5.2 单元测试

一、单项选择题

1~5 DADCA　6~10 DAACC　11~15 CBDDA　16~20 BCDBA　21 B

二、填空题

1. 0 ；$\frac{n(n-1)}{2}$；n-1　2. 2(n-1),　n(n-1)　3. 出度；入度　4. n；2e　5. 先序；层序
6. AEDCB；BCADE　7. n；n(n-1)　8. n　9. 稠密图；稀疏图　10. 活动；顶点间优先关系
11. 无环图　12. 事件；活动

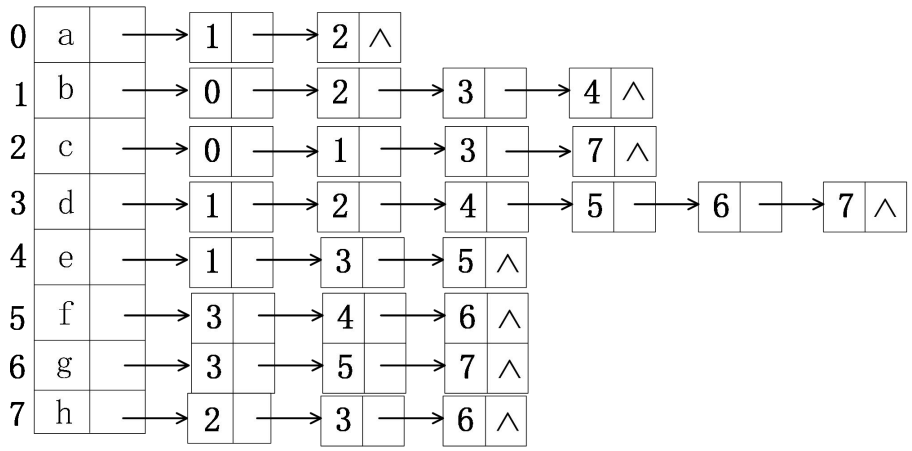
三、应用题

1.【解析】

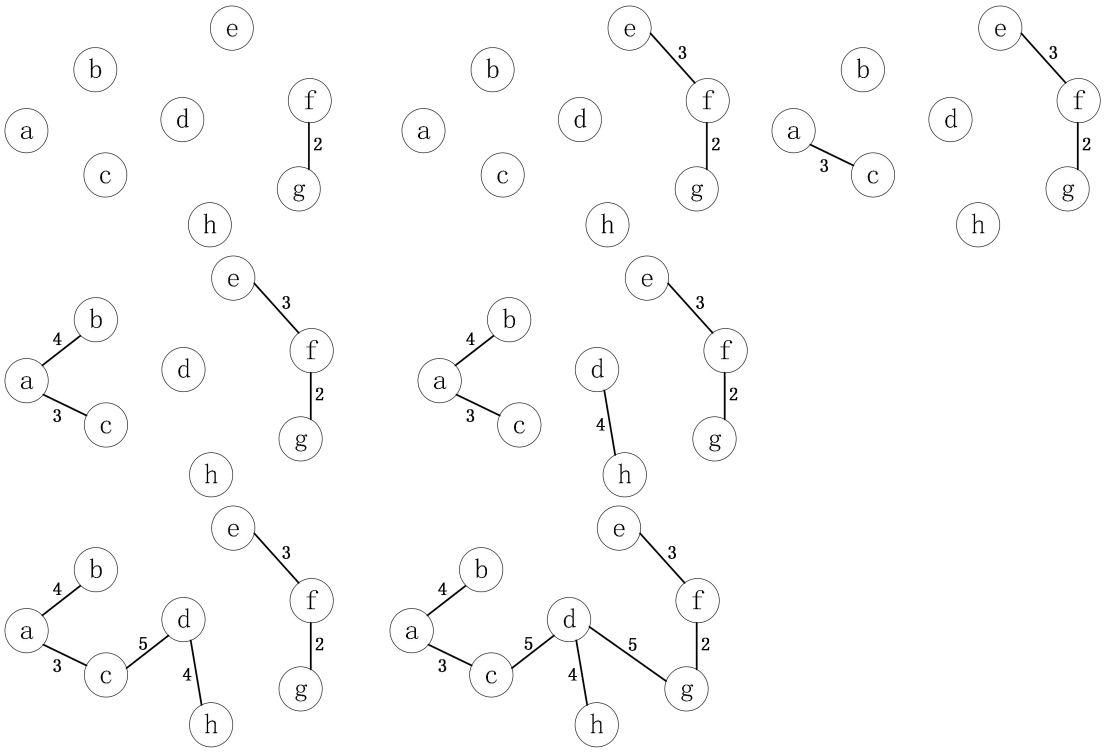
（1）

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

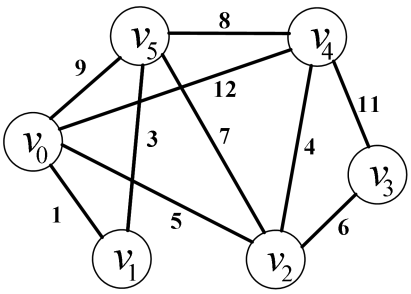
（2）



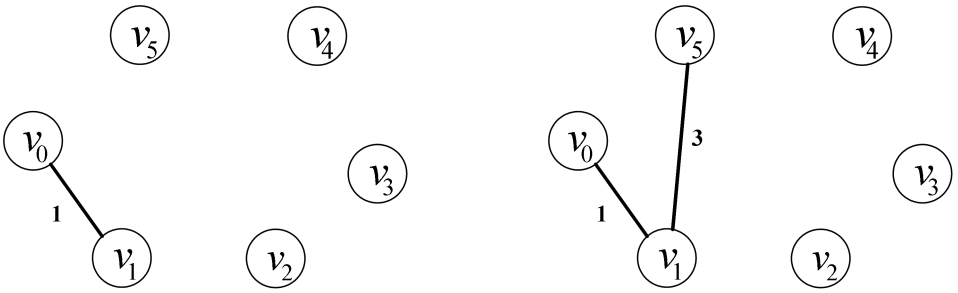
（3）

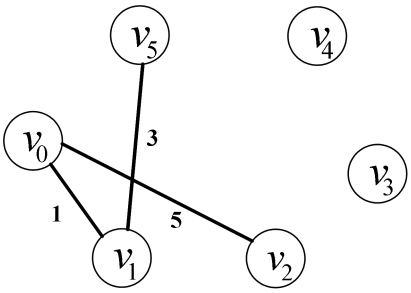


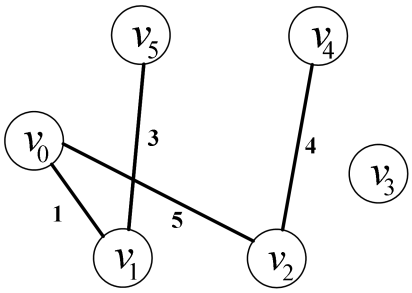
2.【解析】带权无向图 G 为：

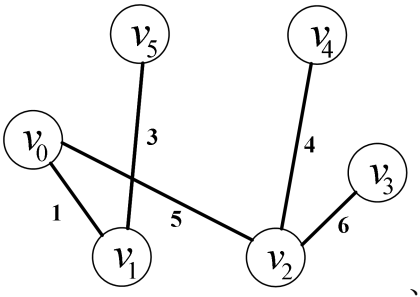


过程：







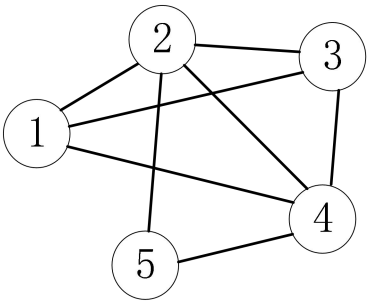


3.【解析】

（与上题类似）略

4.【解析】

（1）



（2）略

（3）略

5.【解析】

（1）43

（2）e(i)表示最早开始时间，l(i)表示最迟开始时间

	1--->2	1--->3	3--->2	2--->4	2--->5	3--->5	4--->6	5--->6
e(i)	0	0	15	19	19	15	29	38
l(i)	17	0	15	27	19	23	37	38

（3）由（2）的表可知，关键活动为：（1,3）（3,2）（2,5）（5,6）

6.【解析】

过程	=> 从 0 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 4 （第 1 步）		选 1 （第 2 步）		选 2 （第 3 步）		选 3 （第 4 步）	
终点	D	P	D	P	D	P	D	P	D	P
1	6	0	5	4	5	4	5	4	5	4
2	9	0	9	0	6	1	6	1	6	1
3	∞		9	4	7	1	7	1	7	1
4	2	0	2	0	2	0	2	0	2	0

由上表可知：

0 到 1 的最短路径为 0--->4--->1，路径长度为 5；

0 到 2 的最短路径为 0--->4--->1--->2，路径长度为 6；

0 到 3 的最短路径为 0--->4--->1--->3，路径长度为 7；

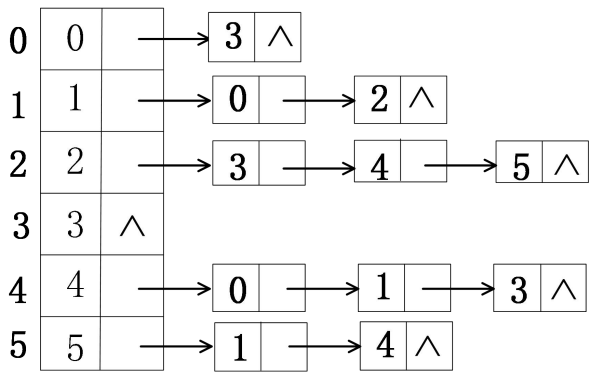
0 到 4 的最短路径为 0--->4，路径长度为 2。

6.【解析】

（1）

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

（2）



（3）略

（4）0，3，1，2，4，5

（5）2，3，4，5，0，1

四、算法设计题

1.//统计出度为 2 的顶点个数 num

```
int count_degree2(frontnodetype G[], int n)
{
    int k ,degree,num=0;
    node *p ;

    for (k=0; k<n; k++)
    {
        p=G[k].next ; //p 指向第 k 个单链表
        while (p!=NULL)        /*   顶点出度统计   */
        {
            degree++ ;
            if(degree > 2)
                break;
            p=p->next ;
        }
        if(degree == 2)
            num++;
    }
    return num;
}
```

2.

```
int countWeight(MGraph DG)   //统计所有边的权值之和
    {   int i,j,countWeight;

       for(i = 0;i < DG.Nv; i++)
          for(j = 0;j < DG.Nv; j++)
           if(DG.G[i][j] >0 && DG.G[i][j] < INFINITY )   //INFINITY 表示无穷大
                countWeight += DG.G[i][j];
            return countWeight;
    }
```

3.

```
int* DegreeOut(MGraph DG)   //统计各顶点的出度
{   int i,j;
    int outdegree[DG.Nv] = {0};
    for(i = 0;i < DG.Nv; i++)
        for(j = 0;j < DG.Nv; j++)
            if(DG.G[i][j] >0 && DG.G[i][j] < INFINITY )   //INFINITY 表示无穷大
```

```
                    outdegree[i]++;
    return outdegree;
}

int* DegreeIn(MGraph DG)   //统计各顶点的入度
{   int i,j;
    int indegree[DG.Nv] = {0};
    for(i = 0;i < DG.Nv; i++)
        for(j = 0;j < DG.Nv; j++)
            if(G[j][i] >0 && G[j][i] < INFINITY )   //INFINITY 表示无穷大
                indegree[i]++;
    return indegree;
}
```

4.

```
//统计出度,存在一维数组 outdegree 中
int* count_outdegree(frontnodetype G[], int n)
{
    int k ; node *p ;
    int outdegree[n]={0}; //初始化为 0

    for (k=0; k< n; k++)
    {
        p=G[k].next ; //p 指向第 k 个单链表
        while (p!=NULL)        /*   顶点入度统计   */
        {
            outdegree[k]++ ;
            p=p->next ;
        }
    }
    return outdegree;
}
```

5. 邻接表-->邻接矩阵

```
MGraph * FrontNodeToMGraph(frontnodetype G[], int n)
{
    int k ,degree,num=0,j=0;
    node *p ;
    MGraph *DG;
    DG->Nv = n;
    for(k = 0;k < DG->Nv; k++)
        for(j = 0;j < DG->Nv; j++)
            DG->G[k][j] = 0; //初始化邻接矩阵
    for (k=0; k<n; k++)
```

```
    {     j=0;   //列号
          p=G[k].next ; //p 指向第 k 个单链表
          while (p!=NULL) //遍历第 k 个单链表的每个节点
          {
               DG->G[k][j] = 1;
               j++;
               p=p->next ;
          }
    }
    return DG;
}
```

6.

```
frontnodetype* MGraphToFrontNode(MGraph DG)
{     int i,j;
      frontnodetype G[DG.Nv]; //定义图的邻接表
      int outdegree[DG.Nv] = {0};
      for(i = 0;i < DG.Nv; i++)
      {
           G[i].next = NULL;//第 i 个单链表
           for(j = 0;j < DG.Nv; j++)
              if(DG.G[i][j] == 1 ){
                 node *p ;
                 p = (node *)malloc(sizeof(node)) ;
                 p->adjvex = j;
                 p->next = G[i].next;
                 G[i].next = p; //头插法
              }
      }
      return G;
}
```