



Data Structures

Ch6



Graphs

2024 年 11 月 15 日

学而不厌 诲人不倦

- ➡ 6.1 引言
- ➡ 6.2 图的逻辑结构
- ➡ 6.3 图的存储结构及实现
- ➡ 6.4 最小生成树
- ➡ **6.5 最短路径**
- ➡ 6.6 有向无环图及其应用
- ➡ 6.7 扩展与提高
- ➡ 6.8 应用实例

6.5 最短路径

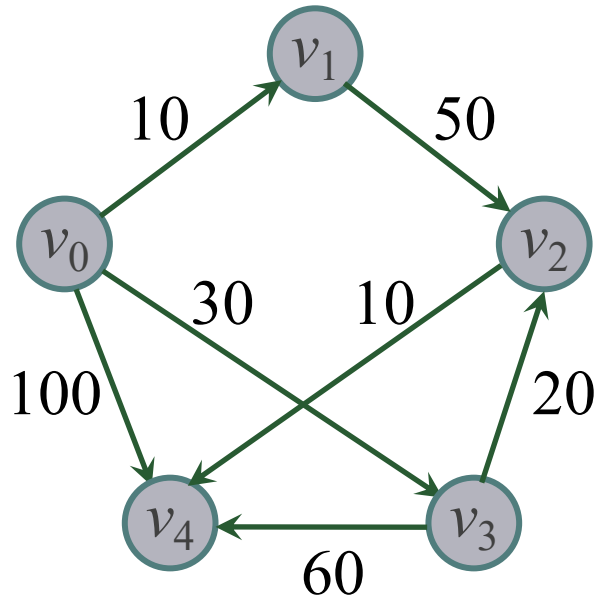
1. 最短路径



最短路径：非带权图——边数最少的路径



最短路径：带权图——边上的权值之和最少的路径



v_0 到 v_4 的最短路径:

$v_0 v_4$: 1

$v_0 v_3 v_4$: 2

$v_0 v_1 v_2 v_4$: 3

$v_0 v_3 v_2 v_4$: 3

v_0 到 v_4 的最短路径:

$v_0 v_4$: 100

$v_0 v_3 v_4$: 90

$v_0 v_1 v_2 v_4$: 70

$v_0 v_3 v_2 v_4$: 60

6.5 最短路径

1. 最短路径

【单源点最短路径问题】

给定带权有向图 $G=(V, E)$ 和源点 $v \in V$, 求从 v 到 G 中其余各顶点的最短路径

【每一对顶点的最短路径问题】

给定带权有向图 $G=(V, E)$, 对任意顶点 v_i 和 v_j ($i \neq j$), 求从顶点 v_i 到顶点 v_j 的最短路径

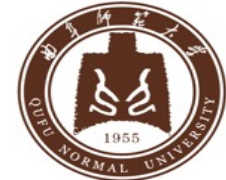
6.5 最短路径

6-5-1 Dijkstra算法

6.5 最短路径

【单源点最短路径问题】

6-5-1 Dijkstra算法



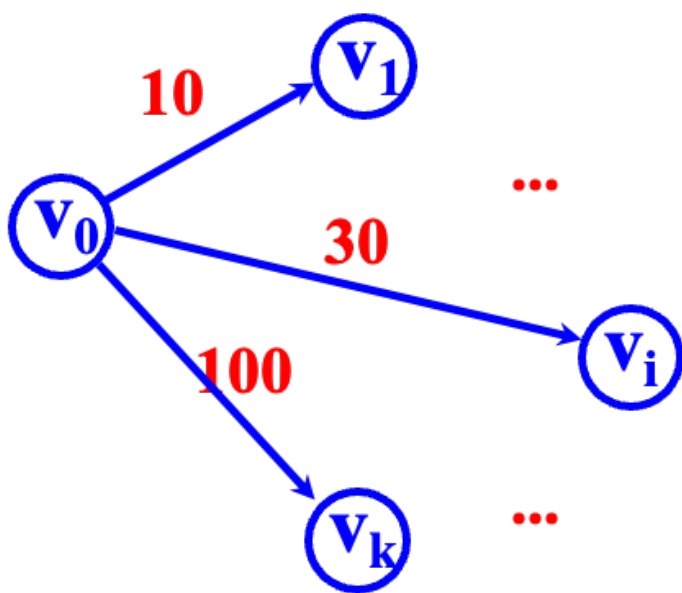
1. Dijkstra算法思想

最小生成树Prim算法、Kruskal算法均为贪心算法，其中Prim算法是对图的节点贪心，而Kruskal算法是对图上的边贪心。

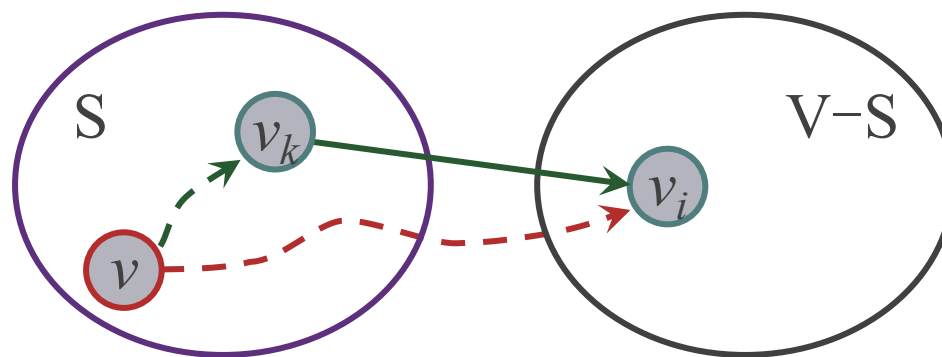
思想：贪心算法(局部最优)，按路径长度递增的次序产生最短路径。

贪心算法：利用局部最优来计算全局最优。

利用已得到的顶点的最短路径来计算其它顶点的最短路径。

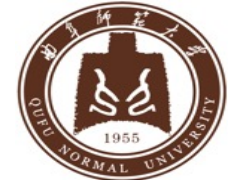


假设图中所示为从源点到其余各点之间的最短路径，则在这些路径中，必然存在一条**长度最短者**。



6.5 最短路径

6-5-1 Dijkstra算法



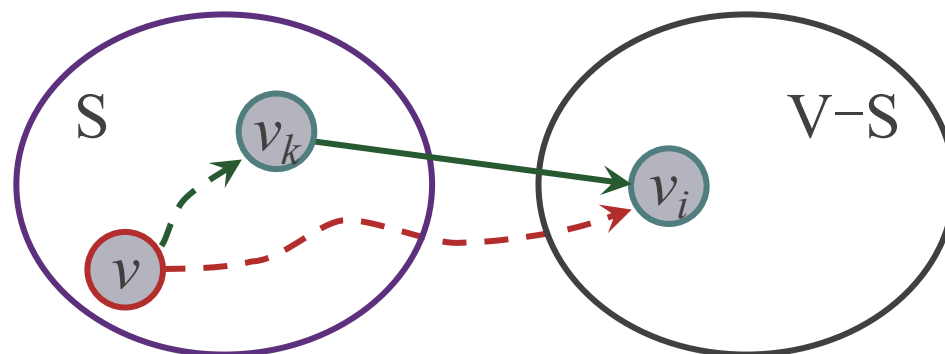
1. Dijkstra算法思想

v : 源点

S : 已经生成最短路径的终点

$w<v, v_i>$: 从顶点 v 到顶点 v_i 的权值

$\text{dist}(v, v_i)$: 表示从顶点 v 到顶点 v_i 的最短路径长度



1. 从**未标记结点**中选择距离源点最近的结点 v_k : $\text{dist}(v, v_k) = \min\{\text{dist}(v, v_j), (j=1..n)\}$;

2. 将结点 v_k 加入最短路径终点集合 S 中: $S = S + \{v_k\}$;

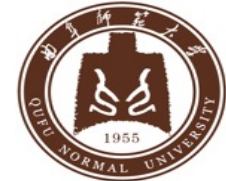
3. 计算从刚加入结点 v_k 到 **V-S** 集合中**未标记邻近结点** v_i 的距离:

$$\text{dist}(v, v_i) = \min\{\text{dist}(v, v_i), \text{dist}(v, v_k) + w<v_k, v_i>\};$$

如果 **源点到 v_k 的距离 + v_k 到 v_i 的距离 < 源点到 v_i 的距离**, 则**更新源点到 v_i 的距离**。

6.5 最短路径

6-5-1 Dijkstra算法



1. Dijkstra算法思想

算法：Dijkstra算法

输入：有向网图 $G=(V, E)$ ，源点 v

输出：从 v 到其他所有顶点的最短路径

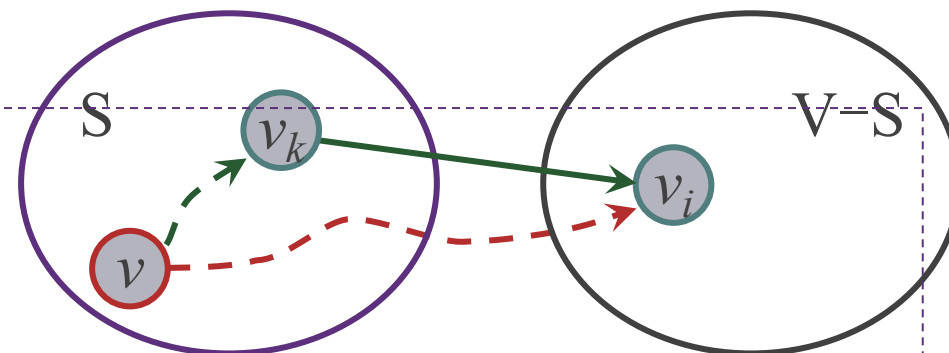
1. 初始化：集合 $S = \{v\}$ ； $\text{dist}(v, v_i) = w\langle v, v_i \rangle, (i=1\dots n)$ ；

2. 重复下述操作直到 $S = V$

2.1 $\text{dist}(v, v_k) = \min\{\text{dist}(v, v_j), (j=1\dots n)\}$ ；

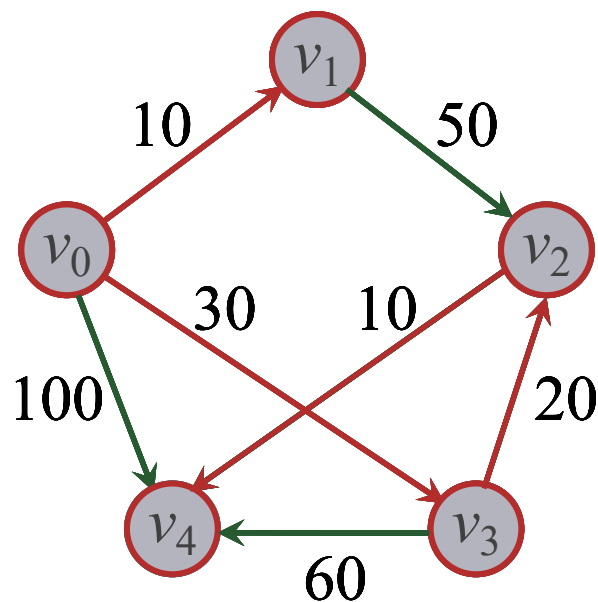
2.2 $S = S + \{v_k\}$ ；

2.3 $\text{dist}(v, v_i) = \min\{\text{dist}(v, v_i), \text{dist}(v, v_k) + w\langle v_k, v_i \rangle\}$ ；





1. Dijkstra算法思想



初始化: $S=\{v_0\}$

$\text{dist}(v, v_i)$: $\langle v_0, v_1 \rangle 10$ $\langle v_0, v_2 \rangle \infty$ $\langle v_0, v_3 \rangle 30$ $\langle v_0, v_4 \rangle 100$

第一次迭代: $S=\{v_0, v_1\}$

$\text{dist}(v, v_i)$: $\langle v_0, v_1, v_2 \rangle 60$ $\langle v_0, v_3 \rangle 30$ $\langle v_0, v_4 \rangle 100$

第二次迭代: $S=\{v_0, v_1, v_3\}$

$\text{dist}(v, v_i)$: $\langle v_0, v_3, v_2 \rangle 50$ $\langle v_0, v_3, v_4 \rangle 90$

第三次迭代: $S=\{v_0, v_1, v_3, v_2\}$

$\text{dist}(v, v_i)$: $\langle v_0, v_3, v_2, v_4 \rangle 60$

第四次迭代: $S=\{v_0, v_1, v_3, v_2, v_4\}$

6.5 最短路径

6-5-1 Dijkstra算法

2. Dijkstra算法存储结构



图采用什么存储结构呢?



邻接矩阵



如何存储 $\text{dist}(v, v_i)$ 呢?



待定路径表 (当前的最短路径)

整型数组 $\text{dist}[n]$: 存储当前最短路径的长度

字符串数组 $\text{path}[n]$: 存储当前的最短路径, 即顶点序列

算法: Dijkstra算法

输入: 有向网图 $G=(V, E)$, 源点 v

输出: 从 v 到其他所有顶点的最短路径

1. 初始化: 集合 $S = \{v\}$; $\text{dist}(v, v_i) = w\langle v, v_i \rangle, (i=0\dots n)$;

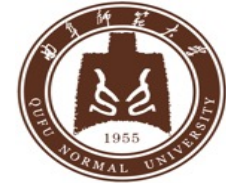
2. 重复下述操作直到 $S = V$

2.1 $\text{dist}(v, v_k) = \min\{\text{dist}(v, v_j), (j=0\dots n)\}$;

2.2 $S = S + \{v_k\}$;

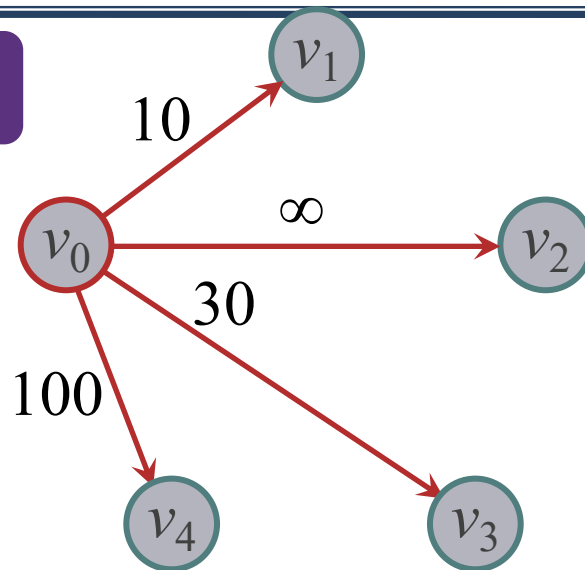
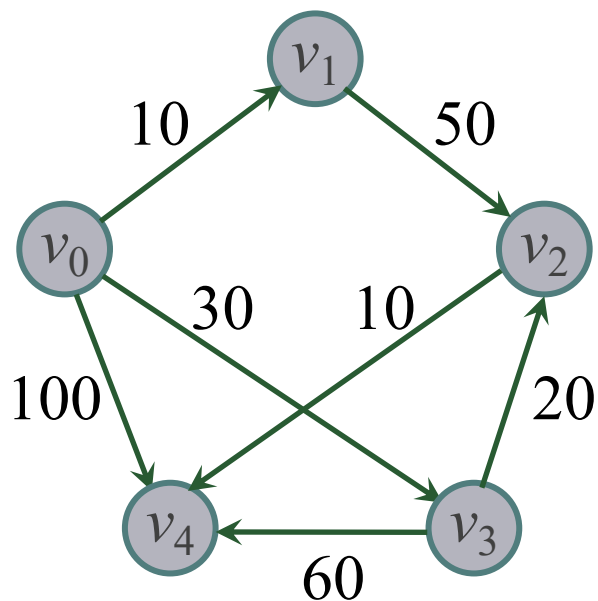
2.3 $\text{dist}(v, v_i) = \min\{\text{dist}(v, v_i), \text{dist}(v, v_k) + w\langle v_k, v_i \rangle\}$;

6.5 最短路径



6-5-1 Dijkstra算法

3. Dijkstra算法实现



当前的最短路径Path:

$\langle v_0, v_1 \rangle 10$

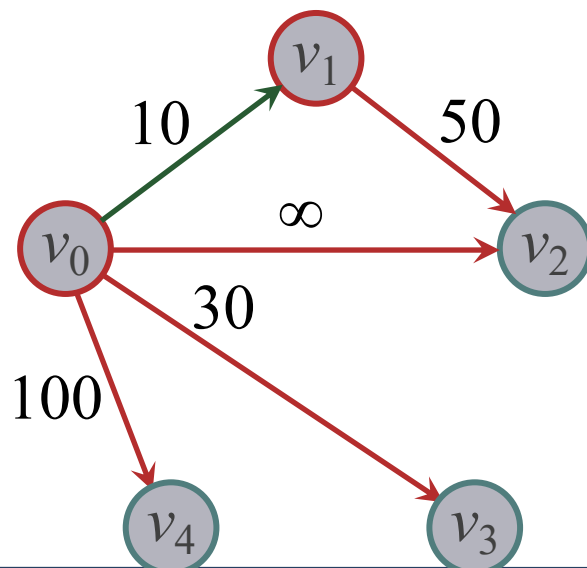
$\langle v_0, v_2 \rangle \infty$

$\langle v_0, v_3 \rangle 30$

$\langle v_0, v_4 \rangle 100$

dist[n]

0	10	∞	30	100
---	----	----------	----	-----



当前的最短路径Path:

$\langle v_0, v_1, v_2 \rangle 60$

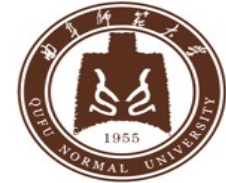
$\langle v_0, v_3 \rangle 30$

$\langle v_0, v_4 \rangle 100$

dist[n]

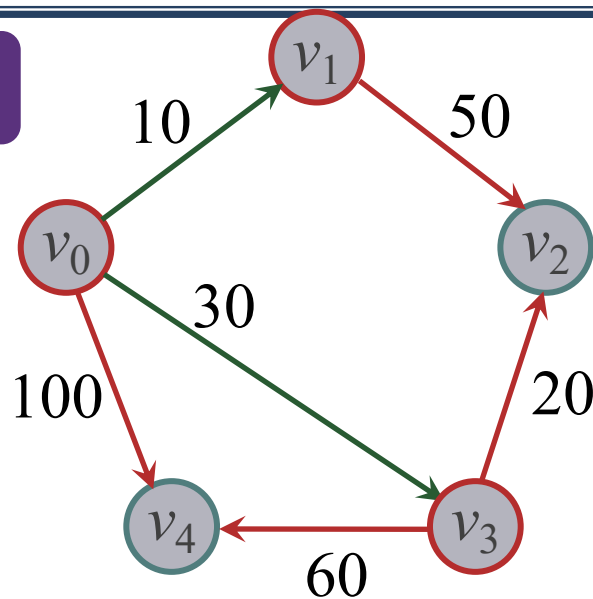
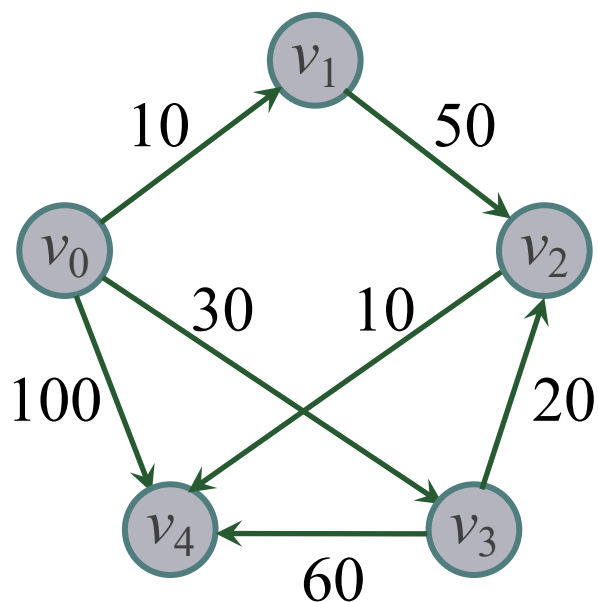
0	10	60	30	100
---	----	----	----	-----

6.5 最短路径



6-5-1 Dijkstra算法

3. Dijkstra算法实现



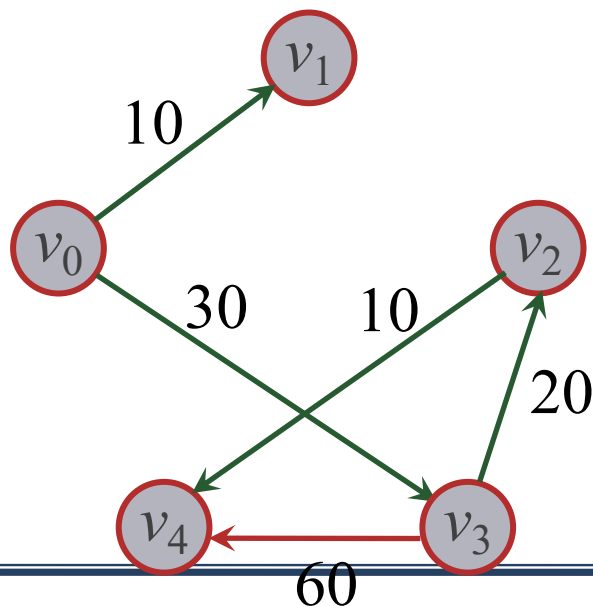
当前的最短路径Path :

$\langle v_0, v_3, v_2 \rangle 50$

$\langle v_0, v_3, v_4 \rangle 90$

dist[n]

0	10	50	30	90
---	----	----	----	----



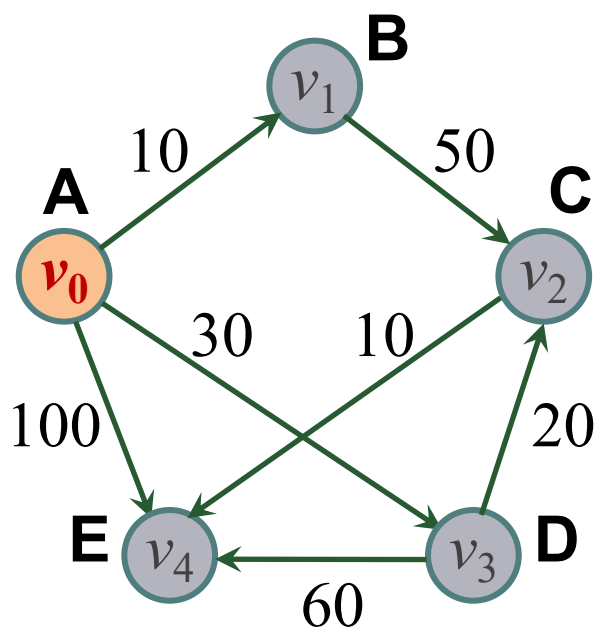
当前的最短路径Path :

$\langle v_0, v_3, v_2, v_4 \rangle 60$

dist[n]

0	10	50	30	60
---	----	----	----	----

Dijkstra算法



图

	v_0	v_1	v_2	v_3	v_4
v_0	0	10	∞	30	100
v_1	∞	0	50	∞	∞
v_2	∞	∞	0	∞	10
v_3	∞	∞	20	0	60
v_4	∞	∞	∞	∞	0

邻接矩阵 $\infty=1000$

终点: v_1 v_2 v_3 v_4 S

num=1, k =1

dist:	10	∞	30	100	$\{v_0\}$
path:	v_0, v_1	v_0, v_2	v_0, v_3	v_0, v_4	

num=2, k =3

dist:	0	60	30	100	$\{v_0, v_1\}$
path:	v_0, v_1	v_0, v_1, v_2	v_0, v_3	v_0, v_4	

局部最优

num=3, k =2

dist:	0	50	0	90	$\{v_0, v_1, v_3\}$
path:	v_0, v_1	v_0, v_3, v_2	v_0, v_3	v_0, v_3, v_4	

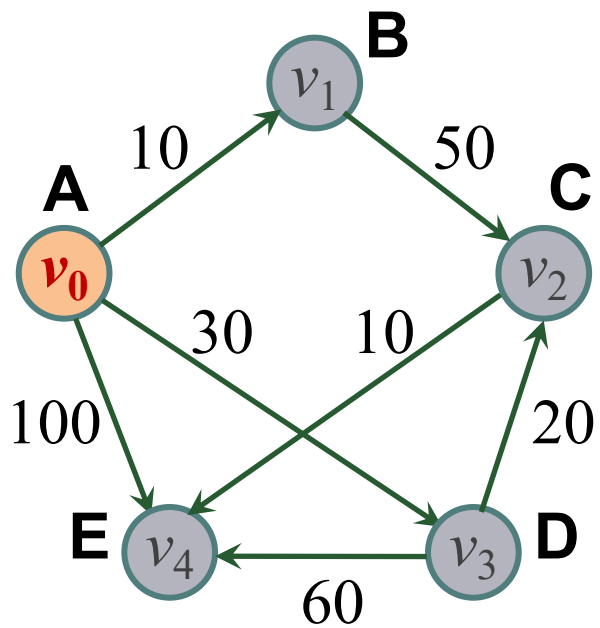
num=4, k =2

dist:	0	0	0	60	$\{v_0, v_1, v_3, v_2\}$
path:	v_0, v_1	v_0, v_3, v_2	v_0, v_3	v_0, v_3, v_2, v_4	

num=5, k =4

$\{v_0, v_1, v_3, v_2, v_4\}$

Dijkstra算法



图

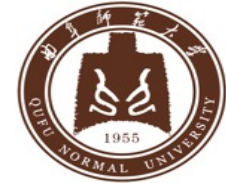
	v_0	v_1	v_2	v_3	v_4
v_0	0	10	∞	30	100
v_1	∞	0	50	∞	∞
v_2	∞	∞	0	∞	10
v_3	∞	∞	20	0	60
v_4	∞	∞	∞	∞	0

邻接矩阵 $\infty=1000$

终点:	v_1	v_2	v_3	v_4	
dist:	10	∞	30	100	S num=1, k =1 v_0
path:	v_0, v_1	v_0, v_2	v_0, v_3	v_0, v_4	
dist:	0	60	30	100	num=2, k =3 v_0, v_1
path:	v_0, v_1	v_0, v_1, v_2	v_0, v_3	v_0, v_4	
		局部最优			
dist:	0	50	0	90	num=3, k =2 v_0, v_2, v_3
path:	v_0, v_1	v_0, v_3, v_2	v_0, v_3	v_0, v_3, v_4	

```
for (num = 1; num < vertexNum; num++)
{
    for (k = 0, i = 0; i < vertexNum; i++)
        if ((dist[i] != 0) && (dist[i] < dist[k])) k = i;
    cout << path[k] << dist[k];
    for (i = 0; i < vertexNum; i++)
        if (dist[i] > dist[k] + edge[k][i]) {
            dist[i] = dist[k] + edge[k][i]; path[i] = path[k] + vertex[i];
        }
    dist[k] = 0;
}
```

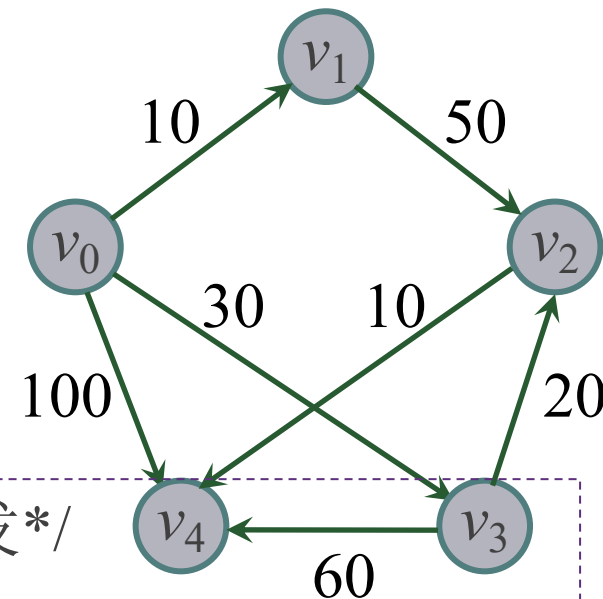
6.5 最短路径



6-5-1 Dijkstra算法

3. Dijkstra算法实现

下标 终点	1 v_1	2 v_2	3 v_3	4 v_4	S
dist	10	∞	30	100	
path	v_0, v_1	v_0, v_2	v_0, v_3	v_0, v_4	v_0



```
void Dijkstra(int v)
```

```
{
```

```
    int i, k, num, dist[MaxSize];
```

```
    string path[MaxSize];
```

```
    for (i = 0; i < vertexNum; i++)
```

```
    {
```

```
        dist[i] = edge[v][i];
```

```
        path[i] = vertex[v] + vertex[i];
```

```
    }
```

```
/*从源点v出发*/
```

```
/*初始化数组dist[n]和path[n]*/
```

```
/*字符串连接*/
```

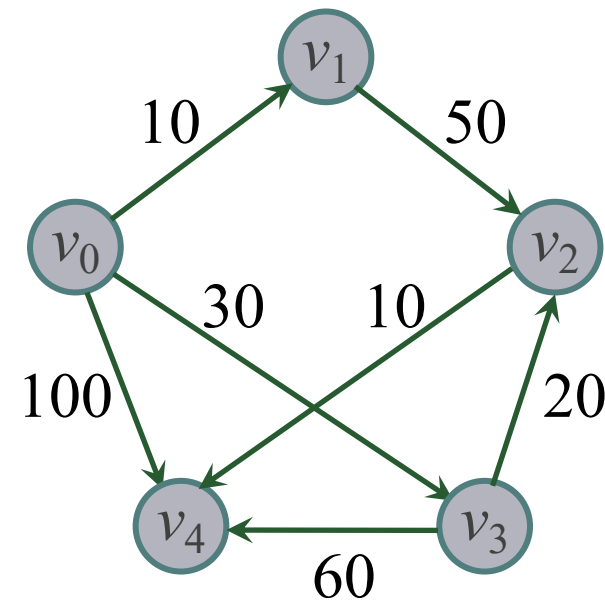
6.5 最短路径

6-5-1 Dijkstra算法



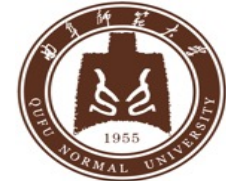
3. Dijkstra算法实现

下标 终点	1 v_1	2 v_2	3 v_3	4 v_4	S
dist path	10 v_0, v_1	∞ v_0, v_2	30 v_0, v_3	100 v_0, v_4	v_0
dist path					v_0, v_1
dist path	<pre>for (num = 1; num < vertexNum; num++) { for (k = 0, i = 0; i < vertexNum; i++) if ((dist[i] != 0) && (dist[i] < dist[k])) k = i; cout << path[k] << dist[k]; }</pre>				
dist path					
dist path					
path					



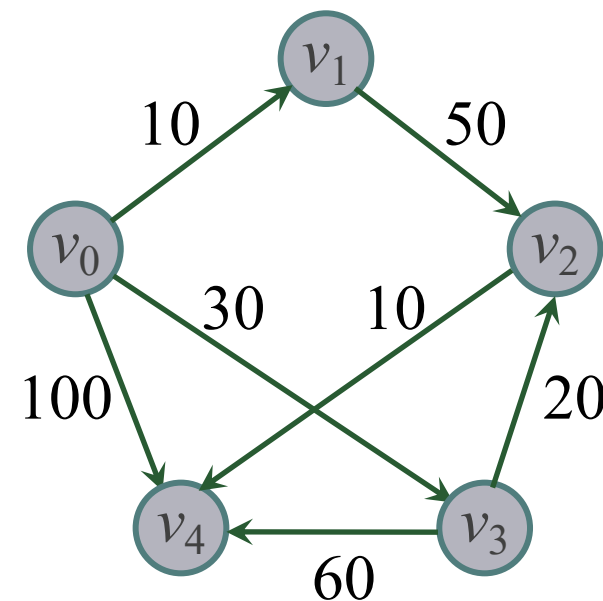
6.5 最短路径

6-5-1 Dijkstra算法



3. Dijkstra算法实现

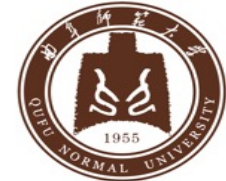
下标 终点	1 v_1	2 v_2	3 v_3	4 v_4	S
dist path	10 v_0, v_1	∞ v_0, v_2	30 v_0, v_3	100 v_0, v_4	v_0
dist path	0 v_0, v_1	60 v_0, v_1, v_2	30 v_0, v_3	100 v_0, v_4	v_0, v_1
dist path	<pre>for (num = 1; num < vertexNum; num++) { for (i = 0; i < vertexNum; i++) if (dist[i] > dist[k] + edge[k][i]) { dist[i] = dist[k] + edge[k][i]; path[i] = path[k] + vertex[i]; } dist[k] = 0; }</pre>				
dist path					
dist path					
dist path					



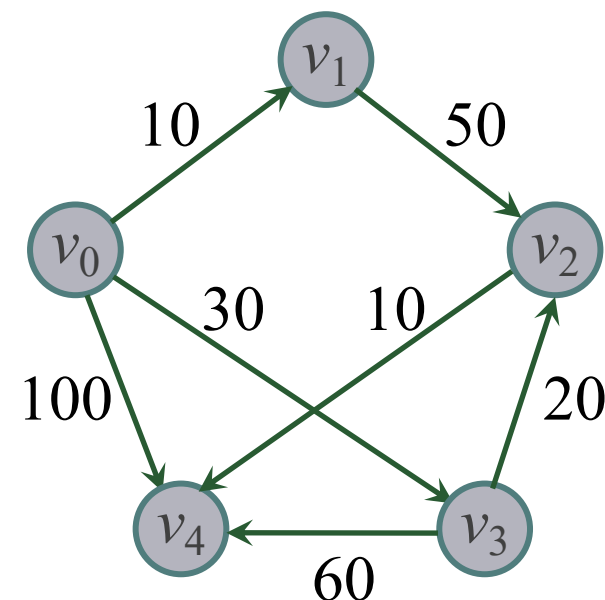
6.5 最短路径

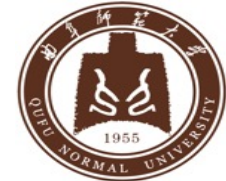
3. Dijkstra算法实现

6-5-1 Dijkstra算法



下标 终点	1 v_1	2 v_2	3 v_3	4 v_4	S
dist path	10 v_0, v_1	∞ v_0, v_2	30 v_0, v_3	100 v_0, v_4	v_0
dist path	0 v_0, v_1	60 v_0, v_1, v_2	30 v_0, v_3	100 v_0, v_4	v_0, v_1
dist path		50 v_0, v_3, v_2	0 v_0, v_3	90 v_0, v_3, v_4	v_0, v_2, v_3
dist path		0 v_0, v_3, v_2		60 v_0, v_3, v_2, v_4	v_0, v_1, v_3, v_2
dist path				0 v_0, v_3, v_2, v_4	v_0, v_1, v_3, v_2, v_4





```
void Dijkstra(int v)
```

/*从源点v出发*/

```
{
```

```
    int i, k, num, dist[MaxSize]; string path[MaxSize];
```

```
    for (i = 0; i < vertexNum; i++)
```

```
    {
```

```
        dist[i] = edge[v][i]; path[i] = vertex[v] + vertex[i];
```

```
    }
```

} $O(n)$

```
    for (num = 1; num < vertexNum; num++)
```

```
    {
```

```
        for (k = 0, i = 0; i < vertexNum; i++)
```

```
            if ((dist[i] != 0) && (dist[i] < dist[k])) k = i; //找最小距离对应顶点k
```

} $O(n)$

```
        cout << path[k] << dist[k];
```

```
        for (i = 0; i < vertexNum; i++)
```

```
            if (dist[i] > dist[k] + edge[k][i]) {
```

```
                dist[i] = dist[k] + edge[k][i]; path[i] = path[k] + vertex[i];
```

} $O(n)$

```
            }
```

```
        dist[k] = 0; //顶点k加入S集合
```

```
    }
```

```
}
```



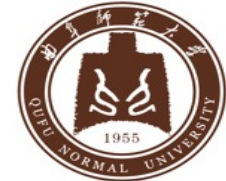
时间复杂度?



$O(n^2)$

6.5 最短路径

6-5-2 Floyd算法



问题提出

每一对顶点的最短路径问题

【问题】 给定带权有向图 $G = (V, E)$, 对任意顶点 v_i 和 v_j ($i \neq j$) , 求从顶点 v_i 到顶点 v_j 的最短路径

算法1: 每次以一个顶点为源头, 重复执行 Dijkstra 算法 n 遍, 便可得到每一对顶点之间的最短路径。

算法时间复杂度: $O(n^3)$

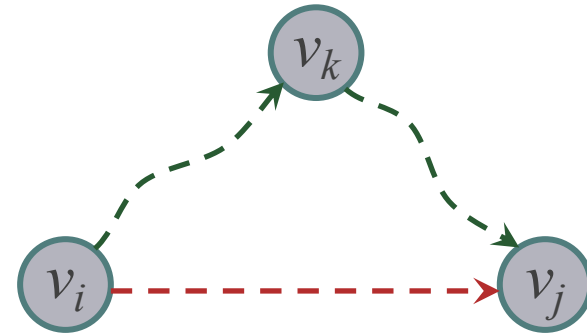
算法2: Floyd 算法

算法时间复杂度: $O(n^3)$

6.5 最短路径

6-5-2 Floyd算法

1. Floyd算法思想



$w\langle v_i, v_j \rangle$: 从顶点 v_i 到顶点 v_j 的权值

$\text{dist}_k(v_i, v_j)$: 从顶点 v_i 到顶点 v_j 经过的顶点编号不大于 k 的最短路径长度

算法: Floyd算法

输入: 带权有向图 $G=(V, E)$

输出: 每一对顶点的最短路径

1. 初始化: 假设从 v_i 到 v_j 的弧是最短路径, 即 $\text{dist}_1(v_i, v_j) = w\langle v_i, v_j \rangle$;
2. 循环变量 k 从 $0 \sim n-1$ 进行 n 次迭代:

$$\text{dist}_k(v_i, v_j) = \min \{ \text{dist}_{k-1}(v_i, v_j), \text{dist}_{k-1}(v_i, v_k) + \text{dist}_{k-1}(v_k, v_j) \}$$

6.5 最短路径

6-5-2 Floyd算法

2. Floyd算法存储结构

 如何存储dist? 如何存储带权有向图? \Rightarrow 邻接矩阵

$$\begin{cases} \text{dist}_1(v_i, v_j) = w\langle v_i, v_j \rangle \\ \text{dist}_k(v_i, v_j) = \min\{\text{dist}_{k-1}(v_i, v_j), \text{dist}_{k-1}(v_i, v_k) + \text{dist}_{k-1}(v_k, v_j)\} \end{cases}$$

算法: Floyd算法

输入: 带权有向图 $G=(V, E)$

输出: 每一对顶点的最短路径

1. 初始化: 假设从 v_i 到 v_j 的弧是最短路径, 即 $\text{dist}_1(v_i, v_j) = w\langle v_i, v_j \rangle$;
2. 循环变量 k 从 $0 \sim n-1$ 进行 n 次迭代:

$$\text{dist}_k(v_i, v_j) = \min\{\text{dist}_{k-1}(v_i, v_j), \text{dist}_{k-1}(v_i, v_k) + \text{dist}_{k-1}(v_k, v_j)\}$$

6.5 最短路径

6-5-2 Floyd算法

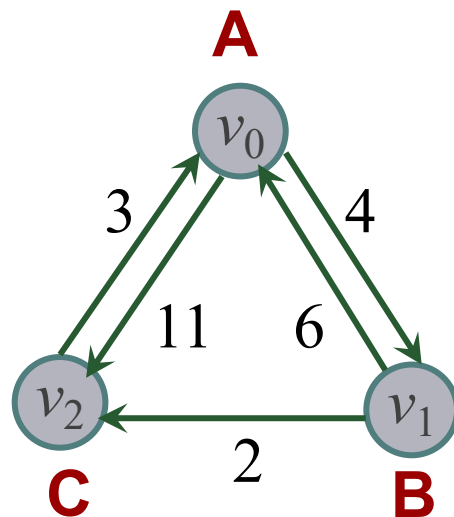


3. Floyd算法实现

初始化

$$\text{dist}_1 = \begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{pmatrix}$$

$$\text{path}_1 = \begin{pmatrix} & \text{AB} & \text{AC} \\ \text{BA} & & \text{BC} \\ \text{CA} & \text{CB} & \end{pmatrix}$$



```
void Floyd( )           string ch[ ]={"A","B","C"};
{
    int i, j, k, dist[MaxSize][MaxSize];
    string path[MaxSize][MaxSize];
    for (i = 0; i < vertexNum; i++)
        for (j = 0; j < vertexNum; j++)
        { dist[i][j] = edge[i][j];
          if (dist[i][j] != 1000)
              path[i][j] = vertex[i] + vertex[j];
          else path[i][j] = "";
        }
}
```


6.5 最短路径

6-5-2 Floyd算法



3. Floyd算法实现

经过 v_0

$$\text{dist}_0 = \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

经过 v_1

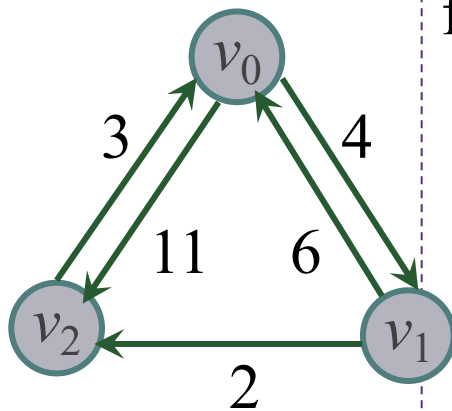
$$\text{dist}_1 = \begin{bmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

经过 v_2

$$\text{dist}_2 = \begin{bmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

初始化

$$\text{dist}_{-1} = \begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix}$$

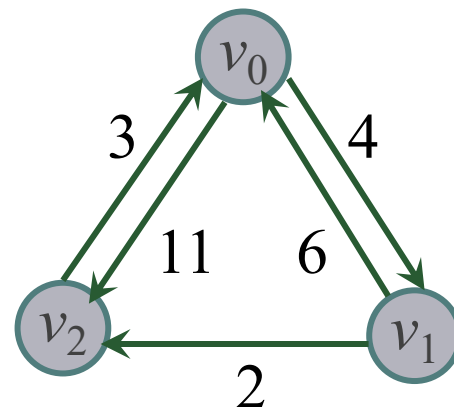


```
for (k = 0; k < vertexNum; k++)  
  for (i = 0; i < vertexNum; i++)  
    for (j = 0; j < vertexNum; j++)  
      if (dist[i][k] + dist[k][j] < dist[i][j])  
      {  
        dist[i][j] = dist[i][k] + dist[k][j];  
        path[i][j] = path[i][k] + path[k][j];  
      }
```

3. Floyd算法实现

```
template <class DataType>
void MGraph<DataType> :: Floyd( )
{
    int i, j, k, dist[MaxSize][MaxSize];
    string path[MaxSize][MaxSize];
    for (i = 0; i < vertexNum; i++)                //初始化矩阵dist和path
        for (j = 0; j < vertexNum; j++)
        {
            dist[i][j] = edge[i][j];
            if (dist[i][j] != 1000)                  //假设1000为边上权的最大值
                path[i][j] = vertex[i] + vertex[j];    //+为字符串连接操作
            else path[i][j] = "";
        }
    for (k = 0; k < vertexNum; k++)                //进行n次迭代
        for (i = 0; i < vertexNum; i++)
            for (j = 0; j < vertexNum; j++)
                if (dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                    path[i][j] = path[i][k] + path[k][j];    //+为字符串连接操作
                }
    for (i = 0; i < vertexNum; i++)
    {
        for (j = 0; j < vertexNum; j++)
            cout << path[i][j] << ":" << dist[i][j] << "\t";
        cout << endl;
    }
}
```

AA:0	AB:4	ABBC:6
BCCA:5	BB:0	BC:2
CA:3	CAAB:7	CC:0



$O(n^2)$

$O(n^3)$

string ch[]={"A","B","C"};



时间复杂度?



$O(n^3)$

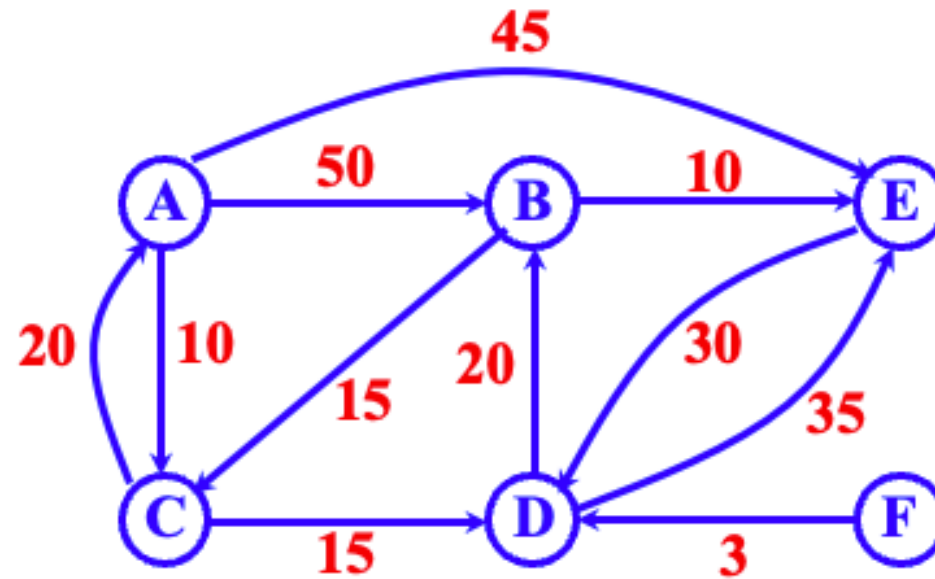
1. 掌握Dijkstra算法及实现方法
2. 理解Floyd算法及实现方法

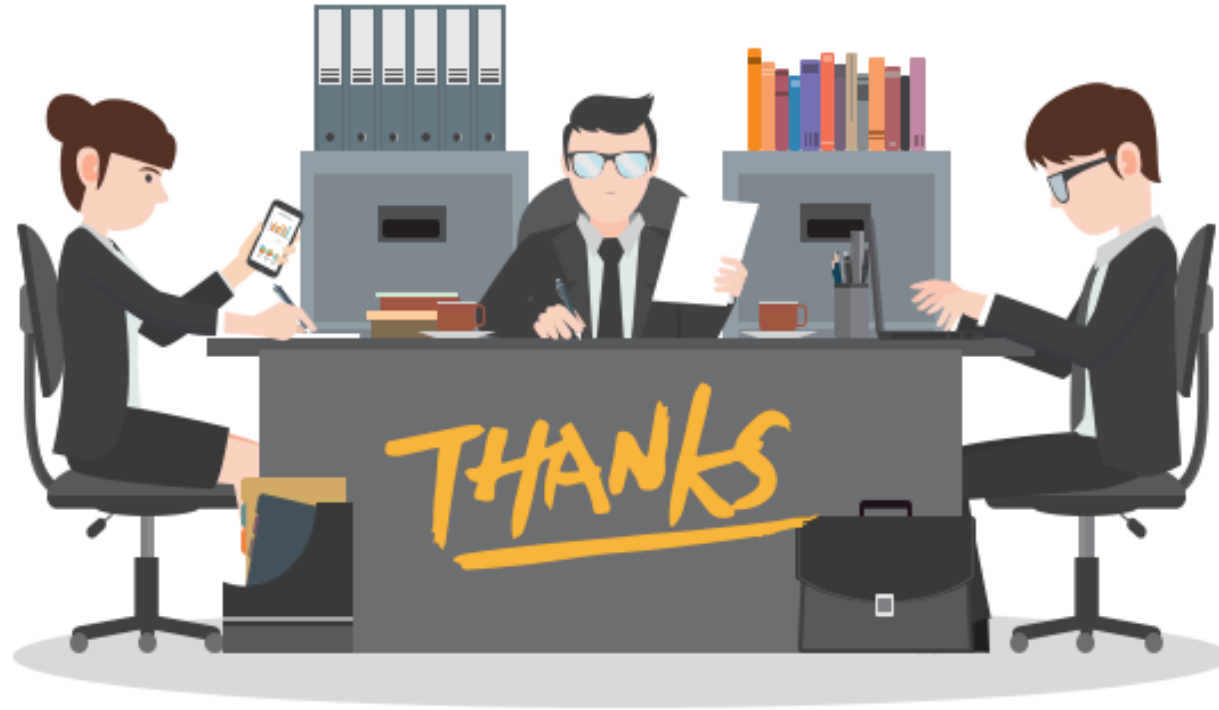
<https://c.runoob.com/compile/12/>

<https://wandbox.org/>

作业

1. 求 A 到其它各点的最短路径及长度 要求：中间过程
2. 求所有顶点对之间的最短路径及长度 要求：中间过程





Thank You !

Q & A