数据结构及其应用算法

2.3 已知顺序表 La 中数据元素按非递减有序排列。试写一个算法，将元素 x 插到 La 的合适位置上，保持该表的有序性。

```
Status ListInsert_Sq(Sqlist &L,ElemType e)
{       if(L.length>=L.listsize) Increment(L);//如果在插入之前顺序表已经处于满状态，需要先增加顺序表容量

        int i;
        for (i=0;i<L.length;)
        {       if(L.elem[i]<=x)   i++;  //确定插入位置
                else{       ElemType *p,*q;
                            if(i==0)    q=&(L.elem[0]); // 如果i为0, x是最小的，插入顺序表最前端
                            else      q=&(L.elem[i-1]); //i>0,插入位置为i-1

                            for(p=&(L.elem[L.length-1]);p>=q;--p)   *(p+1)=*p; // 由后往前逐渐后移元素

                            *q=x; //插入x
                            ++L.length; //更新顺序表长度
                            break;
                    }
         }
        if(i==L.length){  //如果i为L.length，则直接插入顺序表最末端
                L.elem[L.length]=x;
                L.length++;}
        return OK;}
```

2.5 试写一个算法，实现顺序表的就地逆置，即在原表的存储空间将线性表（a1,a2, ..., an-1,an）逆置为(an,an-1, ..., a2,a1)

```
Status ListInvert_Sq(Sqlist list)
{       int i=0;
        if(list.length) //判断顺序表不为空
        {
                while ((i!=list.length-1-i)&&(list.length-1-i!=1))
                {
                        ElemType temp;
                        temp = list.elem[i];
                        list.elem[i] = list.elem[list.length-i-1];
                        list.elem[list.length-i-1] = temp;
                        i++;
                }
        }
        else //顺序表为空
        {
                cout<<"Empty list!"<<endl;
                return FALSE;
        }
        return OK;
}
```

2.6 试写一个算法，对带头结点的单链表实现就地逆置

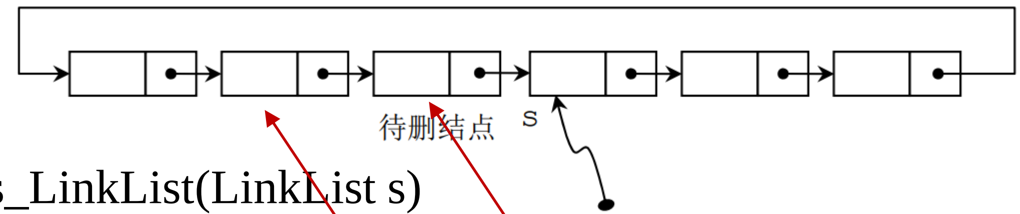
```
Status LinkListwithHeadNodeInvert(LinkList &L)
{       LinkList p,q;
        p=L->next;
        if(!p||!p->next) //L为空表或只有一个结点
                cout<<"empty link list or only contain one element"<<endl;
        else
        {       q=p->next;
                p->next=NULL;
                while(q) //从第二个节点开始打断放到第一个，依次类推，便可逆置

                {
                        p=q->next;
                        q->next=L->next;
                        L->next=q;
                        q=p;
                }

        }
        return OK;
}
```

2.9 设有两个非递减有序的单链表 A 和 B。请写出算法，将A和B“就地” 归并成一个按元素值非递增有序的单链表 C。

```
Status MergeLinkList(LinkList &A, LinkList &B)
{// 假设A 、B都含有头结点，先将A、B合并，再将其逆序
        LinkList p=B->next;
        while(p) //对于B中的每个结点p
        {
          ListInsert_WithHeadNode(A,p->data);//将p->data插入到非递减有序的单链表 A中的合适位置
          p=p->next;
        }
        //
        LinkListwithHeadNodeInvert(A); //参照题2.6，对带头结点的单链表实现就地逆置
        return OK;
}
```

```
Status ListInsert_WithHeadNode(LinkList &L, ElemType e)
{// 已知单链表 La (带头结点)中数据元素按非递减有序排列，将元素 e 插到 La 的合适位置上
        LinkList P,Q;
        P=L->next;
        while(P) // L不是空表
        {        if(P->data<=e) //确定插入位置为P的前驱Q后面
                 {        Q=P;
                          P=P->next;
                 }
                 else{    LinkList S=(LinkList)malloc(sizeof(LNode)); //动态分配空间
                          S->data=e;
                          Q->next=S;
                          S->next=P; //前插，S插到P的前面
                          return OK;
                    }
        }
        //P为空指针，L为空表
        LinkList S=(LinkList)malloc(sizeof(LNode));
        S->data=e;
        P=S;
        S->next=NULL;  // 置S为首元结点，后继为空
        return OK;
}
```

2.10 设有一个<u>**长度大于 1**</u> 的单向循环链表，表中既无头结点，也无头指针，s 为指向表中某个结点的指针,如图 2-1 所示。试编写一个算法，删除链表中指针 s 所指结点的直接前驱。




```
Status DeletePrevious_LinkList(LinkList s)
{           LinkList p,q;
            p=s;
//  if(p->next=s){      cout<<"one element only, so just delete the whole linklist"<<endl;
//                      delete(s);
//                      return OK;
//          }
            while(p->next!=s) //长度大于1的链表，找到s的直接前驱p, q为p的前驱
            {
                    q=p;
                    p=p->next;
            }
            q->next=s; //q的后继置为s,删除p
            delete p; //释放内存空间
            return OK;
}
```

2.12已知线性表用顺序存储结构表示，表中数据元素为 n 个正整数。试写一算法，分离该表中的奇数和偶数，使得所有奇数集中放在左侧，偶数集中放在右侧。要求：(1)**不借助辅助数组**;(2)**时间复杂度为 O(n)**。

```
Status SplitSqlistbyOddandEven(Sqlist &L)
        int i=0,j,k=0;
        j = L.length-1; //i为最左端，j为最右端
        while (i<=j)
        {       while(L.elem[i]%2!=0&&i<=j) //从左边开始找偶数L.elem[i]
                        i++;
                while (L.elem[j]%2==0&&i<=j) // 从右边开始找奇数L.elem[j]
                        j--;
                if(i<j){ //交换 L.elem[i] 与L.elem[j]，使得奇数在左侧，偶数在右侧
                        ElemType temp; temp=L.elem[i];
                        L.elem[i]=L.elem[j]; L.elem[j]=temp;
                        i++;
                        j--;}
        }
        return OK;
}
```

第五章 串和数组

5.1 已知多维数组 A[2][2][3][3]**按行优先方式**存储。试按存储位置的先后次序，列出所有数组元素 A[i][j][k][l]序列（为了简化表达，可以只列出形如“i,j,k,l”的序列，如元素 A[0][0][2][1]可表示为“0,0,2,1”）。

[0,0,0,0]->[0,0,0,1]->[0,0,0,2]->[0,0,1,0]->[0,0,1,1]->…->[1,1,2,0]->[1,1,2,1]->[1,1,2,2]

0 0 0 0, 0 0 0 1, 0 0 0 2,
0 0 1 0, 0 0 1 1, 0 0 1 2,
0 0 2 0, 0 0 2 1, 0 0 2 2,
0 1 0 0, 0 1 0 1, 0 1 0 2,
0 1 1 0, 0 1 1 1, 0 1 1 2,
0 1 2 0, 0 1 2 1, 0 1 2 2,
1 0 0 0, 1 0 0 1, 1 0 0 2,
1 0 1 0, 1 0 1 1, 1 0 12,
1 0 2 0, 1 0 2 1, 1 0 2 2,
1 1 0 0, 1 1 0 1, 1 1 0 2,
1 1 1 0, 1 1 1 1, 1 1 1 2,
1 1 2 0, 1 1 2 1, 1 1 2 2,

5.2 假设有一个二维数组 A[0..5][0..7]，每个元素占 6 个字节，首元素 A[0][0] 的地址为 1000，求：

(1)A 的体积；

(2)最后一个元素 A[5][7]的地址；

(3)按行主序方式存储时，A[2][4]的地址；

(4)按列主序方式存储时，A[2][4]的地址；

（1）6*8*6 = 288

（2）1000+（5*8+7）*6 =1282

或者 1000+（5+7*6）*6 =1282

（3）1000+（2*8+4）*6 =1120

（4）1000+（4*6+2）*6 =1156

二维数组A[m][n]中每个元素占L个存储单元，元素 ai,j 的存储地址

- 按行主序 LOC[i,j] = LOC[0,0]+(i*n+j)*L
- 按列主序 LOC[i,j] = LOC[0,0]+(i+j*m)*L

m = 6, n=8

5.3 设有上三角矩阵 An×n，

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ & a_{22} & a_{23} & \dots & a_{2n} \\ & & a_{33} & \dots & a_{3n} \\ & C & & \dots & \dots \\ & & & & a_{nn} \end{bmatrix}$$

将其上三角的元素逐行存于数组B[0..m-1]中（m充分大），使得B[k]=aij且k＝f1(i)+f2(j)+c。试推导出函数f1、f2和常数c（要求f1和f2中不含常数项）。

k=j-(i*i-2*n*i-i)/2-n-1
f1=-(i*i-2*n*i-i)/2
f2=j,
c=-(n+1)

B[k] = aij : a00,…,aij 共有元素k+1个

一共i行，前i-1行中每行元素个数为 n+1-i,
第i行元素个数为j+1-i

相加等于k+1, 得到k=j-(i*i-2*n*i-i)/2-n-1

5.4设有一个准对角矩阵

$$\begin{bmatrix} a_{11} & a_{12} & & & & & & \\ a_{21} & a_{22} & & & & & & \\ & & a_{33} & a_{34} & & & & \\ & & a_{43} & a_{44} & & & & \\ & & & & \cdots & \cdots & & \\ & & & & \cdots & \cdots & & \\ & & & & & & a_{2m-1,2m-1} & a_{2m-1,2m} \\ & & & & & & a_{2m,2m-1} & a_{2m,2m} \end{bmatrix}$$

按以下方式存于一维数组 B[4m]中：

0	1	2	3	4	5	6		k		4m-2	4m-1
a_{11}	a_{12}	a_{21}	a_{22}	a_{33}	a_{34}	a_{43}	...	a_{ij}	...	$a_{2m-1,2m}$	$a_{2m,2m}$

写出由一对下标(i,j)求 k 的转换公式。

对角线上的元素 aij (i=j) 对应

$$k=\begin{cases} 2*i-1 & i\text{为偶数} \\ 2*i-2 & i\text{为奇数} \end{cases} \Rightarrow k=\begin{cases} i+j-1 & i\text{为偶数} \\ i+j-2 & i\text{为奇数} \end{cases}$$

5.5 已知稀疏矩阵 A4×5 如下：

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 5 \\ 2 & 3 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 7 \end{bmatrix}$$

(1)用三元组表作为存储结构，绘出相应的三元组表示意图； (2)用十字链表作为存储结构，绘出相应的十字链表示意图。

(1)

i,j: 储存非零元素的行和列信息

e:非零元素的值

i, j, e

0,1,1

0,4,5

1,0,2

1,1,3

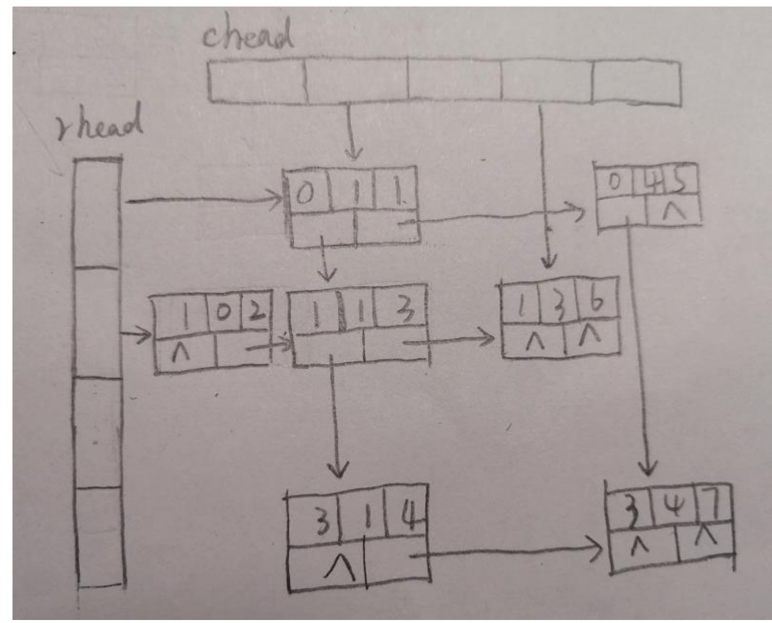
1,3,6

3,1,4

3,4,7

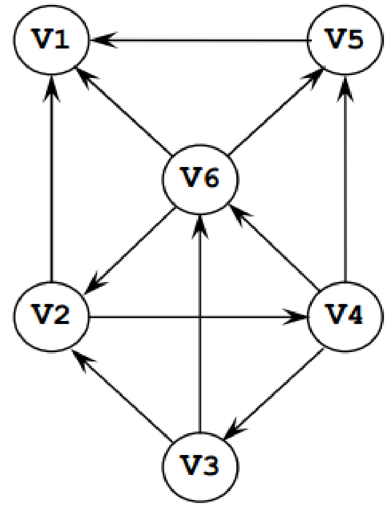
(2)

图 2 十字链表的节点结构



第七章

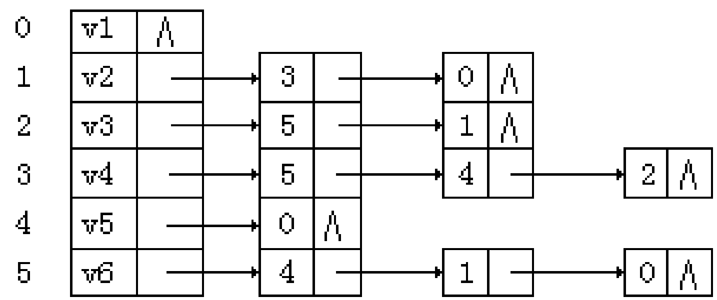
7.1已知有向图如图 7-1 所示，请给出该图的
(1)邻接矩阵示意图
(2)邻接表示意图
(3)逆邻接表
(4)所有强连通分量



非带权图使用邻接矩阵存储时，非0元代表边，0代表两点之间没有边，不是用∞表示

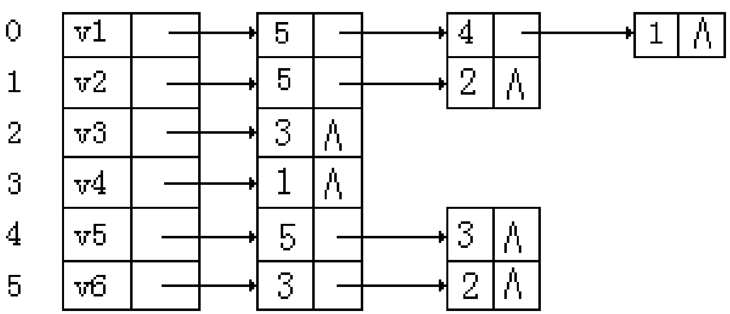
(1)
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

(2)



邻接表中顶点vi的边链表中的结点都是其出边邻接点

(3)

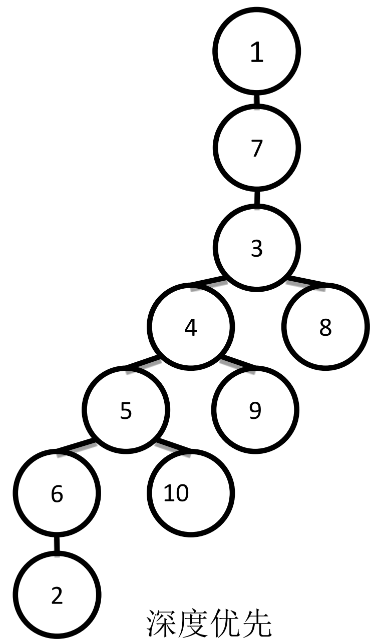


(4) 有三个强连通分量
1、5、2346

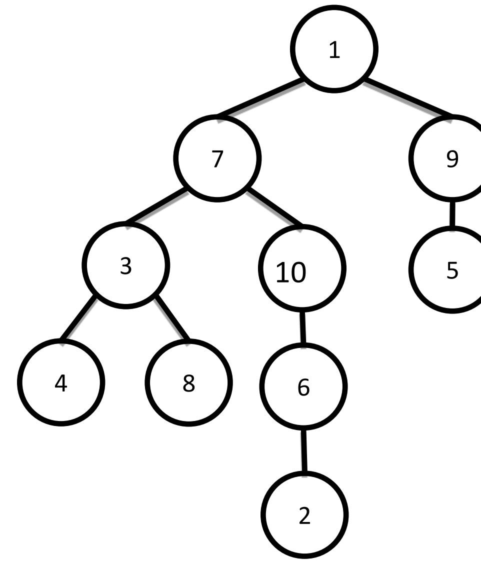
逆邻接表中顶点vi的边链表中的结点都是其入边邻接点

7.2 已知图 G 的邻接矩阵如图 7-2 所示。写出该图**从顶点 1 出发**的深度优先搜索序列和广度优先搜索序列，并画出相应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



深度优先

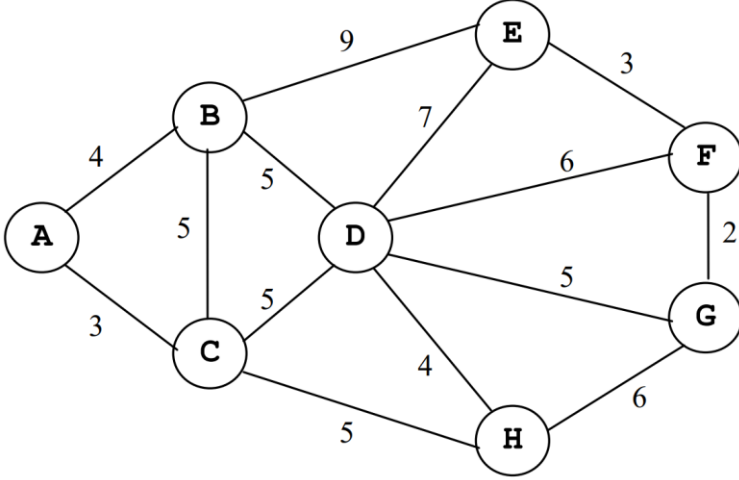


广度优先

深度优先搜索序列：v1 v7 v3 v4 v5 v6 v2 v10 v9 v8(对每一个可能的分支路径深入到不能再深入时再回溯,不重复访问)

广度优先搜索序列：v1 v7 v9 v3 v10 v5 v4 v8 v6 v2(逐层进行，首先访问原点1及其所有邻接点，再依次访问这些点中所有未曾访问的邻接点)

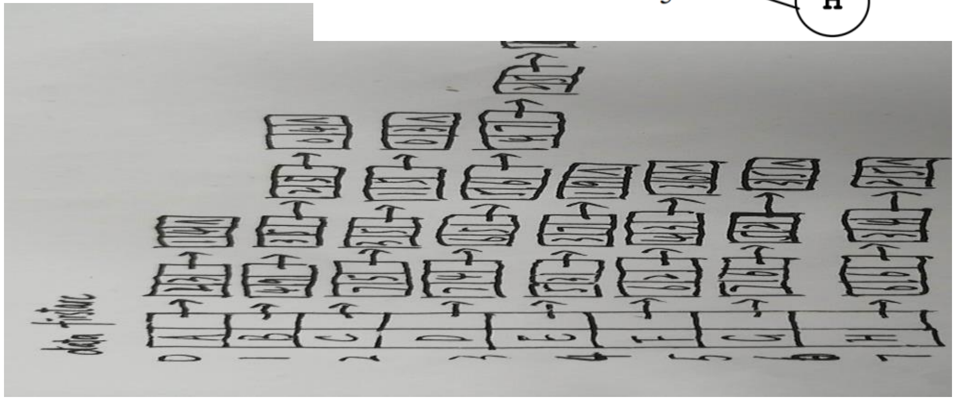
7.3无向带权图如图 7-3 所示，
(1)画出它的邻接矩阵，并按 Prim 算法求其最小生成树。
(2)画出它的邻接表，并按 Kruskal 算法求其最小生成树

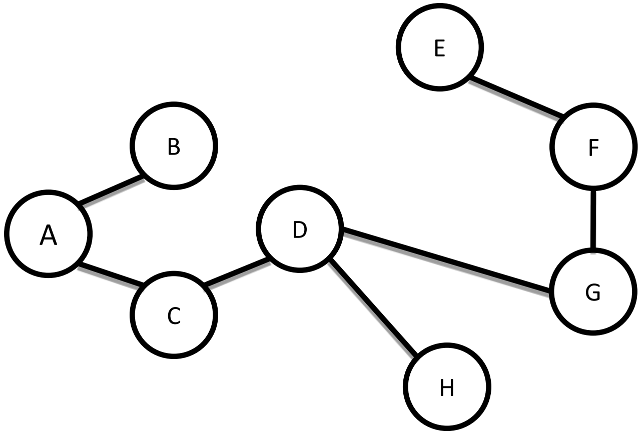


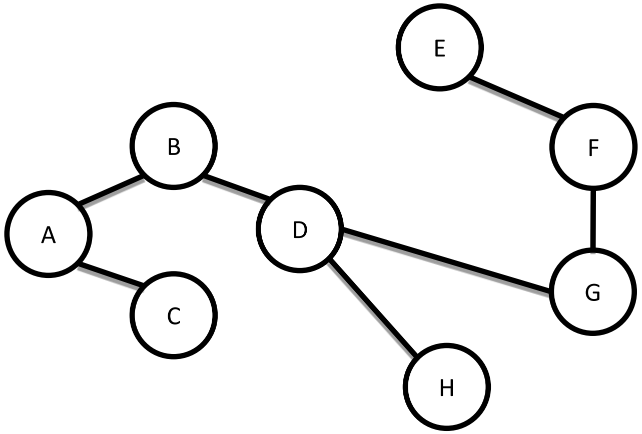
(1)
$$\begin{bmatrix} \infty & 4 & 3 & \infty & \infty & \infty & \infty & \infty \\ 4 & \infty & 5 & 5 & 9 & \infty & \infty & \infty \\ 3 & 5 & \infty & 5 & \infty & \infty & \infty & 5 \\ \infty & 5 & 5 & \infty & 7 & 6 & 5 & 4 \\ \infty & 9 & \infty & 7 & \infty & 3 & \infty & \infty \\ \infty & \infty & \infty & 6 & 3 & \infty & 2 & \infty \\ \infty & \infty & \infty & 5 & \infty & 2 & \infty & 6 \\ \infty & \infty & 5 & 4 & \infty & \infty & 6 & \infty \end{bmatrix}$$

带权图邻接矩阵用∞表示不存在(vi,vj)

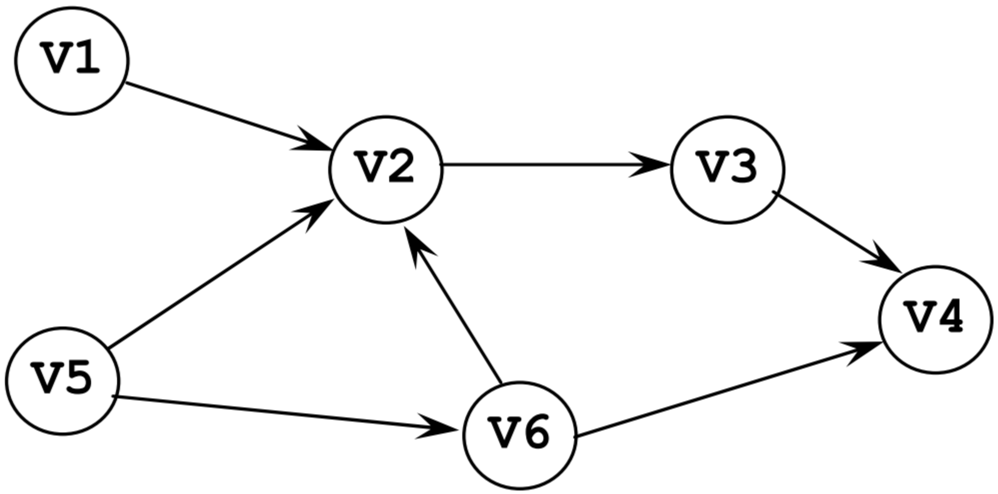
(2)







7.4 有向图如图 7-4 所示，试写出其所有可能的拓扑序列。



(V1, V5, V6, V2, V3, V4)
(V5, V1, V6, V2, V3, V4)
(V5, V6, V1, V2, V3, V4)

7.5 试利用Dijkstra算法求图7-5中顶点A到其他各顶点之间的最短路径。写出执行算法过程中，数组D、P和S各步的状态。

集合S存放已经已经找到最短路径的顶点，初态时只包含一个源点A；
D[i]存放源点到序号为i的顶点的最短路径长度；P[i]存放与D[i]相应路径上的顶点序列

初始化

S	D	P
1	0	
0	15	01
0	2	02
0	12	03
0	∞	
0	∞	
0	∞	

(1)

S	D	P
1	0	
0	15	01
1	2	02
0	12	03
0	10	024
0	6	025
0	∞	

(2)

S	D	P
1	0	
0	15	01
1	2	02
0	12	0253
0	10	024
1	6	025
0	16	0256

(3)

S	D	P
1	0	
0	15	01
1	2	02
0	11	0253
1	10	024
1	6	025
0	16	0256

(4)

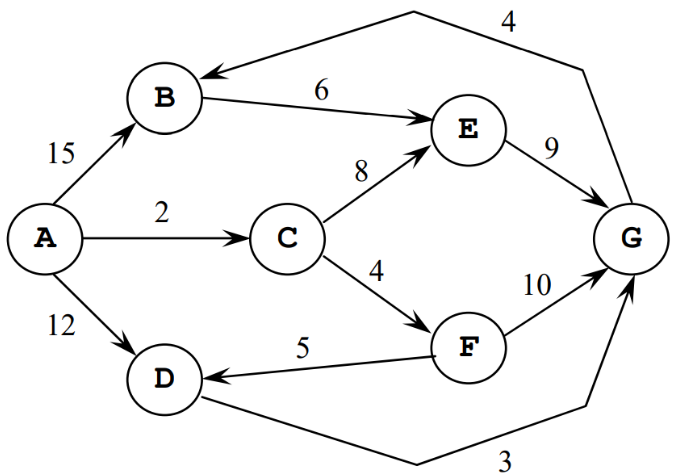
S	D	P
1	0	
0	15	01
1	2	02
1	11	0253
1	10	024
1	6	025
0	14	02536

(5)

S	D	P
1	0	
0	15	01
1	2	02
1	11	0253
1	10	024
1	6	025
1	14	02536

(6)

S	D	P
1	0	
1	15	01
1	2	02
1	11	0253
1	10	024
1	6	025
1	14	02536



7.8 设具有 n 个顶点的有向图用邻接表存储。试写出计算所有顶点入度的算法，可将每个顶点的入度值分别存入一维数组 int Indegree[n]中。

```
//思路:先把邻接表转换成逆邻接表，这样问题简单多了。
void num_Indegree(ALGraph G, ALGraph GOut)
{//设有向图有n个顶点，建逆邻接表的顶点向量。
    for (int i=1;i<=n;i++)
    {
        G[i]. vertices = GOut[i]. vertices;
        G[i].firstarc=null; }
    //邻接表转为逆邻接表
    for (i=1;i<=n;i++)
    {p= GOut[i].firstarc;//取指向邻接表的指针
        while (p!=null)
        {   j=p->adjvex;
            s=(ArcNode *)malloc(sizeof(ArcNode));
            s->adjvex=i;
            s->next=G[j].firstarc;
            G[j].firstarc=s;
            p=p->next;//下一个邻接点
        }//while
    }//for
    //统计各节点的入度
    for (i=0; i<n; i++)
    {
        p = G[i].firstarc;
        while(p ! = null)
        {
            Indegree[i]++;
            p = p->next;
        } // while
    } //for
}//function
```

7.9 假设有向图以邻接表作为存储结构。试基于图的深度优先搜索策略写一算法，判断有向图中是否存在由顶点 Vi 至顶点 Vj(i!=j)的路径。

```
//深度优先判断有向图G中顶点i到顶点j是否有路径,是则返回1,否则返回0
int visited[MAXSIZE]; //指示顶点是否在当前路径上
int exist_path_DFS(ALGraph G,int i,int j) {
   if(i==j) return 1; //i就是j
   else
   {
      visited[i]=1;
      for(p=G.vertices[i].firstarc;p;p=p->nextarc)
      {
          k=p->adjvex;
               if(!visited[k]&&exist_path(k,j))  return  1;//i下游的顶点到j有路径

      }//for
                                                              }//else
}//exist_path_DFS
```

7.10 假设有向图以邻接表作为存储结构。试基于图的广度优先搜索策略写一算法，判断有向图中是否存在由顶点 Vi 至顶点 Vj(i!=j)的路径。

```
//广度优先判断有向图G中顶点i到顶点j是否有路径,是则返回1,否则返回0
int exist_path_BFS(ALGraph G,int i,int j) {
    int visited[MAXSIZE];
    InitQueue(Q);
    EnQueue(Q,i); //结点放入队列
    while(!QueueEmpty(Q))
    {
        DeQueue(Q,u); //出队列，先入先出
        visited[u]=1;
        for(p=G.vertices[i].firstarc;p;p=p->nextarc)
        {k=p->adjvex;
            if(k==j) return 1;
            if(!visited[k]) EnQueue(Q,k);
        }//for
    }//while
    return 0;
}//exist_path_BFS
```

第十章 内部排序

10.1以关键字序列（5，1，6，0，9，2，8，3，7，4）为例，手工执行下列排序算法，写出每一趟排序结束时关键字序列状态

(1) 直接插入排序　　(2) 希尔排序（取增量为5，3，1）
(3) 快速排序　　(4) 冒泡排序
(5) 归并排序　　(6) 堆排序

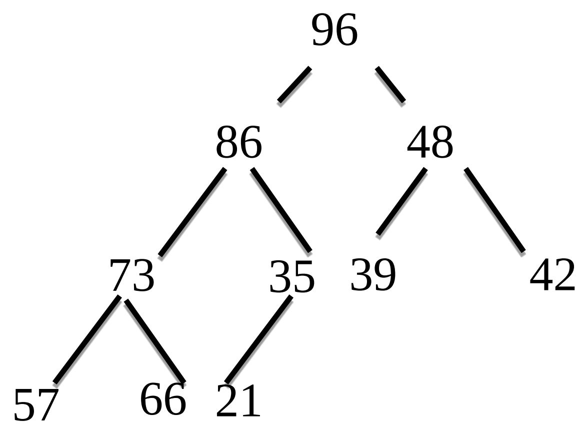
(1) (1,5,6,0,9,2,8,3,7,4)
(2) (2,1,3,0,4,5,8,6,7,9)
(3) (4,1,3,0,2,5,8,9,7,6)
(4) (1,5,0,6,2,8,3,7,4,9)
(5) (1,5,0,6,2,9,3,8,4,7)
(6) (8,7,6,5,4,2,1,3,0,9)

10.3判别以下序列是否为堆（小顶堆或大顶堆），若不是，则吧它调整为堆

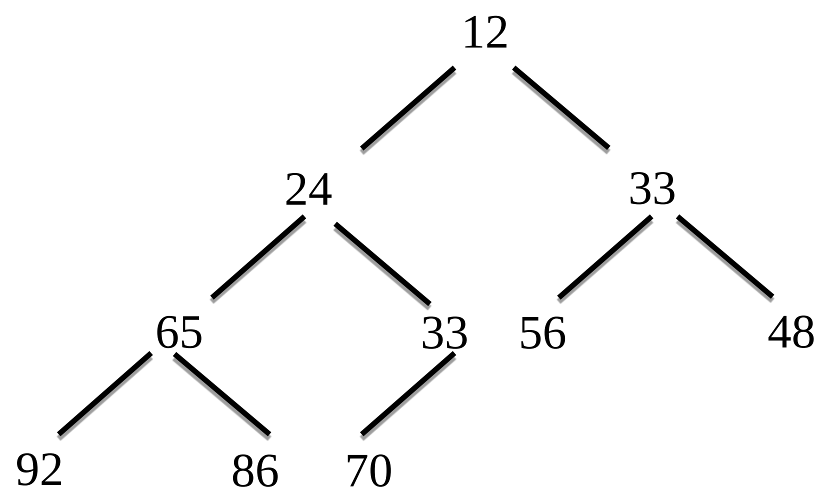
(1) (96,86,48,73,35,39,42,57,66,21);

(2) (12,70,33,65,24,56,48,92,86,33);

(1) 是大顶堆



(2) 不是大顶堆，也不是小顶堆。调整后：



(12,24,33,65,33,56,48,92,86,70)

10.5试以单链表为存储结构，实现简单选择排序算法。

```
void LinkedList_Select_Sort(LinkedList&L)//单链表上的简单选择排序算法
{
    for(p=L;p;p=p->next)//p不为空指针
    {
        min=p;
        for(q=min->next;q;q=q->next)
            if(min->data > q->data) min=q;
        if(min!=p){t=p->data;p->data=min->data;min->data=t;} //交换两个结点数据域
    }
}// LinkedList_Select_Sort
```