数据保持原顺序中的绝对位置不变。
C、排序算法的性能与被排序元素的数量关系不大
D、排序算法的性能与被排序元素的数量关系密切

16 若用冒泡排序方法对序列{10,14,26,29,41,52}从大到小排序,需要进行()次比较。
A、3　　B、10　　C、15　　D、20

下列代码的功能是利用堆排序将 N 个元素按非递减顺序排序。

```
void PercDown( ElementType A[], int p, int N )
{   int Parent, Child;
    ElementType X;
    X = A[p]; /* 取出根结点存放的值 */
    for( Parent=p; (Parent*2+1)<N; Parent=Child ) {
Child = Parent * 2 + 1;
        if( (Child!=N-1) && (__A[Child]<A[Child+1]_) )
            Child++;  /* Child 指向左右子结点的较大者 */
        if( __X < A[Child]_ ) A[Parent] = A[Child]; /* 找到了合适位置 */
else  break;
    }
    _ A[Parent] = X;
}
void HeapSort( ElementType A[], int N )
{   int i;
    for ( i=N/2-1; i>=0; i-- )/* 建立最大堆 */
         PercDown( A, i, N );
    for ( i=N-1; i>0; i-- ) { /* 删除最大堆顶 */
        Swap( &A[0], &A[i] );
       __ PercDown( A, 0, i )_;
    }
}
```

对一组记录关键字序列{54，38，96，23，15，72，60，45，83}进行直接插入排序，当第七个记录的

关键字 60 插入到有序序列时，为寻找插入位置需要比较____3_____次。

在排序方法中，从待排序记录序列中依次取出记录，并将其依次放入有序序列（初始为空）一端的方法，称为__简单选择排序____。

在待排序的记录序列基本有序的前提下，效率最高的排序方法是直接插入排序。

设有 10000 个无序记录，希望用最快速度挑选出其中前 10 个关键字最大的记录，最好选用___堆排序_____排序方法。

在堆排序、快速排序和归并排序中，如果只从存储空间考虑，则应该选取__堆排序___方法；如果只从平均情况下排序最快考虑，则应该选取__快速排序_方法；如果只从在最坏情况下要求排序最快来考虑，则应该选取____堆_____排序。

下列代码的功能是将一系列元素{ r[0]，r[1] … r[n-1] }按非递减顺序排序。简单选择排序是每次仅将一个待排序列的最小元放到正确的位置上，而这个另类的选择排序是每次从待排序列中同时找到最小元和最大元，把它们放到最终的正确位置上。请将代码中几个空白处完善。

```
void  sort( int r[], int n )
{
   int i, j, mini, maxi;
    for (i=0; i<n-i-1; i++) {
      mini = maxi = i;
      for( j=i+1; __j<=n-i-1_ ; ++j ){
         if(  __r[j]<r[mini]___ ) mini = j;
         else if(r[j] > r[maxi]) maxi = j;
      }
      if( ___mini!=i___) swap(&r[mini], &r[i]);
      if( maxi != n-i-1 ){
         if( __maxi==i__) swap(&r[mini], &r[n-i-1]);
         else swap(&r[maxi], &r[n-i-1]);
      }
   }
}
```

希尔排序是稳定的算法。 ×

仅基于比较的算法能得到的最好的”最坏时间复杂度”是 O(NlogN)。 √

排序算法中的比较次数与初始元素序列的排列无关。 ×

在待排序的记录集中，存在多个具有相同键值的记录，若经过排序，这些记录的相对次序仍然保持不变，称这种排序为稳定排序。 √

对 N 个记录进行堆排序，需要的额外空间为 O(N)。 ×

在堆排序中，若要进行非递减排序，则需要建立大根堆。 √

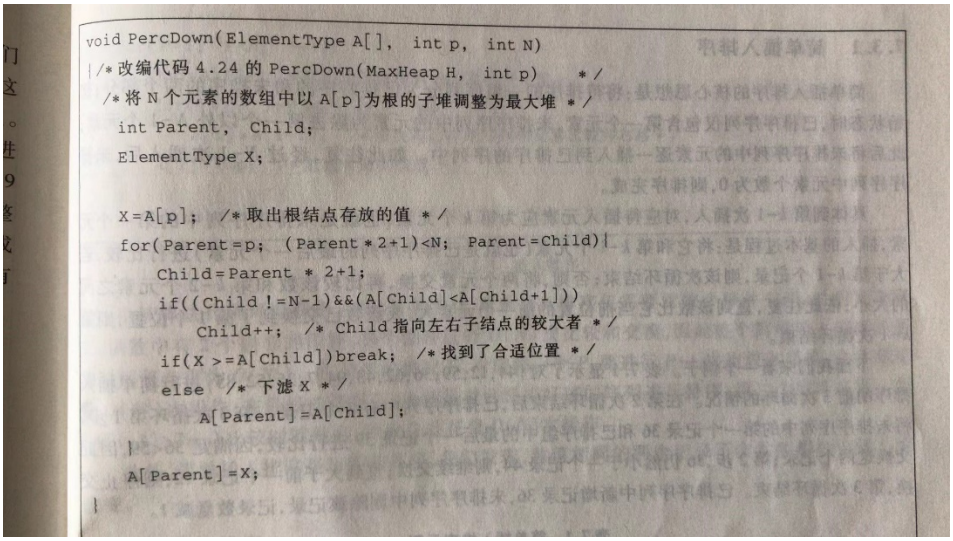
对 N 个记录进行快速排序，在最坏的情况下，其时间复杂度是 O(NlogN) ×

要从 50 个键值中找出最大的 3 个值，选择排序比堆排序快。 √

直接选择排序算法在最好情况下的时间复杂度为 O(n)。 ×

评价基于比较的排序算法的时间性能，主要标准是关键码的比较次数和记录的移动次数。
√

试设计一个算法：已知{x0，x1，x2，……，xn-1}是一个堆，要求删除其中一个记录结点 x[i]后，还是堆。提示：先将 x[i]保存，将堆中最后一个元素换至 x[i]，并将堆长度减 1；再从位置 i 开始向下调整，使其满足堆性质。
其中调整部分的处理参考教材 P267



```
void PercDown( ElementType A[], int p, int N)
/* 改编代码 4.24 的 PercDown( MaxHeap H, int p)    */
/* 将 N 个元素的数组中以 A[p]为根的子堆调整为最大堆 */
  int Parent, Child;
  ElementType X;

  X=A[p]; /* 取出根结点存放的值 */
  for(Parent=p; (Parent*2+1)<N; Parent=Child){
     Child=Parent * 2+1;
     if((Child!=N-1)&&(A[Child]<A[Child+1]))
        Child++; /* Child 指向左右子结点的较大者 */
     if(X>=A[Child])break; /* 找到了合适位置 */
     else /* 下滤 X */
        A[Parent]=A[Child];
  }
  A[Parent]=X;
}
```