



7.6 最短路径



最短路径

广度优先搜索

- 某一地区的一个公路网，给定了该网内的 n 个城市以及这些城市之间的相通公路的距离，能否找到城市A到城市B之间一条最近的通路呢？
- 从A地到B地换车次数最少的路径
- 从A地到B地最短的路径（距离最短，行驶时间最短）



最短路径

- 算法:

1. 迪杰斯特拉算法----从一个源点到其它各点的最短路径
2. 弗洛伊德算法----每一对顶点之间的最短路径



迪杰斯特拉算法

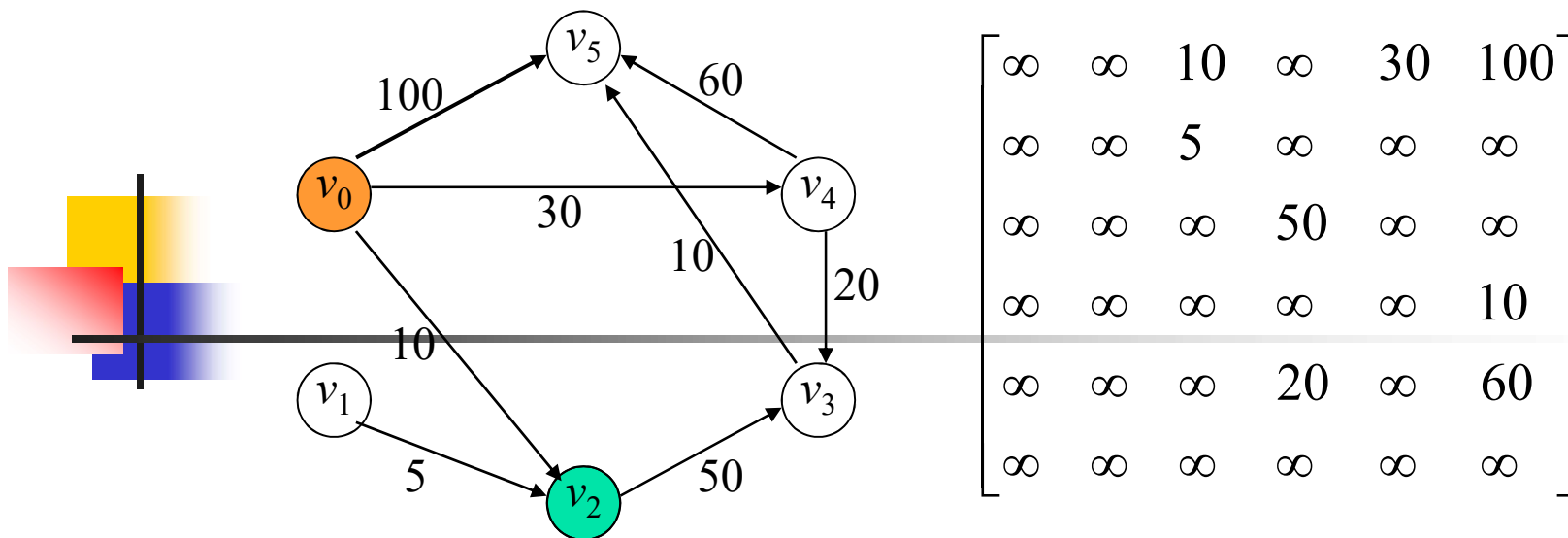
- 按路径长度递增的次序产生最短路径
- 设图 $G=(V,E)$, $V=\{v_0, v_1, \dots, v_{n-1}\}$, **假设**源点为 v_0 , 求 v_0 到其余各点的最短路径。
- **分析:**
 - 设 v_0 到 v_1, \dots, v_{n-1} 的最短路径分别为 P_1, P_2, \dots, P_{n-1} . 若这 $n-1$ 条路径中最短的一条为 $P_i (1 \leq i \leq n-1)$, 那么它一定是弧 $\langle v_0, v_i \rangle$;
 - 若这 $n-1$ 条路径中第二短的为 $P_j (1 \leq i \neq j \leq n-1)$, 那么它一定是弧 $\langle v_0, v_j \rangle$ 或者是路径 $\langle v_0, v_i \rangle \langle v_i, v_j \rangle$;
 -

该算法只适用于静态网络
网络上边的权值不能为负数



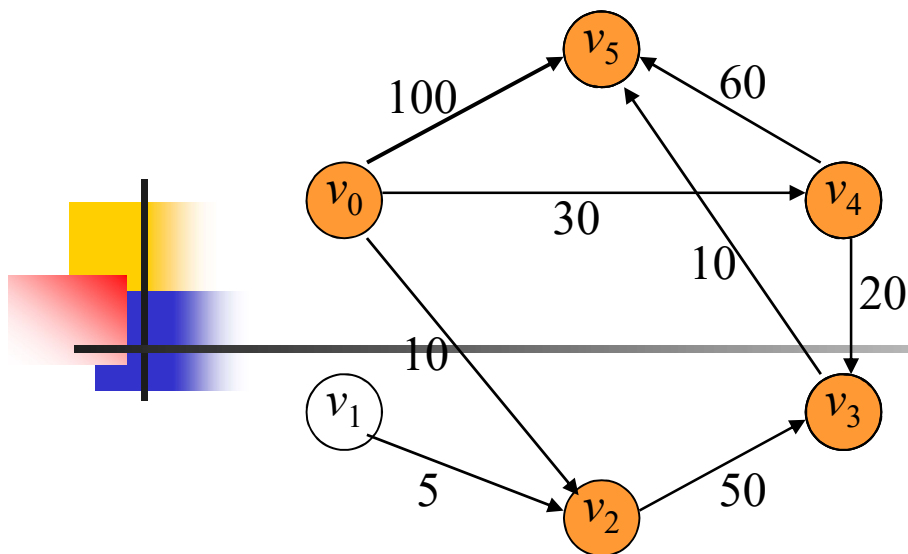
迪杰斯特拉算法

- **基本思想：** 设集合S中存放已找到最短路径的顶点，集合 $T = V - S$ 存放当前还未找到最短路径的顶点。
 - (1) 初态: S中只包含源点 v_0 ， v_0 到其余各点的**弧**为各点**当前**的“最短”路径。
 - (2) 从T中选取**当前**的“最短”路径长度最短的顶点u加入到S中。S每加入一个新的顶点u，都要修改顶点 v_0 到T中剩余顶点的最短路径长度，T中各顶点新的最短路径长度值为原来的最短路径长度值与顶点u的最短路径长度值加上u到该顶点的路径长度值中的较小值。
 - (3) 重复(2)，直到T的顶点全部加入到S中为止。



| 初态 | $i=1$ | $i=2$ | $i=3$ | $i=4$ | $i=5$ |
|-------|------------------------------|-------|-------|-------|-------|
| v_1 | ∞ | | | | |
| v_2 | 10 (v_0, v_2) | | | | |
| v_3 | ∞ | | | | |
| v_4 | 30 (v_0, v_4) | | | | |
| v_5 | 100 (v_0, v_5) | | | | |
| 选择点 | | | | | |
| S | $\{v_0\}$ | | | | |

迪杰斯特拉算法求解过程



v_0 到 v_1 的最短路径:无穷

v_0 到 v_2 的最短路径:10, $v_0 \rightarrow v_2$

v_0 到 v_3 的最短路径:50, $v_0 \rightarrow v_4 \rightarrow v_3$

v_0 到 v_4 的最短路径:30, $v_0 \rightarrow v_4$

v_0 到 v_5 的最短路径:60, $v_0 \rightarrow v_4 \rightarrow v_3 \rightarrow v_5$

| 初态 | $i=1$ | $i=2$ | $i=3$ | $i=4$ | $i=5$ |
|-------|------------------------------|----------------------------------|----------------------------------|---------------------------------------|----------|
| v_1 | ∞ | ∞ | ∞ | ∞ | ∞ |
| v_2 | 10 (v_0, v_2) | | | | |
| v_3 | ∞ | 60 (v_0, v_2, v_3) | 50 (v_0, v_4, v_3) | | |
| v_4 | 30 (v_0, v_4) | 30 (v_0, v_4) | | | |
| v_5 | 100 (v_0, v_5) | 100 (v_0, v_5) | 90 (v_0, v_4, v_5) | 60 (v_0, v_4, v_3, v_5) | |
| 选择点 | v_2 | v_4 | v_3 | v_5 | |
| S | $\{v_0, v_2\}$ | $\{v_0, v_2, v_4\}$ | $\{v_0, v_2, v_3, v_4\}$ | $\{v_0, v_2, v_3, v_4, v_5\}$ | |

迪杰斯特拉算法求解过程



迪杰斯特拉算法

- 实现迪杰斯特拉算法尚需解决以下几个问题：
 - 图的存储？
 - 如何区分已经求出最短路径的点，即：如何区分S中的点和V-S中的点？
 - 如何表示源点到顶点i的最短路径？

迪杰斯特拉算法实现--图的存储

图的存储：邻接矩阵和邻接表都可以，下面以**邻接矩阵**为例给出算法实现

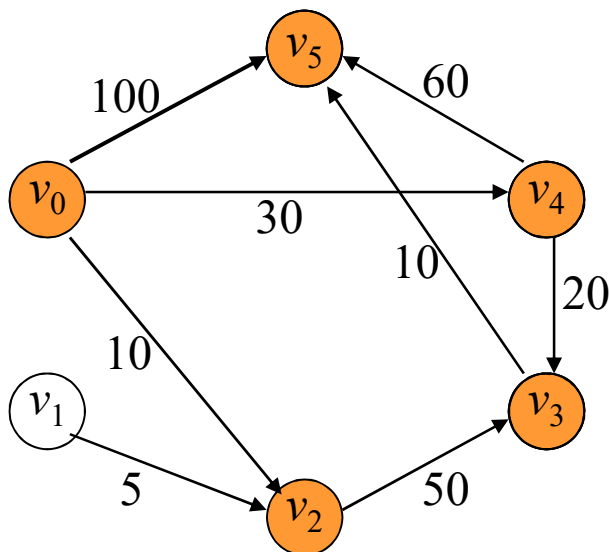
```
#define max 100
```

```
typedef struct {
```

```
    int arcs[max][max];
```

```
    int vexnum, arcnum; } AGraphs;
```

AGraphs G;



| | | | | | |
|----------|----------|----------|----------|----------|----------|
| ∞ | ∞ | 10 | ∞ | 30 | 100 |
| ∞ | ∞ | 5 | ∞ | ∞ | ∞ |
| ∞ | ∞ | ∞ | 50 | ∞ | ∞ |
| ∞ | ∞ | ∞ | ∞ | ∞ | 10 |
| ∞ | ∞ | ∞ | 20 | ∞ | 60 |
| ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |

迪杰斯特拉算法实现--S中的点的表示

| | | | | | |
|----------|----------|----------|----------|----------|----------|
| ∞ | ∞ | 10 | ∞ | 30 | 100 |
| ∞ | ∞ | 5 | ∞ | ∞ | ∞ |
| ∞ | ∞ | ∞ | 50 | ∞ | ∞ |
| ∞ | ∞ | ∞ | ∞ | ∞ | 10 |
| ∞ | ∞ | ∞ | 20 | ∞ | 60 |
| ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |

■ 方法一:

- 设一个一维数组 `int final[max];`
- `final[i]=1` 表示从源点到顶点i的最短路径已经求出, i在S中
- `final[i]=0` 表示从源点到顶点i的最短路径尚未求出, i在V-S中

■ 方法二:

- 利用邻接矩阵主对角线的位置 `G.arcs[i][i]` 表示i是否在S中
- `G.arcs[i][i]=1` 表示从源点到顶点i的最短路径已经求出, i在S中
- `G.arcs[i][i]=0` 表示从源点到顶点i的最短路径尚未求出, i在V-S中

迪杰斯特拉算法实现--最短路径的表示

- 一维数组 `int D[max]` 表示最短路径的长度
 - $D[i]$: 从源点到点 v_i 的最短路径的长度
 - **初态为**：若从源点到 v_i 有弧，则 $D[i]$ 为弧上的权值；否则置 $D[i]$ 为 ∞ ，即： $D[i]=G.arcs[k][i]$ ；//**说明**： k 为源点
- 二维数组 `int P[max][max]` 表示最短路径包含的顶点
 - $P[i][]$: 从源点到点 v_i 的最短路径
 - $P[i][j]=0$: v_j 不在从源点到点 v_i 的最短路径上
 - $P[i][j]=1$: v_j 位于从源点到点 v_i 的最短路径上。
 - **说明**：1. 书上的这种表示最短路径的方式只是给出了最短路径经过的顶点有哪些，没有给出这些顶点在这条路径上的顺序。2. 也可以采用其它的方式存储表示最短路径----例如**利用最短路径定理，存放终点前一步的位置**

迪杰斯特拉算法实现--最短路径的表示

■ 二维数组 $\text{int } p[\text{max}][\text{max}]$ 表示最短路径包含的顶点

➤ $P[i][j]$: 从源点到点 v_i 的最短路径

• $P[i][j]=0$: v_j 不在从源点到点 v_i 的最短路径上

• $P[i][j]=1$: v_j 位于从源点到点 v_i 的最短路径上。

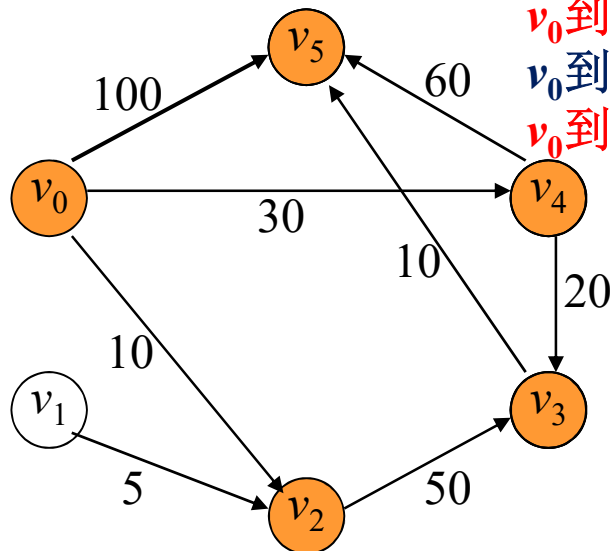
v_0 到 v_1 的最短路径: 无穷

v_0 到 v_2 的最短路径: 10, $v_0 \rightarrow v_2$

v_0 到 v_3 的最短路径: 50, $v_0 \rightarrow v_4 \rightarrow v_3$

v_0 到 v_4 的最短路径: 30, $v_0 \rightarrow v_4$

v_0 到 v_5 的最短路径: 60, $v_0 \rightarrow v_4 \rightarrow v_3 \rightarrow v_5$



$P[5][0]=1;$

$P[5][1]=0;$

$P[5][2]=0;$

$P[5][3]=1;$

$P[5][4]=1;$

$P[5][5]=1;$

$P[3][0]=1;$

$P[3][1]=0;$

$P[3][2]=0;$

$P[3][3]=1;$

$P[3][4]=1;$

$P[3][5]=0;$



算法描述(v_k 为源点)

(1) 初始化操作:

$D[i] = G.\text{arcs}[k][i]; D[k] = 0;$

$\text{final}[i] = 0 \ (v_i \in V); \text{final}[k] = 1;$

$p[i][j] = 0; v_i, v_j \in V;$

$p[i][i] = 1; p[i][k] = 1;$

(2) 选择 j , 使得 $D[j] = \text{Min}\{D[i] \mid v_i \in V-S\};$

$S = S \cup \{j\}; // \text{final}[j] = 1$

(3) 修改从 k 出发到集合 $V-S$ 上任一顶点的最短路径长度:

若 $D[j] + G.\text{arcs}[j][w] < D[w]$, 则 $D[w] = D[j] + G.\text{arcs}[j][w]$

(4) 重复操作(2),(3)最多 $n-1$ 次

```
void ShortestPath(AGraphs G,int k,int P[ ][ ], int D[ ])
```

```
{  int i,w,j,min;
```

```
    for (i=0;i<G.vexnum; i++)
```

```
    {  final[i]=0; D[i]=G.arcs[k][i];
```

```
        for(w=0;w<G.vexnum; w++) P[i][w]=0;
```

```
        if (D[i]<INFINITY) {  P[i][k]=1; P[i][i]=1; }
```

```
    }
```

```
    D[k]=0; final[k]=1;
```

```
    for(i=1; i<G.vexnum; i++)
```

```
    {  min=INFINITY;
```

```
        for (w=0;w<G.vexnum; w++)
```

```
        if (!final[w]&&D[w]<min) {j=w; min=D[w];}
```

```
        if(min== INFINITY) return;
```

```
        final[j]=1;
```

```
        for(w=0;w<G.vexnum; w++)
```

```
        if(!final[w]&&(min+G.arcs[j][w]<D[w]))
```

```
        {  D[w]=min+G.arcs[j][w];
```

```
            P[w]=P[j]; P[w][w]=1;  }
```

```
    }
```

```
}
```



弗洛伊德算法

- 求每一对顶点之间的最短路径
- 从 v_i 到 v_j 的所有可能存在的路径中，选出一条长度最短的路径。
- 若 $\langle v_i, v_j \rangle$ 存在，则存在路径 (v_i, v_j)
- 若 $\langle v_i, v_0 \rangle, \langle v_0, v_j \rangle$ 存在，则存在路径 (v_i, v_0, v_j)
- 若 $(v_i, \dots, v_1), (v_1, \dots, v_j)$ 存在，则存在一条路径 $(v_i, \dots, v_1, \dots, v_j)$
- ...
- 依次类推，则 v_i 至 v_j 的最短路径应是上述这些路径中，路径长度最小者。



弗洛伊德算法——顶点 v_i 到顶点 v_j 的最短路径

- 如果 $\langle v_i, v_j \rangle \in E(G)$, 则从 v_i 到 v_j 存在一条路径 (v_i, v_j) ;
- 该路径是否为最短路径尚需进行 n 次试探。
- 首先考虑路径 (v_i, v_0, v_j) , 若其存在, 比较路径 (v_i, v_0, v_j) 和路径 (v_i, v_j) 的长度, 取其中较小者为从 v_i 到 v_j 的中间顶点序号不大于0的最短路径;
- 在路径上再加一个顶点 v_1 , 若 (v_i, \dots, v_1) 和 (v_1, \dots, v_j) 分别是当前找到的中间顶点的序号不大于0最短路径, 那么将 $(v_i, \dots, v_1, \dots, v_j)$ 和已找到的中间结点的序号不大于0的最短路径比较, 取其中较小的为从 v_i 到 v_j 的中间顶点序号不大于1的最短路径;
- 再增加一个顶点 v_2 , 继续进行试探, 依次类推。经过 n 次试探后可求得从顶点 v_i 到顶点 v_j 的最短路径。



弗洛伊德算法

图采用邻接矩阵形式存放

```
#define max 100
```

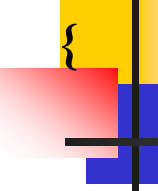
```
typedef struct {  
    int arcs[max][max];  
    int vexnum, arcnum;}AGraphs;
```

```
AGraphs G;
```



弗洛伊德算法

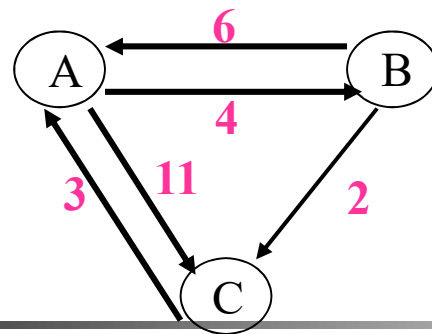
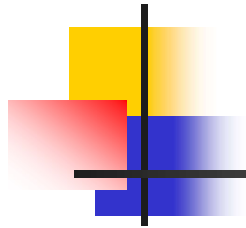
- 弗洛伊德算法递推地产生一个n阶矩阵序列： $D^{(-1)}, D^{(0)}, \dots, D^{(k)}, \dots, D^{(n-1)}$
- 其中 $D^{(-1)}[i][j] = G.\text{arcs}[i][j]$;
 $D^{(k)}[i][j] = \min \{D^{(k-1)}[i][j], D^{(k-1)}[i][k] + D^{(k-1)}[k][j]\} \quad (0 \leq k \leq n-1)$
- $D^{(k)}[i][j]$ 表示从 v_i 到 v_j 的中间顶点序号不大于 k 的最短路径的长度； $D^{(n-1)}[i][j]$ 表示从 v_i 到 v_j 的最短路径的长度
- $P[i][j]$ 表示从 v_i 到 v_j 的路径.
 - $P[i][j][x] = 1$ 表示从 v_i 到 v_j 的路径有顶点 x
 - $P[i][j][x] = 0$ 表示从 v_i 到 v_j 的路径没有顶点 x



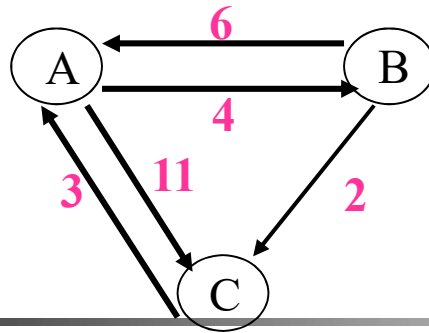
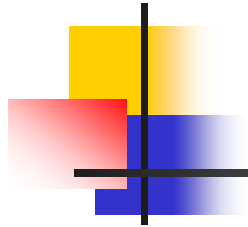
```

void s1(int D[ ][ ],int p[ ][ ][ ], Agraphs G)
{
    int i,j,k;
    for(i=0;i<G. vexnum;i++)
    for(j=0;j<G. vexnum;j++)
    {
        D[i][j]=G.arcs[i][j];
        for(k=0;k<G.vnum;k++) p[i][j][k]=0;
        if(D[i][j]<INFINITY)
            { p[i][j][i]=1; p[i][j][j]=1;}}
    for (k=0;k<G. vexnum;k++)
        for (i=0;i<G. vexnum;i++)
            for(j=0;j<G vexnum;j++)
                if(D[i][k]+D[k][j]<D[i][j])
                    {D[i][j]=D[i][k]+D[k][j];
                     for(w=0;w<G. vexnum;w++)
                         p[i][j][w]=p[i][k][w]||p[k][j][w];}
    }

```



| | | |
|---|----------|----|
| 0 | 4 | 11 |
| 6 | 0 | 2 |
| 3 | ∞ | 0 |



$$\begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix}$$

$$D^{(-1)} = \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix} \quad D^{(0)} = \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \quad D^{(1)} = \begin{bmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \quad D^{(2)} = \begin{bmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

$$P^{(-1)} = \begin{bmatrix} AB & AC \\ BA & BC \\ CA & \end{bmatrix} \quad P^{(0)} = \begin{bmatrix} AB & AC \\ BA & BC \\ CA & CAB \end{bmatrix} \quad P^{(1)} = \begin{bmatrix} AB & ABC \\ BA & BC \\ CA & CAB \end{bmatrix} \quad P^{(2)} = \begin{bmatrix} AB & ABC \\ BCA & BC \\ CA & CAB \end{bmatrix}$$