数据结构

课后习题答案

目 录

第 6 章 图

1．选择题

（1）在一个图中，所有顶点的度数之和等于图的边数的（　　）倍。

A．1/2　　B．1　　C．2　　D．4

答案：C

（2）在一个有向图中，所有顶点的入度之和等于所有顶点的出度之和的（　　）倍。

A．1/2　　B．1　　C．2　　D．4

答案：B

解释：有向图所有顶点入度之和等于所有顶点出度之和。

（3）具有 n 个顶点的有向图最多有（　　）条边。

A．n　　B．n(n-1)　　C．n(n+1)　　D．n^2

答案：B

解释：有向图的边有方向之分，即为从 n 个顶点中选取 2 个顶点有序排列，结果为 n(n-1)。

（4）n 个顶点的连通图用邻接距阵表示时，该矩阵至少有（　　）个非零元素。

A．n　　B．2(n-1)　　C．n/2　　D．n^2

答案：B

（5）G 是一个非连通无向图，共有 28 条边，则该图至少有（　　）个顶点。

A．7　　B．8　　C．9　　D．10

答案：C

解释：8 个顶点的无向图最多有 8*7/2=28 条边，再添加一个点即构成非连通无向图，故至少有 9 个顶点。

（6）若从无向图的任意一个顶点出发进行一次深度优先搜索可以访问图中所有的顶点，则该图一定是（　　）图。

A．非连通　　B．连通　　C．强连通　　D．有向

答案：B

解释：即从该无向图任意一个顶点出发有到各个顶点的路径，所以该无向图是连通图。

（7）下面（　）算法适合构造一个稠密图 G 的最小生成树。

A．　Prim 算法　　B．Kruskal 算法　　C．Floyd 算法　　D．Dijkstra 算法

答案：A

解释：Prim 算法适合构造一个稠密图 G 的最小生成树，Kruskal 算法适合构造一个稀疏图 G 的最小生成树。

（8）用邻接表表示图进行广度优先遍历时，通常借助（　　）来实现算法。

A．栈　　B. 队列　　C.　树　　D．图

答案：B

解释：广度优先遍历通常借助队列来实现算法，深度优先遍历通常借助栈来实现算法。

（9）用邻接表表示图进行深度优先遍历时，通常借助（　　）来实现算法。

A．栈　　B．队列　　C．树　　D．图

答案：A

解释：广度优先遍历通常借助队列来实现算法，深度优先遍历通常借助栈来实现算法。

（10）深度优先遍历类似于二叉树的（　　）。

A．先序遍历　　B．中序遍历　　C．后序遍历　　D．层次遍历

答案：A

（11）广度优先遍历类似于二叉树的（　　）。

A．先序遍历　　B．中序遍历　　C．后序遍历　　D．层次遍历

答案：D

（12）图的 BFS 生成树的树高比 DFS 生成树的树高（　　）。

A．小　　B．相等　　C．小或相等　　D．大或相等

答案：C

解释：对于一些特殊的图，比如只有一个顶点的图，其 BFS 生成树的树高和 DFS 生成树的树高相等。一般的图，根据图的 BFS 生成树和 DFS 树的算法思想，BFS 生成树的树高比 DFS 生成树的树高小。

（13）已知图的邻接矩阵如图 6.1 所示，则从顶点 v_0 出发按深度优先遍历的结果是（　　）。

$$\begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array}\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

图 6.1　邻接矩阵

A．0 2 4 3 1 5 6

B．0 1 3 6 5 4 2

C．0 1 3 4 2 5 6

D．0 3 6 1 5 4 2

（14）已知图的邻接表如图 6.2 所示，则从顶点 v_0 出发按广度优先遍历的结果是（　　），按深度优先遍历的结果是（　　）。

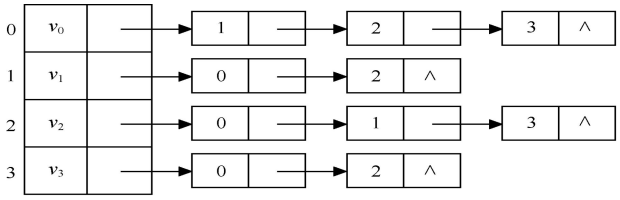


图 6.2　邻接表

A．0 1 3 2　　B．0 2 3 1　　C．0 3 2 1　　D．0 1 2 3

答案：D、D

（15）下面（　　）方法可以判断出一个有向图是否有环。

A．深度优先遍历　　B．拓扑排序　　C．求最短路径　　D．求关键路径

答案：B

2．应用题

（1）已知图 6.3 所示的有向图，请给出：

① 每个顶点的入度和出度；

② 邻接矩阵；

③ 邻接表；

④ 逆邻接表。

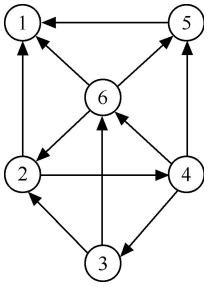


图 6.3　有向图

答案：

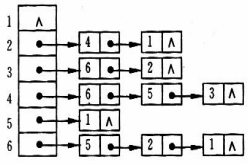
(1)

顶点	1	2	3	4	5	6
入度	3	2	1	1	2	2
出度	0	2	2	3	1	3

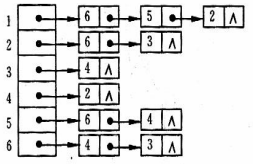
(2) 邻接矩阵

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

(3) 邻接表



(4) 逆邻接表



（2）已知如图 6.4 所示的无向网，请给出：

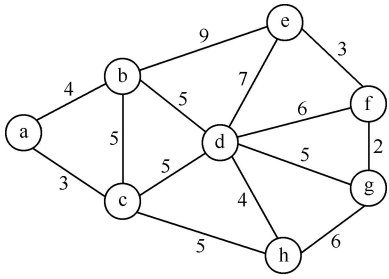


图 6.4 无向网

① 邻接矩阵；

② 邻接表；

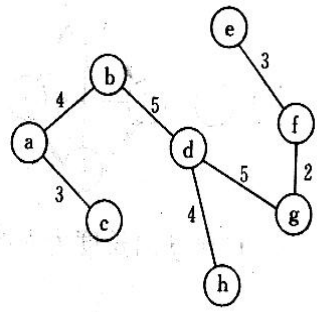
③ 最小生成树

答案：

①

$$\begin{bmatrix} \infty & 4 & 3 & \infty & \infty & \infty & \infty & \infty \\ 4 & \infty & 5 & 5 & 9 & \infty & \infty & \infty \\ 3 & 5 & \infty & 5 & \infty & \infty & \infty & 5 \\ \infty & 5 & 5 & \infty & 7 & 6 & 5 & 4 \\ \infty & 9 & \infty & 7 & \infty & 3 & \infty & \infty \\ \infty & \infty & \infty & 6 & 3 & \infty & 2 & \infty \\ \infty & \infty & \infty & 5 & \infty & 2 & \infty & 6 \\ \infty & \infty & 5 & 4 & \infty & \infty & 6 & \infty \end{bmatrix}$$

③



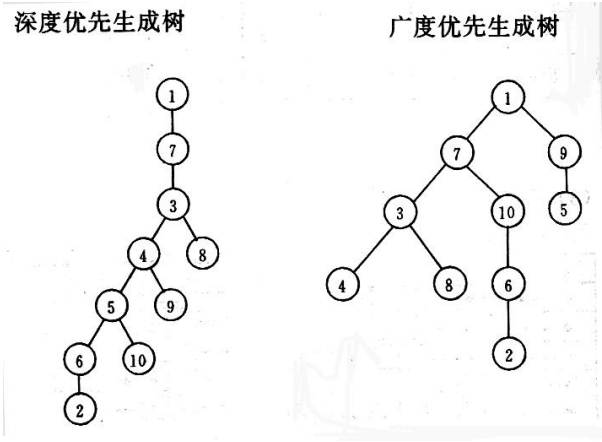
②

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

（3）已知图的邻接矩阵如图 6.5 所示。试分别画出自顶点 1 出发进行遍历所得的深度优先生成树和广度优先生成树。

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	1	0	1	0
2	0	0	1	0	0	0	1	0	0	0
3	0	0	0	1	0	0	0	1	0	0
4	0	0	0	0	1	0	0	0	1	0
5	0	0	0	0	0	1	0	0	0	1
6	1	1	0	0	0	0	0	0	0	0
7	0	0	1	0	0	0	0	0	0	1
8	1	0	0	1	0	0	0	0	1	0
9	0	0	0	0	1	0	1	0	0	1
10	1	0	0	0	0	1	0	0	0	0

图 6.5　邻接矩阵



（4）有向网如图 6.6 所示，试用迪杰斯特拉算法求出从顶点 a 到其他各顶点间的最短路径，完成表 6.1。

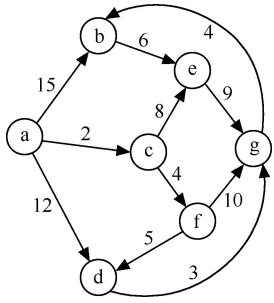


图 6.6　有向网

表 6.1

D 终点	i=1	i=2	i=3	i=4	i=5	i=6
b	15 (a,b)	15 (a,b)	15 (a,b)	15 (a,b)	15 (a,b)	**15** **(a,b)**
c	**2** **(a,c)**					
d	12 (a,d)	12 (a,d)	11 (a,c,f,d)	**11** **(a,c,f,d)**		
e	∞	10 (a,c,e)	**10** **(a,c,e)**			
f	∞	**6** **(a,c,f)**				

g	∞	∞	16 (a,c,f,g)	16 (a,c,f,g)	**14** **(a,c,f,d,g)**	
S 终 点 集	{a,c}	{a,c,f}	{a,c,f,e}	{a,c,f,e,d}	{a,c,f,e,d,g}	{a,c,f,e,d,g,b}

（5）试对图 6.7 所示的 AOE-网：

① 求这个工程最早可能在什么时间结束；

② 求每个活动的最早开始时间和最迟开始时间；

③ 确定哪些活动是关键活动

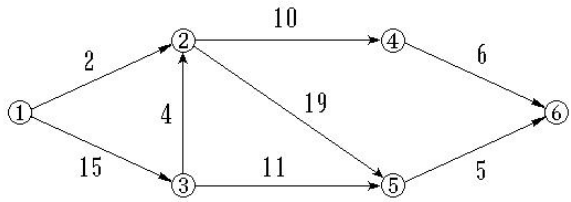


图 6.7　AOE-网

答案：按拓扑有序的顺序计算各个顶点的最早可能开始时间 *Ve* 和最迟允许开始时间 *Vl*。然后再计算各个活动的最早可能开始时间 e 和最迟允许开始时间 *l*，根据 *l* - *e* = 0? 来确定关键活动，从而确定关键路径。

	1 ∂	2 ÷	3 •	4 ≠	5 ≡	6 ≈
Ve	0	19	15	29	38	43
Vl	0	19	15	37	38	43

	<1, 2>	<1, 3>	<3, 2>	<2, 4>	<2, 5>	<3, 5>	<4, 6>	<5, 6>
e	0	0	15	19	19	15	29	38
l	17	0	15	27	19	27	37	38
-e	17	0	0	8	0	12	8	0

此工程最早完成时间为 43。关键路径为<1, 3><3, 2><2, 5><5, 6>

3．算法设计题

（1）分别以邻接矩阵和邻接表作为存储结构，实现以下图的基本操作：

① 增加一个新顶点 v，InsertVex(G, v)；

② 删除顶点 v 及其相关的边，DeleteVex(G, v);

③ 增加一条边<v，w>，InsertArc(G, v, w);

④ 删除一条边<v，w>，DeleteArc(G, v, w)。

[算法描述]

假设图 G 为有向无权图，以邻接矩阵作为存储结构四个算法分别如下：

① 增加一个新顶点 v

```
Status Insert_Vex(MGraph &G, char v)//在邻接矩阵表示的图 G 上插入顶点 v
{
  if(G.vexnum+1)>MAX_VERTEX_NUM return INFEASIBLE;
  G.vexs[++G.vexnum]=v;
  return OK;
}//Insert_Vex
```

② 删除顶点 v 及其相关的边，

```
Status Delete_Vex(MGraph &G,char v)//在邻接矩阵表示的图 G 上删除顶点 v
{
n=G.vexnum;
if((m=LocateVex(G,v))<0) return ERROR;
G.vexs[m]<->G.vexs[n]; //将待删除顶点交换到最后一个顶点
for(i=0;i<n;i++)
{
G.arcs[m]=G.arcs[n];
G.arcs[m]=G.arcs[n]; //将边的关系随之交换
}
G.arcs[m][m].adj=0;
G.vexnum--;
return OK;
}//Delete_Vex
```

分析:如果不把待删除顶点交换到最后一个顶点的话,算法将会比较复杂,而伴随着大量元素的移动,时间复杂度也会大大增加。

③ 增加一条边<v，w>

```
Status Insert_Arc(MGraph &G,char v,char w)//在邻接矩阵表示的图 G 上插入边(v,w)
{
  if((i=LocateVex(G,v))<0) return ERROR;
  if((j=LocateVex(G,w))<0) return ERROR;
  if(i==j) return ERROR;
  if(!G.arcs[j].adj)
{
  G.arcs[j].adj=1;
  G.arcnum++;
}
     return OK;
}//Insert_Arc
```

④ 删除一条边<v，w>

```
Status Delete_Arc(MGraph &G,char v,char w)//在邻接矩阵表示的图 G 上删除边(v,w)
{
if((i=LocateVex(G,v))<0) return ERROR;
if((j=LocateVex(G,w))<0) return ERROR;
if(G.arcs[j].adj)
{
G.arcs[j].adj=0;
G.arcnum--;
}
return OK;
}//Delete_Arc
```

以邻接表作为存储结构，本题只给出 Insert_Arc 算法.其余算法类似。

```
Status Insert_Arc(ALGraph &G,char v,char w)//在邻接表表示的图 G 上插入边(v,w)
{
if((i=LocateVex(G,v))<0) return ERROR;
if((j=LocateVex(G,w))<0) return ERROR;
p=new ArcNode;
p->adjvex=j;p->nextarc=NULL;
if(!G.vertices.firstarc) G.vertices.firstarc=p;
else
{
for(q=G.vertices.firstarc;q->q->nextarc;q=q->nextarc)
if(q->adjvex==j) return ERROR; //边已经存在
q->nextarc=p;
}
G.arcnum++;
return OK;
}//Insert_Arc
```

（2）设计一个算法，求图 G 中距离顶点 v 的最短路径长度最大的一个顶点，设 v 可达其余各个顶点。

[题目分析]

利用 Dijkstra 算法求 v0 到其它所有顶点的最短路径，分别保存在数组 D[i]中，然后求出 D[i]中值最大的数组下标 m 即可。

[算法描述]

```
int ShortestPath_MAX(AMGraph G, int v0){
    //用 Dijkstra 算法求距离顶点 v0 的最短路径长度最大的一个顶点 m
    n=G.vexnum;                              //n 为 G 中顶点的个数
    for(v = 0; v<n; ++v){                    //n 个顶点依次初始化
      S[v] = false;                          //S 初始为空集
      D[v] = G.arcs[v0][v];                  //将 v0 到各个终点的最短路径长度初始化
      if(D[v]< MaxInt)   Path [v]=v0;        //如果 v0 和 v 之间有弧，则将 v 的前驱置为 v0
      else Path [v]=-1;                      //如果 v0 和 v 之间无弧，则将 v 的前驱置为-1
    }//for
    S[v0]=true;                              //将 v0 加入 S
    D[v0]=0;                                 //源点到源点的距离为 0
    /*开始主循环，每次求得 v0 到某个顶点 v 的最短路径，将 v 加到 S 集*/
    for(i=1;i<n; ++i){                       //对其余 n−1 个顶点，依次进行计算
      min= MaxInt;
      for(w=0;w<n; ++w)
        if(!S[w]&&D[w]<min)
            {v=w; min=D[w];}                 //选择一条当前的最短路径，终点为 v
      S[v]=true;                             //将 v 加入 S
      for(w=0;w<n; ++w)                      //更新从 v0 到 V−S 上所有顶点的最短路径长度
      if(!S[w]&&(D[v]+G.arcs[v][w]<D[w])){
          D[w]=D[v]+G.arcs[v][w];             //更新 D[w]
          Path [w]=v;                        //更改 w 的前驱为 v
      }//if
    }//for
/*最短路径求解完毕，设距离顶点 v0 的最短路径长度最大的一个顶点为 m */
Max=D[0];
m=0;
for(i=1;i<n;i++)
    if(Max<D[i]) m=i;
return m;
}
```

第 7 章　查找

1．选择题

（1）对 n 个元素的表做顺序查找时，若查找每个元素的概率相同，则平均查找长度为（　　）。

A．(n-1)/2　　B．　n/2　　C．(n+1)/2　　D．n

答案：C

解释：总查找次数 N=1+2+3+…+n=n(n+1)/2，则平均查找长度为 N/n=(n+1)/2。

（2）适用于折半查找的表的存储方式及元素排列要求为（　　）。

A．链接方式存储，元素无序　　B．链接方式存储，元素有序

C．顺序方式存储，元素无序　　D．顺序方式存储，元素有序

答案：D

解释：折半查找要求线性表必须采用顺序存储结构，而且表中元素按关键字有序排列。

（3）如果要求一个线性表既能较快的查找，又能适应动态变化的要求，最好采用(　　)查找法。

A．顺序查找　　B．折半查找

C．分块查找　　D．哈希查找

答案：C

解释：分块查找的优点是：在表中插入和删除数据元素时，只要找到该元素对应的块，就可以在该块内进行插入和删除运算。由于块内是无序的，故插入和删除比较容易，无需进行大量移动。如果线性表既要快速查找又经常动态变化，则可采用分块查找。

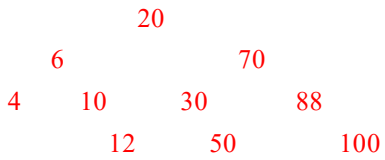
（4）折半查找有序表（4，6，10，12，20，30，50，70，88，100）。若查找表中元素 58，则它将依次与表中（　　）比较大小，查找结果是失败。

A．20，70，30，50　　B．30，88，70，50

C．20，50　　D．30，88，50

答案：A

解释：表中共 10 个元素，第一次取⌊(1+10)/2⌋=5，与第五个元素 20 比较，58 大于 20，再取⌊(6+10)/2⌋=8，与第八个元素 70 比较，依次类推再与 30、50 比较，最终查找失败。



（5）对 22 个记录的有序表作折半查找，当查找失败时，至少需要比较（　　）次关键字。

A．3　　B．4　　C．5　　D．6

答案：B

解释：22 个记录的有序表，其折半查找的判定树深度为$\lfloor \log_2 22 \rfloor + 1 = 5$，且该判定树不是满二叉树，即查找失败时至多比较 5 次，至少比较 4 次。

（6）折半搜索与二叉排序树的时间性能（　　）。

A．相同　　B．完全不同

C．有时不相同　　D．数量级都是 $O(\log_2 n)$

答案：C

解释：折半查找复杂度恒定是 log2n，但二叉排序树最优时间复杂度是 log2n，只有平衡二叉树才是 log2n。

（7）分别以下列序列构造二叉排序树，与用其它三个序列所构造的结果不同的是（　　）。

A．（100，80，　90，　60，　120，110，130）

B．（100，120，110，130，80，　60，　90）

C．（100，60，　80，　90，　120，110，130）

D．(100，80，　60，　90，　120，130，110)

答案：C

解释：A、B、C、D 四个选项构造二叉排序树都以 100 为根，易知 A、B、D 三个序列中第一个比 100 小的关键字为 80，即 100 的左孩子为 80，而 C 选项中 100 的左孩子为 60，故选 C。

（8）在平衡二叉树中插入一个结点后造成了不平衡，设最低的不平衡结点为 A，并已知 A 的左孩子的平衡因子为 0 右孩子的平衡因子为 1，则应作（　　）型调整以使其平衡。

A．LL　　B．LR　　C．RL　　D．RR

答案：C

（9）下列关于 m 阶 B-树的说法错误的是（　　）。

A．根结点至多有 m 棵子树

B．所有叶子都在同一层次上

C．非叶结点至少有 m/2 (m 为偶数)或 m/2+1（m 为奇数）棵子树

D．根结点中的数据是有序的

答案：D

（10）下面关于 B-和 B+树的叙述中，不正确的是（　　）。

A．B-树和 B+树都是平衡的多叉树　　B．B-树和 B+树都可用于文件的索引结构

C．B-树和 B+树都能有效地支持顺序检索　D．B-树和 B+树都能有效地支持随机检索

答案：C

（11）m 阶 B-树是一棵（　　）。

A．m 叉排序树　　B．m 叉平衡排序树

C．m-1 叉平衡排序树　　D．m+1 叉平衡排序树

答案：B

（12）下面关于哈希查找的说法，正确的是（　　）。

A．哈希函数构造的越复杂越好，因为这样随机性好，冲突小

B．除留余数法是所有哈希函数中最好的

C．不存在特别好与坏的哈希函数，要视情况而定

D．哈希表的平均查找长度有时也和记录总数有关

答案：C

（13）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

答案：A

解释：在同义词构成的单链表中，查找该单链表表中不同元素，所消耗的时间不同。

（14）设哈希表长为 14，哈希函数是 H(key)=key%11，表中已有数据的关键字为 15，38，61，84 共四个，现要将关键字为 49 的元素加到表中，用二次探测法解决冲突，则放入的位置是（　　）。

A．8　　B．3　　C．5　　D．9

答案：D

解释：关键字 15 放入位置 4，关键字 38 放入位置 5，关键字 61 放入位置 6，关键字 84 放入位置 7，再添加关键字 49，计算得到地址为 5，冲突，用二次探测法(49+1)解决冲突得到新地址为 6，仍冲突，再用二次探测法(49-1)解决冲突，得到新地址为 4，仍冲突，再用二次探测法(49+2 的平方)解决冲突，得到新地址为 9，不冲突，即将关键字 49 放入位置 9。

（15）采用线性探测法处理冲突，可能要探测多个位置，在查找成功的情况下，所探测的这些位置上的关键字（　　）。

A．不一定都是同义词　　B．一定都是同义词

C．一定都不是同义词　　D．都相同

答案：A

解释：所探测的这些关键字可能是在处理其它关键字冲突过程中放入该位置的。

2．应用题

（1）假定对有序表：（3，4，5，7，24，30，42，54，63，72，87，95）进行折半查找，试回答下列问题：

① 画出描述折半查找过程的判定树；

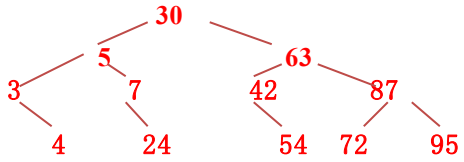
② 若查找元素 54，需依次与哪些元素比较？

③ 若查找元素 90，需依次与哪些元素比较？

④ 假定每个元素的查找概率相等，求查找成功时的平均查找长度。

答案：

①先画出判定树如下（注：mid=⌊(1+12)/2⌋=6）：



②查找元素 54，需依次与 30, 63, 42, 54 元素比较；

③查找元素 90，需依次与 30, 63,87, 95 元素比较；

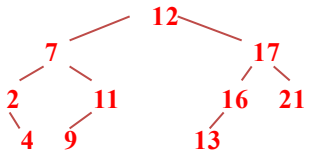
④求 ASL 之前，需要统计每个元素的查找次数。判定树的前 3 层共查找 1＋2×2＋4×3=17 次；

但最后一层未满，不能用 8×4，只能用 5×4=20 次，

所以 ASL＝1/12（17＋20）＝37/12≈3.08

（2）在一棵空的二叉排序树中依次插入关键字序列为 12，7，17，11，16，2，13，9，21，4，请画出所得到的二叉排序树。

答案：



验算方法：　用中序遍历应得到排序结果：2,4,7,9,11,12,13,16,17,21

（3）已知如下所示长度为 12 的表：（Jan, Feb, Mar, Apr, May, June, July, Aug, Sep, Oct, Nov, Dec）

① 试按表中元素的顺序依次插入一棵初始为空的二叉排序树，画出插入完成之后的二叉排序树，并求其在等概率的情况下查找成功的平均查找长度。

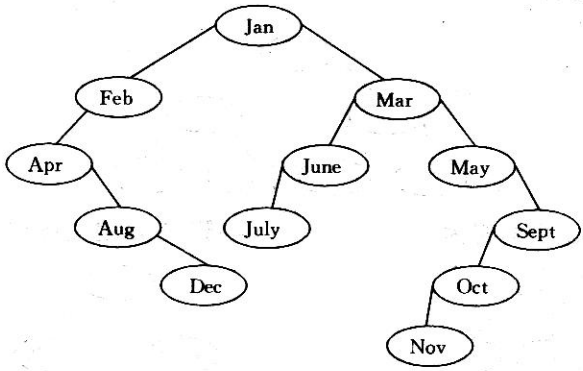
② 若对表中元素先进行排序构成有序表，求在等概率的情况下对此有序表进行折半查找时查找成功的平均查找长度。

③ 按表中元素顺序构造一棵平衡二叉排序树，并求其在等概率的情况下查找成功的平均查找长度。

答案：

(1) 求得的二叉排序树如下图所示，在等概率情况下查找成功的平均查找长度为

$$ASL_{succ}=\frac{1}{12}(1\times 1+2\times 2+3\times 3+4\times 3+5\times 2+6\times 1)=\frac{42}{12}$$



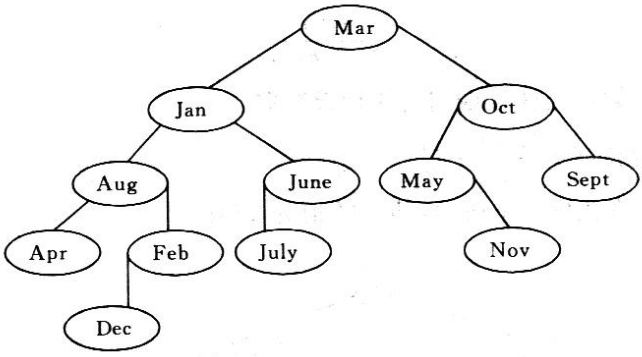
(2) 经排序后的表及在折半查找时找到表中元素所经比较的次数对照如下：

Apr	Aug	Dec	Feb	Jan	July	June	Mar	May	Nov	Oct	Sept
3	4	2	3	4	1	3	4	2	4	3	4

等概率情况下查找成功时的平均查找长度为

$$ASL_{succ}=\frac{1}{12}(1\times 1+2\times 2+3\times 4+4\times 5)=\frac{37}{12}$$

(3) 平衡二叉树为



它在等概率情况下的平均查找长度为

$$ASL=\frac{1}{12}(1\times 1+2\times 2+3\times 4+4\times 4+5\times 1)=\frac{38}{12}$$

（4）设哈希表的地址范围为 0～17，哈希函数为：H（key）=key%16。用线性探测法处理冲突，输入关键字序列：（10，24，32，17，31，30，46，47，40，63，49），构造哈希表，试回答下列问题：

① 画出哈希表的示意图；

② 若查找关键字 63，需要依次与哪些关键字进行比较？

③ 若查找关键字 60，需要依次与哪些关键字比较？

④ 假定每个关键字的查找概率相等，求查找成功时的平均查找长度。

答案：

①画表如下：

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
32	17	63	49					24	40	10				30	31	46	47

②查找 63,首先要与 H(63)=63%16=15 号单元内容比较，即 63 与 31 比较 ,不匹配;然后顺移，与 46,47,32,17,63 相比，一共比较了 6 次！

③查找 60,首先要与 H(60)=60%16=12 号单元内容比较，但因为 12 号单元为空（应当有空标记），所以应当只比较这一次即可。

④对于黑色数据元素，各比较 1 次；共 6 次；

对红色元素则各不相同，要统计移位的位数。“63”需要 6 次，“49”需要 3 次，“40”需要 2 次，“46”需要 3 次，“47”需要 3 次，

所以 ASL=1/11（6＋2＋3×3+6）＝23/11

（5）设有一组关键字（9，01，23，14，55，20，84，27），采用哈希函数：H（key）=key %7 ，表长为 10，用开放地址法的二次探测法处理冲突。要求：对该关键字序列构造哈希表，并计算查找成功的平均查找长度。

答案：

散列地址	0	1	2	3	4	5	6	7	8	9
关键字	14	1	9	23	84	27	55	20		
比较次数	1	1	1	2	4	3	1	2		

平均查找长度：ASL_{succ}=（1+1+1+2+4+3+1+2）/8=15/8

以关键字 27 为例：H（27）=27%7=6（冲突） H_1=（6+1^2）%10=7（冲突）

H_2=（6-1^2）%10=5 所以比较了 3 次。

（6）设哈希函数 H（K）=3 K mod 11，哈希地址空间为 0～10，对关键字序列（32，13，49，24，38，21，4，12），按下述两种解决冲突的方法构造哈希表，并分别求出等概率下查找成功时和查找失败时的平均查找长度 ASL_{succ} 和 ASL_{unsucc}。

① 线性探测法；

② 链地址法。

答案：

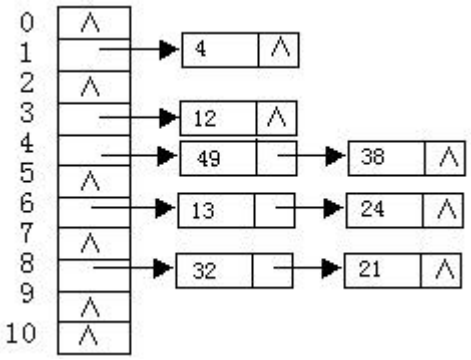
①

散列地址	0	1	2	3	4	5	6	7	8	9	10
关键字		4		12	49	38	13	24	32	21	
比较次数		1		1	1	2	1	2	1	2	

ASL_{succ} =（1+1+1+2+1+2+1+2）/8=11/8

ASL_{unsucc}=（1+2+1+8+7+6+5+4+3+2+1）/11=40/11

②



ASL_{succ} =（1*5+2*3）/8=11/8

ASL_{unsucc}=（1+2+1+2+3+1+3+1+3+1+1）/11=19/11

3．算法设计题

（1）试写出折半查找的递归算法。

[算法描述]

```
int  BinSrch（rectype r[ ]，int  k，low，high）
//在长为 n 的有序表中查找关键字 k，若查找成功，返回 k 所在位置，查找失败返回 0。
{if（low≤high）  //low 和 high 分别是有序表的下界和上界
  {mid=（low+high）/2;
   if（r[mid].key==k）return  （mid）;
   else if（r[mid].key>k）return  （BinSrch（r,k，mid+1，high））;
        else return  （BinSrch（r,k，low，mid-1））;
   }
 else  return  （0）; //查找失败。
}//算法结束
```

（2）试写一个判别给定二叉树是否为二叉排序树的算法。

[题目分析] 根据二叉排序树中序遍历所得结点值为增序的性质，在遍历中将当前遍历结点与其前驱结点值比较，即可得出结论，为此设全局指针变量 pre（初值为 null）和全局变量 flag，初值为 true。若非二叉排序树，则置 flag 为 false。

[算法描述]

```
#define true 1
#define false 0
typedef struct node
{datatype data; struct node *lchild,*rchild;} *BTree;
void  JudgeBST（BTree T,int  flag）
// 判断二叉树是否是二叉排序树，本算法结束后，在调用程序中由 flag 得出结论。
{ if（T!=null && flag）
     { Judgebst（T->lchild,flag）; // 中序遍历左子树
       if（pre==null）pre=T; // 中序遍历的第一个结点不必判断
       else  if（pre->data<T->data）pre=T; //前驱指针指向当前结点
               else{flag=flase; }   //不是完全二叉树
       Judgebst （T->rchild,flag）; // 中序遍历右子树
                          }//JudgeBST 算法结束
```

第 8 章　排序

1．选择题

（1）从未排序序列中依次取出元素与已排序序列中的元素进行比较，将其放入已排序序列的正确位置上的方法，这种排序方法称为（　　）。

A．归并排序　　B．冒泡排序　　C．插入排序　　D．选择排序

答案：C

（2）从未排序序列中挑选元素，并将其依次放入已排序序列（初始时为空）的一端的方法，称为（　　）。

A．归并排序　　B．冒泡排序　　C．插入排序　　D．选择排序

答案：D

（3）对 n 个不同的关键字由小到大进行冒泡排序，在下列（　　）情况下比较的次数最多。

A．从小到大排列好的　　B．从大到小排列好的

C．元素无序　　D．元素基本有序

答案：B

解释：对关键字进行冒泡排序，关键字逆序时比较次数最多。

（4）对 n 个不同的排序码进行冒泡排序，在元素无序的情况下比较的次数最多为（　　）。

A．n+1　　B．n　　C．n-1　　D．n(n-1)/2

答案：D

解释：比较次数最多时，第一次比较 n-1 次，第二次比较 n-2 次……最后一次比较 1 次，即(n-1)+(n-2)+…+1= n(n-1)/2。

（5）快速排序在下列（　　）情况下最易发挥其长处。

A．被排序的数据中含有多个相同排序码

B．被排序的数据已基本有序

C．被排序的数据完全无序

D．被排序的数据中的最大值和最小值相差悬殊

答案：C

解释：B 选项是快速排序的最坏情况。

（6）对 n 个关键字作快速排序，在最坏情况下，算法的时间复杂度是（　　）。

A．O(n)　　B．$O(n^2)$　　C．$O(n\log_2 n)$　　D．$O(n^3)$

答案：B

解释：快速排序的平均时间复杂度为 $O(n\log_2 n)$，但在最坏情况下，即关键字基本排好序的情况下，时间复杂度为 $O(n^2)$。

（7）若一组记录的排序码为（46, 79，56，38，40，84），则利用快速排序的方法，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

答案：C

（8）下列关键字序列中，（　　）是堆。

A．16，72，31，23，94，53　　B．94，23，31，72，16，53

C．16，53，23，94，31，72　　D．16，23，53，31，94，72

答案：D

解释：D 选项为小根堆

（9）堆是一种（　　）排序。

A．插入　　B．选择　　C．交换　　D．归并

答案：B

（10）堆的形状是一棵（　　）。

A．二叉排序树　　B．满二叉树　　C．完全二叉树　　D．平衡二叉树

答案：C

（11）若一组记录的排序码为（46，79，56，38，40，84），则利用堆排序的方法建立的初始堆为（　　）。

A．79，46，56，38，40，84　　B．84，79，56，38，40，46

C．84，79，56，46，40，38　　D．84，56，79，40，46，38

答案：B

（12）下述几种排序方法中，要求内存最大的是（　　）。

A．希尔排序　　B．快速排序　　C．归并排序　　D．堆排序

答案：C

解释：堆排序、希尔排序的空间复杂度为 O(1)，快速排序的空间复杂度为 $O(\log_2 n)$，归并排序的空间复杂度为 O(n)。

（13）下述几种排序方法中，（　　）是稳定的排序方法。

A．希尔排序　　B．快速排序　　C．归并排序　　D．堆排序

答案：C

解释：不稳定排序有希尔排序、简单选择排序、快速排序、堆排序；稳定排序有直接插入排序、折半插入排序、冒泡排序、归并排序、基数排序。

（14）数据表中有 10000 个元素，如果仅要求求出其中最大的 10 个元素，则采用(　　)算法最节省时间。

A．冒泡排序　　B．快速排序　　C．简单选择排序　　D．堆排序

答案：D

（15）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．希尔排序　　B．快速排序　　C．冒泡排序　　D．堆排序

答案：A

解释：快速排序的每趟排序能将作为枢轴的元素放到最终位置；冒泡排序的每趟排序能将最大或最小的元素放到最终位置；堆排序的每趟排序能将最大或最小的元素放到最终位置。

2．应用题

（1）设待排序的关键字序列为{12，2，16，30，28，10，16*，20，6，18}，试分别写出使用以下排序方法，每趟排序结束后关键字序列的状态。

① 直接插入排序

② 折半插入排序

③ 希尔排序（增量选取 5，3，1）

④ 冒泡排序

⑤ 快速排序

⑥ 简单选择排序

⑦ 堆排序

⑧ 二路归并排序

答案：

①直接插入排序

[2 12] 16 30 28 10 16* 20 6 18
[2 12 16] 30 28 10 16* 20 6 18
[2 12 16 30] 28 10 16* 20 6 18
[2 12 16 28 30] 10 16* 20 6 18
[2 10 12 16 28 30] 16* 20 6 18
[2 10 12 16 16* 28 30] 20 6 18
[2 10 12 16 16* 20 28 30] 6 18
[2 6 10 12 16 16* 20 28 30] 18
[2 6 10 12 16 16* 18 20 28 30]

② 折半插入排序 排序过程同①

③ 希尔排序（增量选取 5，3，1）

10 2 16 6 18 12 16* 20 30 28 （增量选取 5）
6 2 12 10 18 16 16* 20 30 28 （增量选取 3）
2 6 10 12 16 16* 18 20 28 30 （增量选取 1）

④ 冒泡排序

2 12 16 28 10 16* 20 6 18 [30]
2 12 16 10 16* 20 6 18 [28 30]
2 12 10 16 16* 6 18 [20 28 30]
2 10 12 16 6 16* [18 20 28 30]

2 10 12 6 16 [16* 18 20 28 30]
2 10 6 12 [16 16* 18 20 28 30]
2 6 10 [12 16 16* 18 20 28 30]
2 6 10 12 16 16* 18 20 28 30]

⑤ 快速排序

12 [6 2 10] **12** [28 30 16* 20 16 18]
6 [2] **6** [10] 12 [28 30 16* 20 16 18]
28 2 6 10 12 [18 16 16* 20] **28** [30]
18 2 6 10 12 [16* 16] **18** [20] 28 30
16* 2 6 10 12 16* [16] 18 20 28 30
左子序列递归深度为 1，右子序列递归深度为 3

⑥ 简单选择排序

2 [12 16 30 28 10 16* 20 6 18]
2 6 [16 30 28 10 16* 20 12 18]
2 6 10 [30 28 16 16* 20 12 18]
2 6 10 12 [28 16 16* 20 30 18]
2 6 10 12 16 [28 16* 20 30 18]
2 6 10 12 16 16* [28 20 30 18]
2 6 10 12 16 16* 18 [20 30 28]
2 6 10 12 16 16* 18 20 [28 30]
2 6 10 12 16 16* 18 20 28 [30]

⑦堆排序（红色加粗为本次需要交换的，黑色为已经排完序的）

12 **2** 16 **30** 28 10 16* 20 6 18
12 **30** 16 2 28 10 16* 20 6 18
30 12 16 **2** 28 10 16* **20** 6 18
30 **12** 16 20 **28** 10 16* 2 6 18
30 28 16 20 12 10 16* 2 6 **18**
18 **28** 16 20 12 10 16* 2 6 30
28 18 16 20 12 10 16* 2 **6** 30
6 **18** 16 **20** 12 10 16* 2 28 30
6 **20** 16 18 12 10 16* 2 28 30
20 6 16 18 12 10 16* **2** 28 30
2 **6** 16 **18** 12 10 16* 20 28 30
2 **18** 16 6 12 10 16* 20 28 30
18 2 16 6 12 10 **16*** 20 28 30

16* **2** 16 6 **12** 10 18 20 28 30

16* 12 16 6 2 **10** 18 20 28 30

10 12 **16** 6 2 16* 18 20 28 30

16 12 10 6 **2** 16* 18 20 28 30

2 **12** 10 6 16 16* 18 20 28 30

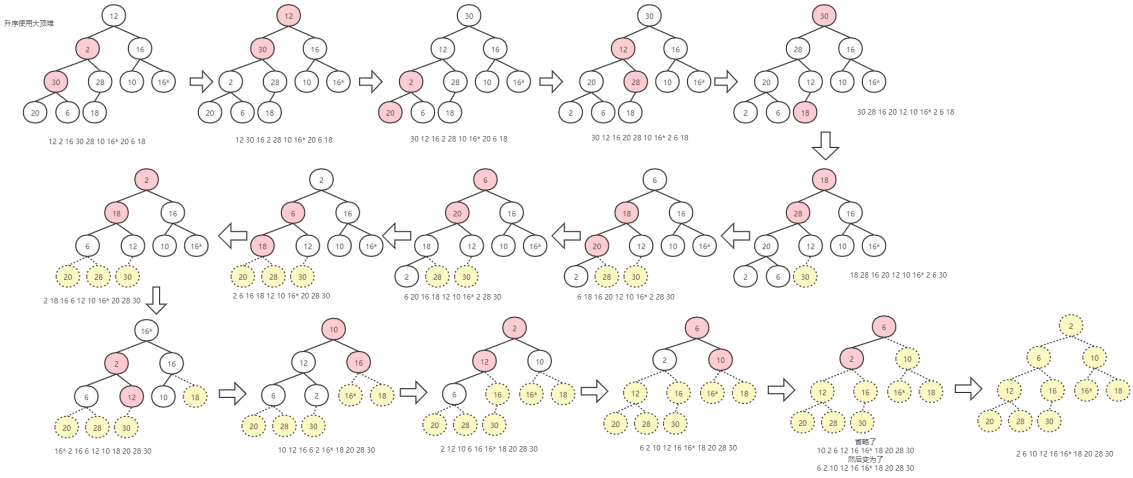
12 2 10 **6** 16 16* 18 20 28 30

6 2 **10** 12 16 16* 18 20 28 30

10 2 **6** 12 16 16* 18 20 28 30

6 **2** 10 12 16 16* 18 20 28 30

2 6 10 12 16 16* 18 20 28 30



⑧ 二路归并排序

2 12　　16 30　　10 28　　16 * 20　　6 18

2 12 16 30　　10 16* 20 28　　6 18

2 10 12 16 16* 20 28 30　　6 18

2 6 10 12 16 16* 18 20 28 30

（2）给出如下关键字序列｛321，156，57，46，28，7，331，33，34，63｝，试按链式基数排序方法，列出每一趟分配和收集的过程。

答案：

按最低位优先法　→321→156→57→46→28→7→331→33→34→63

分配 [0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

321　33　34　156　57　28

331　63　46　7

收集　→321→331→33→63→34→156→46→57→7→28