中国风

数据结构实验课

实验4.1内容算法提示

赋权图最短路径

题目：给一个赋权图（无向图），求某个结点到其余所有结点的最短路径的长度。
总体思路：主要以起始点为中心向外层层扩展，边扩展边更新起点到各个节点的最短路径，直到扩展到终点为止。

最短路径算法：
（1）单源最短路径：定起始顶点s，找出从s到图中其它各顶点的最短路径。
Dijkstra算法：解决所有边的权为非负的单源最短路径问题。
SPFA算法：可以解决权值可以为负数 的单源最短路径问题，但其时间复杂度过高
（2）全源最短路径：找出连接图中所有顶点到其他顶点的最短路径。
Floyd算法：可以检测图中的负环并可以解决不包括负环的图中的全源最短路径问题。
Johnson算法：也是决不包含负环的图的全源最短路径问题，但是其算法效率更高。

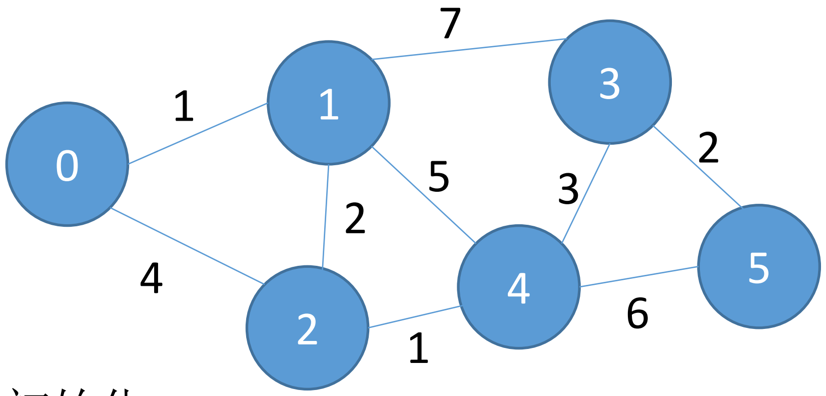
实验4.1 SPFA算法提示

SPFA算法介绍： SPFA算法是求解单源最短路径问题的一种算法，由理查德·贝尔曼（Richard Bellman） 和莱斯特·福特创立的。有时候这种算法也被称为 Moore-Bellman-Ford 算法，因为 Edward F. Moore 也为这个算法的发展做出了贡献。它的原理是对图进行V-1次松弛操作，得到所有可能的最短路径。

其优于迪科斯彻算法的方面是边的权值可以为负数、实现简单，缺点是时间复杂度过高，高达 O(VE)。但算法可以进行若干种优化，提高了效率。

算法思想： 用数组dist记录每个结点的最短路径估计值，用邻接表或邻接矩阵来存储图G。采取动态逼近法：设立一个先进先出的队列用来保存待优化的结点，优化时每次取出队首结点u，并且用u点当前的最短路径估计值对离开u点所指向的结点v进行松弛操作，dist[v]=min(dist[u]+w[u][v],dist[v]),如果v点的最短路径估计值有所调整，且v点不在当前的队列中，就将v点放入队尾。这样不断从队列中取出结点来进行松弛操作，直至队列空为止。

实验4.1 SPFA算法提示




```
AM = { {0, 1, 4, 10000, 10000, 10000},
       {1, 0, 2, 7, 5, 10000},
       {4, 2, 0, 10000, 1, 10000},
       {10000, 7, 10000, 0, 3, 2},
       {10000, 5, 1, 3, 0, 6},
       {10000, 10000, 10000, 2, 6, 0}}
```

1初始化

Node	0	1	2	3	4	5
dist[i]	0	∞	∞	∞	∞	∞

queue

0

2.Queue不为空，循环，0出队，0可到达1，2节点，松弛成功，1，2不在队列中，入队

Node	0	1	2	3	4	5
dist[i]	0	1	4	∞	∞	∞

queue

2
1

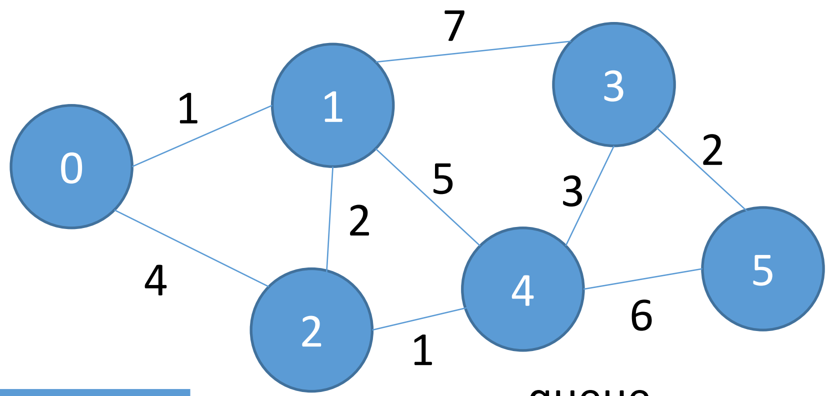
3.Queue不为空，循环，1出队，1可到达0，2，3，4节点，2，3，4松弛成功，且3，4不在队列中，入队

Node	0	1	2	3	4	5
dist[i]	0	1	3	8	6	∞

queue

4
3
2

实验4.1 SPFA算法提示



4 Queue不为空，循环，2出队，2可到达0，1，4节点，4松弛成功，4已在队列中

Node	0	1	2	3	4	5
dist[i]	0	1	3	8	4	∞

queue

4
3

5.Queue不为空，循环，3出队，3可到达1，4，5节点，5松弛成功，5不在队列中，入队

Node	0	1	2	3	4	5
dist[i]	0	1	3	8	4	10

queue

5
4

6.Queue不为空，循环，4出队，4可到达1，2，3，5节点，3松弛成功，3不在队列，入队

Node	0	1	2	3	4	5
dist[i]	0	1	3	7	4	10

queue

3
5

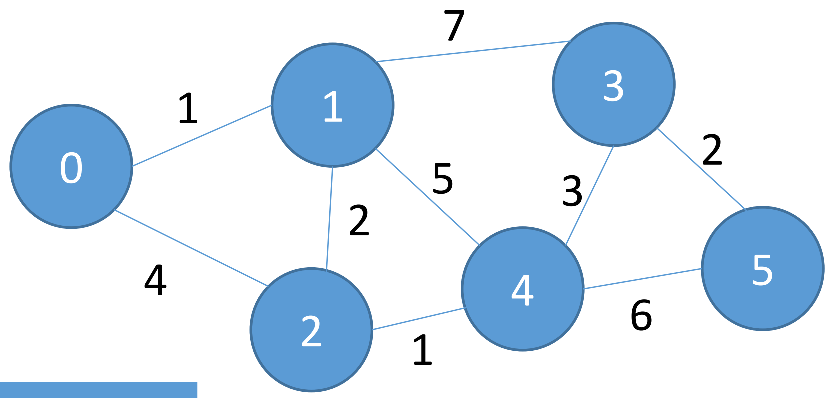
7.Queue不为空，循环，5出队，5可到达3，4节点，松弛不成功

Node	0	1	2	3	4	5
dist[i]	0	1	3	7	4	10

queue

3

实验4.1 SPFA算法提示



8 Queue不为空，循环，3出队，3可到达1，4，5节点，5松弛成功，5已未在队列，入队

Node	0	1	2	3	4	5
dist[i]	0	1	3	7	4	9

queue

5

9.Queue不为空，循环，5出队，5可到达3，4节点，松弛不成功

Node	0	1	2	3	4	5
dist[i]	0	1	3	7	4	9

queue

10.Queue为空，退出循环

最终从起点0到其余各个结点最短路径如下

dist[i]	0	1	3	7	4	9

sample_output	0 1 3 7 4 9

实验4.1 内容算法提示

由上述分析可设计数据结构：

```
struct Graph
{
   int Node;  //结点个数;
   vector<int>  dist;  //保存起点到所有结点当
                                   前最短路径
  vector<vector<int>> AM;   //邻接矩阵
  queue<int> q;    //保存等待访问结点
  Graph(int node)
  {           Node=node;
              dist.resize(Node);
              AM.resize(Node);
  };
};
```

参考代码：

```
int Node = 6;
vector<int> dist(Node, INT_MAX);
vector<vector<int>> AM = {{0, 1, 4, 10000, 10000, 10000},
                          {1, 0, 2, 7, 5, 10000},
                          {4, 2, 0, 10000, 1, 10000},
                          {10000, 7, 10000, 0, 3, 2},
                          {10000, 5, 1, 3, 0, 6},
                          {10000, 10000, 10000, 2, 6, 0}};
queue<int> q;
q.push(0);
dist[0] = 0;
while (!q.empty())
{
    int now = q.front();
    q.pop();
    for (int i = 0; i < AM[now].size(); ++i)
    {
        if (AM[now][i] != 10000 && AM[now][i] != 0)  //寻找now结点可达的结点
        {
            int to = i;
            if (dist[now] + AM[now][to] < dist[to])   //松弛成功
            {
                dist[to] = dist[now] + AM[now][to];   //修改dist
                if (!find(q, to))                    //判断to结点是否在队列q中
                    q.push(to);
            }
        }
    }
}
```

实验4.1 算法分析

时间复杂度分析

时间消耗主要是：任意一个结点要V条边做松弛操作，而一共有N个结点，所以时间复杂度为O(N*V)。

空间复杂度分析

空间消耗主要是：需要额外队列q，其长度小于等于N，所以空间复杂度为O(N).

实验4.2内容算法提示

题目：用迪杰斯特拉算法求一点到其余所有结点的最短路径。

先输入一个小于100的正整数n，然后输入图的邻接矩阵（10000表示无穷大，即两点之间没有边），最后输入两个0到n-1的整数表示两个点。

先用迪杰斯特拉算法求给定的第一个点到其余所有结点的最短路径。然后再输出给定的两个点之间的最短路径（按顺序输出最短路径上的每一个点，每个数据占一行）。

用迪杰斯特拉算法求赋权图中的最短路径

实验4.2内容算法提示

用迪杰斯特拉算法求赋权图中的最短路径

Dijkstra算法

用途：

用于求图中指定两点之间的**最短路径**，或者是**指定一点到其它**所有点之间的最短路径。实质上是**贪心算法。**

基本思想：

（1）设置两个顶点的集合S和T＝V－S，集合S中存放已找到最短路径的顶点，集合T存放当前还未找到最短路径的顶点；

（2）初始状态时，集合S中只包含源点v0；

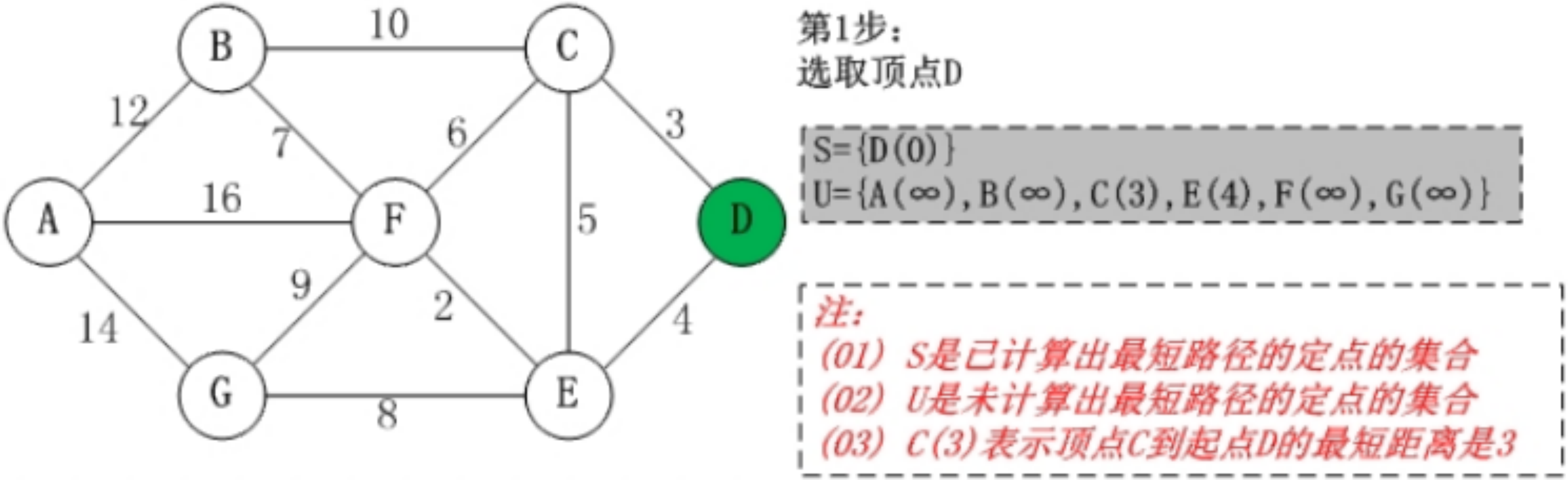
（3）从集合T中选取到某个顶点vi（要求vi到v0的路径长度最小）加入到S中；

（4）S中每加入一个顶点vi，都要修改顶点v0到T中剩余顶点的最短路径长度值，它们的值为原来值与新值的较小者，新值是vi的最短路径长度加上vi到该顶点的路径长度；

（5）不断重复（3）和（4），直到S包含全部顶点。

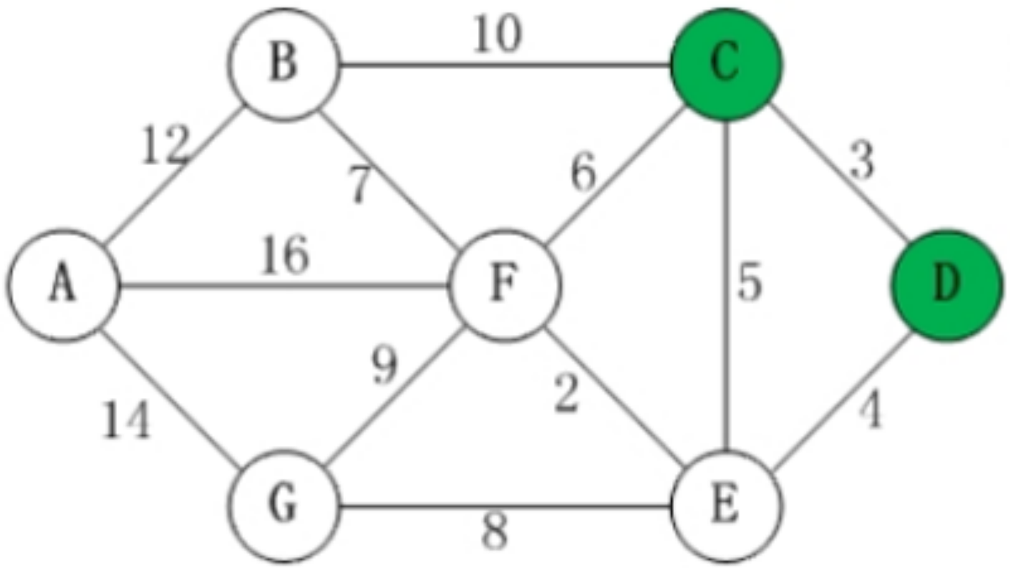
实验4.2内容算法提示

以下图为例，来对迪杰斯特拉进行算法演示(以第4个顶点D为起点)。



初始状态：S是已计算出最短路径的顶点集合，U是未计算除最短路径的顶点的集合！
第1步：将顶点D加入到S中。
此时，S={D(0)}, U={A(∞),B(∞),C(3),E(4),F(∞),G(∞)}。 注:C(3)表示C到起点D的距离是3。

实验4.2内容算法提示

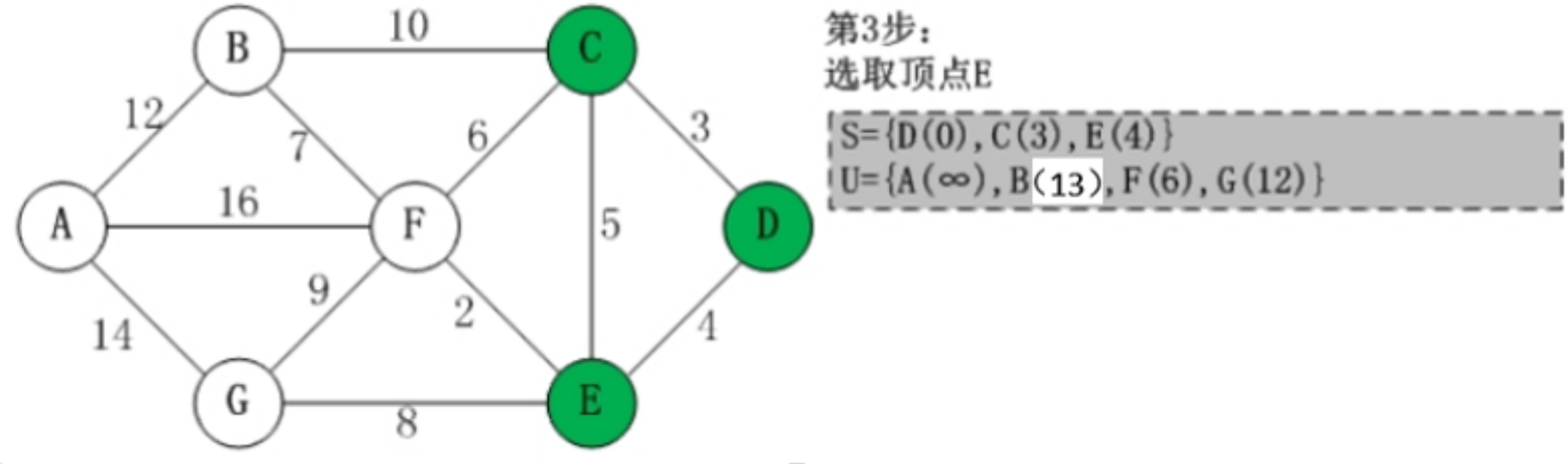


第2步：
选取顶点C

S={D(0),C(3)}
U={A(∞),B(13),E(4),F(9),G(∞)}

第2步：将顶点C加入到S中。 上一步操作之后，U中顶点C到起点D的距离最短；因此，将C加入到S中，同时更新U中顶点的距离。以顶点F为例，之前F到D的距离为∞；但是将C加入到S之后，F到D的距离为9=(F,C)+(C,D)。 此时，S={D(0),C(3)}, U={A(∞),B(23),E(4),F(9),G(∞)}。

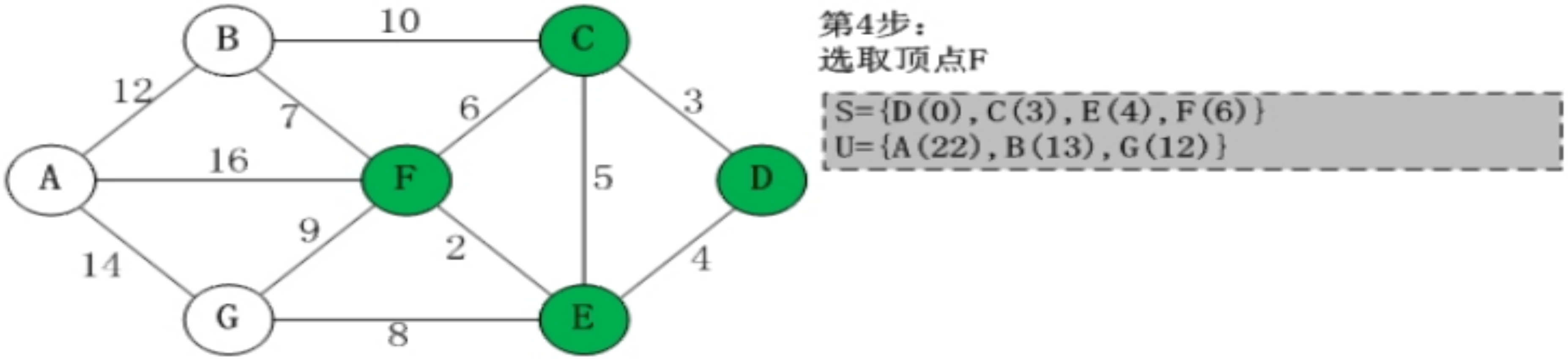
实验4.2内容算法提示

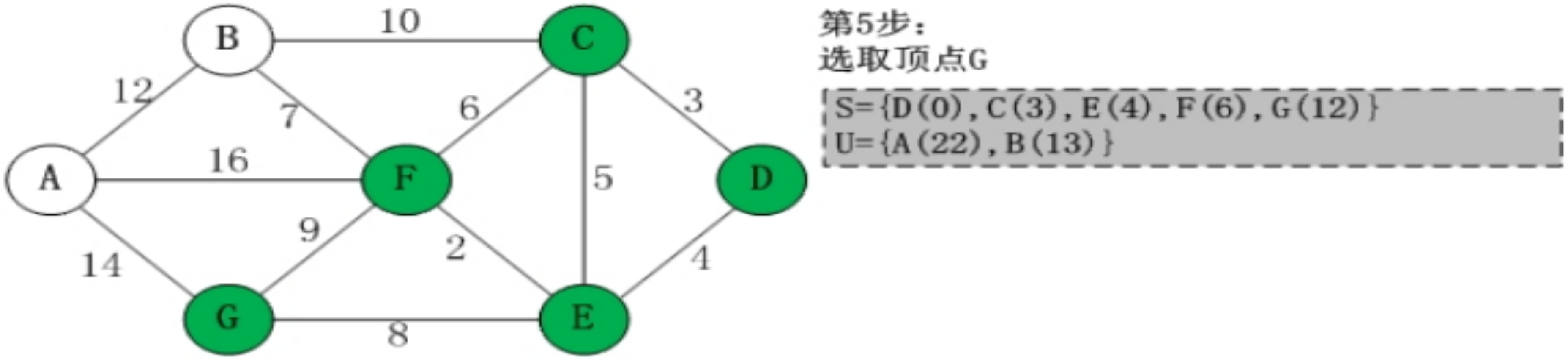


第3步：将顶点E加入到S中。 上一步操作之后，U中顶点E到起点D的距离最短；因此，将E加入到S中，同时更新U中顶点的距离。还是以顶点F为例，之前F到D的距离为9；但是将E加入到S之后，F到D的距离为6=(F,E)+(E,D)。 此时，S={D(0),C(3),E(4)}, U={A(∞),B(13),F(6),G(12)}。

用迪杰斯特拉算法求赋权图中的最短路径

实验4.2内容算法提示

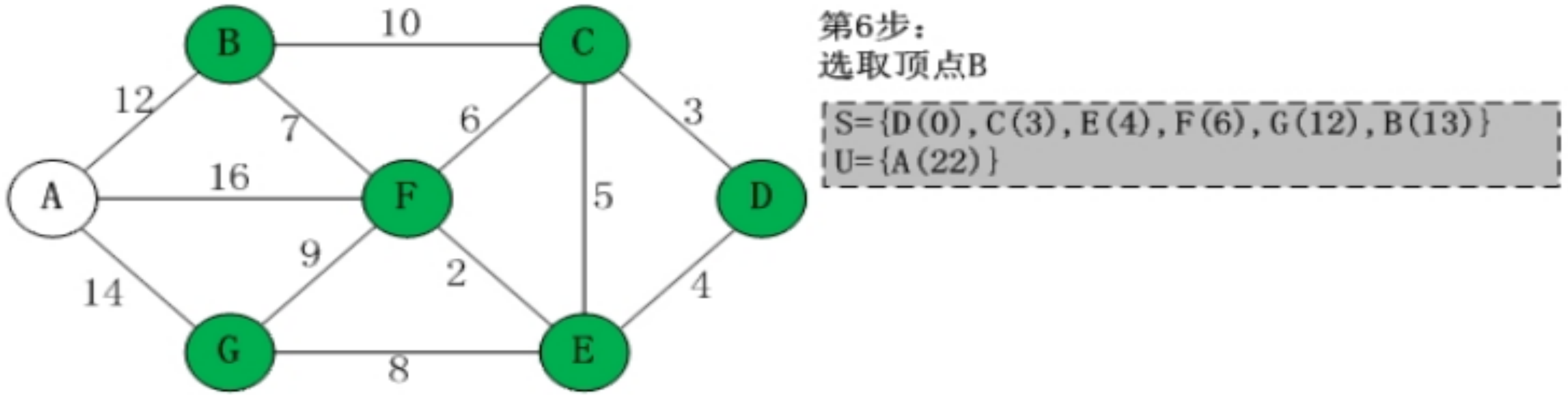


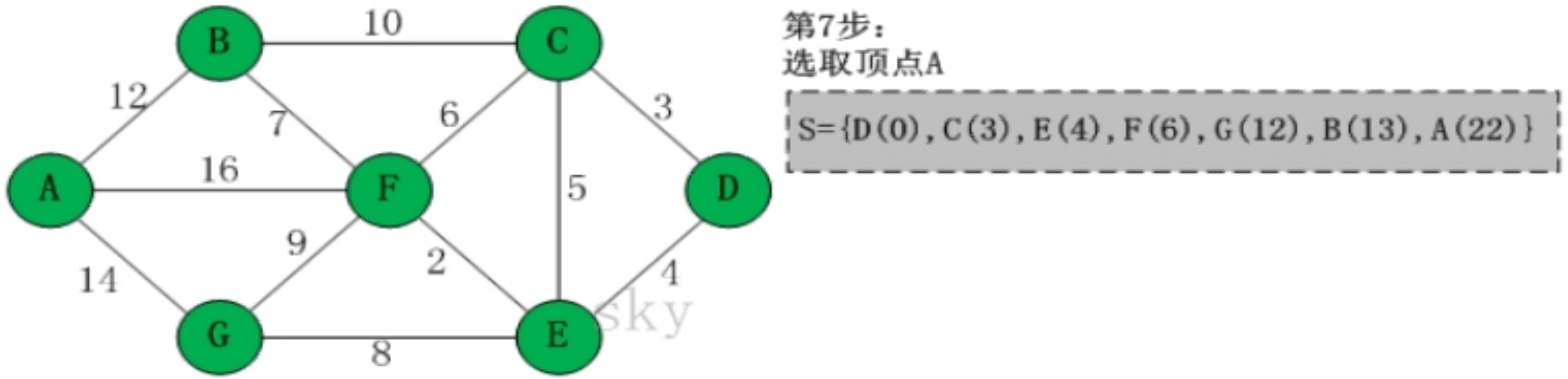


第4步：将顶点F加入到S中。 此时，S={D(0),C(3),E(4),F(6)}, U={A(22),B(13),G(12)}。

第5步：将顶点G加入到S中。 此时，S={D(0),C(3),E(4),F(6),G(12)}, U={A(22),B(13)}。

实验4.2内容算法提示





第6步：将顶点B加入到S中。 此时，S={D(0),C(3),E(4),F(6),G(12),B(13)}, U={A(22)}。
第7步：将顶点A加入到S中。 此时，S={D(0),C(3),E(4),F(6),G(12),B(13),A(22)}。
此时，起点D到各个顶点的最短距离就计算出来了：A(22) B(13) C(3) D(0) E(4) F(6) G(12)。

实验4.2内容算法提示

用迪杰斯特拉算法求赋权图中的最短路径

算法设计：

1.首先函数里面运用二维数组cost[n][n]实现图的邻接矩阵存储，数组dist[n]表示源点到节点n的最短距离，S[n]表示某一节点n是否已经进入集合S，如果进入则将S[i]置为1，否则为0。pre[n]表示当前节点n的前驱节点（用来输出路径）。

2.在开始遍历之前，首先给数组D[n]赋值为源点到该点的距离，这样便能第一次找到源点到相邻节点的最短距离（dist[i]=cost[v][i];）。

3.下面找出最短距离：

```
int u; //u为待选顶点
int min=inf; //令初始最小值>max，使距离值为max的顶点也能加到S中
if((!S[j])&&(dist[j]<min))  //寻找距离S最小的顶点u
 {min=dist[j];
     u=j;}
```

4.更新各节点的最短距离：

```
for(int k=0;k<n;k++)
{if((!S[k])&&(dist[k]>dist[u]+cost[u][k]))  //调整未加入S的点的距离值
{D[k]=D[u]+cost[u][k];
 pre[k]=u; //若通过u减小了k的距离值，则修改k的前趋为u
}}
```

5.另外，若从原点无法到达顶点x，则令其前趋为-1：pre[x]=-1，在输出判别一下就可以了。还要提醒的一点是输入输出的顶点的标号与实际存储的数组下标相差为1,应该要分辨清楚。

实验4.2内容算法提示

这个时间复杂度是O(N^2)。

其中每次找到离1号顶点最近的顶点的时间复杂度是O(N)，这里可以用“堆”来优化使降低到O(logN)，

另外对于边数M少于N^2的稀疏图来说（M<<N^2的图称为稀疏图，而M较大的图称为稠密图），我们可以用邻接表来代替邻接矩阵存储，使得整个时间复杂度优化到O(M+N)logN。

实验4.3内容算法提示

题目：用弗洛伊德算法求赋权图的两点间的最短路径的长度。

Input:先输入一个小于100的正整数n，然后输入图的邻接矩阵（10000表示无穷大，即两点之间没有边），之后再输入一个小于100的正整数m，最后的m行每行输入两个不同的0到n-1之间的整数表示两个点。

弗洛伊德算法求最短路径长度

实验4.3内容算法提示

注意：

图最初始输入时，仅记录了两点之间直接相连的距离，如上图

A-B 12

B-C 10

A-C ∞

当以B为中介，A-C 22

弗洛伊德算法核心思想是将每个点都作为中介，去更新其他点与点之间的路径

弗洛伊德算法求最短路径长度

实验4.3内容算法提示

核心思想：

```
//这里是弗洛伊德算法的核心部分
    //k为中间点
    for(k = 0; k < G.vexnum; k++){
        //v为起点
        for(v = 0 ; v < G.vexnum; v++){
            //w为终点
            for(w =0; w < G.vexnum; w++){
                if(D[v][w] > (D[v][k] + D[k][w])){
                    D[v][w] = D[v][k] + D[k][w];//更新最小路径
                }
            }
        }
    }
```

弗洛伊德算法求最短路径长度

实验4.3内容算法提示

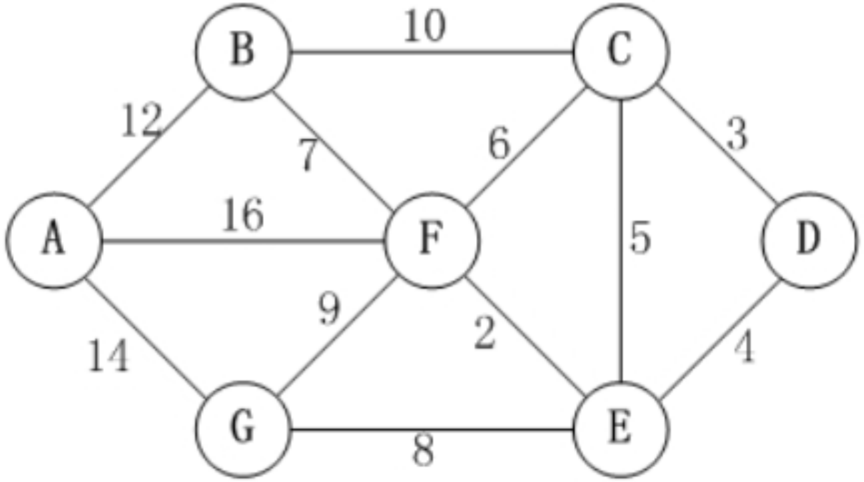
时间复杂度： n ^ 3

空间复杂度： n ^ 2

弗洛伊德算法求最短路径长度

实验4.3内容算法提示

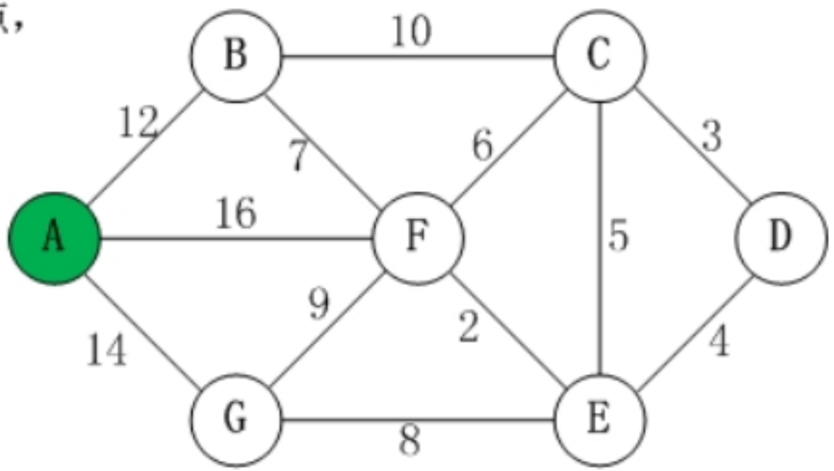
第1步：
初始化矩阵S



$$
\begin{array}{c}
\begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \\ G \end{array}
\begin{bmatrix}
0 & 12 & INF & INF & INF & 16 & 14 \\
12 & 0 & 10 & INF & INF & 7 & INF \\
INF & 10 & 0 & 3 & 5 & 6 & INF \\
INF & INF & 3 & 0 & 4 & INF & INF \\
INF & INF & 5 & 4 & 0 & 2 & 8 \\
16 & 7 & 6 & INF & 2 & 0 & 9 \\
14 & INF & INF & INF & 8 & 9 & 0
\end{bmatrix}
\end{array}
$$

实验4.3内容算法提示

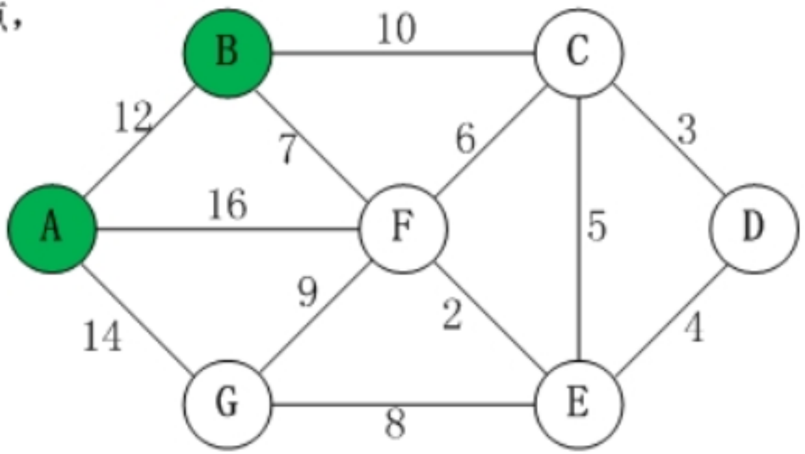
第2步：
以顶点A为中介点，
更新矩阵S。



$$
\begin{array}{c}
 \\
A \\ B \\ C \\ D \\ E \\ F \\ G
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\
\begin{bmatrix}
0 & 12 & INF & INF & INF & 16 & 14 \\
12 & 0 & 10 & INF & INF & 7 & 26 \\
INF & 10 & 0 & 3 & 5 & 6 & INF \\
INF & INF & 3 & 0 & 4 & INF & INF \\
INF & INF & 5 & 4 & 0 & 2 & 8 \\
16 & 7 & 6 & INF & 2 & 0 & 9 \\
14 & 26 & INF & INF & 8 & 9 & 0
\end{bmatrix}
\end{array}
$$

实验4.3内容算法提示

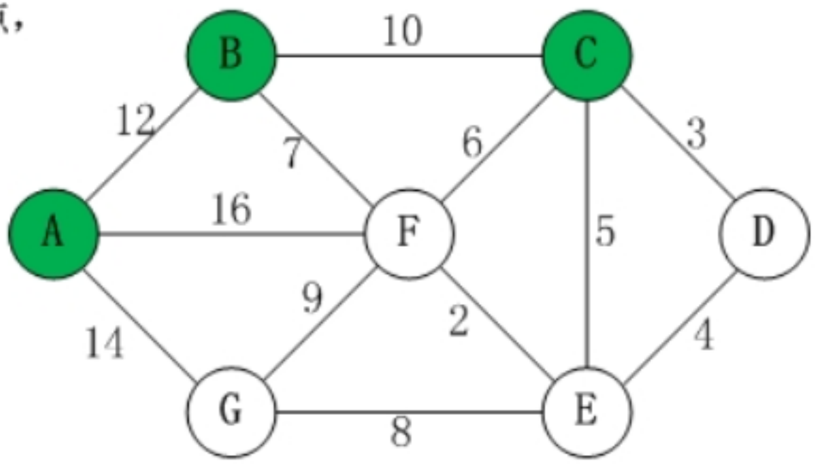
第3步：
以顶点B为中介点，
更新矩阵S。



$$
\begin{array}{c|ccccccc}
 & A & B & C & D & E & F & G \\
\hline
A & 0 & 12 & 22 & INF & INF & 16 & 14 \\
B & 12 & 0 & 10 & INF & INF & 7 & 26 \\
C & 22 & 10 & 0 & 3 & 5 & 6 & 36 \\
D & INF & INF & 3 & 0 & 4 & INF & INF \\
E & INF & INF & 5 & 4 & 0 & 2 & 8 \\
F & 16 & 7 & 6 & INF & 2 & 0 & 9 \\
G & 14 & 26 & 36 & INF & 8 & 9 & 0
\end{array}
$$

实验4.3内容算法提示

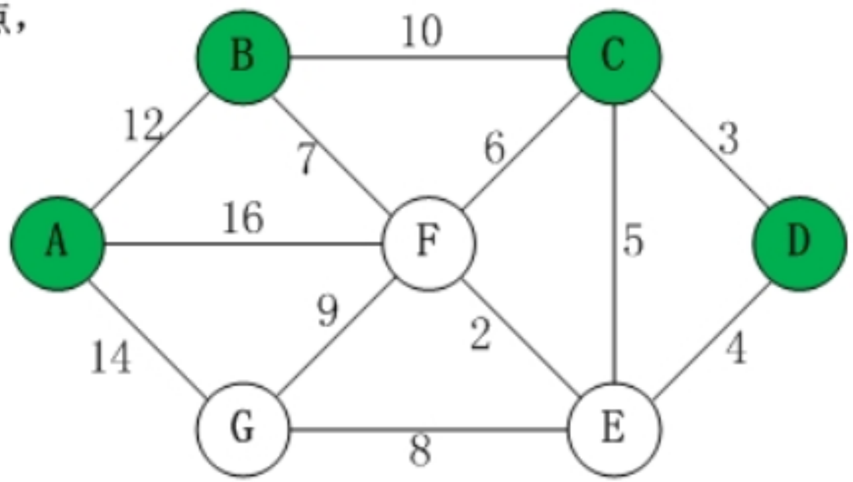
第4步：
以顶点C为中介点，
更新矩阵S。



$$
\begin{array}{c}
\\
\begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \\ G \end{array}
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\
\begin{bmatrix}
0 & 12 & 22 & 25 & 27 & 16 & 14 \\
12 & 0 & 10 & 13 & 15 & 7 & 26 \\
22 & 10 & 0 & 3 & 5 & 6 & 36 \\
25 & 13 & 3 & 0 & 4 & 9 & 39 \\
27 & 15 & 5 & 4 & 0 & 2 & 8 \\
16 & 7 & 6 & 9 & 2 & 0 & 9 \\
14 & 26 & 36 & 39 & 8 & 9 & 0
\end{bmatrix}
\end{array}
$$

实验4.3内容算法提示

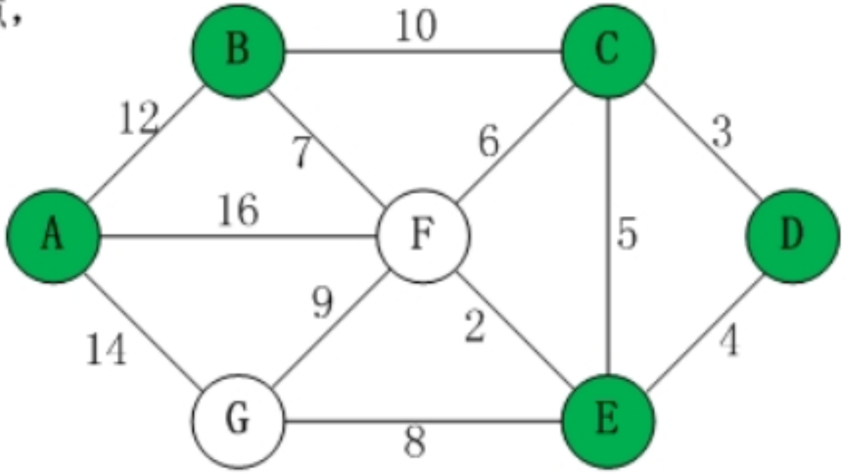
第5步：
以顶点D为中介点，
更新矩阵S。



$$
\begin{array}{c}
\\
A \\ B \\ C \\ D \\ E \\ F \\ G
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\
\begin{bmatrix}
0 & 12 & 22 & 25 & 27 & 16 & 14 \\
12 & 0 & 10 & 13 & 15 & 7 & 26 \\
22 & 10 & 0 & 3 & 5 & 6 & 36 \\
25 & 13 & 3 & 0 & 4 & 9 & 39 \\
27 & 15 & 5 & 4 & 0 & 2 & 8 \\
16 & 7 & 6 & 9 & 2 & 0 & 9 \\
14 & 26 & 36 & 39 & 8 & 9 & 0
\end{bmatrix}
\end{array}
$$

实验4.3内容算法提示

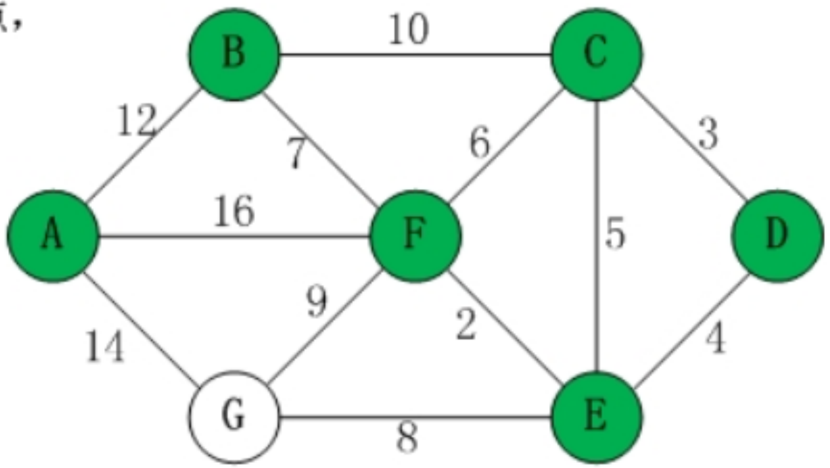
第6步：
以顶点E为中介点，
更新矩阵S。



$$
\begin{array}{c}
\\ A \\ B \\ C \\ D \\ E \\ F \\ G
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\
\begin{bmatrix}
0 & 12 & 22 & 25 & 27 & 16 & 14 \\
12 & 0 & 10 & 13 & 15 & 7 & 23 \\
22 & 10 & 0 & 3 & 5 & 6 & 13 \\
25 & 13 & 3 & 0 & 4 & 6 & 12 \\
27 & 15 & 5 & 4 & 0 & 2 & 8 \\
16 & 7 & 6 & 6 & 2 & 0 & 9 \\
14 & 23 & 13 & 12 & 8 & 9 & 0
\end{bmatrix}
\end{array}
$$

实验4.3内容算法提示

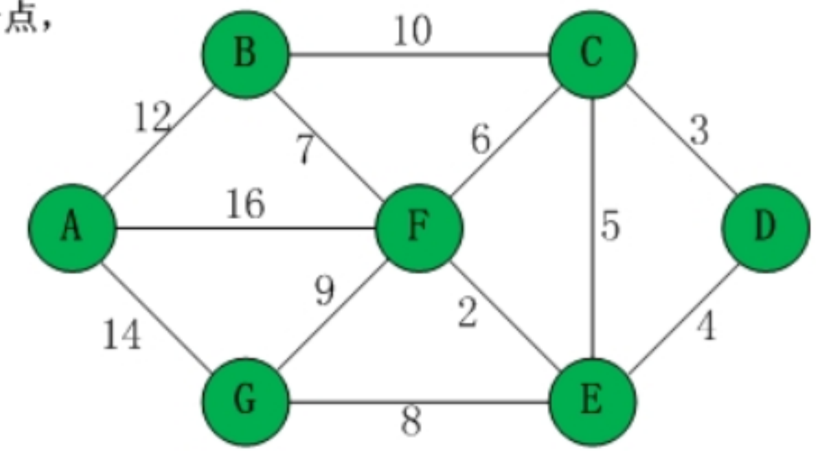
第7步：
以顶点F为中介点，
更新矩阵S。



$$
\begin{array}{c|c}
 & \begin{matrix} A & B & C & D & E & F & G \end{matrix} \\
\begin{matrix} A \\ B \\ C \\ D \\ E \\ F \\ G \end{matrix} &
\begin{bmatrix}
0 & 12 & 22 & 22 & 18 & 16 & 14 \\
12 & 0 & 10 & 13 & 9 & 7 & 16 \\
22 & 10 & 0 & 3 & 5 & 6 & 13 \\
22 & 13 & 3 & 0 & 4 & 6 & 12 \\
18 & 9 & 5 & 4 & 0 & 2 & 8 \\
16 & 7 & 6 & 6 & 2 & 0 & 9 \\
14 & 16 & 13 & 12 & 8 & 9 & 0
\end{bmatrix}
\end{array}
$$

实验4.3内容算法提示

第8步：
以顶点G为中介点，
更新矩阵S。



	A	B	C	D	E	F	G
A	0	12	22	22	18	16	14
B	12	0	10	13	9	7	16
C	22	10	0	3	5	6	13
D	22	13	3	0	4	6	12
E	18	9	5	4	0	2	8
F	16	7	6	6	2	0	9
G	14	16	13	12	8	9	0

实验4.4内容算法提示

题目：用弗洛伊德算法求任意两点间的最短路径，并输出指定的m对结点间的最短路径。

Input:先输入一个小于100的正整数n，然后输入图的邻接矩阵（10000表示无穷大，即两点之间没有边），之后再输入一个小于100的正整数m，最后的m行每行输入两个不同的0到n-1之间的整数表示两个点。

弗洛伊德算法求最短路径

实验4.4内容算法提示

注意：

与上一题思想大致相同，需要保存中介点信息以便输出路径

新建矩阵P[n][n]，用来存储中介节点信息

弗洛伊德算法求最短路径

实验4.4内容算法提示

核心思想：

```
//这里是弗洛伊德算法的核心部分
    //k为中间点
    for(k = 0; k < G.vexnum; k++){
        //v为起点
        for(v = 0 ; v < G.vexnum; v++){
            //w为终点
            for(w =0; w < G.vexnum; w++){
                if(D[v][w] > (D[v][k] + D[k][w])){
                    D[v][w] = D[v][k] + D[k][w];//更新最小路径
                    P[v][w] = P[v][k];//更新最小路径中间顶点
                }
            }
        }
    }
```

弗洛伊德算法求最短路径

实验4.4内容算法提示

时间复杂度： n ^ 3

空间复杂度： n ^ 2

弗洛伊德算法求最短路径长度

再会