第七章 图

7.6 最短路径

一. 最短路径

- 若用带权图表示交通网，图中顶点表示地点，边代表两地之间有直接道路，边上的权值表示路程(或所花费用或时间）。从一个地方到另一个地方的路径长度表示该路径上各边的权值之和。问题：
  - 两地之间是否有通路?
  - 在有多条通路的情况下，哪条最短?
- 考虑到交通网的有向性，直接讨论的是带权有向图的最短路径问题，但解决问题的算法也适用于无向图。
- 将一个路径的起始顶点称为源点，最后一个顶点称为终点。

7.6 最短路径

一. 最短路径

- 基本概念
  - 路径长度：一条路径上所经过的边的数目
  - 带权路径长度：一条路径所经过的边上的权值之和
  - 最短路径：带权路径长度值最小的那条路径，其长度就是最短路径长度或最短距离。
  - 单源最短路径是求从网中某一顶点（源点），到其余各顶点的最短路径。

7.6 最短路径

二. 最短路径迪杰斯特拉(Dijkstra)算法

- 问题描述：

对于给定的有向图G=(V，E)及单个源点V_s，求V_s到G的其余各顶点的最短路径。

Edsger Dijkstra

ALGOL的推广者，率先实现了ALGOL60的编译器；
在图论、算法和操作系统有很大的贡献；
提出了操作系统中的PV操作；图论中求最短路径的方法……。
获1972年图灵奖。

7.6 最短路径

二. 最短路径迪杰斯特拉(Dijkstra)算法

- 针对单源点的最短路径问题，**Dijkstra**提出了一种按路径长度递增次序产生最短路径的算法，其基本思想是：

从图的给定源点到其它各个顶点之间客观上应存在一条最短路径，在这组最短路径中，按其长度的递增次序，依次生成从源点到不同顶点的最短路径，并求出路径长度。

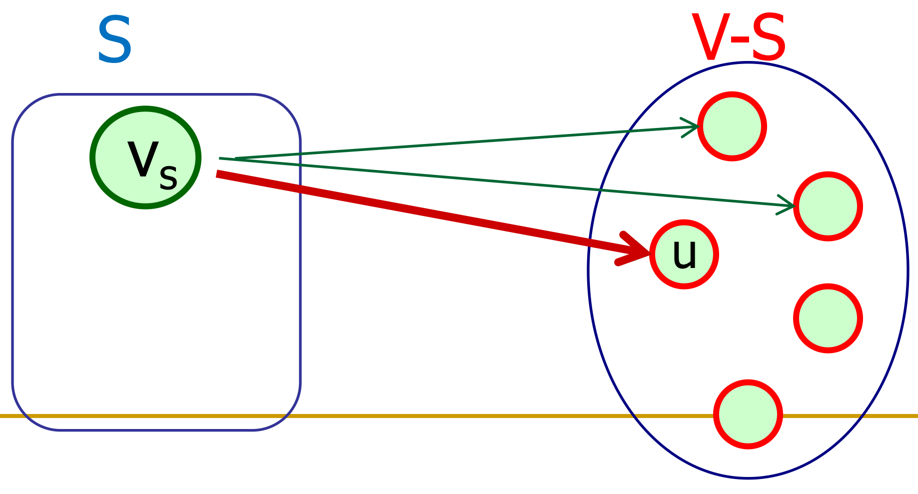
7.6 最短路径

二. 最短路径迪杰斯特拉(Dijkstra)算法

算法说明:

把图G=(V， E) 中的顶点集合V分成2组 ：

- 第1组为已求出最短路径的顶点集合S（开始时S={V_S}， 以后每求得一条最短路径V_S,… ,u ， 就将u加入到集合S中，直到全部顶点都加入到S中，此时算法结束）。
- 第2组为其余未求出最短路径的顶点集合（用V-S表示）。



7.6 最短路径

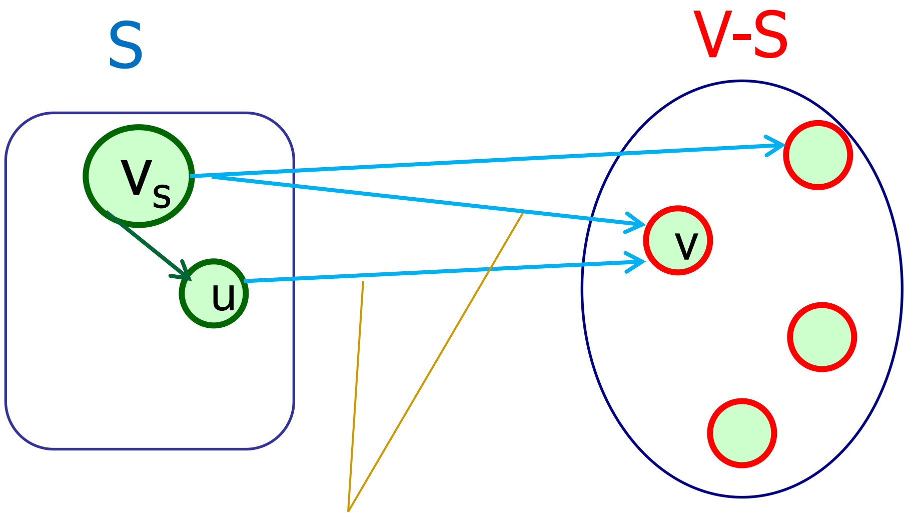
二. 最短路径迪杰斯特拉(Dijkstra)算法

算法说明：

- 随着顶点u加入到S中，V_S到V-S中其它顶点V_j的最短路径可能有更新，以下两项中长度较小者确定为新的最短路径：

  ① 已有的源点 V_s 到 V_j 的最短路径；

  ② 路径V_s，....，u, V_j



两条路径进行比较：若经过u 的路径长度更短，则更新路径(vs,v)为(vs,u,v) 及相应长度值

7.6 最短路径

二. 最短路径迪杰斯特拉(Dijkstra)算法

算法说明:

设给定源点为V_S，S为已求得最短路径的终点集，开始时令S={V_S}。当求得第一条最短路径(V_S，V_i)后，$V_i \in$ V-S，将V_i加入S中，即S变为{V_S，V_i}。根据以下结论继续求下一条最短路径，直到S =V。

设下一条最短路径终点为V_j，则V_j只能是下面两种路径的较小者的终点:

① 源点 V_s 到 V_j 有直接的弧<V_s，V_j>;

② 从V_s出发到V_j的这条最短路径所经过的所有中间顶点必定在S中，即只有这条最短路径的最后一条弧才是从S内某个顶点连接到V-S中的顶点V_j。

7.6 最短路径

二. 最短路径迪杰斯特拉(Dijkstra)算法

■ 算法实现

① 设置两个顶点集合S和V-S ，集合S中存放已找到最短路径的顶点，集合V-S中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，设为V_0；

③ 然后从集合V-S中选择到源点V_0路径长度最短的顶点V_j加入到集合S中；

④ 集合S中每加入一个新的顶点V_j都可能要修改源点V_0到集合V-S中剩余顶点的当前最短路径长度值，集合V-S中各顶点的新的当前最短路径长度值，为原来的当前最短路径长度值与从源点V_0过顶点V_j到达该顶点的路径长度中的较小者。

⑤ 转到3，此过程不断重复，直到集合V-S中的顶点全部加入到集合S中为止。

7.6 最短路径

- 迪杰斯特拉(Dijkstra)算法
  - 求v0到其他顶点的所有最短路径
  - 算法流程

    1、初始化，将v0加入已搜索顶点集合

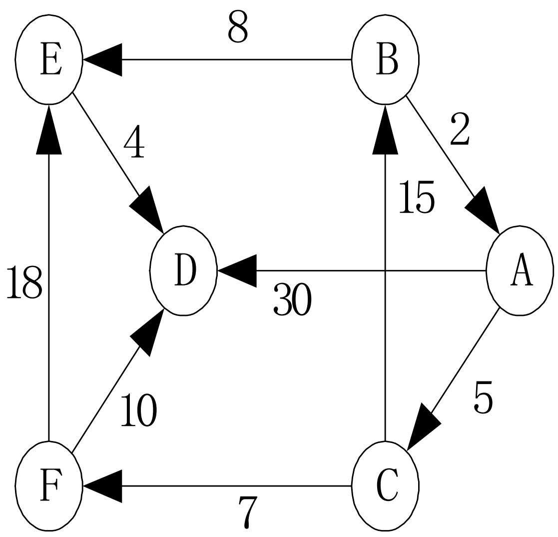
    2、找出离v0最近的顶点v，加入集合

    3、比较v0到其他顶点、v0到v再到其他顶点的路径，从而更新其他顶点的最短路径

    4、重复步骤2、3

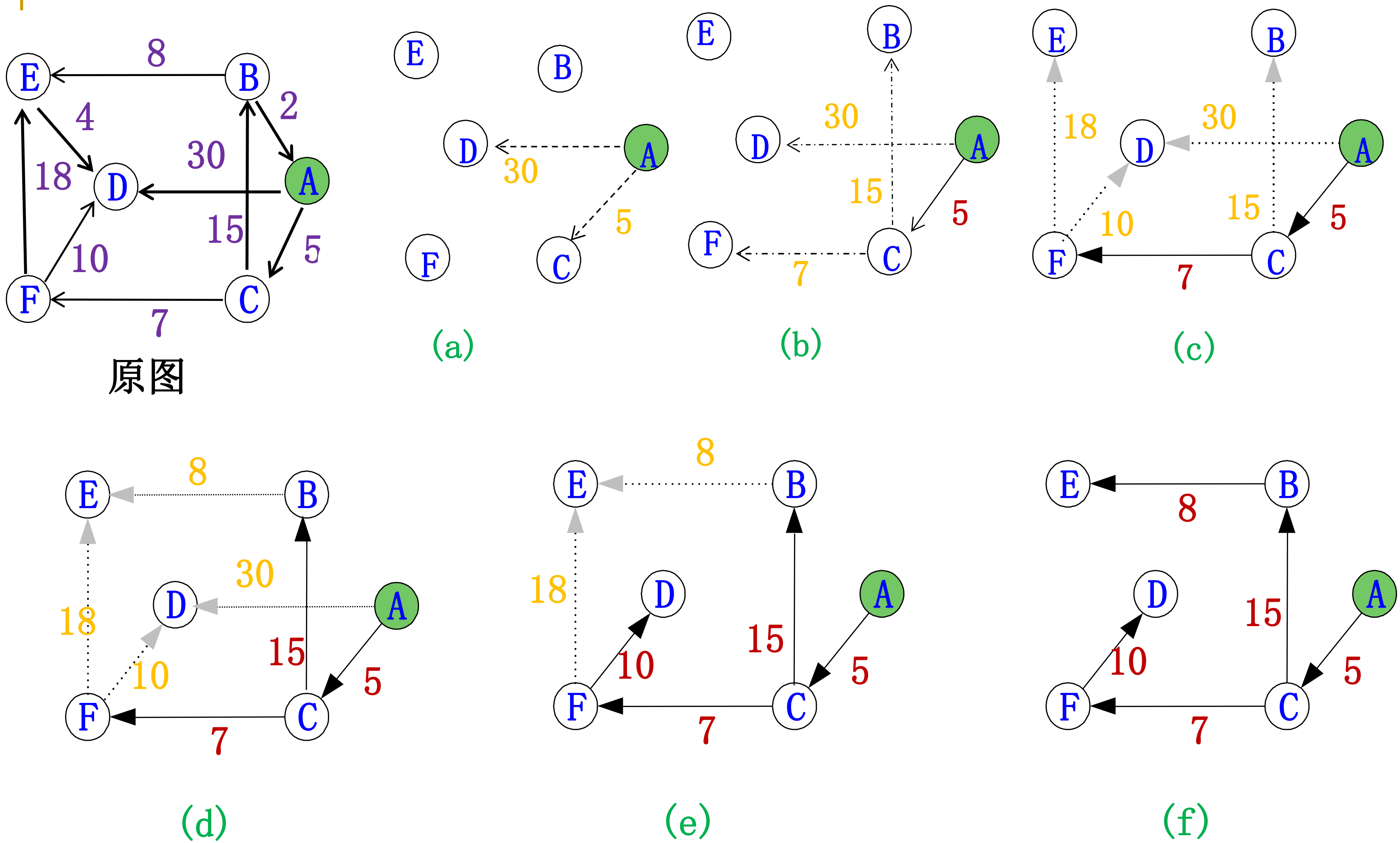
7.6 最短路径

例：求下图A顶点到各顶点的最短路径。



$$\begin{bmatrix} 0 & \infty & 5 & 30 & \infty & \infty \\ 2 & 0 & \infty & \infty & 8 & \infty \\ \infty & 15 & 0 & \infty & \infty & 7 \\ \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & 4 & 0 & \infty \\ \infty & \infty & \infty & 10 & 18 & 0 \end{bmatrix}$$

7.6 最短路径



E B D A F C 8 2 4 30 18 15 5 10 7
原图
E B D A F C 30 5
(a)
E B D A F C 30 15 5 7
(b)
E B D A F C 18 30 10 15 5 7
(c)
E B D A F C 8 30 18 15 10 5 7
(d)
E B D A F C 8 18 15 10 5 7
(e)
E B D A F C 8 15 10 5 7
(f)

二. 最短路径迪杰斯特拉(Dijkstra)算法

- 按路径长度递增的次序逐步产生最短路径
  - 在Dijkstra算法中，引进了一个辅助向量D
    - 每个分量D[i]表示当前所找到的从始点V_0到每个终点V_i的最短路径长度。
    - D[i]初值为起始点V_0到各终点V_i的直接距离，即若从起始点到某终点有(出)弧，则为弧上的权值，否则为∞。

7.6 最短路径

- Dijkstra算法实现

1. 令**S={Vs}**，用带权的邻接矩阵表示有向图，对图中每个顶点**Vi**按以下原则置初值：

$$D[i]=\begin{cases} 0 & i=s \\ W_{si} & i\neq s \text{ 且 } \langle v_s,v_i\rangle \in E, \quad w_{si}\text{为弧上的权值} \\ \infty & i\neq s \text{ 且 不存在} \langle v_s,v_i\rangle \end{cases}$$

2. 选择一个顶点$V_j \in V-S$，使得：

   $D[j] = Min\{ D[i] \mid i \in V-S \}$，　V_j就是求得的下一条最短路径终点，将 V_j 并入到S中，即 $S = S \cup \{j\}$；

3. 对V-S中的每个顶点V_k，修改D[k]，方法是：

   若 $D[j]+W_{jk} < D[k]$，则修改为：$D[k] = D[j]+W_{jk}$ $(\forall V_k \in V-S)$

4. 判断：若 S = V，则算法结束，否则转步骤 2。

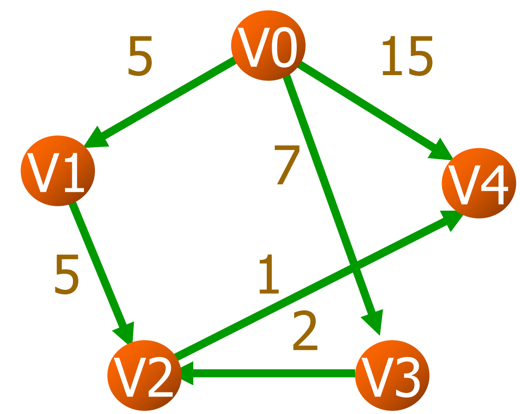
7.6 最短路径

二. 最短路径Dijkstra算法

- 对于右图，如果始点是v_0，设S为已求得的最短路径的终点的集合，初始时$S=\{v_0\}$
- D[i]的初值为：$D[i]=\{5, \infty, 7, 15\}$
- $D[j]=Min\{D[i] \mid v_i \in V\}$ 是从始点v_0出发的长度最短的一条路径$\langle v_0, v_j\rangle$
- 下一条长度次短的最短路径(设其终点为v_j)为以下之一：
  1. 弧$\langle v_0, v_j\rangle$
  2. 中间只经过S中的顶点v_k而后到达顶点v_j的路径

  则，$D[j]=Min\{ D_{0j} , D[k]+\langle v_k, v_j\rangle\}$

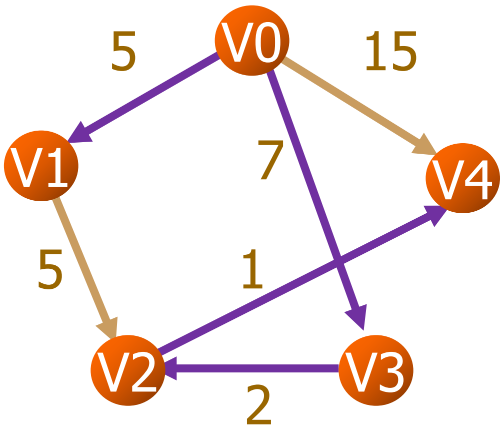
  $v_k \in S$ $v_j \in V-S$



7.6 最短路径

- Dijkstra算法举例

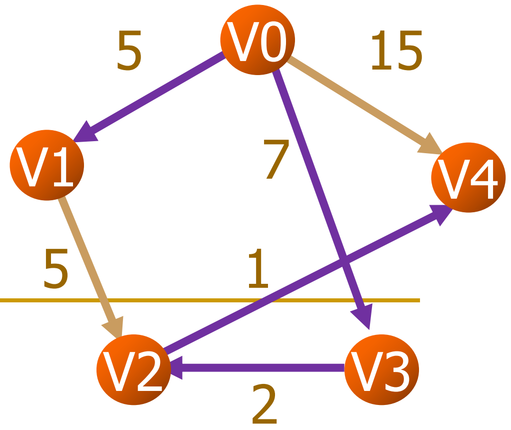
顶点	D[i]			
1	5 {0,1}			
2	∞	10 {0,1,2}	9 {0,3,2}	
3	7 {0,3}	7 {0,3}		
4	15 {0,4}	15 {0,4}	15 {0,4}	10 {0,3,2,4}
终点j	v1	v3	v2	v4
S	{v0, v1}	{v0,v1,v3}	{v0,v1,v3, v2}	{v0,v1,v3,v2, v4}



7.6 最短路径

- 算法实现分析
  - 能够找出V0到其他顶点的最短距离
  - 能够找出V0到其他顶点的最短路径的顶点
  - 但是不能找到最短路径的顶点排列顺序（经过哪些顶点？）
- 求每条最短路径的顶点序列的方法
  - 设置一个数组path[]，初始化清空，然后根据算法不断在里面加入路径的顶点，也就是用来存放得到的从源点v0到其余各顶点vi的最短路径上到达目标顶点vi的前一顶点下标。

	0	1	2	3	4
path	-1	0	3	0	2



7.6 最短路径

- Dijkstra算法

```
void ShortestPath_DIJ(MGraph G, int v0, PathMatrix &P, ShortPathTable &D)
{ // 若P[v][w]为TRUE，则w是从v0到v当前求得最短路径上的顶点
  // final[v]为TRUE当且仅当v∈S，即已经求得从v0到v的最短路径。
  int i=0, j,  v, w, min;
  bool final[MAX_VERTEX_NUM];
  for (v=0; v<G.vexnum; ++v) {
    final[v] = FALSE;             //初始化已找到最短路径的顶点标志final[i]
    D[v] = G.arcs[v0][v].adj;    //初始化最短路径值数组D[i]
    for (w=0; w<G.vexnum; ++w)  P[v][w] = FALSE;  // 设空路径
    if (D[v] < INFINITY) { P[v][v0] = TRUE;  P[v][v] = TRUE; }
  }
```

7.6 最短路径

- Dijkstra算法

```
  D[v0] = 0;  final[v0] = TRUE;          // 初始化，v0顶点属于S集合
  //--- 开始主循环，每次求得v0到某个顶点v的最短路径，并加v到S集合中
  for (i=1; i<G.vexnum; ++i) {           // 其余G.vexnum-1个顶点
    min = INFINITY;                      // 当前所知离v0顶点的最近距离
    for (w=0; w<G.vexnum; ++w)
      if (!final[w])                     // 比较，寻找在V-S中距离v0顶点最近的w顶点
        if (D[w]<min) { v = w;  min = D[w]; }
    final[v] = TRUE;                     // 离v0顶点最近的v=w加入S集合中
    for (w=0; w<G.vexnum; ++w)           // 更新当前最短路径及距离
      if (!final[w] && (min+G.arcs[v][w].adj<D[w])) { // 修改D[w]和P[w], w∈V-S
        D[w] = min + G.arcs[v][w].adj;
        for(j=0;j<G.vexnum;j++) P[w][j] = P[v][j]; //第v行赋值于第w行
        P[w][w] = TRUE;   // P[w] = P[v]+[w]
      }//if
  }//for
} // ShortestPath_DIJ
```

7.6 最短路径

二. 最短路径迪杰斯特拉(Dijkstra)算法

- 算法分析

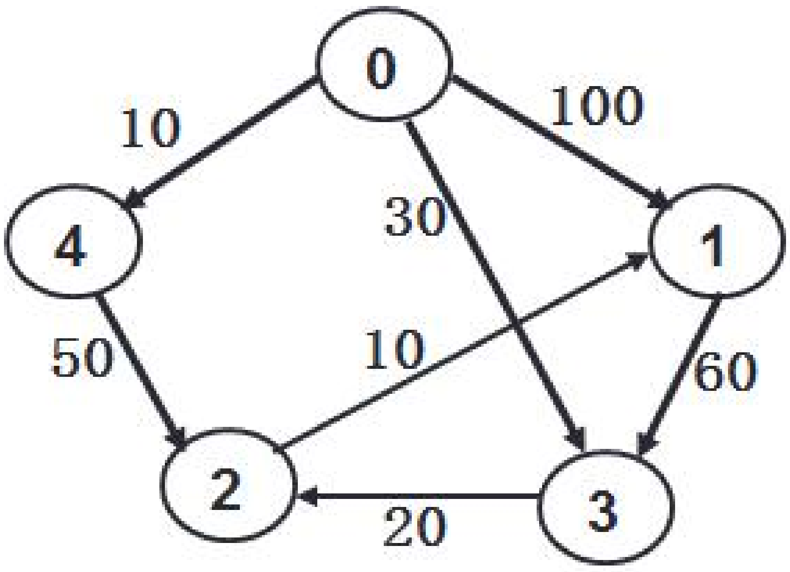
Dijkstra算法的主要执行是：

- 数组变量的初始化：时间复杂度是**O(n)**；
- 求最短路径的二重循环：时间复杂度是**O(n²)**；

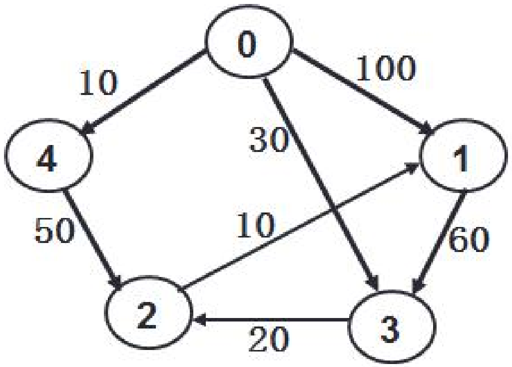
因此，整个算法的时间复杂度是**O(n²)**。

7.6 最短路径

- **练习1：**已知带权有向图如下，请求出顶点0到其他顶点的最短路径和长度，要求使用迪杰斯特拉算法写出求解过程



练习参考答案



顶点	D[i]			
1	100 {0, 1}	100 {0, 1}	100 {0, 1}	60 {0, 3, 2, 1}
2	∞	60 {0, 4, 2}	50 {0, 3, 2}	
3	30 {0, 3}	30 {0, 3}		
4	10 {0, 4}			
新加入的顶点	4	3	2	1
S	{0, 4}	{0, 4, 3}	{0, 4, 3, 2}	{0, 4, 3, 2, 1}

7.6 最短路径

- 求n个顶点之间的最短路径
  - 用**Dijkstra**算法也可以求得有向图**G=(V，E)**中每一对顶点间的最短路径。方法是：设置二维数组D[i][j]，数组每一行D[i]表示从顶点vi出发到其它顶点的最短路径，即每次以一个不同的顶点vi为源点重复**Dijkstra**算法便可求得每一对顶点间的最短路径，时间复杂度是**$O(n^3)$**。
  - 弗罗伊德**(Floyd)**算法，其时间复杂度仍是**$O(n^3)$**，但算法形式更为简明，步骤更为简单，是基于图的邻接矩阵。

7.6 最短路径

弗罗伊德算法实现

- 定义一个n阶方阵序列 $D^{(-1)}, D^{(0)}, D^{(1)}, \ldots, D^{(k)}, \ldots, D^{(n-1)}$,
  其中:

$$D^{(-1)}[i][j] = G.arcs[i][j]$$

$$D^{(k)}[i][j] = \mathrm{Min}\{\ D^{(k-1)}[i][j],\ \ D^{(k-1)}[i][k] + D^{(k-1)}[k][j]\ \}$$

$D^{(1)}[i][j]$是从vi到vj的中间顶点序号不大于1的最短路径的长度, $D^{(k)}[i][j]$是从vi到vj的中间顶点序号不大于k的最短路径的长度;…依此类推, $D^{(n-1)}[i][j]$就是从vi到vj的最短路径的长度。

7.6 最短路径

- 弗罗伊德算法实现

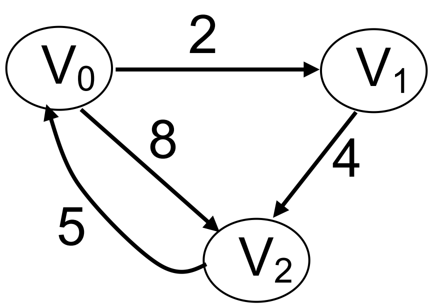
  - 定义二维数组Path[n][n](n为图的顶点数)，元素Path[i][j]保存从V_i到V_j的最短路径所经过的顶点。

    若 Path[i][j]=k：从V_i到V_j经过V_k，最短路径序列是(V_i, …, V_k, …, V_j)，则路径序列：(V_i, …, V_k)和(V_k, …, V_j)一定是从V_i到V_k和从V_k到V_j的最短路径。从而可以根据Path[i][k]和Path[k][j]的值再找到该路径上所经过的其它顶点，…依此类推。

7.6 最短路径

- **弗罗伊德算法实现**

初始时令**Path[i][j]=-1**，表示从V_i到V_j不经过任何(**S**中的中间)顶点。当某个顶点V_k使**D[i][j]**变小，令**Path[i][j]=k**。



$$\begin{bmatrix} 0 & 2 & 8 \\ \infty & 0 & 4 \\ 5 & \infty & 0 \end{bmatrix}$$

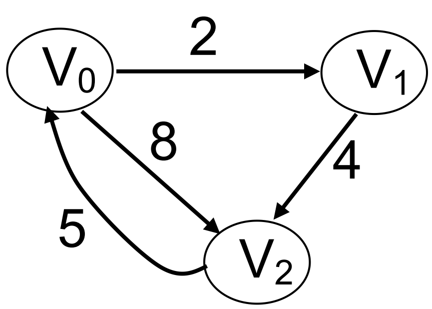
$D^{(-1)}$

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

Path[ ][ ]

7.6 最短路径

步骤	初始	k=0	K=1	K=2
D	$\begin{bmatrix} 0 & 2 & 8 \\ \infty & 0 & 4 \\ 5 & \infty & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 2 & 8 \\ \infty & 0 & 4 \\ 5 & 7 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 2 & 6 \\ \infty & 0 & 4 \\ 5 & 7 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 2 & 6 \\ 9 & 0 & 4 \\ 5 & 7 & 0 \end{bmatrix}$
Path	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & -1 \\ -1 & -1 & -1 \\ -1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & -1 & -1 \\ -1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ 2 & -1 & -1 \\ -1 & 0 & -1 \end{bmatrix}$



根据上述过程中Path[i][j]数组，得出：

V_0到V_1： 最短路径是{ V0, V1 }， 路径长度是2；

V_0到V_2： 最短路径是{ V0, V1, V2 }， 路径长度是6；

V_1到V_0： 最短路径是{ V1, V2, V0 }， 路径长度是9；

V_2到V_1： 最短路径是{ V2, V0, V1 }， 路径长度是7。

第七章总结

- 图的数据结构：G＝(V, E)，V是顶点，E是边或弧
  - 无向图的边(x,y)，有向图的弧<x,y>，带权值的图称为网
- 图的术语：
  - 度、入度、出度的计算
  - 路径、回路（环）、简单路径、连通、生成树
- 图的存储结构：
  - 邻接矩阵
  - 邻接表和逆邻接表
- 图的遍历：深度优先搜索、广度优先搜索
- 生成树：DFS生成树和BFS生成树
- 最小生成树：
  - 普里姆(Prim)算法，从顶点出发
  - 克鲁斯卡尔(Kruskal)算法，从边中选择
- 拓扑排序，根据AOV网求拓扑有序序列
- 关键路径，根据AOE图求关键路径（活动的最早和最晚开始时间）
- 最短路径：迪杰斯特拉(Dijkstra)算法