第11章 网络分析算法

主要内容

- 网络分析概述
- 网络分析实现
- 连通性分析实现

1. 网络分析(NETWORK)

网络是用于实现资源运输和信息交流的一系列相互联接的线性特征组合。

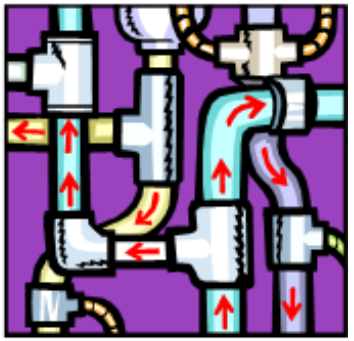
1.1 什么是网络分析?

在GIS中，网络分析是指依据网络拓扑关系（结点与弧段拓扑、弧段的连通性），通过考察网络元素的空间及属性数据，以数学理论模型为基础，对网络的性能特征进行多方面研究的一种分析计算。

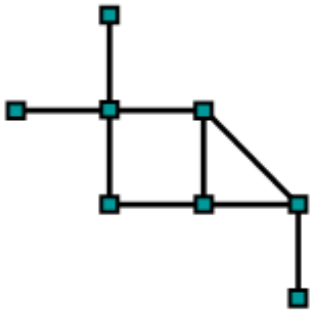
- 由一系列相互连通的点和线组成，用来描述地理要素（资源）的流动情况。

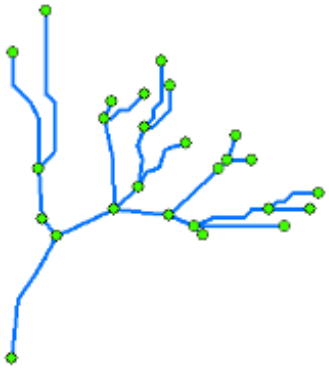
道路

流水



实体





社交网

道路网

电网

河网

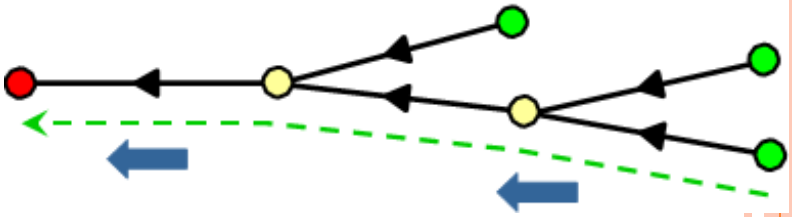
1. 网络分析(NetWork)

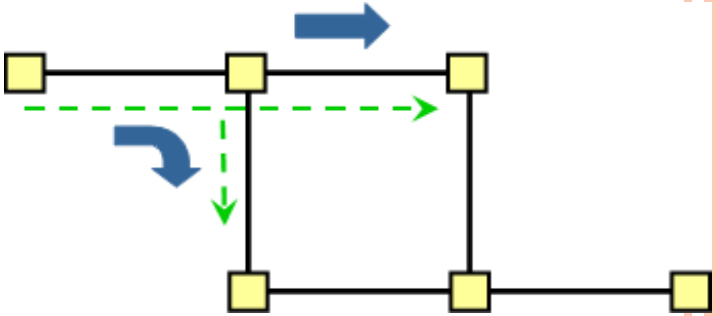
1.2 网络分析主要内容

- ✓ 网络数据模型
- ✓ 网络分析功能
  - 网络跟踪(Trace)
  - 路径分析(PathFinding)
  - 资源分配(Allocation)
  - 其他网络分析

地理网络的类型

- 定向网络
  - 流向由源（source）至汇（sink）
  - 网络中流动的资源自身不能决定流向（如：水流、电流）
- 非定向网络
  - 流向不完全由系统控制
  - 网络中流动的资源可以决定流向
  - (如：交通系统)





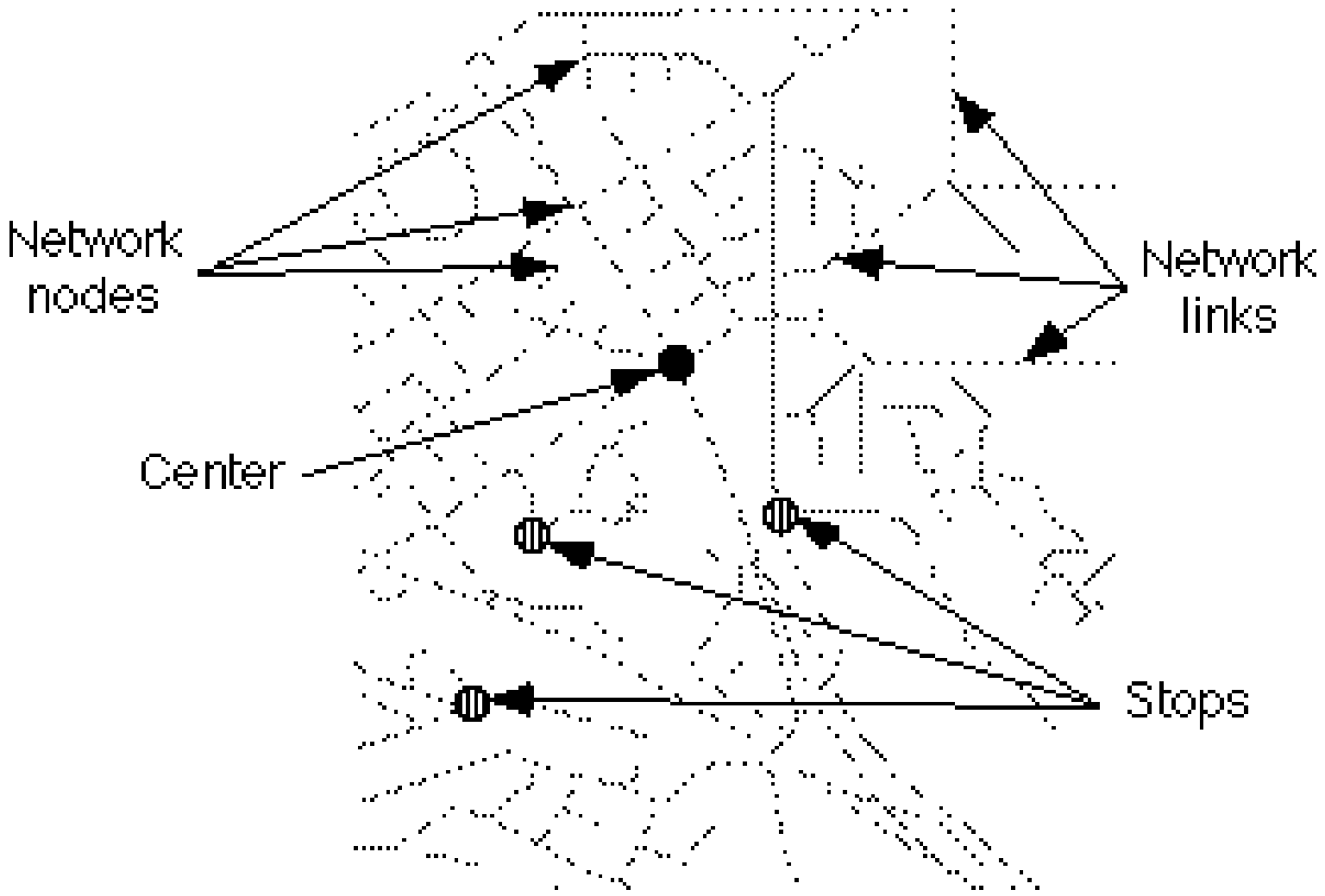
网络数据模型

- 网络模型是对现实世界网络的抽象。在模型中，网络由链(Link)、结点(Node)、站点(Stop)、中心(Center)和转向点(Turn)组成。
- 建立一个好的网络模型的关键是清楚地认识现实网络的各种特性与以网络模型的要素(Link, Node, Stop, Center, Turn)表示的特性之间的关系。

✓ 网络组成要素

- 结点(Node)：网络中任意两条线段的交点，属性如资源数量等
- 链(Link)：连接两个结点的弧段。供物体运营的通道，链间的连接关系由弧段-结点拓扑数据结构来表达。属性如资源流动的时间、速度等
- 中心(Center)：网络中位于结点处，具有沿着链收集和发放资源能力的设施，如邮局、电站、水库等
- 站点(Stop)：资源沿着网络路径流动时被分配或收集的位置，如邮件投放点、公共汽车站，属性如资源需求量
- 转向点(拐点，Turn)：链路相交处，资源流向发生改变的点

✓ 网络组成要素



网络分析功能

- 网络跟踪
- 路径分析
- 资源分配
- 定位配置分析
- 地址地理编码

✓网络跟踪(TRACE)

- 概念：网络中用于研究网络中资源和信息的流向就是网络跟踪的过程。

  在点污染研究中，可以跟踪污染物从污染源开始，沿河流向下游扩散的过程。在电网应用中，可以根据不同开关的开、关状态，确定电力的流向。

- 数据结构的拓扑基础：网络跟踪中涉及的一个重要概念是“连通性”(Connectivity)，这定义了网络中弧段与弧段的连接方式，也决定了资源与信息在网络中流动时的走向。

✓路径分析

- 在网络分析过程中，路径系统起着相当重要的作用。事实上很多网络分析的结果都是以路径系统的形式体现出来的。
- 内容：路径分析是用于模拟两个或两个以上地点之间资源流动的路径寻找过程。当选择了起点、终点和路径必须通过的若干中间点后，就可以通过路径分析功能按照指定的条件寻找最优路径

☑ 路径选择(PathFinding)

- 应用：在远距离送货、物资派发、急救服务和邮递等服务中，经常需要在一次行程中同时访问多个站点(收货方、邮件主人、物资储备站等)，如何寻找到一个最短和最经济的路径，保证访问到所有站点，同时最快最省地完成一次行程，这是很多机构经常遇到的问题。
- 这类分析中，道路网络的不同弧段(网络模型中的Link)有不同的影响物流通过的因素， **路径选择**分析必须充分考虑到这些因素，在保证遍历需要访问的站点的同时，为用户寻找出一条最佳(距离、时间或费用等)的运行路径。

☑ 路径选择(PathFinding)

- 两种方式(Path和Tour)。ArcInfo有两个路径选择分析命令：Path和Tour。

  共同点：都是在网络中寻找遍历所有站点最经济的路径。

  区别：在遍历网络的所有站点过程中，处理站点的顺序有所不同。

  - PATH：必须按照指定的顺序访问网站中的所有站点。例如，救护车必须从急救中心(STOP 1)出发，然后前往事故地点(STOP 2)，然后负责将伤员送往最近的医院(STOP 3)，最后返回急救中心(STOP 4).

☑ 路径选择(PathFinding)

- TOUR：进行路径选择分析时，在保证在一次行程中访问所有站点的前提下，访问站点的次序是由TOUR自己决定的。因此TOUR分析的结果既包括所选择的路径，也包括它所确定的最优的访问次序。例如：卡车司机要在一天时间内向若干个站点送货，只要保证在当天内将货物送到每一个站点就可以了，先送哪个站点，后送哪个站点，完全由司机本人决定。TOUR就负责完成确定访问次序，并寻找最经济路径的任务。

资源分配(Allocation)

- 反映现实世界网络中资源的供需关系模型。可以解决资源的有效利用和合理分配；确定最近中心，实现最佳服务

- “供(Supply)”代表一定数量的资源或货物，它们位于被称之为“CENTER”的设施中。“需(Demand)”指对资源的利用。Allocate分析就是在空间中的一个或多个点之间分配资源的过程。
- 为了实现供需关系，在网络中必然存在资源的运输和流动。资源要么由供方送到需方，要么由需需方到供方索取。

☑关于ALLOCATE的两个例子

- Supply-To-Demand的例子：负荷设计、时间与距离损耗估算

  电能从电站产生，并通过电网传送到客户那里去。在这里，电站就是网络模型中的“Center”,因为它可以提供电力供应。电能的客户沿电网的线路(网络模型中的Link)分布，他们产生了“Demand”。在这种情况下，资源是通过网络由供方传输到需方来实现资源分配。可用来分析输电系统是否超载；停电的社会、经济影响估计等。

☑关于Allocate的两个例子

⇨ Demand-To-Supply的例子：学校选址

学校与学生的关系也构成一种在网络中供需分配关系。学校是资源提供方，它负责提供名额供适龄儿童入学。适龄儿童是资源的需求方，他们要求入学。作为需求方的适龄儿童沿街道网络分布，他们产生了对作为供给方的学校的资源--学生名额的需求。这种情况下，“资源”的流向是由适龄儿童前往学校

✓定位配置分析(选址和分区)

☑Location-Allocation(选址和分区)分析

Location-allocation分析是决定一个或多个服务设施的最优位置的过程，它的定位力求保证服务设施可以以最经济有效的方式为它所服务的人群提供服务。在此分析中，即有定位过程，也有资源分配过程。

✓定位配置分析(选址和分区)

☑定位配置分析的实质是线性规划问题。主要的算法包括：

- p-中心问题：在m个候选点中，选择p个供应点，为n个需求点服务，并使得从服务中心到需求点之间的距离（或时间、费用）最小。
- 中心服务范围的确定：中心服务范围是指一个服务设施在给定的时间或距离内，能够到达的区域。
- 中心资源的分配范围：资源分配就是将空间网络的边或者结点，按照中心的供应量及网络边和结点的需求量，分配给一个中心的过程，用来模拟空间网络上资源的供需关系(Allocation）。

✓地址编码与匹配(GeoCoding)

☑地址编码与匹配(GeoCoding)

利用人们习惯的地址(街道门牌号)信息确定它在地图上的确切位置的技术.

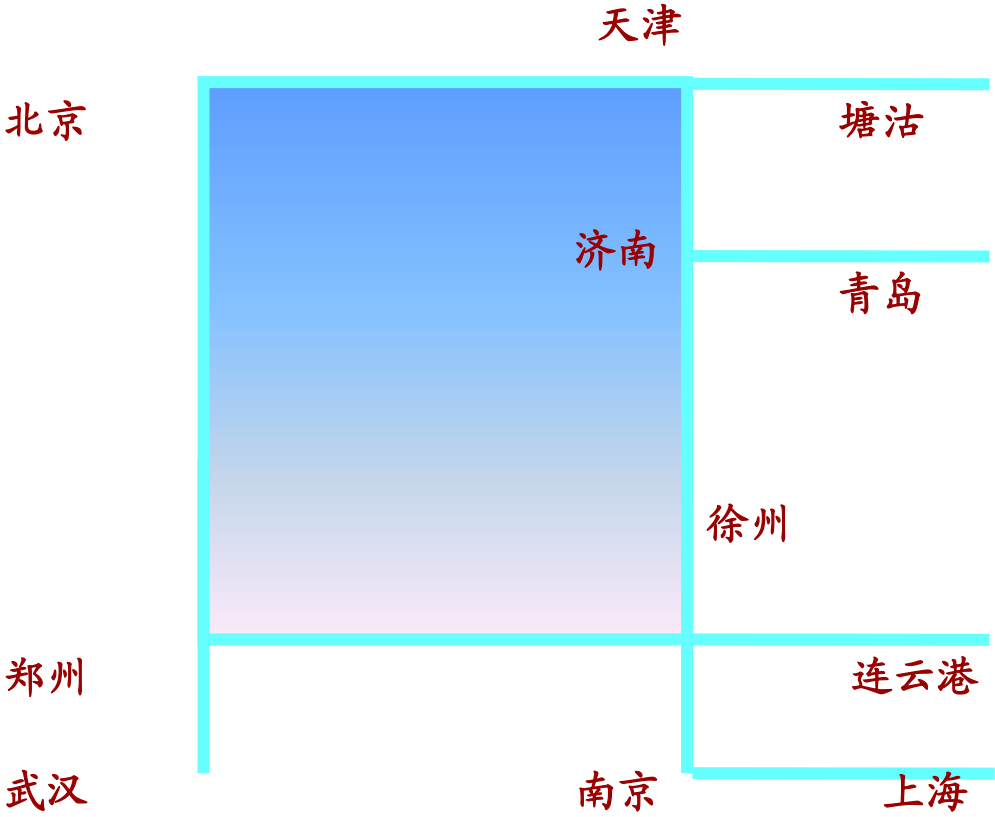
地址编码与匹配就是在含地址的表格数据与相关图层之间建立联系，并为表格数据创建一个相应的点要素层。当对表格数据进行编码后，就可以对表格数据进行空间定位查询和分析。

小结

- ✓ 网络数据模型
- ✓ 网络分析功能
  - 网络跟踪(Trace)
  - 路径分析(PathFinding)
  - 资源分配(Allocation)
  - 定位配置分析(Location-Allocation)
  - 地址地理编码 (GeoCoding)

2. 网络分析(NetWork)实现

- 网络分析中的数据结构
- 网络分析实现
  - 最佳路径分析



用点表示城市，用点与点之间的线表示城市之间的铁路线。

图的矩阵表示

图的矩阵表示主要方法有：权矩阵、邻接矩阵。

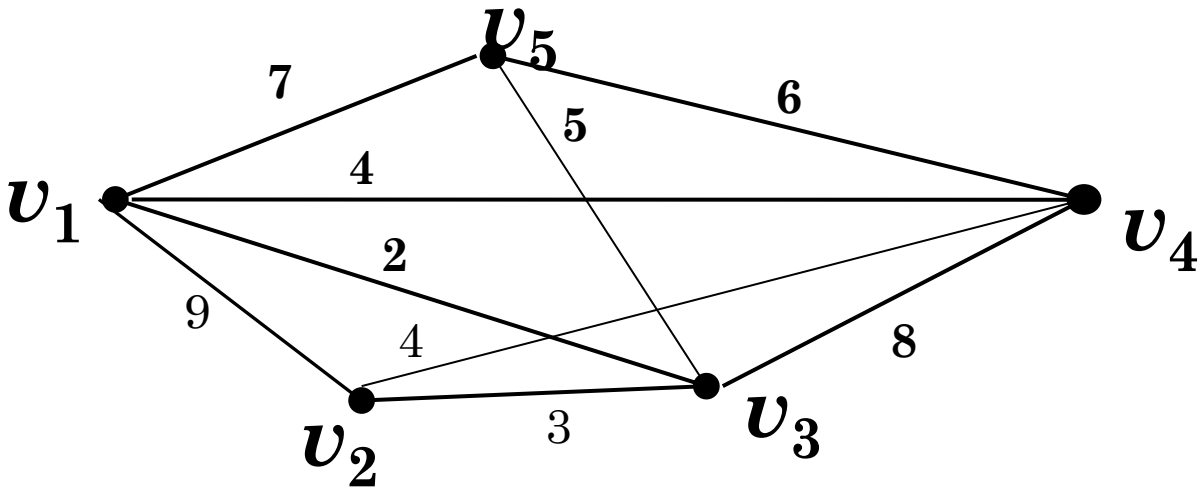
- **权矩阵**

网络图N=（V，E），边$[v_i, v_j]$的权为w_{ij}，构造矩阵

$$A=(a_{ij})_{n\times n}，其中$$

$$a_{ij}=\begin{cases} w_{ij} & [v_i,v_j]\in E \\ 0 & 其它 \end{cases}$$

称矩阵A为网络N的权矩阵。



其权矩阵为：

$$A=\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array}\begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{bmatrix} 0 & 9 & 2 & 4 & 7 \\ 9 & 0 & 3 & 4 & 0 \\ 2 & 3 & 0 & 8 & 5 \\ 4 & 4 & 8 & 0 & 6 \\ 7 & 0 & 5 & 6 & 0 \end{bmatrix} \end{array}$$

注：当G为无向图时，权矩阵为对称矩阵。

图的矩阵表示

- 邻接矩阵

图G=（V，E），$p=n$，构造矩阵

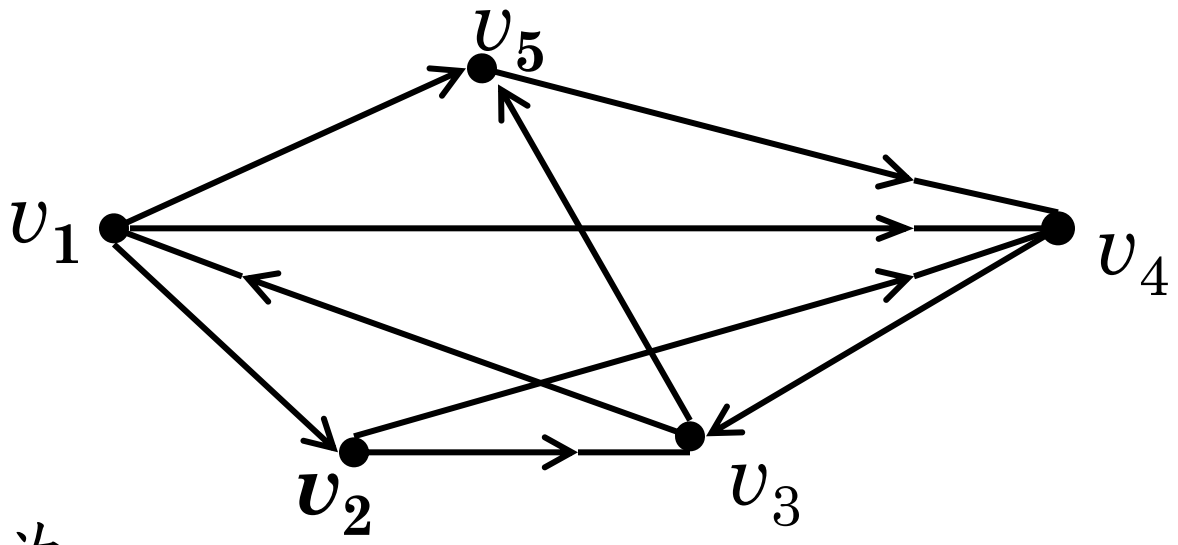
$$A=(a_{ij})_{n\times n},\ \text{其中}$$

$$a_{ij}=\begin{cases}1 & [v_i,v_j]\in E\\ 0 & \text{其它}\end{cases}$$

v_i与v_j不相邻

称矩阵A为G的邻接矩阵。

?



其邻接矩阵为:

$$A=\begin{bmatrix}0&1&0&1&1\\0&0&1&1&0\\1&0&0&0&1\\0&0&1&0&0\\0&0&0&1&0\end{bmatrix}$$

注: 当G为无向图时，邻接矩阵为对称矩阵。

最短路径分析

- 固定起点的最短路

  Dijkstra(狄克斯拉)(荷兰)算法

  逐次逼近算法(Ford(美国)算法)

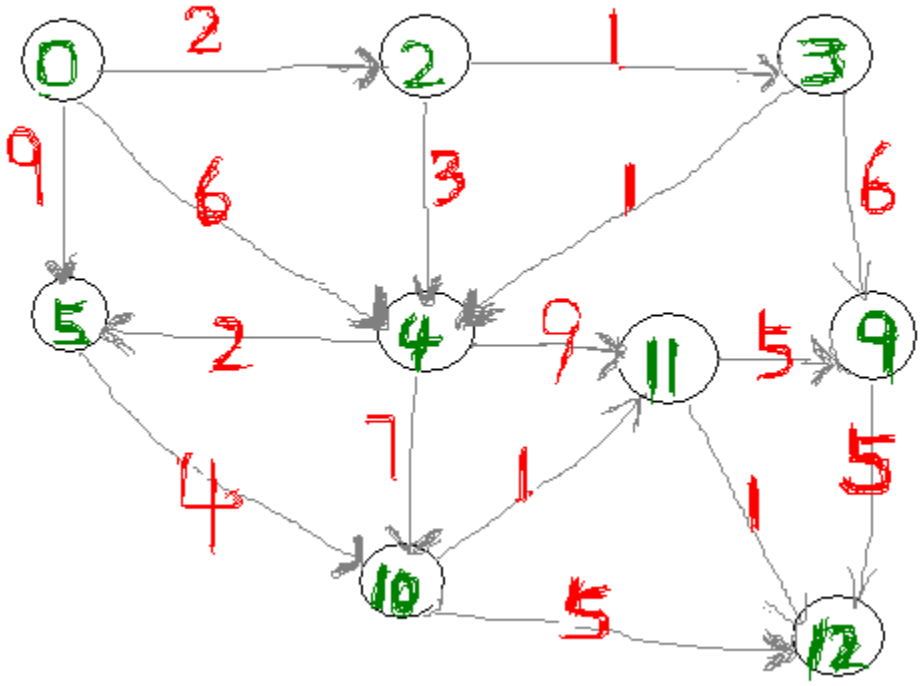
- 每对顶点之间的最短路

  路矩阵算法(Floyd(佛洛伊德)算法)

最佳路径分析

- Dijkstra算法（狄克斯特拉算法）

求从 0 节点到 12 节点 找一条最优路径

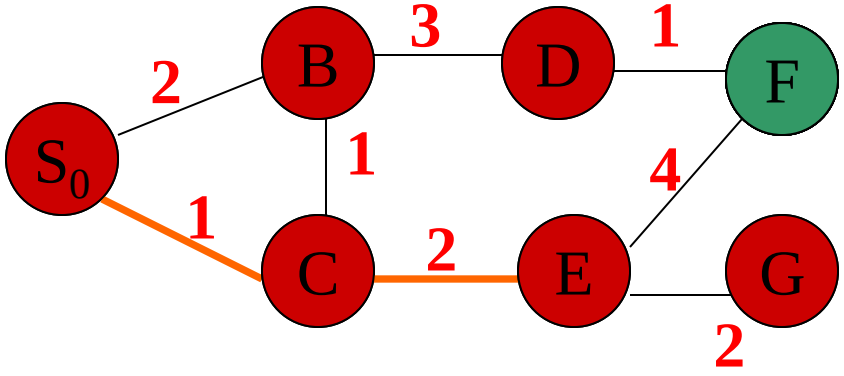


绿色的是节点

红色的为权值

箭头为可通行的标志

DIJKSTRA最短路径算法过程（无向图）



CLOSE表中记录的节点顺序有无规律?

OPEN表

节点号	父节点编码	路径

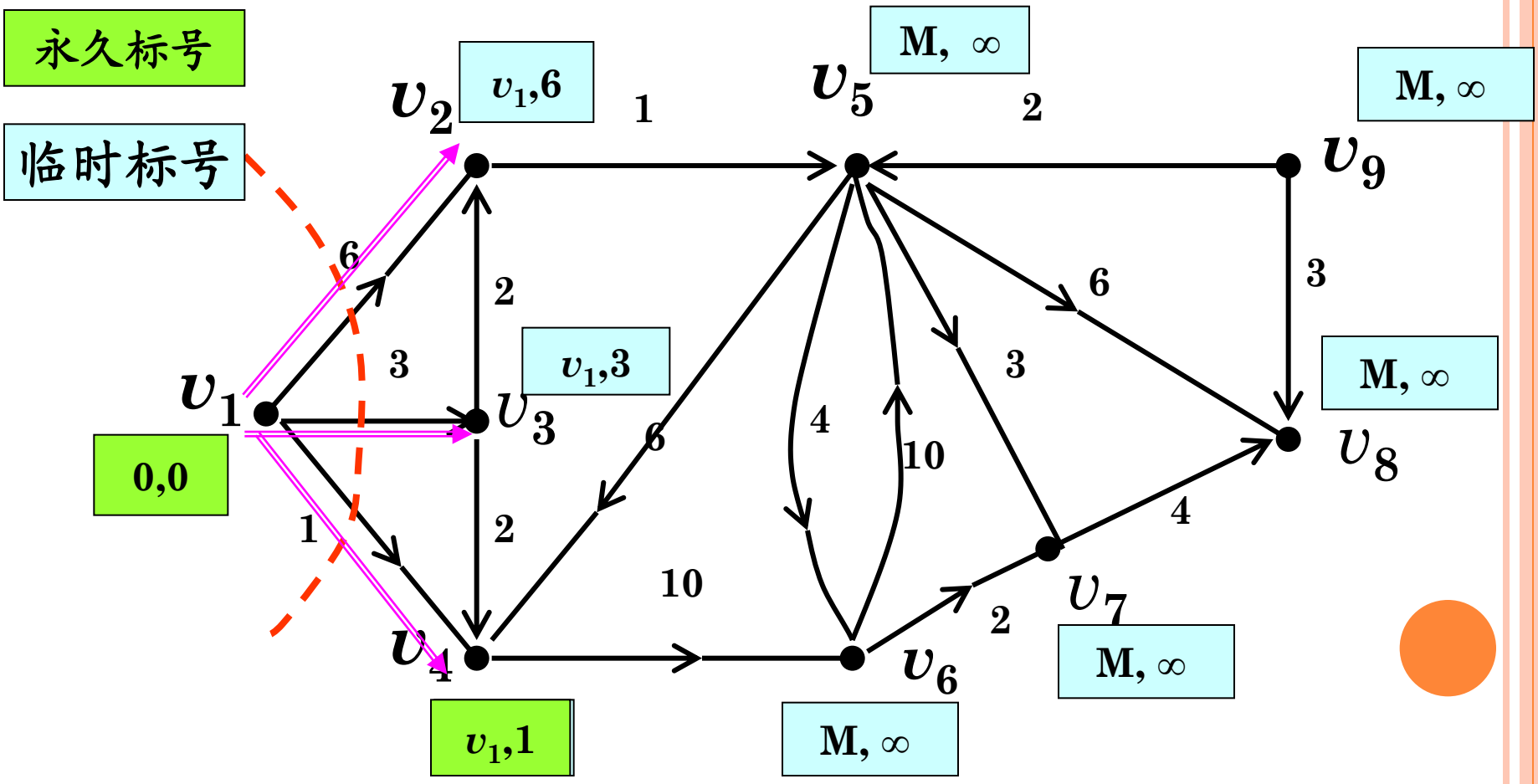
变量n　6

CLOSE表

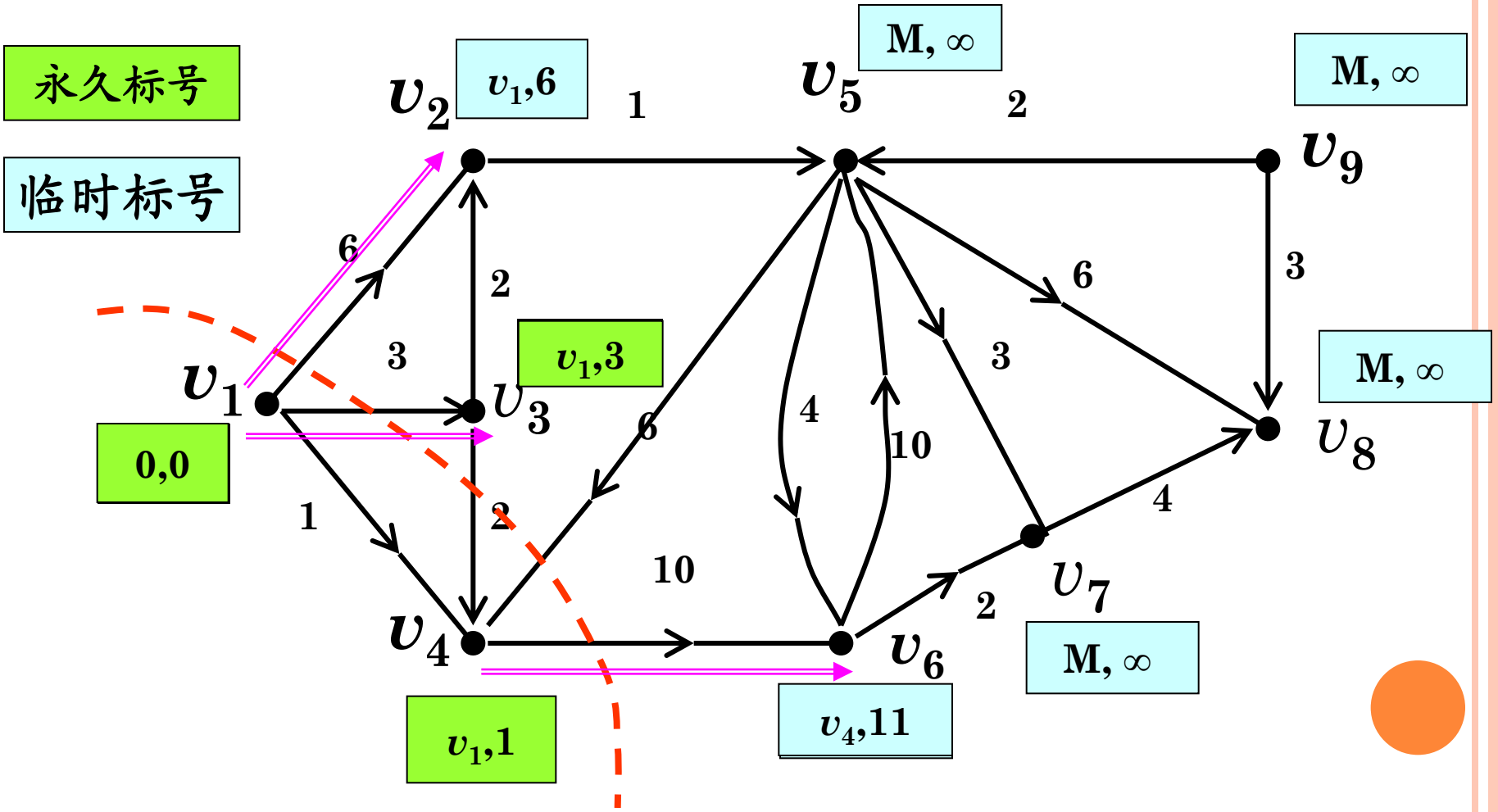
编号	节点号	父节点编码	路径
0	S_0	0	0
1	C	0	1
2	B	0	2
3	E	1	3
4	D	2	5
5	G	3	5
6	F	4	6

DIJKSTRA最短路径算法过程（有向图）

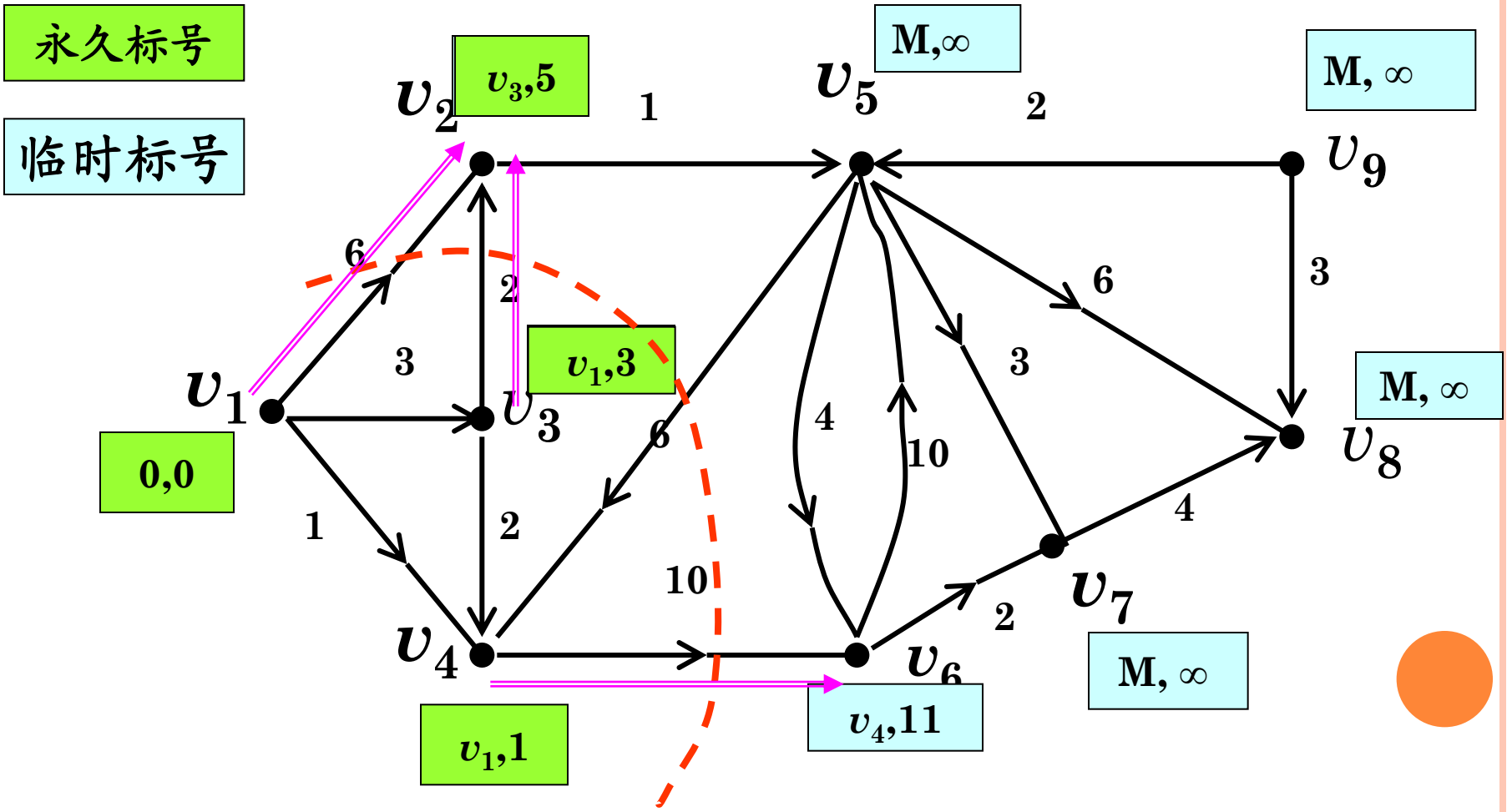
- 计算v1出发到v8去，使总费用最小的旅行路线



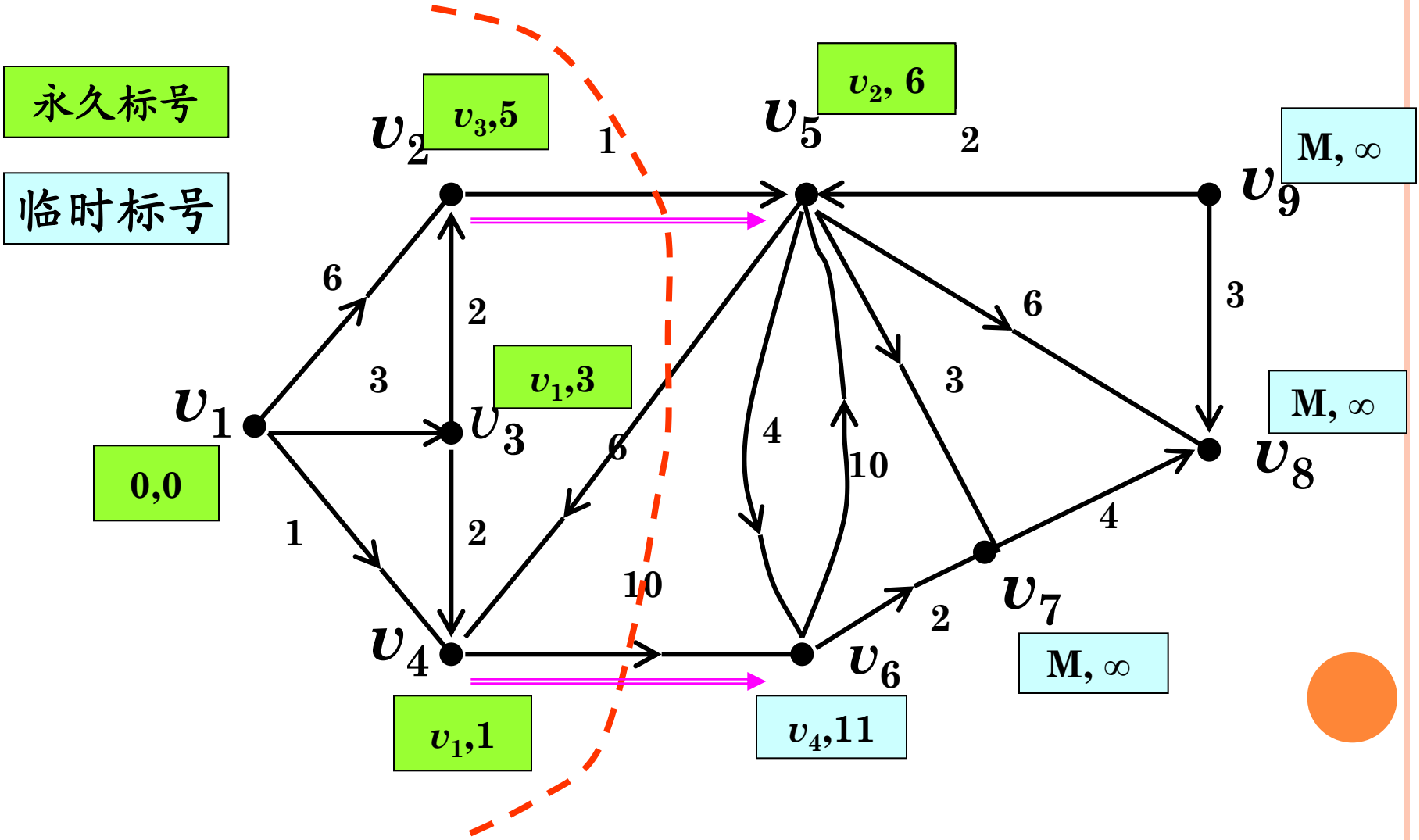
DIJKSTRA最短路径算法过程（有向图）



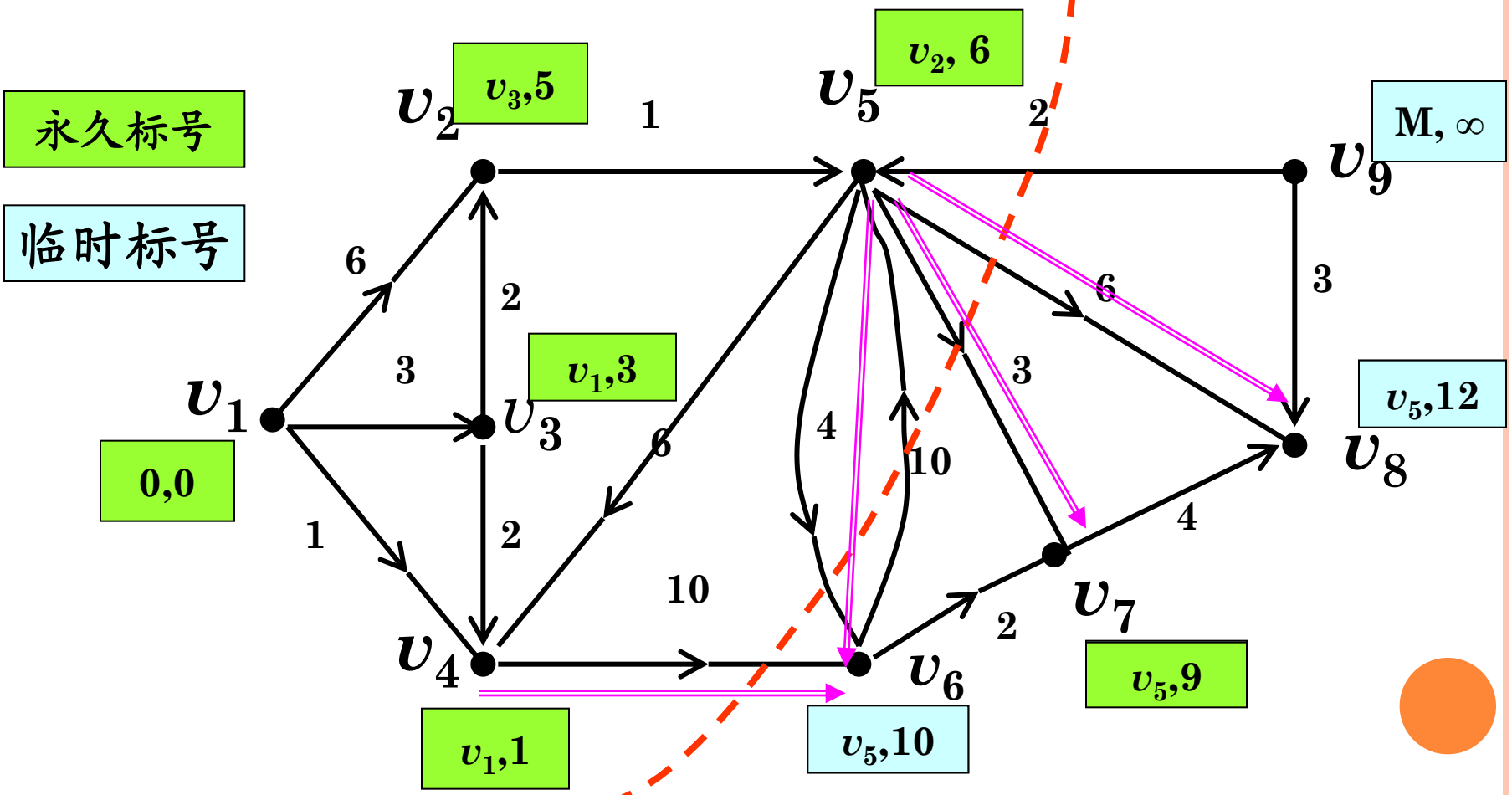
DIJKSTRA最短路径算法过程（有向图）



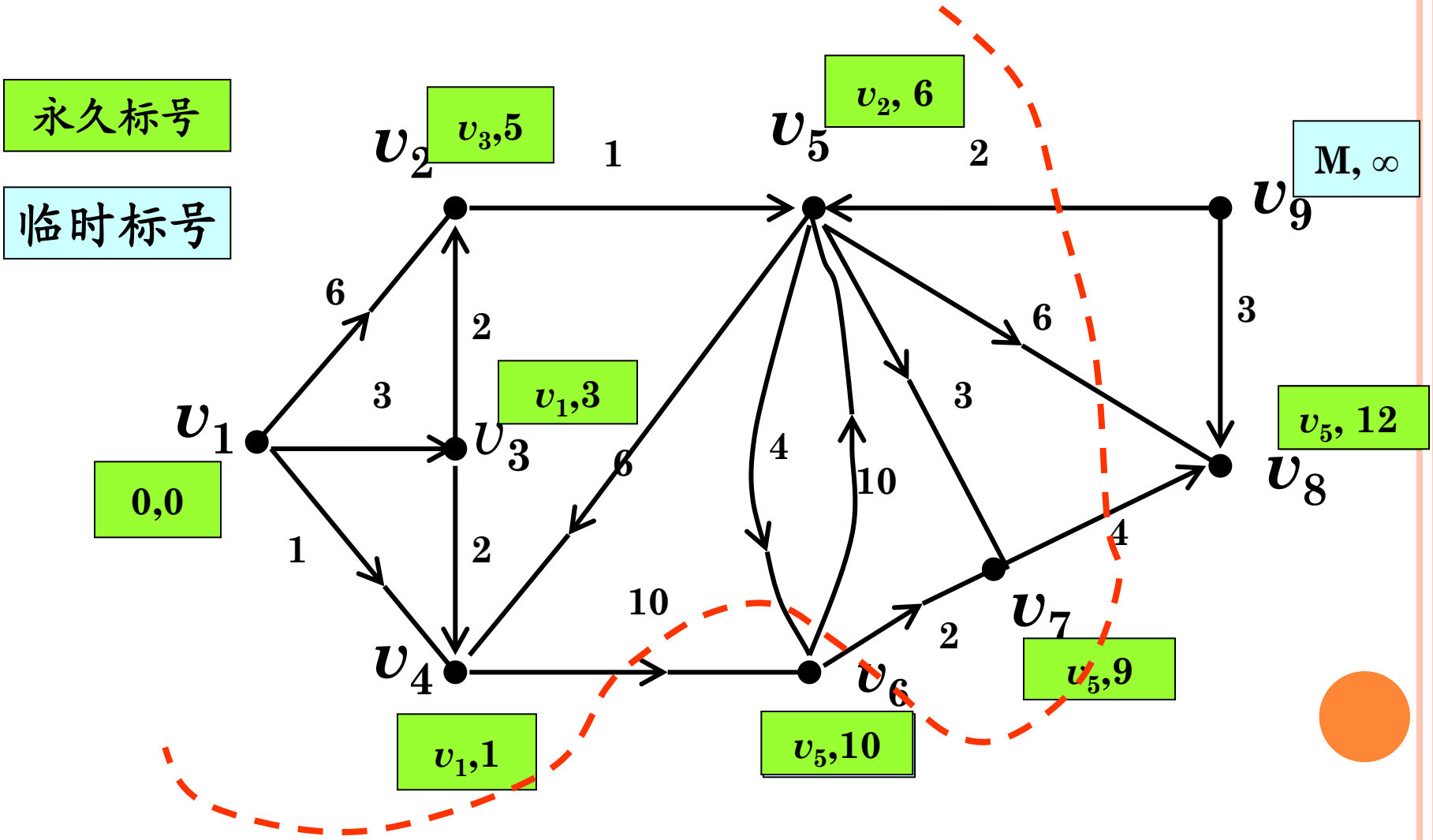
DIJKSTRA最短路径算法过程（有向图）



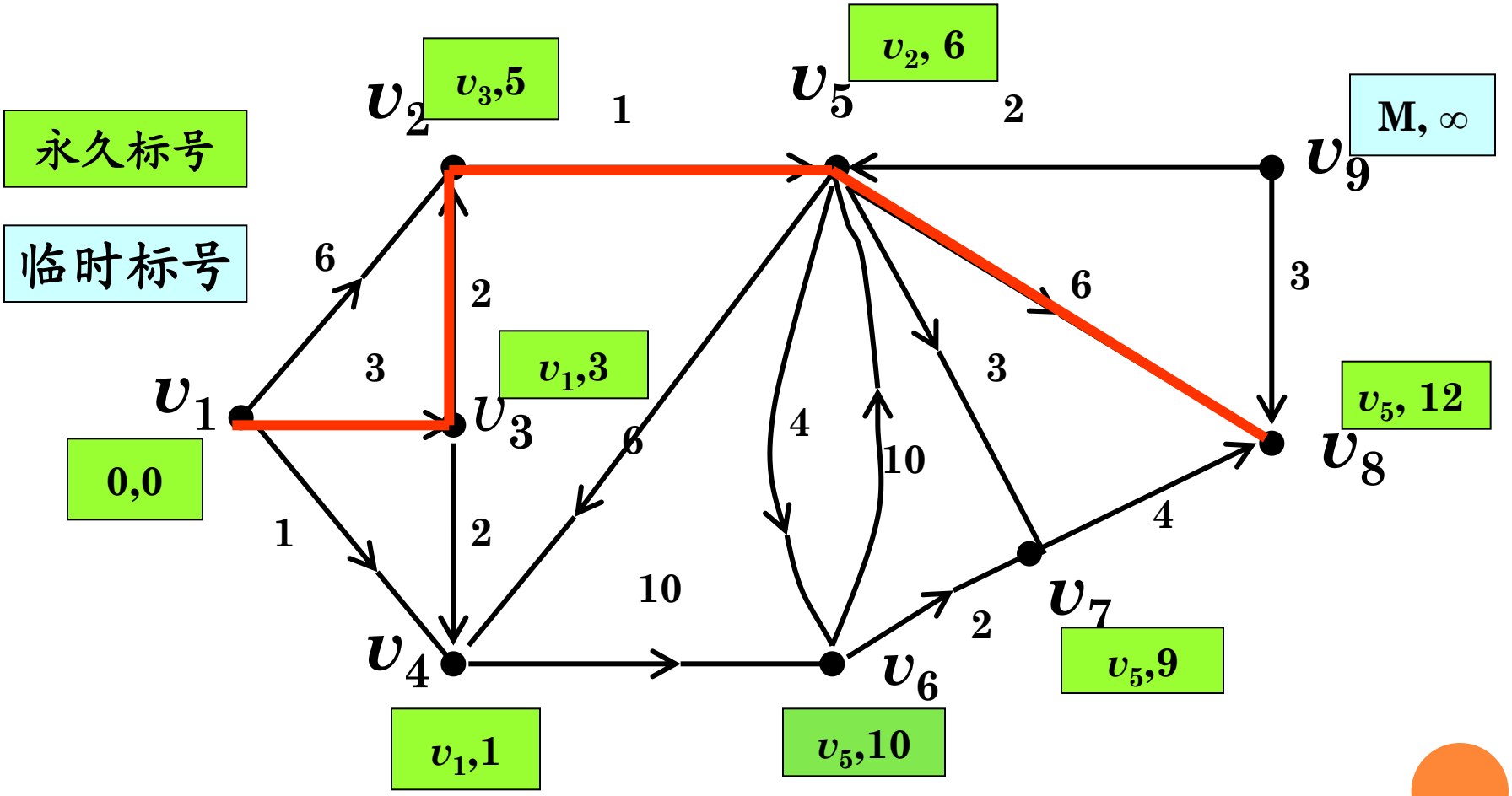
DIJKSTRA最短路径算法过程（有向图）



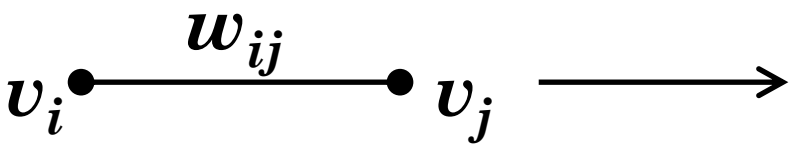
DIJKSTRA最短路径算法过程（有向图）



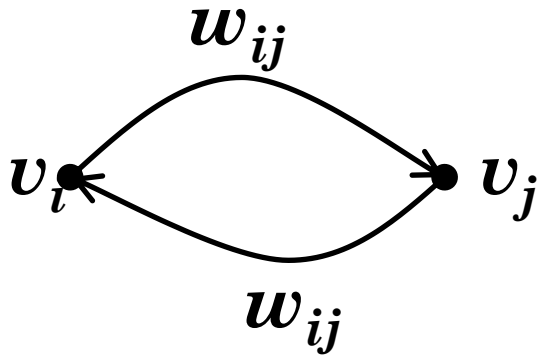
DIJKSTRA最短路径算法过程（有向图）

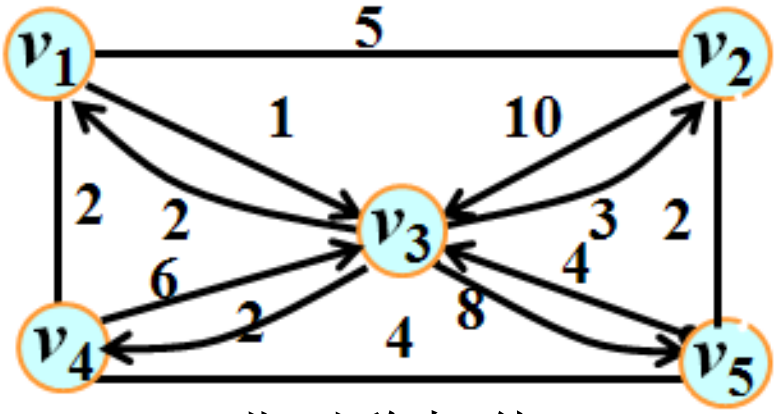


标号结束　　反向追踪

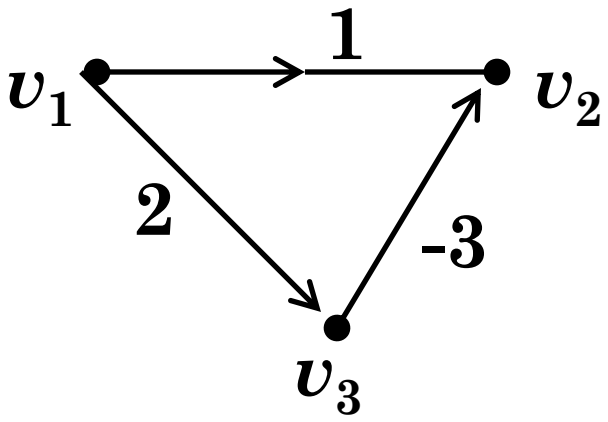


对称权值





非对称权值
多点对距离



负权值

路矩阵算法

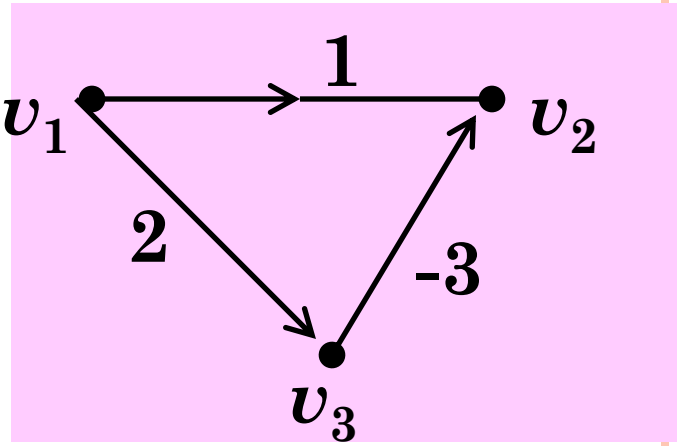
逐步逼近算法

逐次逼近算法

- 算法说明：

$$P_{1j}^{(k)} = \underset{1\le i\le n}{Min}\{ P_{1i}^{(k-1)} + w_{ij} \}$$

$$P_{1j}^{(k)} = P_{1j}^{(k-1)}$$



该公式表明，$P^{(1)}{}_{1j}$中的第j个分量等于$P^{(0)}{}_{1j}$的分量与基本表(权矩阵)中的第j列相应元素路长的最小值，它相当于在v_1与v_j之间插入一个转接点($v_1, v_2, \ldots, v_n$中的任一个，含点v_1与v_j)后所有可能路中的最短路的路长；每迭代一次，就相当与增加一个转接点，而$P^{(k)}{}_{1j}$中的每一个分量则随着k的增加而呈不增的趋势！

逐次逼近算法

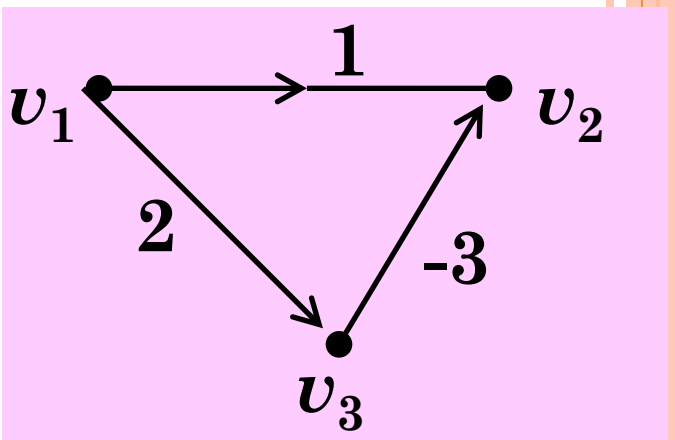
- 逐次逼近算法步骤：

用于计算带有负权弧指定点v_1到其余各点的最短路

1. 令 $P_{1j}^{(0)} = w_{1j}$ $j=1,2,\ldots,n.$

2. 计算 $P_{1j}^{(k)} = \underset{1\le i\le n}{Min}\{P_{1i}^{(k-1)} + w_{ij}\}$

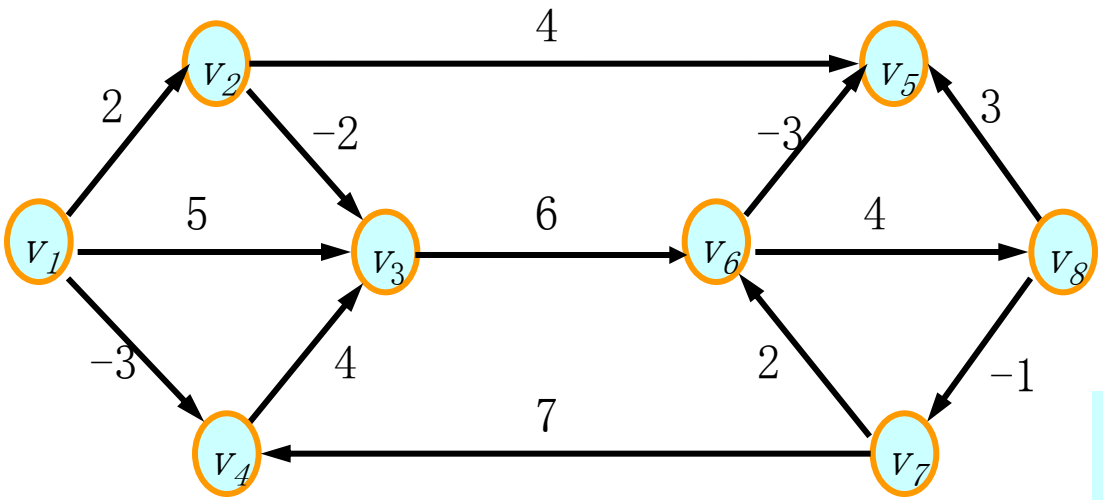
$j=1,2,\ldots,n.$ $k=1,2,\ldots,$ $t\leqslant n$



3. 当出现 $P_{1j}^{(k)} = P_{1j}^{(k-1)}$ 时，

其元素即是v_1到v_j的最短路长。

计算从点v_1到所有其它顶点的最短路



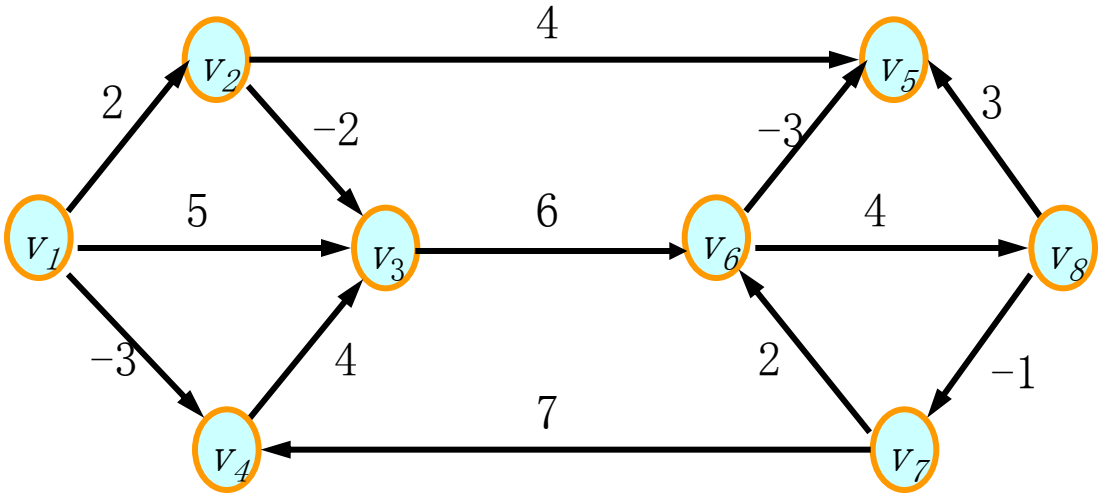
$$P_{1j}^{(0)} = w_{1j}$$

解：初始条件为 $P_{11}^{(0)} = 0, P_{12}^{(0)} = 2, P_{13}^{(0)} = 5, P_{14}^{(0)} = -3$

以后按照公式 $P_{1j}^{(k)} = \underset{1 \le i \le n}{Min}\{ P_{1i}^{(k-1)} + w_{ij} \}$ 迭代。

直到得到 $P_{1j}^{(t)} = P_{1j}^{(t-1)}$ 迭代停止。

计算从点 v_1 到所有其它顶点的最短路

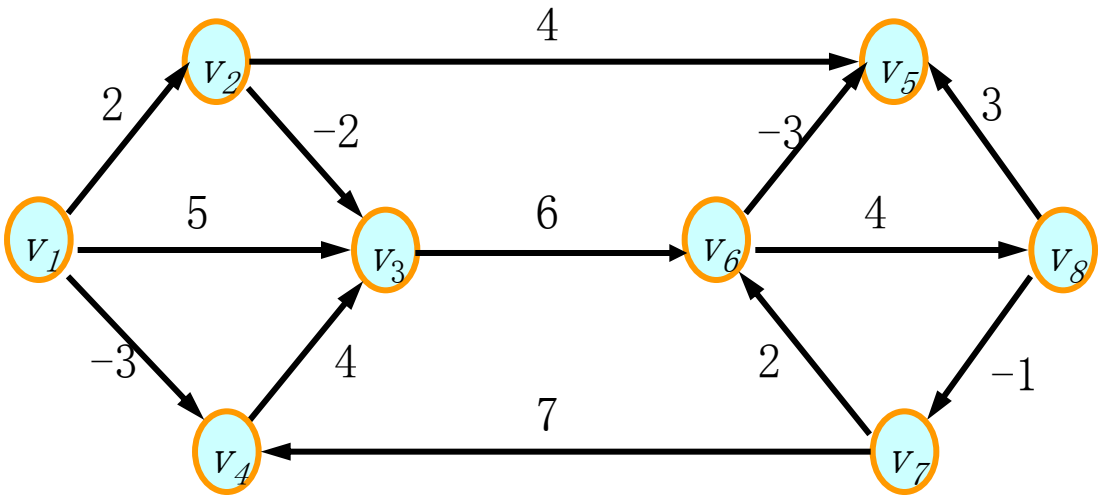


解： 初始条件为：

$$P_{11}^{(1)}=0,\quad P_{12}^{(1)}=2,\quad P_{13}^{(1)}=5,\quad P_{14}^{(1)}=-3,$$

$$P_{15}^{(1)}=P_{16}^{(1)}=P_{17}^{(1)}=P_{18}^{(1)}=\infty$$

计算从点v_1到所有其它顶点的最短路

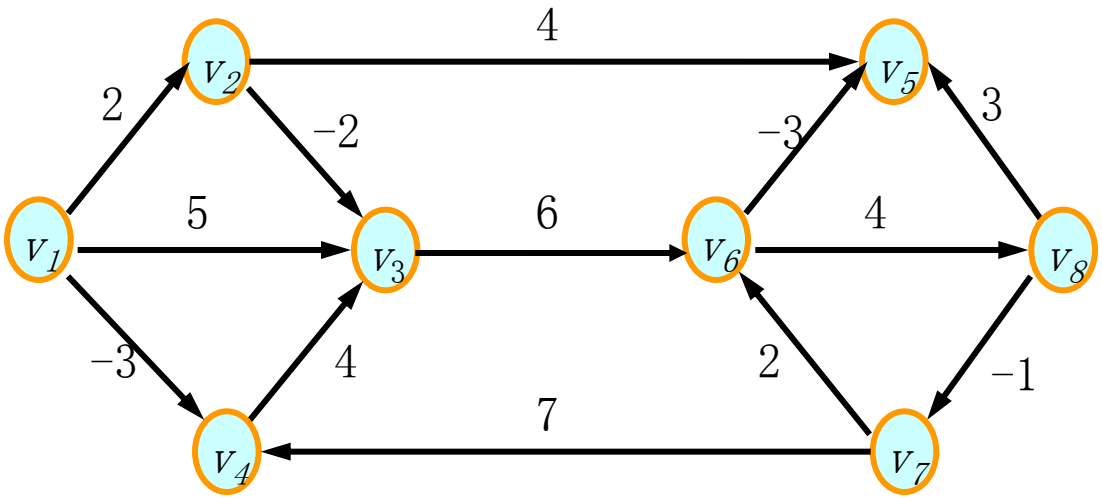


第一轮迭代：

$$P_{11}^{(2)} = \min\{P_{11}^{(1)} + l_{11},\ P_{12}^{(1)} + l_{21},\ P_{13}^{(1)} + l_{31},\ P_{14}^{(1)} + l_{41},$$
$$P_{15}^{(1)} + l_{51},\ \ldots,\ P_{18}^{(1)} + l_{81}\}$$
$$= \min\{0+0,\ 2+\infty, 5+\infty, -3+\infty, \infty, \infty, \infty, \infty\} = 0$$

$$P_{12}^{(2)} = \min\{P_{11}^{(1)} + l_{12},\ P_{12}^{(1)} + l_{22},\ P_{13}^{(1)} + l_{32},\ P_{14}^{(1)} + l_{42},$$
$$P_{15}^{(1)} + l_{52},\ \ldots,\ P_{18}^{(1)} + l_{82}\}$$
$$= \min\{0+2,\ 2+0, 5+\infty, -3+\infty, \infty, \infty, \infty, \infty\} = 2$$

计算从点 v_1 到所有其它顶点的最短路



第一轮迭代：

类似可得：

$$P_{13}^{(2)}=0,\quad P_{14}^{(2)}=-3,\quad P_{15}^{(2)}=6,$$

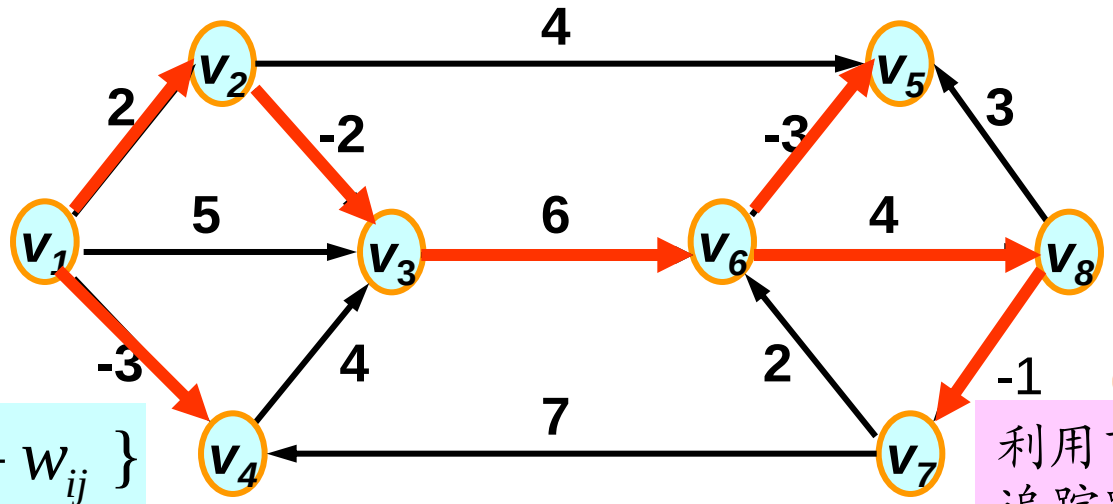
$$P_{16}^{(2)}=11,\quad P_{17}^{(2)}=P_{18}^{(2)}=\infty,$$

可以看出 $P_{1j}^{(2)}$ 表示V1两步到Vj的最短路径

	w_{ij}								$P_{1j}^{(0)}$	$P_{1j}^{(1)}$	$P_{1j}^{(2)}$	$P_{1j}^{(3)}$	$P_{1j}^{(4)}$	$P_{1j}^{(5)}$
	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8						
v_1	0	2	5	-3					0	0	0	0	0	0
v_2		0	-2		4				2	2	2	2	2	2
v_3			0			6			5	0_{123}	0_{123}	0_{123}	0_{123}	0_{123}
v_4			4	0					−3	−3	−3	−3	−3	−3
v_5					0					6_{125}	6_{125}	3_{12365}	3_{12365}	3_{12365}
v_6					-3	0		4		11_{136}	6_{1236}	6_{1236}	6_{1236}	6_{1236}
v_7				7		2	0					14_{13687}	9_{123687}	9_{123687}
v_8					3		-1	0			15_{1368}	10_{12368}	10_{12368}	10_{12368}

基本表

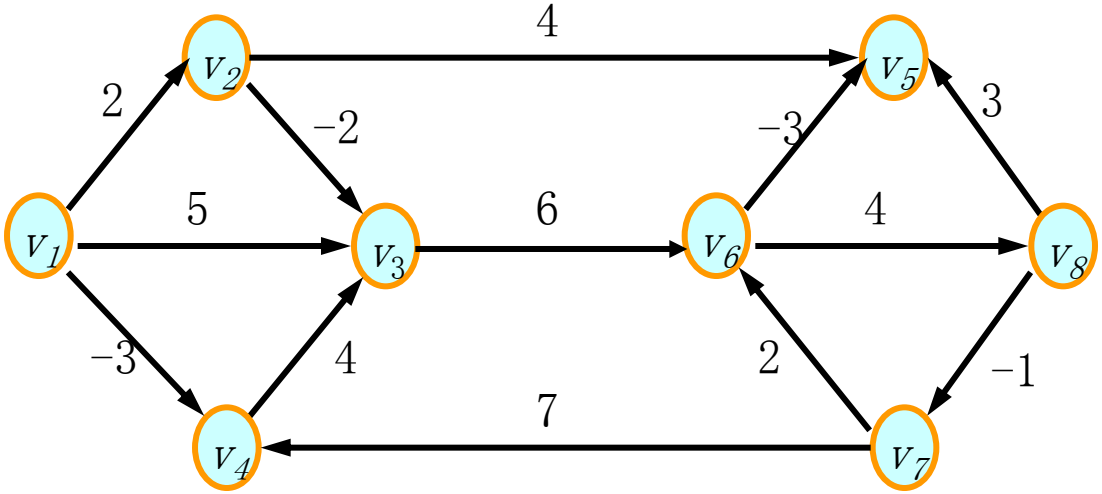
空格为无穷大



$$P_{1j}^{(k)} = \underset{1\le i\le n}{Min}\{ P_{1i}^{(k-1)} + w_{ij} \}$$

利用下标追踪路径

计算从点v_1到所有其它顶点的最短路



如需求V1到V8的最短路径：

- 已知$P_{18}=10$，而$P_{18}=\min\{P_{1i}+L_{i8}\}$，在表中寻求满足等式的Vi点，容易知道$P_{16}+L_{68}=10$，记下<V6,V8>;
- 再考察V6，由于$P_{16}=6$，而$6=0+6=P_{13}+L_{36}$，记下<V3,V6>;
- 再考察V3，由于$P_{13}=0$，而$0=2+(-2)=P_{12}+L_{23}$，记下<V2,V3>;
- 再考察V2，由于$P_{12}=2$，而$2=0+2=P_{11}+L_{12}$，记下<V1,V2>;
- 所以V1到V8的最短路径为*v1 → v2 → v3 → v6 → v8*

Floyd算法(路矩阵法)思想

某些问题需要求网络上任意两点间的最短路。当然，它也可以用标号算法依次改变始点的办法来计算，但是比较麻烦。

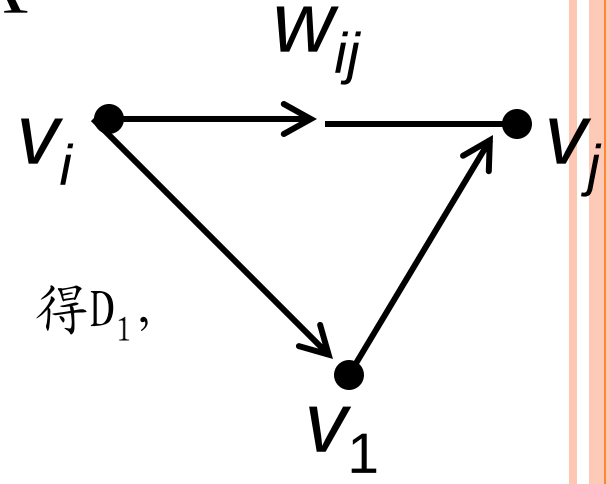
这里介绍Floyd在1962年提出的路矩阵法，它可直接求出网络中任意两点间的最短路。

Floyd算法(路矩阵法)思想

网络D=（V，A，W），令U=$(d_{ij})_{n\times n}$，d_{ij}表示D中v_i到v_j的最短路的长度。

考虑D中任意两点v_i，v_j，如将D中v_i，v_j以外的点都删掉，得只剩v_i，v_j的一个子网络D_0，记

$$d_{ij}^{(0)}=\begin{cases} w_{ij} & \text{当}(v_i,v_j)\in A \\ \infty & \text{否} \end{cases}$$



w_{ij}为弧（v_i，v_j）的权。

在D_0中加入v_1及D中与v_i，v_j，v_1相关联的弧，得D_1，D_1中v_i到v_j的最短路记为　，则一定有 $d_{ij}^{(1)}$

$$d_{ij}^{(1)}=\min\left\{d_{ij}^{(0)},\ d_{i1}^{(0)}+d_{1j}^{(0)}\right\}$$

Floyd算法(路矩阵法)思想

$$d_{ij}^{(1)} = \min\left\{d_{ij}^{(0)},\ d_{i1}^{(0)} + d_{1j}^{(0)}\right\}$$

再在D_1中加入v_2及D中与v_i，v_j，v_1，v_2相关联的弧，得D_2，D_2中v_i到v_j的最短路长记为 $d_{ij}^{(2)}$，则有

$$d_{ij}^{(2)} = \min\left\{d_{ij}^{(1)},\ d_{i2}^{(1)} + d_{2j}^{(1)}\right\}$$

随着转接点的逐步增加，我们会发现

- 二指定点间路的条数也会随之增加，但是其最短路及路长可以确定；
- 当V中所有的点都可以作为转接点时，指定点v_i到v_j间所有路中的最短路自然是图中指定点v_i到v_j间的最短路。

路矩阵序列的含义

$D^{(0)}$

其中的任一元素表示相应两点间无转接点时最短路路长。

一阶路矩阵$D^{(1)}$

其中的元素表示相应两点间可能以点v_1为转接点的所有路中路长最短的路的路长；……；

K阶路矩阵$D^{(K)}$

其中的元素表示相应两点间可能以点v_1、v_2、…、v_k为转接点的所有路中路长最短的路的路长。

n阶路矩阵$D^{(n)}$

其中的元素$d^{(n)}_{ij}$就是v_i到v_j的可能以点v_1、v_2、…、v_n为转接点的所有路中路长最短的路的路长。既是v_i到v_j的最短路的路长。

为使计算程序化，转接点按顶点下标的顺序依次加入

Floyd算法(路矩阵法)步骤

设有有向网络D=（V，A），其权矩阵为$A=(a_{ij})_{n\times n}$，如下构造路矩阵序列：

$$a_{ij}=\begin{cases} w_{ij},(v_i,v_j)\in A \\ 0,i=j \\ \infty,(v_i,v_j)\notin A \end{cases}$$

1. 令权矩阵A为初始路矩阵$D^{(0)}$，即令$D^{(0)}=A$

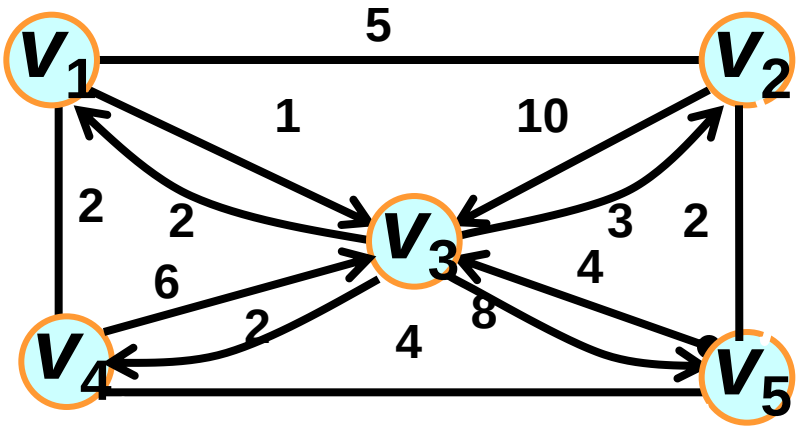
2. 依次计算K阶路矩阵$D^{(K)}=(d^{(k)}_{ij})_{n\times n}$，$k=1,2,\ldots,n$，

这里 $$d_{ij}^{(k)}=Min\{d_{ij}^{(k-1)},d_{ik}^{(k-1)}+d_{kj}^{(k-1)}\}$$

则n阶路矩阵$D^{(n)}$中的元素$d^{(n)}_{ij}$就是v_i到v_j的最短路的路长。

路矩阵法算法步骤

计算如下交通网络中各对点间最短路路长。

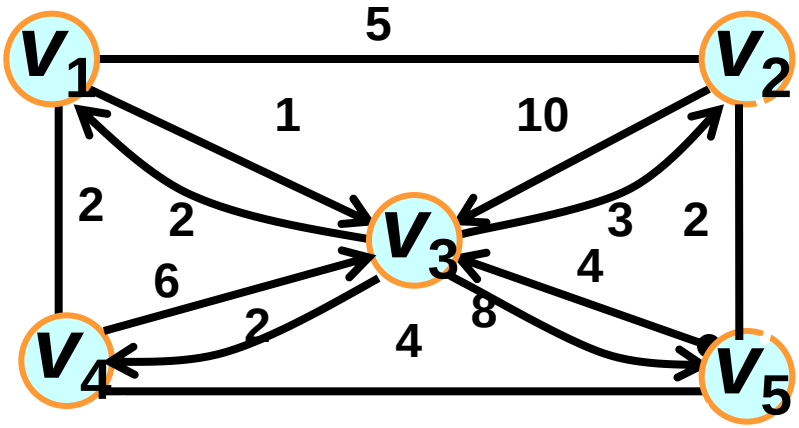


该图的权矩阵为：

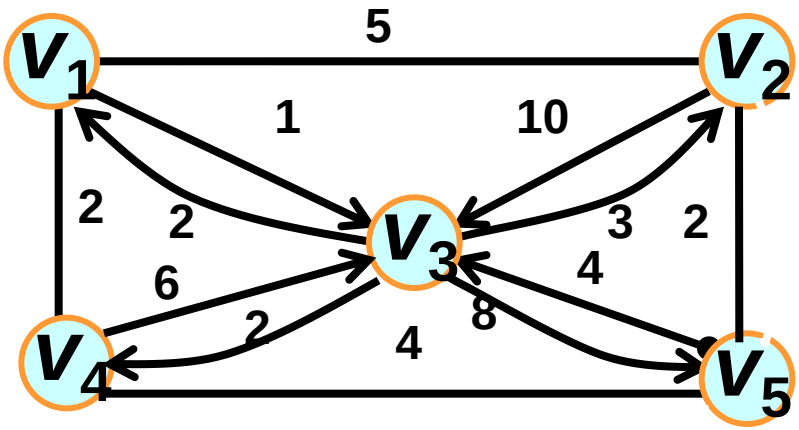
$$A=\begin{bmatrix} 0 & 5 & 1 & 2 & \infty \\ 5 & 0 & 10 & \infty & 2 \\ 2 & 3 & 0 & 2 & 8 \\ 2 & \infty & 6 & 0 & 4 \\ \infty & 2 & 4 & 4 & 0 \end{bmatrix}$$

路矩阵法算法步骤

计算如下交通网络中各对点间最短路路长。



$$D^{(0)}=A=\begin{bmatrix} 0 & 5 & 1 & 2 & \infty \\ 5 & 0 & 10 & \infty & 2 \\ 2 & 3 & 0 & 2 & 8 \\ 2 & \infty & 6 & 0 & 4 \\ \infty & 2 & 4 & 4 & 0 \end{bmatrix}$$



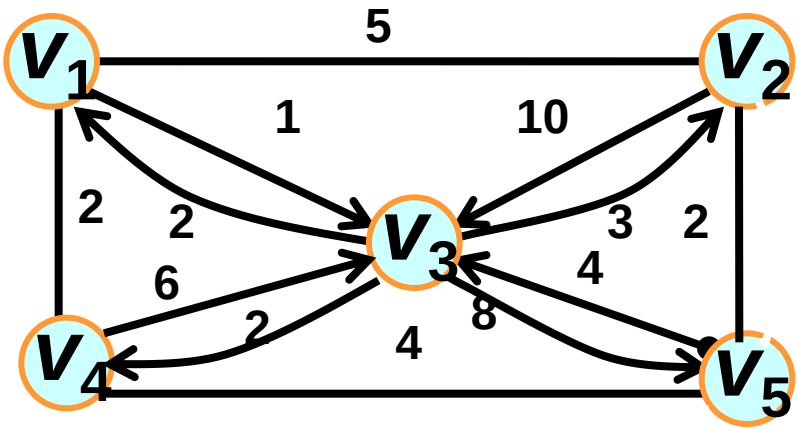
$$D^{(0)}=A=\begin{bmatrix} 0 & 5 & 1 & 2 & \infty \\ 5 & 0 & 10 & \infty & 2 \\ 2 & 3 & 0 & 2 & 8 \\ 2 & \infty & 6 & 0 & 4 \\ \infty & 2 & 4 & 4 & 0 \end{bmatrix}$$

利用公式

$$d_{ij}^{(1)}=\min\left\{d_{ij}^{(0)},\ d_{i1}^{(0)}+d_{1j}^{(0)}\right\}$$

发现第一行，第一列元素不变

$$D^{(1)}=\begin{bmatrix} 0 & 5 & 1 & 2 & \infty \\ 5 & 0 & 6_{213} & 7_{214} & 2 \\ 2 & 3 & 0 & 2 & 8 \\ 2 & 7_{412} & 3_{413} & 0 & 4 \\ \infty & 2 & 4 & 4 & 0 \end{bmatrix}$$



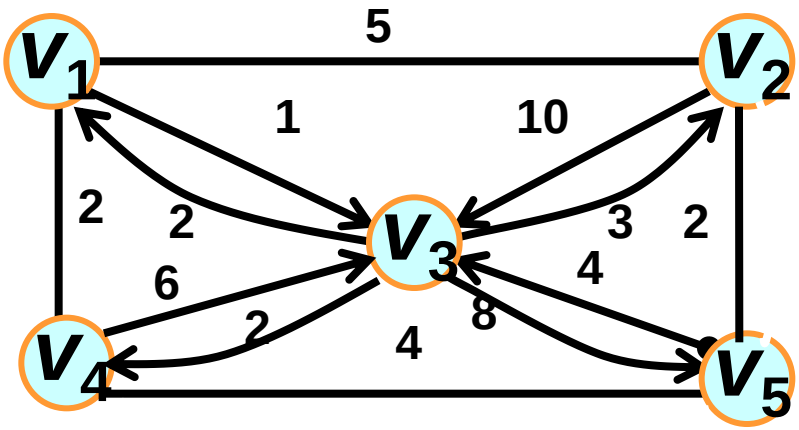
$$D^{(1)}=\begin{bmatrix} 0 & 5 & 1 & 2 & \infty \\ 5 & 0 & 6_{213} & 7_{214} & 2 \\ 2 & 3 & 0 & 2 & 8 \\ 2 & 7_{412} & 3_{413} & 0 & 4 \\ \infty & 2 & 4 & 4 & 0 \end{bmatrix}$$

利用公式

$$d_{ij}^{(2)}=\min\left\{d_{ij}^{(1)},\ d_{i2}^{(1)}+d_{2j}^{(1)}\right\}$$

发现第二行，第二列元素不变

$$D^{(2)}=\begin{bmatrix} 0 & 5 & 1 & 2 & 7_{125} \\ 5 & 0 & 6_{213} & 7_{214} & 2 \\ 2 & 3 & 0 & 2 & 5_{325} \\ 2 & 7_{412} & 3_{413} & 0 & 4 \\ 7_{521} & 2 & 4 & 4 & 0 \end{bmatrix}$$



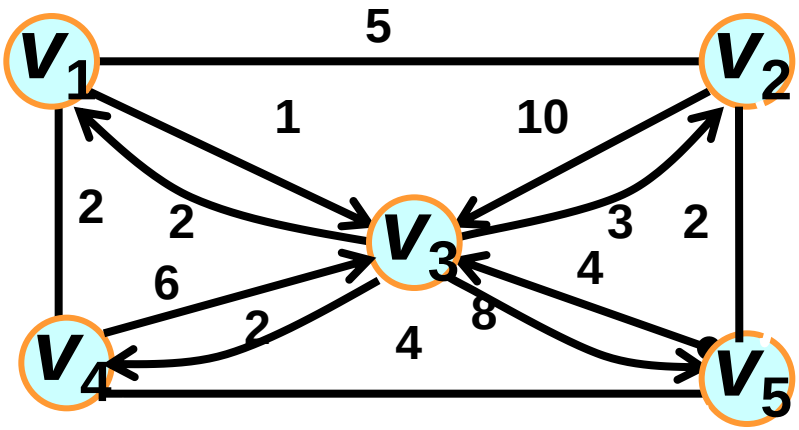
$$D^{(2)}=\begin{bmatrix} 0 & 5 & 1 & 2 & 7_{125} \\ 5 & 0 & 6_{213} & 7_{214} & 2 \\ 2 & 3 & 0 & 2 & 5_{325} \\ 2 & 7_{412} & 3_{413} & 0 & 4 \\ 7_{521} & 2 & 4 & 4 & 0 \end{bmatrix}$$

利用公式

$$d_{ij}^{(3)}=\min\left\{d_{ij}^{(2)},\ d_{i3}^{(2)}+d_{3j}^{(2)}\right\}$$

$$D^{(3)}=\begin{bmatrix} 0 & 4_{132} & 1 & 2 & 6_{1325} \\ 5 & 0 & 6_{213} & 7_{214} & 2 \\ 2 & 3 & 0 & 2 & 5_{325} \\ 2 & 6_{4132} & 3_{413} & 0 & 4 \\ 6_{531} & 2 & 4 & 4 & 0 \end{bmatrix}$$

发现第三行，第三列元素不变



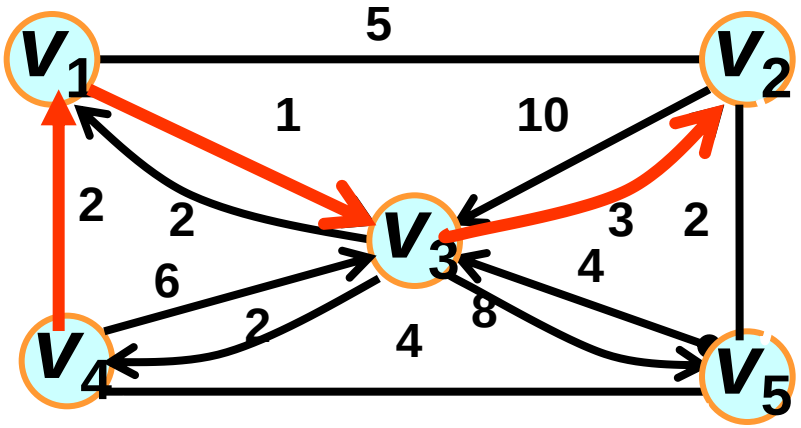
$$D^{(3)}=\begin{bmatrix} 0 & 4_{132} & 1 & 2 & 6_{1325} \\ 5 & 0 & 6_{213} & 7_{214} & 2 \\ 2 & 3 & 0 & 2 & 5_{325} \\ 2 & 6_{4132} & 3_{413} & 0 & 4 \\ 6_{531} & 2 & 4 & 4 & 0 \end{bmatrix}$$

利用公式

$$d_{ij}^{(4)}=\min\left\{d_{ij}^{(3)},\ d_{i4}^{(3)}+d_{4j}^{(3)}\right\}$$

发现第四行，第四列元素不变

$$D^{(4)}=\begin{bmatrix} 0 & 4_{132} & 1 & 2 & 6_{1325/145} \\ 5 & 0 & 6_{213} & 7_{214} & 2 \\ 2 & 3 & 0 & 2 & 5_{325} \\ 2 & 6_{4132} & 3_{413} & 0 & 4 \\ 6_{531/541} & 2 & 4 & 4 & 0 \end{bmatrix}$$



$$D^{(4)}=\begin{bmatrix} 0 & 4_{132} & 1 & 2 & 6_{1325/145} \\ 5 & 0 & 6_{213} & 7_{214} & 2 \\ 2 & 3 & 0 & 2 & 5_{325} \\ 2 & 6_{4132} & 3_{413} & 0 & 4 \\ 6_{531/541} & 2 & 4 & 4 & 0 \end{bmatrix}$$

$D^{(5)}$中的元素给出相应两点间的最短路，其下标给出最短路个顶点下标。

$$D^{(5)}=\begin{bmatrix} 0 & 4_{132} & 1 & 2 & 6_{1325/145} \\ 5 & 0 & 6_{213} & 6_{254} & 2 \\ 2 & 3 & 0 & 2 & 5_{325} \\ 2 & 6_{4132} & 3_{413} & 0 & 4 \\ 6_{531/541} & 2 & 4 & 4 & 0 \end{bmatrix}$$

网络分析算法小结

- 固定起点的最短路

  Dijkstra(狄克斯拉)(荷兰)算法

  逐次逼近算法(Ford(美国)算法)

- 每对顶点之间的最短路

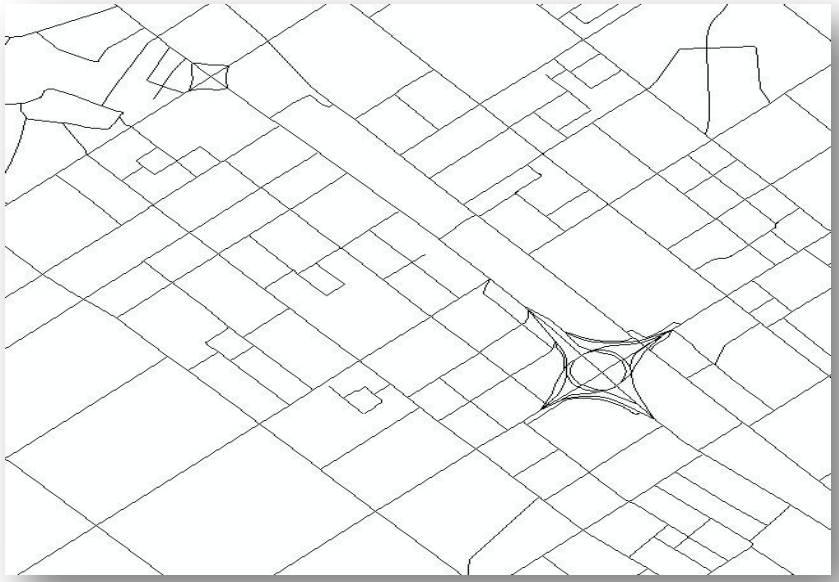
  路矩阵算法(Floyd(佛洛伊德)算法)

连通性分析算法

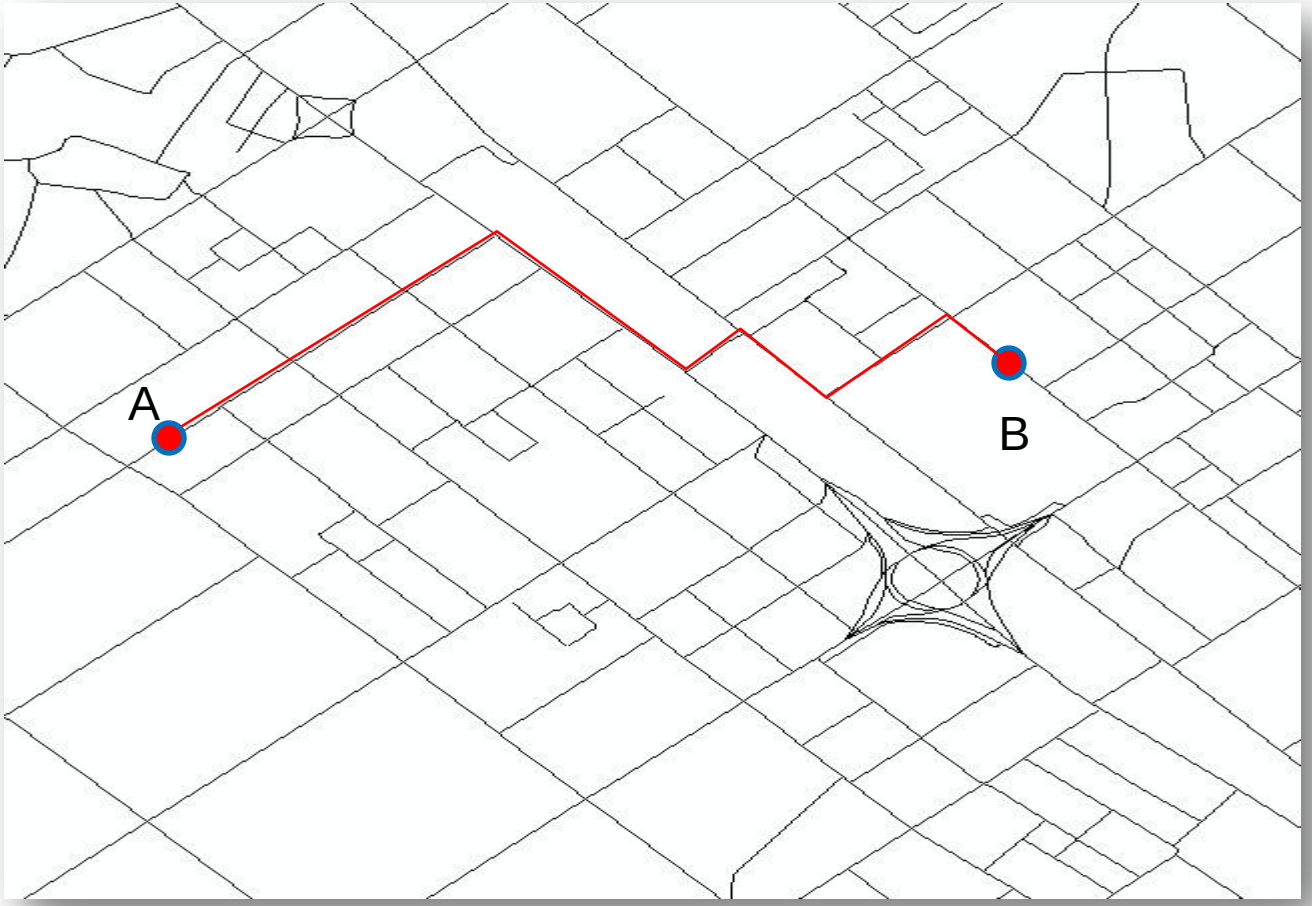
- 现实生活中，我们需要从某点出发，到达城市的任何节点或位置的路线，这一问题统称为连通分量求解。
- 另一分析方法是找出费用最少的连通方案，即在消耗最下的情况下，使得全部结点相互连通。
- 连通分析问题对应于图的生成树求解。
- 求连通分量往往采用深度优先遍历或广度优先遍历形成深度或广度优先生成树。
- 最小费用连通方案即求解图的最小生成树：
  - Prim算法
  - Kruskal算法

连通性分析算法

- Prim算法（加点法）
- Kruskal算法（贪心法）

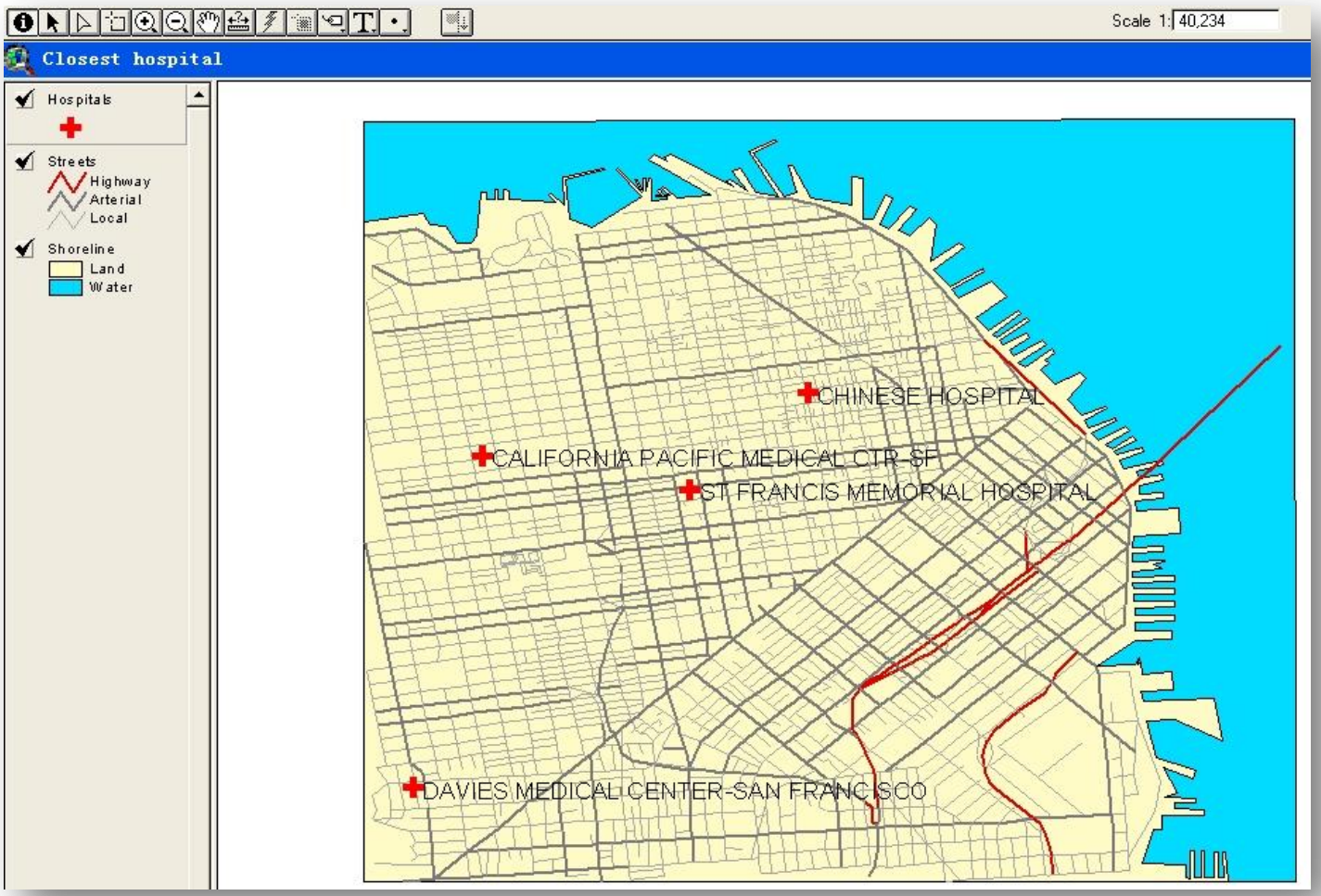


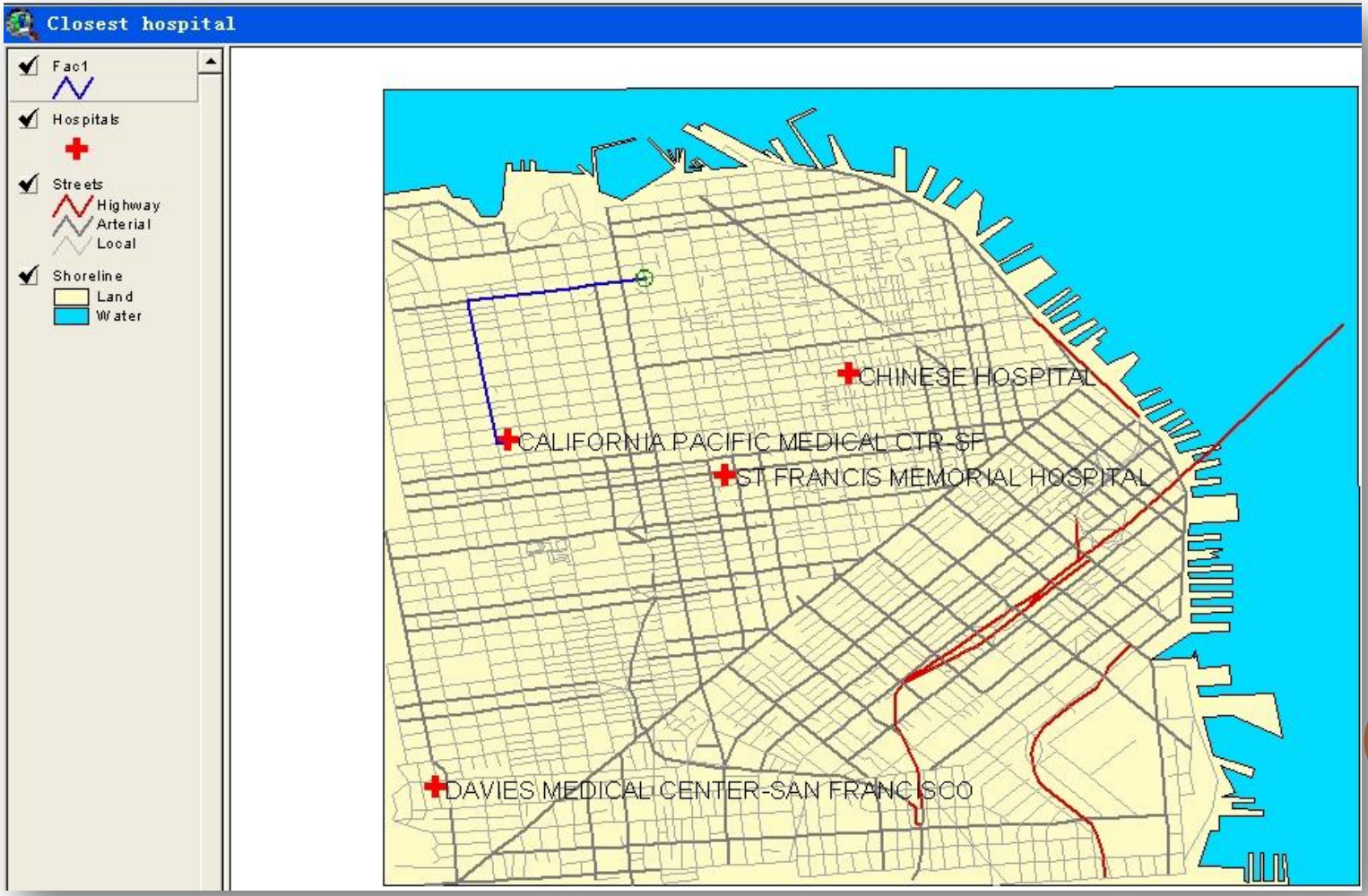
- 两点路径查询



连通性分析算法

- 引导救护车到最近的医院




Closest hospital
Fac1
Hospitals
Streets
Highway
Arterial
Local
Shoreline
Land
Water
CHINESE HOSPITAL
CALIFORNIA PACIFIC MEDICAL CTR-SF
ST FRANCIS MEMORIAL HOSPITAL
DAVIES MEDICAL CENTER-SAN FRANCISCO

连通性分析算法

最小生成树 Minimum-cost Spanning Tree

- 连通无向带权图 —— 网络。
- 网络（带权图）的生成树中生成树各边的权值加起来称为生成树的权，把权值最小的生成树称为最小生成树 (简称为MST)。

连通性分析算法

MST性质（prim算法原理）

- G=(V，E)是一个网络，U是顶点集合V的一个真子集。
- 如果u∈U，v∈V-U，且边(u,v)是图G中所有一个端点在U里，另一端点在V-U里的边中权值最小的边，则一定存在G的一棵最小生成树包括此边(u,v)。

MST必包含连通图中任意两个顶点划分之间的最小权的边。

（任意割集中的最小边）

连通性分析算法

Prim算法（找MST）

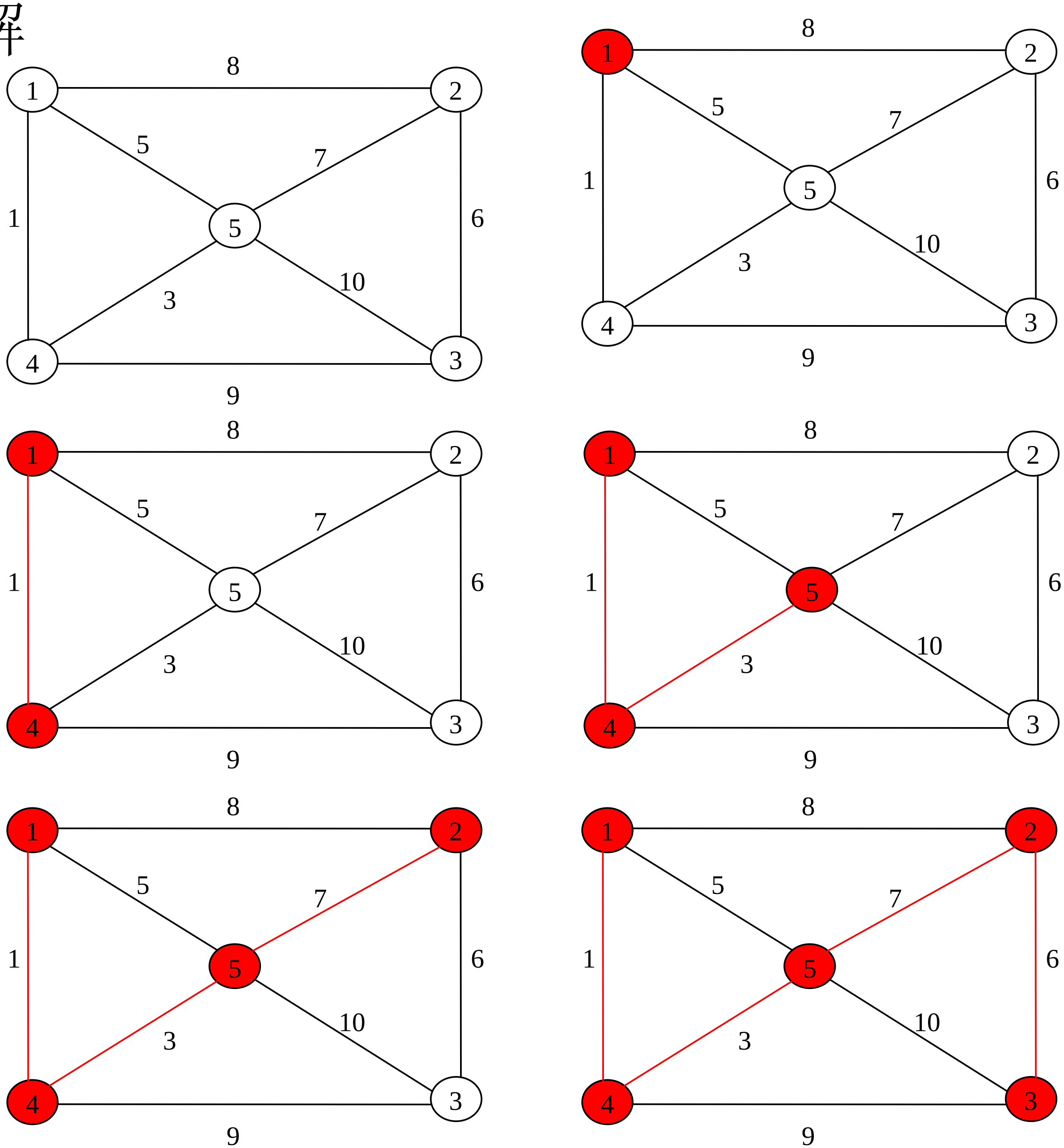
prim算法的基本思想是：

① 首先从集合V中任取一顶点(例如取顶点v_0)放入集合U中
这时U={v_0}，TE=NULL

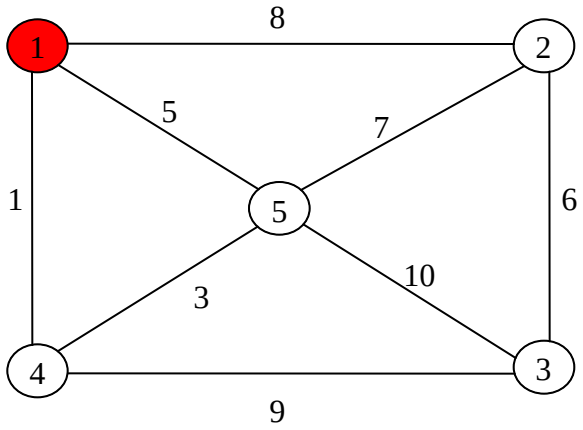
② 然后在所有一个顶点在集合U里，另一个顶点在集合V-U里的边中，找出权值最小的边(u,v)(u∈U，v∈V-U)，将边加入TE，并将顶点v加入集合U

③ 重复上述操作直到U=V为止。这时TE中有n-1条边，T=(U，TE)就是G的一棵最小生成树

Prim算法图解



算法过程

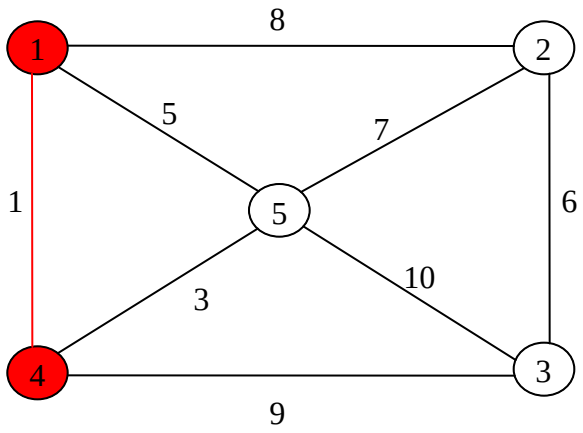


T	c	v(红点)	sb(白点)	b
[]	0	1	2,3,4,5	1 1 1 1 2 3 4 5 8 inf 1 5

矩阵a

	1	2	3	4	5
1	0	8	inf	1	5
2	8	0	6	inf	7
3	inf	6	0	9	10
4	1	inf	9	0	3
5	5	7	10	3	0

算法过程

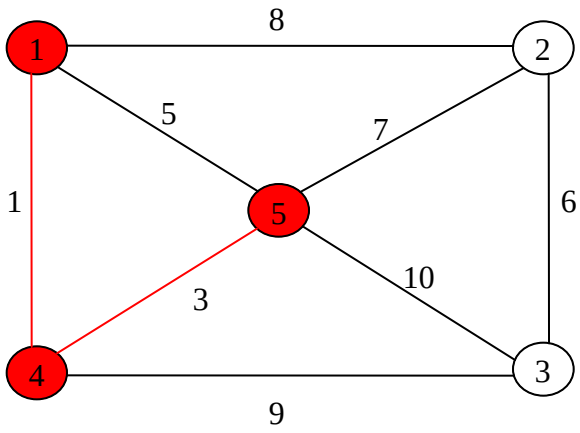


T	c	v(红点)	sb(白点)	b
[]	0	1	2,3,4,5	1 1 1 1 2 3 4 5 8 inf 1 5
1 4 1	1	4	2,3,5	1 4 4 2 3 5 8 9 3

矩阵a

	1	2	3	4	5
1	0	8	inf	1	5
2	8	0	6	inf	7
3	inf	6	0	9	10
4	1	inf	9	0	3
5	5	7	10	3	0

算法过程

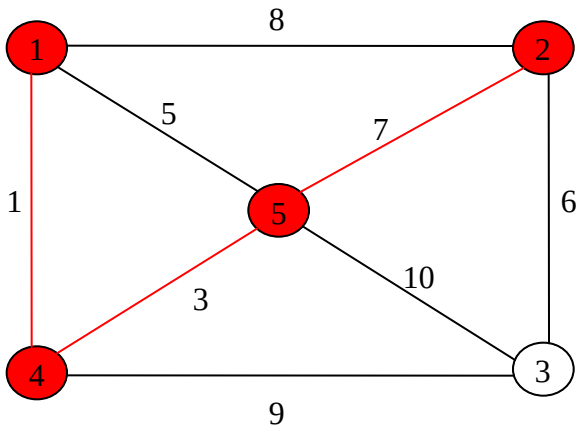


T	c	v(红点)	sb(白点)	b
[]	0	1	2,3,4,5	1 1 1 1 2 3 4 5 8 inf 1 5
1 4 1	1	4	2,3,5	1 4 4 2 3 5 8 9 3
1 4 4 5 1 3	4	5	2,3	5 4 2 3 7 9

矩阵a

	1	2	3	4	5
1	0	8	inf	1	5
2	8	0	6	inf	7
3	inf	6	0	9	10
4	1	inf	9	0	3
5	5	7	10	3	0

算法过程

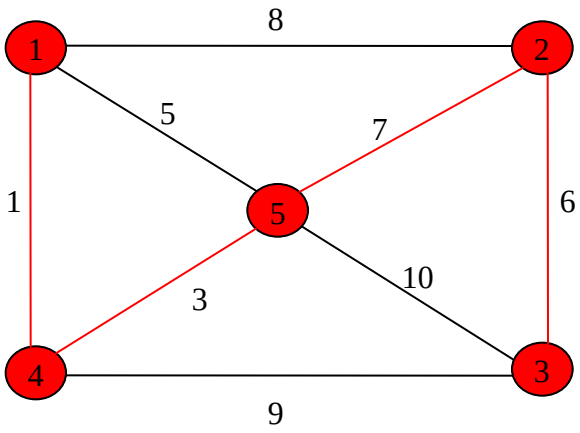


矩阵a

	1	2	3	4	5
1	0	8	inf	1	5
2	8	0	6	inf	7
3	inf	6	0	9	10
4	1	inf	9	0	3
5	5	7	10	3	0

T	c	v(红点)	sb(白点)	b
[]	0	1	2,3,4,5	1 1 1 1 2 3 4 5 8 inf 1 5
1 4 1	1	4	2,3,5	1 4 4 2 3 5 8 9 3
1 4 4 5 1 3	4	5	2,3	5 4 2 3 7 9
1 4 5 4 5 2 1 3 7	11	2	3	2 3 6

算法过程



矩阵a

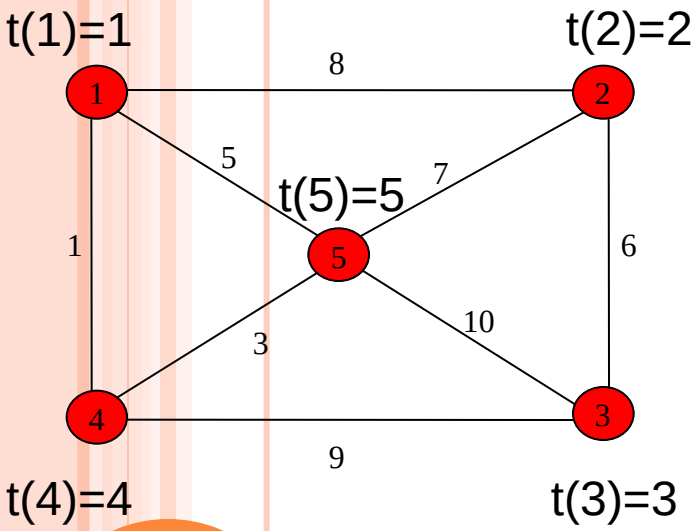
	1	2	3	4	5
1	0	8	inf	1	5
2	8	0	6	inf	7
3	inf	6	0	9	10
4	1	inf	9	0	3
5	5	7	10	3	0

T	c	v(红点)	sb(白点)	b
[]	0	1	2,3,4,5	1 1 1 1 2 3 4 5 8 inf 1 5
1 4 1	1	4	2,3,5	1 4 4 2 3 5 8 9 3
1 4 4 5 1 3	4	5	2,3	5 4 2 3 7 9
1 4 5 4 5 2 1 3 7	11	2	3	2 3 6
1 4 5 2 4 5 2 3 1 3 7 6	17	3	[]	[]

Kruskal算法

- 设G=(V，E)是网络，最小生成树的初始状态为只有n个顶点而无边的非连通图T=(V，φ)，T中每个顶点自成为一个连通分量。
- 将集合E中的边按权递增顺序排列，从小到大依次选择顶点分别在两个不同连通分量中的边加入图T，则原来的两个连通分量由于该边的连接而成为一个连通分量。
- 依次类推，直到T中所有顶点都在同一个连通分量上为止，该连通分量就是G的一棵最小生成树。

Kruskal算法过程



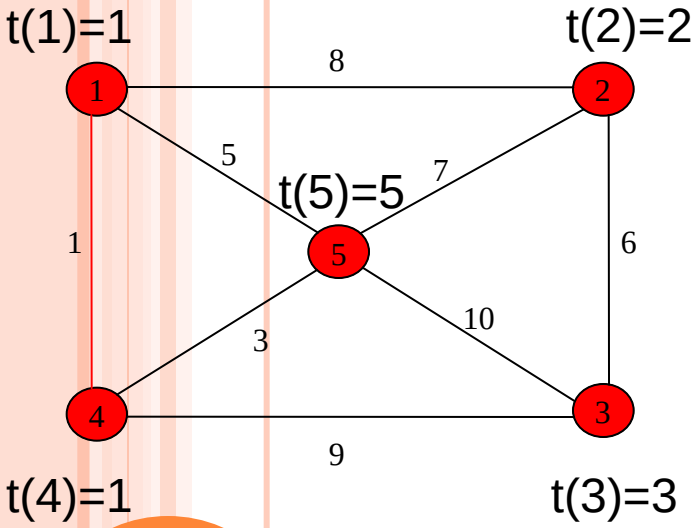
b= 1 1 1 2 2 3 3 4
2 4 5 3 5 4 5 5
8 1 5 6 7 9 10 3

按边权排序后：

B =1 4 1 2 2 1 3 3
4 5 5 3 5 2 4 5
1 3 5 6 7 8 9 10

j(列)	T	c(权值)	k(边数)	t(1)	t(2)	t(3)	t(4)	t(5)
0	[]	0	0	1	2	3	4	5

Kruskal算法过程



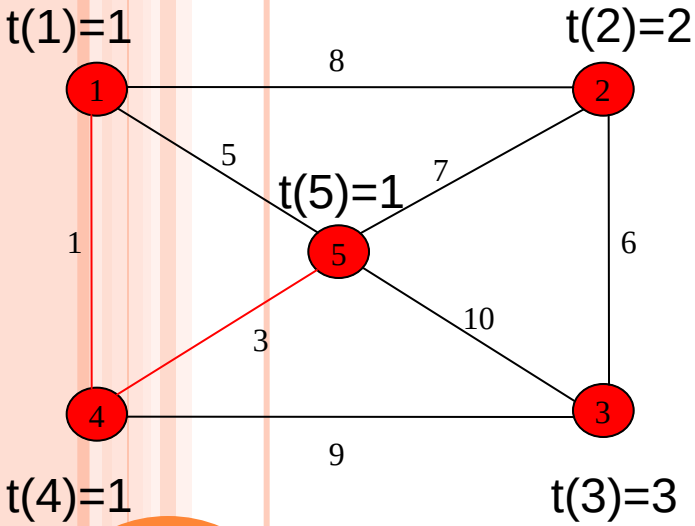
b= 1 1 1 2 2 3 3 4
2 4 5 3 5 4 5 5
8 1 5 6 7 9 10 3

按边权排序后：

B =1 4 1 2 2 1 3 3
4 5 5 3 5 2 4 5
1 3 5 6 7 8 9 10

j(列)	T	c(权值)	k(边数)	t(1)	t(2)	t(3)	t(4)	t(5)
0	[]	0	0	1	2	3	4	5
1	1 4;	1	1	1	2	3	1	5

Kruskal算法过程



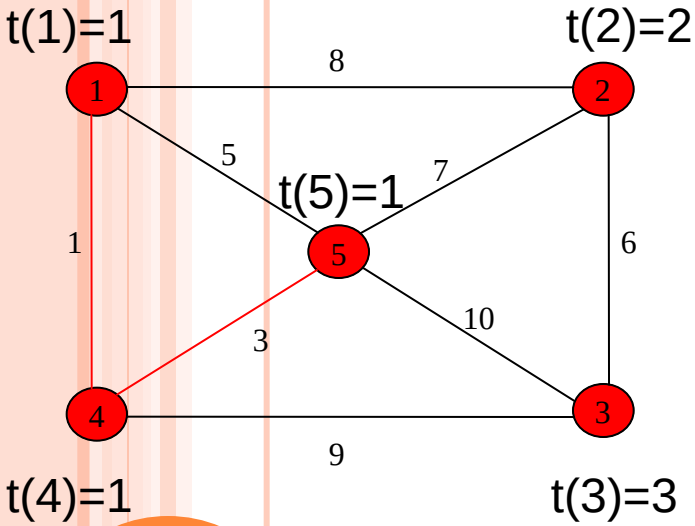
b= 1 1 1 2 2 3 3 4
2 4 5 3 5 4 5 5
8 1 5 6 7 9 10 3

按边权排序后：

B =1 4 1 2 2 1 3 3
4 5 5 3 5 2 4 5
1 3 5 6 7 8 9 10

j(列)	T	c(权值)	k(边数)	t(1)	t(2)	t(3)	t(4)	t(5)
0	[]	0	0	1	2	3	4	5
1	1 4;	1	1	1	2	3	1	5
2	1 4;4 5;	4	2	1	2	3	1	1

Kruskal算法过程



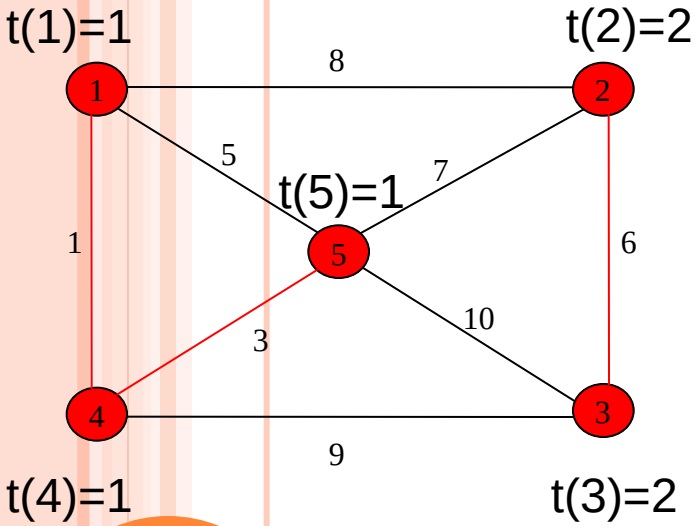
b= 1 1 1 2 2 3 3 4
2 4 5 3 5 4 5 5
8 1 5 6 7 9 10 3

按边权排序后：

B =1 4 1 2 2 1 3 3
4 5 5 3 5 2 4 5
1 3 5 6 7 8 9 10

j(列)	T	c(权值)	k(边数)	t(1)	t(2)	t(3)	t(4)	t(5)
0	[]	0	0	1	2	3	4	5
1	1 4;	1	1	1	2	3	1	5
2	1 4;4 5;	4	2	1	2	3	1	1
3	1 4;4 5;	4	2	1	2	3	1	1

Kruskal算法过程



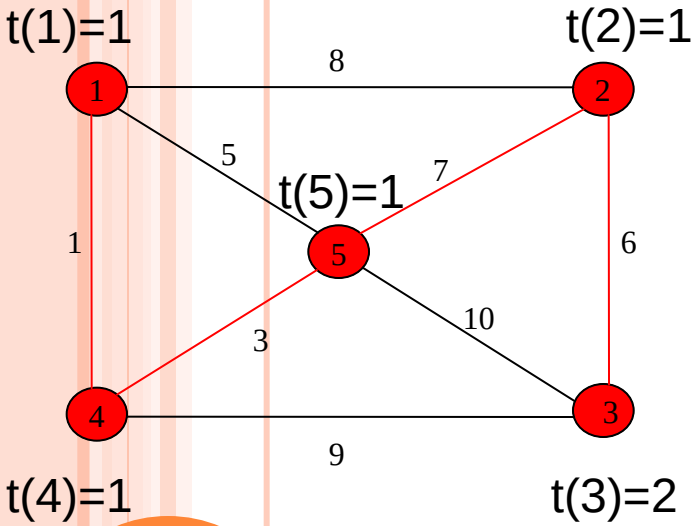
b= 1 1 1 2 2 3 3 4
2 4 5 3 5 4 5 5
8 1 5 6 7 9 10 3

按边权排序后：

B =1 4 1 2 2 1 3 3
4 5 5 3 5 2 4 5
1 3 5 6 7 8 9 10

j(列)	T	c(权值)	k(边数)	t(1)	t(2)	t(3)	t(4)	t(5)
0	[]	0	0	1	2	3	4	5
1	1 4;	1	1	1	2	3	1	5
2	1 4;4 5;	4	2	1	2	3	1	1
3	1 4;4 5;	4	2	1	2	3	1	1
4	1 4;4 5;2 3;	10	3	1	2	2	1	1

Kruskal算法过程



b= 1 1 1 2 2 3 3 4
2 4 5 3 5 4 5 5
8 1 5 6 7 9 10 3

按边权排序后：

B =1 4 1 2 2 1 3 3
4 5 5 3 5 2 4 5
1 3 5 6 7 8 9 10

j(列)	T	c(权值)	k(边数)	t(1)	t(2)	t(3)	t(4)	t(5)
0	[]	0	0	1	2	3	4	5
1	1 4	1	1	1	2	3	1	5
2	1 4;4 5	4	2	1	2	3	1	4
3	1 4;4 5	4	2	1	2	3	1	4
4	1 4;4 5;2 3	10	3	1	2	2	1	4
5	1 4;4 5;2 3;2 5	17	4	1	1	2	1	4

连通性分析小结

- Prim算法（加点法）
- Kruskal算法（贪心法）