第 5 章 图结构

1. 选择题

（1）B （2）A（3）B （4）C （5）BD （6）D （7）A （8）A （9）C （10）B

2. 判断题

（1）X（2）√（3）√ （4）X （5）X （6）√ （7）√（8）X（9）X （10）√
（11）X（12）√（13）√ （14）X （15）X

3. 简答题

（1）对于如图 5-37 所示的有向图，试给出：

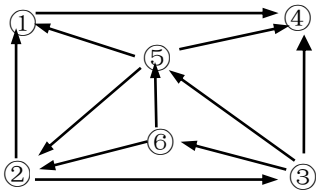
1）每个顶点的入度和出度；

2）邻接矩阵；

3）邻接表；

4）逆邻接表；

5）强连通分量。



（图 5-37）

【解答】

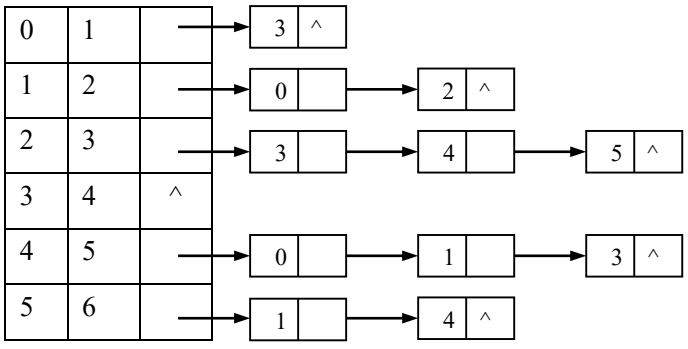
1）每个顶点的入度和出度：

顶点 1（2,1）、顶点 2（2,2）、顶点 3（1,3）、
顶点 4（3,0）、顶点 5（2,3）、顶点 6（1,2）。

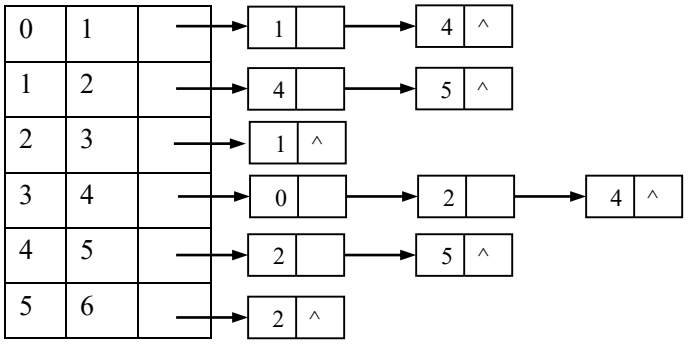
2）邻接矩阵：

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

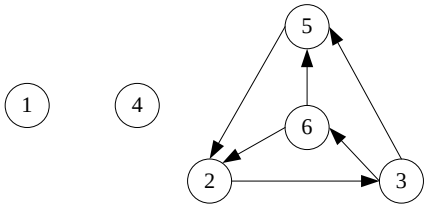
3）邻接表：



4）逆邻接表：



5）强连通分量：



（2）设无向图 G 如图 5-38 所示，试给出：

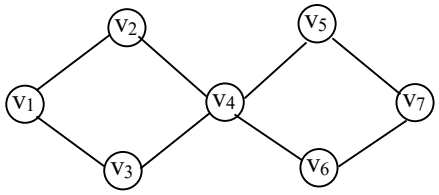
1）该图的邻接矩阵；

2）该图的邻接表；

3）该图的多重邻接表；

4）从 v_1 出发的“深度优先”遍历序列；

5）从 v_1 出发的“广度优先”遍历序列。



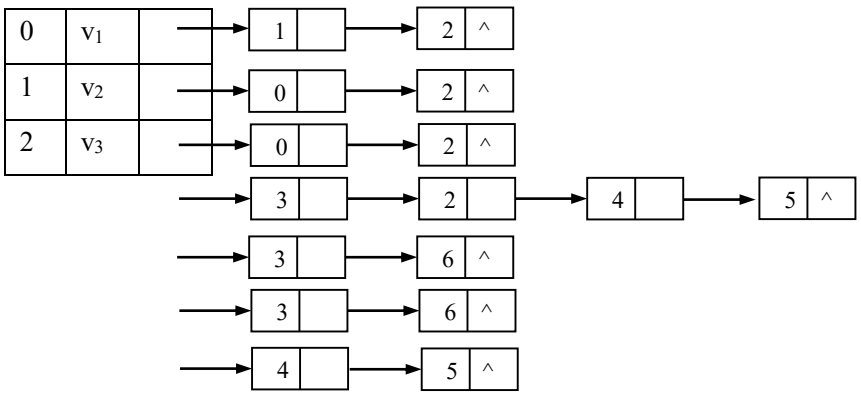
（图 5-38）

【解答】

1） 该图的邻接矩阵：

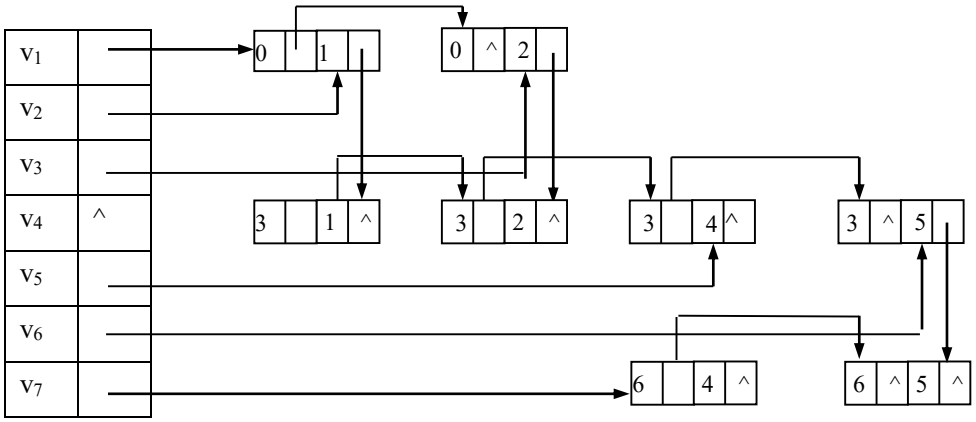
$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

2）该图的邻接表：



3	v_4	
4	v_5	
5	v_6	
6	v_7	

3）该图的多重邻接表：



4）从 v_1 出发的“深度优先”遍历序列：$v_1v_2v_4v_3v_5v_7v_6$

5）从 v_1 出发的“广度优先”遍历序列：$v_1v_2v_3v_4v_5v_6v_7$

（3）用邻接矩阵表示图时，矩阵元素的个数与顶点个数是否相关？与边的条数是否有关？

【解答】

设图的顶点个数为 n($n \geqslant 0$)，则邻接矩阵元素个数为 n^2，即顶点个数的平方。矩阵元素的个数与图的边数无关。

（4）设有向图 G 如图 5-39 所示，试画出图 G 的十字链表结构，并写出图 G 的两个拓扑序列。

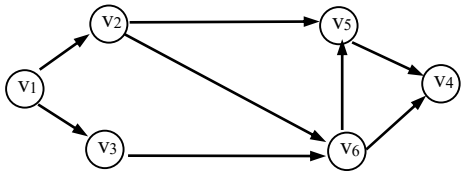
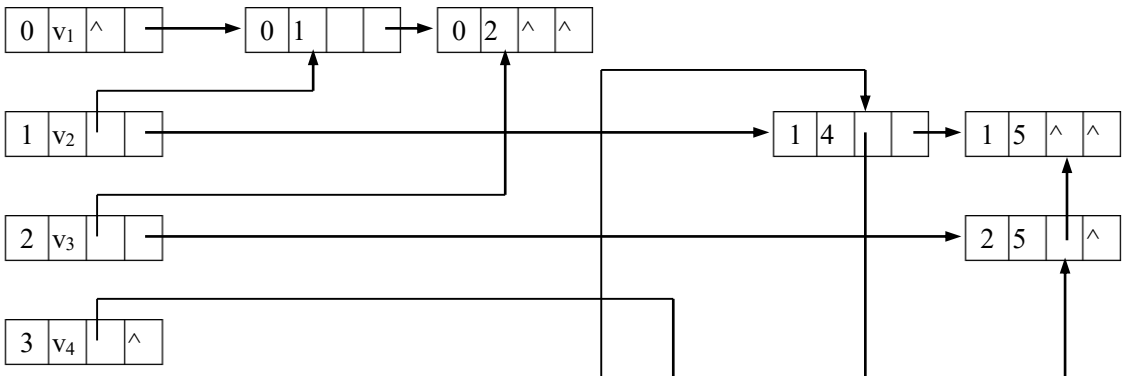


图 5-39

【解答】

1）G 的十字链表结构：



2）G 的两个拓扑序列：$v_1v_2v_3v_6v_5v_4$; v_1v_3 $v_2v_6v_5v_4$

（5）如图 5-40 所示 AOE 网：

1）列出各事件的最早、最迟发生时间；

2）列出各活动的最早、最迟发生时间；

3）找出该 AOE 网中的关键路径，并回答完成该工程需要的最短时间。

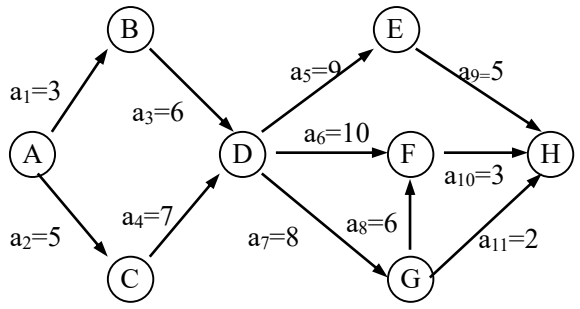


图 5-40

1）ve(A)=0　　ve(B)= ve(A)+3=3　　ve(C)= ve(A)+5=5

ve(D)=max(ve(B)+6, ve(C)+7)=12　　ve(E)= ve(D)+9=21　　ve(G)=ve(D)+8=20

ve(F)= max(ve(D)+10, ve(G)+6)=26　　ve(H)= max(ve(E)+E, ve(G)+2, ve(F)+3)=29

vl(H)=29　　vl(E)= vl(H)–5=24　　vl(F)= vl(H)–3=26

vl(G)=min(vl(H)–2, vl(F)–6)=20　　vl(D)=min(vl(E)–9, vl(F)–10, vl(G)–8)=12

vl(B)=vl(D)–6=6　　vl(C)=vl(D)–7=5　　vl(A)=min(vl(B)–3, vl(C)–5)= 0

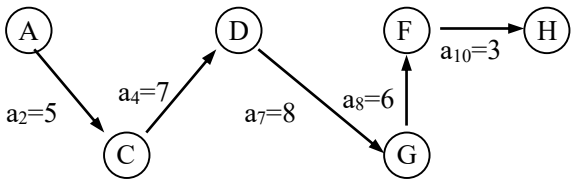
2）　e(a_1)=0　　e(a_2)=0　　e(a_3)=3　　e(a_4)=5　　e(a_5)=12　　e(a_6)=12

e(a_7)=12　　e(a_8)=20　　e(a_9)=21　　e(a_{10})=26　　e(a_{11})=20

l(a_1)=3　　l(a_2)=0　　l(a_3)=6　　l(a_4)=5　　l(a_5)=15　　l(a_6)=16

(a_7)=12　　l(a_8)=20　　l(a_9)=24　　l(a_{10})=26　　l(a_{11})=27

3）关键路径如下图，完成该工程需要的最短时间：29



2. 算法设计题

（1）试以邻接矩阵为存储结构实现图的基本操作：InsertVex(G,v)、InsertArc(G,v,w)、DeleteVex(G,v)和 DeleteArc(G,v,w)。

【解答】算法如下：

```
void InsertVex (MMGraph *G,int v)          /*在图 G 中插入一个顶点做为第 v 个顶点*/
  {int i,j;
                                  G->n++;
   for(i=G->n;i>v;i--)                     /*腾出第 v 个顶点值的位置*/
       G->vexs[i]=G->vexs[i-1];
   G->vexs[v]=getchar();                   /*插入第 v 个顶点的值*/
   for(i=G->n;i>v;i--)
     {for(j=G->n;j>v;j--)                 /*第 v 行以后和第 v 列以后的元素后移一行一列*/
      G->edges[i][j]=G->edges[i-1][j-1];
      G->edges[i][v]=0;                    /*将第 v 行以后行的 v 列赋 0*/
      for(j=v-1;j>=0;j--)                  /*第 v 行以后第 v 列以前的元素后移一行*/
         G->edges[i][j]=G->edges[i-1][j];
     }
   for(j=G->n;j>=0;j--) G->edges[k][j]=0;  /*将第 v 行的元素赋 0*/
   for(i=v-1;i>=0;j--)                     /*第 v 行以前第 v 列以后的元素后移一列*/
     {for(j=G->n;j>k;j--)
      G->edges[i][j]=G->edges[i][j-1];
      G->edges[i][v]=0                     /*将第 v 行以前的 v 列赋 0*/
     }
  }
void InsertArc (MMGraph *G, int v, int w)  /*在图 G 中序号为 v,w 的顶点之间插入一条边*/
  {G->edges[v][w]=1;
   G->edges[w]p[v]=1;
   G->e++;
  }
void DeleteVex (MMGraph *G, int v)         /*在邻接矩阵表示的图 G 中删除序号为 v 的点*/
  {int i,j;
   G->n--;
   for(i=v;i<=G->n;i++)
       G->vexs[i]=G->vexs[i+1];            /*删除顶点值*/
   for(i=0;i<v;i++)                        /*k 行以前 k 列以后前移一列*/
   for(j=v;j<=n;j++)
       G->edges[i][j]=G->edges[i][j＋1];
   for(i=v;i<=G->n;i++)                    /*k 行以后的前移一行*/
      {for(j=0;j<k;j++)
          G->edges[i][j]=G->edges[i+1][j];
       for(j=v;j<=n;j++)                   /*k 行以后 k 列以后前移一列*/
          G->edges[i][j]=G->edges[i][j＋1];
      }
```

```
}
void DeleteArc (MMGraph *G,int v,int w)   /*在图 G 中删除序号为 v,w 的顶点的边*/
 {G->edges[v][w]=0;
                                  G->edges[w][v]=0;
  G->e--;
 }
```

（2）试以邻接表为存储结构实现算法设计题（1）中所列图的基本操作。

【解答】

在这些操作中图的顶点数和边数发生了变化，定义图的类型如下：

```
typedef struct
  {VertexNode adjlist[MAX];          /*顶点结点向量（MAX 为最大顶点数）*/
   int n, e;                         /*顶点数和边数*/
  }AAlGraph;
void InsertVex (AALGraph *G,int v)        /*在图 G 中插入一个顶点，做为第 v 个顶点*/
  {int i;
   G->n++;
   for(i=G->n;i>v;i--)                  /*将原图中从 n 到 v 的结点后移一个位置*/
       {G->adjlist[i].vertex=G->adjlist[i-1].vertex;
        G->adjlist[i].firstedge=G->adjlist[i-1].Firstedge;
       }
   G->adjlist[v].vertex=getchar( );     /*将插入点放入*/
   G->adjlist[v].firstedge=NULL;
  }
void InsertArc (AALGraph *G, int v, int w)  /*在图 G 中序号为 v,w 的顶点之间插入一条边*/
  {EdgeNode *E;
   EdgeNode *sm=(EdgeNode *)malloc(sizeof(EdgeNode));
   EdgeNode *sn=(EdgeNode *)malloc(sizeof(EdgeNode));
   sm->adjvex=w;                        /*将边结点插入到边表首部*/
   sm->next=G->adjlist[m].firstedge;
   sn->adjvex=v;
   sn->next= G->adjlist[m].firstedge;
   G->adjlist[v].firstedge＝sm；
   G->adjlist[v].firstedge＝sn；
   G->e++;
  }
void DeleteVex (AALGraph *G, int v)          /*在邻接表表示的图 G 中删除序号为 v 的点*/
  {int i;
   G->n--;
   EdgeNode *e,*s,*p,*q;
   for(q=e=G->adjlist[v].firstedge;e;q=e,e=e->next,free(q))
     {s=G->adjlist[e->adjvex].firstedge;
      p=s;
      for(;s;p=s, s=s->next)
      if(s->adjvex==v)
```

```
            {if(p!=s)    p->next=s->next;
             else    (G->adjlist[e->adjvex].firstedge=s->next;)}
             if(s)    free(s);
             break;
            }
         }
      for(i=v;i<G->n;i++)                          /*删除顶点值*/
        {G->adjlist[i].vertex=G->adjlist[i+1].vextex;
         G->adjlist[i].first[i].firstedge=G->adjlist[i+1].firstedge;
        }
      }
    void DeleteArc(AALGraph *G,int v,int w)        /*在图 G 中删除序号为 v,w 的顶点之的边*/
      {EdgeNode *s,*p;
       s=G->adjlist[v].firstedge;
       p=s;
       for(;s;s=s->next);                          /*在与 m 邻接的点中找 n*/
         {if(s->adjvex==w)                         /*若找到邻接点 n，则将该边从边表中脱出*/
             {if(p!=s)    p->next=s->next;
              else    G->adjlist[v].firstedge=s->next;
             }
          if(s) free(s);                           /*释放要删除的边结点*/
                                              }
       s=G->adjlist[w].firstedge;p=s;
       for(;s;p=s,s=s->next)                       /*在与 n 邻接的点中找 m*/
         {if(s->adjvex==v)                         /*若找到邻接点 m，则将该边从边表中脱出*/
            {if(p!=s)    p->next=s->next;
             else      G->adjlist[w].firstedge=s->next; }
          if(s)    free(s);                        /*释放要删除的边结点*/
         }
       G->e--;
      }
```

（3）试以十字链表为存储结构实现算法设计题（1）中所列图的基本操作。

算法略。

（4）试以邻接多重表为存储结构实现算法设计题（1）中所列图的基本操作。

算法略。

（5）对于含有 n 个顶点的有向图，编写算法由其邻接表构造相应的逆邻接表。

【解答】

```
Void InvertAdjList(ALGraph G, ALGraph *H)    /*由有向图的邻接表 G 建立其逆邻接表 H*/
  {for (i=1;i<=n;i++)                        /*设有向图有 n 个顶点，建逆邻接表的顶点向量*/
     {H[i]->vertex=G.adjlist[i].vertex; H->firstedge=NULL;}
   for (i=0; i<n; i++)                       /*邻接表转为逆邻接表*/
     {p= G.adjlist[i].firstedge;             /*取指向邻接表的指针*/
      while (p!=null)
                                        {j=p->adjvex;
```

```
            s=(ArcNode *)malloc(sizeof(EdgeNode));  /*申请边结点空间*/
            s->adjvex=i;
            s->next= H[j]->firstedge;
            H[j]->firstedge=s;
            p=p->next;                                  /*下一个邻接点*/
           }
        }
     }
```

（6）试写一算法，由图的邻接表存储得到图的十字链表存储。

【解答】设图的结点数和边数分别为 N 和 E。十字链表存储的图的类型定义为：

```
typedef struct
 {VexNode    vlist[N];
 }OLGraph;
void CreateOG(ALGraph *H, OLGraph *G)        /*由邻接表 H 得到十字链表 G*/
   {for(i=0;i<N;i++)
       {G->vlist[i].vertex=H->adjlist[i].vertex;
        G->vlist[i].firstin= G->xlist[i].firstout=NULL;
       }                                              /*初始化十字链表*/
    for(i=0;i<N;i++)
      {p=H->adjlist[i].firstedge;
       while(p)
          {s=new ArcBox;
           j=p->adjvex;
           s->tailvex=i;
           s->headvex=j;
           s->tlink=G->xlist[i].firstout;
           G->vlist[i].firstout=s;
            s->hlink=G->xlist[j].firstin;
            G->vlist[j].firstin=s;
            p=p->next;
          }                                             /*生成十字链表*/
       }
   }
```

（7）写一算法，由依次输入图的顶点数目、边的数目、各顶点的信息和各条边的信息建立无向图的邻接多重表。

【解答】以邻接多重表存储的图的类型定义为：

```
#define MAX_VERTEX_NUM    20
 typedef struct
 {VexBox adjmulist[MAX_VERTEX_NUM];
  int vernum,edgenum;
 }AMLGraph;
 void CreateAMLGraph(AMLGraph *G)
   {scanf(G->vexnum, G->edgenum);
    for(i=0;i<vexnum;i++)                          /*读入顶点集，初始化邻接多重表*/
```

```
    {scanf(G->adjmulist[i].data);
     G-> adjmulist[i].firstedge=NULL;}
  for(k=0;k<G->edgenum;k++)                    /*读入边集，生成邻接多重表*/
     {scanf(i, j); s=new EBox;
      scanf(s->info);
      s->ivex=i;
      s->jvex=j;
      s->ilink=G->adjmulist[i].firstedge;
      G->adjmulist[i].firstedge=s;
      s->jlink=G->adjmulist[j].firstedge;
      G->adjmulist[j].firstedge=s;
     }
  }
```

（8）试写一个算法，判别以邻接表方式存储的有向图中是否存在由顶点 v_i 到顶点 v_j 的路径（i≠j）。假设分别基于下述策略：

1）图的深度优先搜索；

2）图的广度优先搜索。

【提示】 在有向图中，判断顶点 v_i 和顶点 v_j 间是否有路径，可采用遍历的方法，从顶点 v_i 出发，不论是深度优先遍历（dfs）还是广度优先遍历（bfs），在未退出 dfs 或 bfs 前,若访问到 V_j，则说明有通路，否则无通路。设一全程变量 flag。初始化为 0，若有通路，则 flag=1。

1）基于图的深度优先搜索算法：

```
int visited[]=0;                          /*全局变量，访问数组初始化*/
int DFS(ALGraph G, int vi)
    /*以邻接表为存储结构的有向图 G，判断顶点 vi 到 vj 是否有通路,返回 1 或 0 表示有或无*/
    /*若顶点 vi 和 vj 不是编号，必须先用顶点定位函数,查出其在邻接表顶点向量中的下标 i 和 j*/
  {visited[vi]=1;                         /*visited 是访问数组，设顶点的信息就是顶点编号*/
   p=G.adjlist[vi].firstedge;             /*第一个邻接点*/
   while(p!=NULL)
      {if(vj==j)
         {flag=1;
           Return(1);}                   /*vi 和 vj 有通路*/
       if(visited[j]==0) DFS(G,p->adjvex);
         p=p->next;
       }
   if (!flag) return(0);
  }
```

2）基于图的广度优先搜索算法：思想是用一个栈存放遍历的顶点，遇到顶点 v_j 时输出路径。

```
  void BFS(ALGraph G, int i)
/*有向图 G 的顶点 vi（编号 i）和顶点 vj（编号 j）间是否有路径，如有，则输出。*/
    {int *n;
     Init_Stack(S);                                  /*S 是存放顶点编号的栈*/
     visited[i]=1;                                   /*visited 数组在进入 BFS 前已初始化*/
     Push_Stack(S,i);
     p=G.adjlist[i].firstedge;                       /*求第一个邻接点*/
```

```
while (p)
   {if(p->adjvex==j)
       {Push_Stack(S,j);
        printf("顶点 vi 和 vj 的路径为：\n");
            for(i=1;i<=top; i++)
               {Pop_Stack(S,n);
                printf("%4d",*n);
                exit(0);}
               }
    else if(visited[p->adjvex]==0)
               {BFS(G, p->adjvex);
                Pop_Stack(S,x);
                p=p->next;}
   }
}
```

（9）试修改 Prim 算法，使之能在邻接表存储结构上实现求图的最小生成森林，并分析其时间复杂度（森林的存储结构为孩子－兄弟链表）。

【提示】对于非连通的带权图，采用 Prim 算法会产生最小生成森林，设计算法时要考虑对非连通图的各个连通分支分别求最小生成树，算法由三重循环来实现。

算法略。

（10）以邻接表作存储结构实现求从源点到其余各顶点的最短路径的 Dijkstra 算法。

【解答】设图的顶点数为 N，算法如下：

```
void Shortest_Dijkstra(ALGraph G, vertype v0)
/*在带权邻接表 G 中，用 Dijkstra 方法求从顶点 v0 到其它顶点的最短路径*/
   {int d[],s[];                    /*d 数组存放最短路径，s 数组存顶点是否找到最短路径的信息*/
    for(i=1;i<=N;i++)
        {d[i]=INFINITY;
         s[i]=0; }                  /*初始化，INFINITY 是机器中最大的数，代表无穷大*/
         s[v0]=1;
         p=G.adjlist[v0].firstedge;
         while(p)
           {d[p->adjvex]=p->info; /*info 是边对应的权值*/
               p=p->next;}          /*顶点的最短路径赋初值*/
         for(i=1;i<N;i++)           /*在尚未确定最短路径的顶点集中选有最短路径的顶点 u*/
           {mindis=INFINITY;
            for(j=1;j<=N;j++)
                if(s[j]==0&&d[j]<mindis)
                   {u=j;
                    mindis=d [j];}
            s[u]=1;                 /*顶点 u 已找到最短路径*/
            p=G.adjlist[u].firstedge;
            while(p)                /*修改从 v0 到其它顶点的最短路径*/
               {j=p->adjvex;
                if(s[j]==0&&d[j]>d[u]+p->info)
```

```
                    d[j]=d[u]+p->info;
                p=p->next;}
            }
}
```