参考答案

填空题（1×25＝25 分）

1．算法的 5 个重要特性是（有穷性），（确定性），（可行性），（输入）和（输出）。

2．在树形结构中，树根结点没有（前驱）结点，其余每个结点有且只有（一）个前驱结点，叶子结点没有（后续）结点。

3．在图形结构中，每个结点的前驱节点数和后续结点数可以有（任意多个），6 个顶点的无向图至少要（5）条边才是连通图。

4．从一个具有 n 个结点的单链表中查找其值等于 x 的结点时，在查找成功的情况下，平均需要比较（ (n+1)/2 ）个结点。

5．若 n×n 的下三角矩阵 A 已经压缩存储到一维数组 S[1..n*(n+1)/2]中，则 A[i][j]对应的 S 中的存储位置是（i(i+1)/2+j+1）。

6．已知某二叉树的后序遍历序列是 dabec，中序遍历序列是 debac，那么它的前序遍历序列是（cedba）。

7．对于一个 n 个结点的满二叉树，假设该树有 m 个树叶，深度为 h，则(n= 2^h-1).

8. 具有 5 层结点的二叉平衡树至少有（15）个结点，至多有（31）个结点；广义表（a,b,c,d）的表头是（a），表尾是（(b,c,d)）。

9．设计 Hash 函数时要求其函数值应按（ 平均概率 ）取其值域的每一个值。

10．用冒泡排序法对 n 个记录进行排序，第一趟要比较（ n-1 ）个元素，第二趟要比较（ n-2 ）个元素。

11．对给定的一个有 n 个元素的数组，建立一个有序单链表的算法的时间复杂度是（O(n^2)）。

12．有一个长度为 12 的有序表，按二分查找法对该表进行查找，在表内各元素等概率情况下查找成功所需的平均比较次数为（ 37/12）。

13．设高度为 h 的二叉树上只有度为 0 和度为 2 的结点，则此类二叉树中所包含的结点数至少为（2h-1），至多为（2^h-1）。

14.在所有排序方法中，关键字比较的次数与记录的初始排序次序无关的排序方法是(选择排序)。直接存取文件是按（哈希）方法组织的。

简答

1、在单链表、双链表和单循环链表中，若仅知道指针 p 指向某内部结点，不知道头指针，能否将结点*p 从相应的链表中删除？若可以，其时间复杂度各为多少？

答：单链表：可以删除。将*p 结点的数据和其后继结点的数据交换位置，再删除其后继结点，时间复杂度为 O（1）；　双链表：可以删除，其时间复杂度为 O（1）；单循环：可以删除。其时间复杂度为 O（n），若采用上述单链表的删除方法，时间复杂度也为 O（1）。

2、已知一棵度为 m 的树中有 n_1 个度为 1 的结点，n_2 个度为 2 的结点，…，n_m 个度为 m 的结点，问该树中有多少片叶子？

答：（1）总结点数 N =n_0 +n_1 +n_2 +…+n_{m-1} +n_m　①

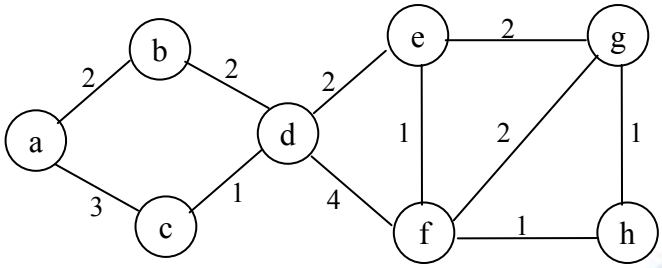
（2）因为 1 度结点发出一个分支，2 度结点发出两个分支，…，m 度结点发出 m 个分支，叶结点不发出分支，所以总分支数 B 为：B= n_1 +2n_2 +…+m*n_m　②

（3）又因为除根结点之外，其他结点有且仅有一个进入支，因此，总分支数应等于总结点数减 1，即 B＝N-1　③

（4）由②、③两式可得：N＝n_1 +2n_2 +…+m*n_m＋1　④

（5）由①、④两式可得叶结点数：n_0＝n_2 +2n_3 +3n_4 +…+(m-1)n_m

3、对下图所示的连通图，请分别用 Prim 和 Kruskal 算法构造其最小生成树（按先后顺序写出所选边来表达树的生成过程）。并说明 Prim 和 Kruskal 算法构造最小生成树的时间复杂度各为多少？它们分别适合于哪类图？



Prim 算法构造其最小生成树的过程：
（a,b）、（b,d）、（d,c）、（d,e）、（e,f）、（f,h）、（h,g）
Kruskal 算法构造最小生成树的过程：
（c,d）、（e,f）、（f,h）、（h,g）、（a,b）、（b,d）、（d,e）
时间复杂度分别为：　Prim 算法：$O(n^2)$；Kruskal 算法：O(e*loge)。
Prim 适用于稠密图（点少边多）；Kruskal 适用于稀疏图（点多边少）。

4、对于一组给定的固定不变的关键字序列，有可能设计出无冲突的散列函数 H，此时称 H 为完备的散列函数。若 H 能无冲突地将关键字完全填满散列表，则称 H 是最小完备的散列函数。请问：
（1）已知关键字集合为（81，129，301，38，434，216，412，487，234），散列函数为 H（X）＝（X＋18）/ 63，请问 H 是完备的吗？它是最小完备的吗？
（2）考虑由字符串构成的关键字集合（Bret, Jane, Shirley, Bryce, Michelle, Heather），试为散列表 T［0…6］设计一个完备的散列函数。
答：（提示：考虑每个字符串的第 3 个字符，即 S［2］）
（1）上述散列地址从 0 到 8，连续且没有重复，正好填满散列表 T［9］，所以 H 是完备的，且是最小完备的。
（2）各字符串的第 3 个字母（S［2］）依次为 e、n、I、y、c、a，它们在字母表中的顺序号依次为 5、14、9、25、3、1。因为散列表为 T［0…6］，用除留余数法构造散列函数，其模数可取作 7。令 S［2］在 26 个字母表中的顺序号为 k，则散列函数可取为 H＝kMOD7，于是可得各关键字字符串的散列地址不会发生冲突，所以 H＝kMOD7 是一个完备的散列函数。

5、简述如何根据不同的实际情况和要求选择不同的排序算法。
答：排序算法主要有简单排序、快速排序、堆排序、归并排序、基数排序。
（1）从平均时间性能而言，快速排序最佳，其所需时间最省，但快速排序在最坏情况下的时间性能不如堆排序和归并排序，在 n 较大时，归并排序所需时间较堆排序省，但它所需的辅助存储量最多。
（2）“简单排序”包括除希尔排序之外的所有插入排序，起泡排序和简单排序，其中直接插入排序最为简单，当序列中的记录“基本有序”或 n 值较小时，它是最佳排序方法，因此常将它和其他的排序方法，诸如快速排序、归并排序等结合在一起使用。
（3）基数排序的时间复杂度也可写成 O（d*n）。因此，它最适用于 n 值很大而关键字较小的序列。若关键字也很大，而序列中大多数记录的“最高位关键字”均不同，则亦可先按“最

高位关键字”不同将序列分成若干“小”的子序列，而后进行直接插入排序。

（4）从方法的稳定性来比较，基数排序是稳定的内排方法，所有时间复杂度为 O（n^2）的简单排序法也是稳定的，然而，快速排序、堆排序和希尔排序等时间性能较好的排序方法都是不稳定的。

综合

1、 A 是一个线性表（a_1 ，a_2 ，…，a_n），若采用顺序存储结构，则在等概率的前提下，平均每插入一个元素需要移动的元素个数为多少？若元素插在 a_i 与 a_{i+1} 之间（$0 \leqslant i \leqslant n-1$）的概率为 $2(n-i)/n(n+1)$，则平均每插入一个元素所要移动的元素个数又是多少？

答：在等概率的前提下，平均每插入一个元素需要移动的元素个数为$(0+1+2+\cdots+n)/(n+1) = n/2$。若元素插在 a_i 与 a_{i+1} 之间（$0 \leqslant i \leqslant n-1$）的概率为 $2(n-i)/n(n+1)$，则平均每插入一个元素所要移动的元素个数是

$\sum_{i=0}^{n-1} 2(n-i)^2/n(n+1) = 2(n+1)/3$。

2、设有上三角矩阵（a_{ij}）$_{n\times n}$，将其上三角元素逐行存于数组 B［m］中（m 充分大），使得 B［k］= a_{ij} 且 $k = f_1(i) + f_2(j) + c$。试推导出函数 f_1 ，f_2 和常数 c（要求 f_1 和 f_2 中不含常数项）。

答：$k = ni - (n-j) - i(i-1)/2 - 1$，$(i \leqslant j)$

则得 $f_1(i) = (n+1/2)i - i^2/2$，$f_2(j) = j$，$c = -(n+1)$。

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为 0.07、0.19、0.02、0.06、0.32、0.03、0.21、0.10。试为这 8 个字母设计哈夫曼编码（注意：构造哈夫曼树时确保左子树权值取小）。使用等长编码表示电文是另一种编码方案。比较两种方案的优缺点。

频数	7	19	2	6	32	3	21	10
哈夫曼编码	1010	00	10000	1001	11	10001	01	1011

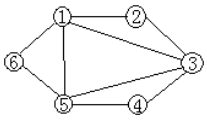
对于哈夫曼编码：带权路径长度为 2.6，

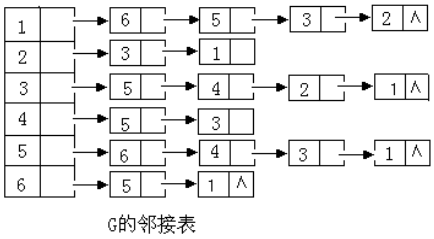
WPL＝｛（2+3）*5+（6+7+10）*4+（32+21+19）*2｝、100＝2.61

对于等长编码：带权路径长度为 3。

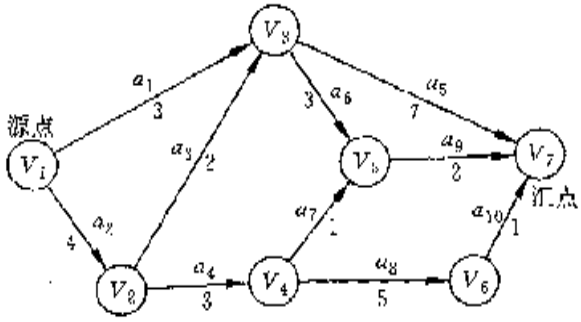
显然哈夫曼编码可以大大提高通信信道的利用率，提高报文发送速度、节省存储空间。

4、设无向图有 6 个结点，9 条边：（1，2）、（1，3）、（1，5）、（1，6）、（2，3）、（3，4）、（3，5）、（4，5）、（5，6）。画出无向图 G，画出 G 的邻接矩阵和邻接表。根据你的邻接表，从顶点 3 出发，分别画出 DFS 和 BFS 生成树。





5、有如下 AOE-网，求每一事件的最早发生时间和最迟发生时间，求每个活动的最早开始时间和最迟开始时间，找出关键活动和关键路径。



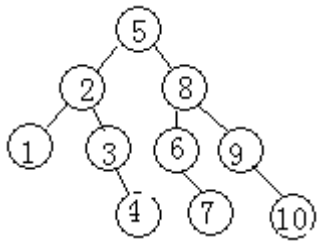
	ve	vl
V1	0	0
V2	4	4
V3	6	6
V4	7	7
V5	9	11
V6	12	12
V7	13	13

	ee	el	el-ee
a1	0	3	3
a2	0	0	0
a3	4	4	0
a4	4	4	0
a5	6	6	0
a6	6	8	2
a7	7	10	3
a8	7	7	0
a9	9	11	2
a10	12	12	0

关键活动边：a2, a3, a4, a5, a8, a10

关键路径：{a2,a3,a5}和{a2,a4,a8,a10}

6、画出对长度为 10 的有序表进行折半查找的判定树，并计算出在等概率的情况下查找成功的平均查找长度和不成功的平均查找长度。



成功：(1+2x2+3x4+4x3)/10=1+4+12+12/10=29/10

不成功：（5x3+6x4）/10=(15+24)/10=39/10

7、试给出含有 7 个排序码的最好和最坏的快速排序情况，并分别写出它的每一趟排序结果。

对快速排序来说，最好情况是基准排序码是所有排序码的中间值，能把序列分成均匀的两个子序列，且初始序列的左半部分排序码都小于基准排序码，右半部分的排序码都大于基准排序码，这样元素的调整次数将最小。假设 7 个排序码的值分别为 1~7，则最好的排序码序列为：4，1，3，2，6，5，7。

初态：　4　1　3　2　6　5　7
第 1 趟：　[2　1　3]　4　[6　5　7]
第 2 趟：　[1]　2　[3]　4　[5]　6　[7]
第 3 趟：　1　2　3　4　5　6　7

对于快速排序，最坏情况是初始序列为正序或反序。例如排序码序列为：7，6，5，4，3，2，1。

初态：　7　6　5　4　3　2　1
第 1 趟：　[1　6　5　4　3　2]　7
第 2 趟：　1　[6　5　4　3　2]　7
第 3 趟：　1　[2　5　4　3]　6　7
第 4 趟：　1　2　[5　4　3]　6　7
第 5 趟：　1　2　[3　4]　5　6　7
第 6 趟：　1　2　3　[4]　5　6　7
第 7 趟：　1　2　3　4　5　6　7

算法设计

1、 试分别用顺序表和单链表作为存储结构，实现将线性表（a_0， a_1， a_2，……，a_{n-1}）就地逆置的操作，所谓“就地”指辅助空间应为 0（1）。

（1）用顺序表作存储结构(算法 1)

```
struct SqList
 {
    ElemType *elem; //  存储空间基址
    int length; //  当前长度
    ……
 };
void InvertSqList(SqList &L)
{
    int i;ElemType temp;
    for (i=0;i<L.length/2;i++)
        {
        temp=L.elem[i];
        L.elem[i]=L.elem[L.length-i-1];
        L.elem[L.length-i-1]= temp;
        }
}
```

用顺序表作存储结构(算法 2)

```
Void InvertSqList(SqList &L)
{
    int i,j;ElemType temp;
    i=0;j= L.length-1
    while(i<j)
        {
        temp=L.elem[i];
        L.elem[i]=L.elem[j];
        L.elem[j]= temp;
        i++;j--;
        }
}
```

（2）用带头结点的单链表作存储结构

```
struct LNode
 {
    ElemType data;
    LNode *next;
 };
 typedef LNode *LinkList;
Void InvertLinkList(LinkList &L)
{
      p=L;L=NULL;
      while(p){
            s=p;p=p->next;
            s->next=L;L=s;
      }
}
```

2、试写出二分查找的递归算法

```
  typeof    int KeyType;
  typeof struct{
     KeyType key;
     ……
     }ElemType;
    typeof struct{
     ElemType *elem;
     int length
}SSTable;

   int binsrch(SSTable r, int low, int high, KeyType k)
    {int mid;low=1;high=r.length
     if (low>high)
        return(0);
```

```
    else
       {mid=(low+high)/2;
        if(k==r.elem[mid].key)
            return(mid);
        if(k> r.elem[mid].key)
            return(binsrch(r, mid+1, high, k));
        if(k> r.elem[mid].key)
            return(binsrch(r, low, mid-1, k));
        }
    }
```

3、二叉链表为存储结构，写出二叉树宽度的算法，所谓宽度是指二叉树的各层上，具有结点数最多的那一层上的结点总数。

```
typedef struct BiTNode
 {
   TElemType data;
   BiTNode *lchild,*rchild; // 左右孩子指针
 }BiTNode,*BiTree;
    int width(BiTree    T)
    {BiTree P=T, q[M];
     int front=-1, rear=-1;
     int count=0, right;
     int max=0;
     if (P!=NULL)
        {q[++rear]=P; max=1; right=rear;}
     while (front!=rear)
        {P=q[++front];
          if (T->lchild!=NULL)
             {q[++rear]= T->lchild; count++;}
          if (T->rchild!=NULL)
             {q[++rear]= T->rchild; count++;}
          if (front==right)
             {if (max < count) max=count;
              count=0;
              right=rear;
            }
          }
        return(max);
  }
```

4、已知无向图 G＝（V，E）的邻接表，给出求图 G 的连通分量个数的算法。

```
        struct ArcNode
         {
           int adjvex;
```

```
    ArcNode *nextarc;
    InfoType *info;
  };
  typedef struct
  {
    VertexType data;
    ArcNode *firstarc;
  }VNode,AdjList[MAX_VERTEX_NUM];
  struct ALGraph
  {
    AdjList adjlist;
    int n,e;
   } ALGraph;
 static int visited[n];
 void dfs (ALGraph   g, int v);
 int dfscount(ALGraph   g)
 {
      int i, j; j=0;
      for(i=0; i<g->n; i++)
          if(visited[i]==0){
              j++;
              dfs(g, i);
          }
}
 viod dfs(ALGraph   g, int v)
 {
       ArcNode*p;
       visited[v]=1;
       p=g->adjlist[v].firstarc;
       while(p!=Null){
            if(visited[p->adjvex]==0)
            dfs(g, p->adjvex);
            p=p->nextarc;}
 }
```

5、已知奇偶转换排序如下所述：第一趟对所有奇数的 i，将 a[i]和 a[i+1]进行比较，第二趟对所有偶数的 i，将 a[i]和 a[iI+1]进行比较，每次比较时，若 a[i]>a[i+1]，则将二者交换，以后重复上述过程，直至整个数组有序。试编写实现该算法的函数。

```
Void   oesort（int   a[n]）
  {     int    i,flag；
        do {
            flag＝0；
            for（i=1；i<=n；i＝i+2）
```

```
            if（a[i]>a[i+1]）{ flag＝1；  t＝a[i+1]；  a[i+1]＝a[i]；  a[i]＝t;｝
        for（i=2；i<=n；i＝i+2）
            if（a[i]>a[iI+1]）{ flag＝1；  t＝a[i+1]；  a[i+1]＝a[i]；  a[i]＝t;｝
    ｝while（flag！=0）；
}
```