

第八章 图

张史梁

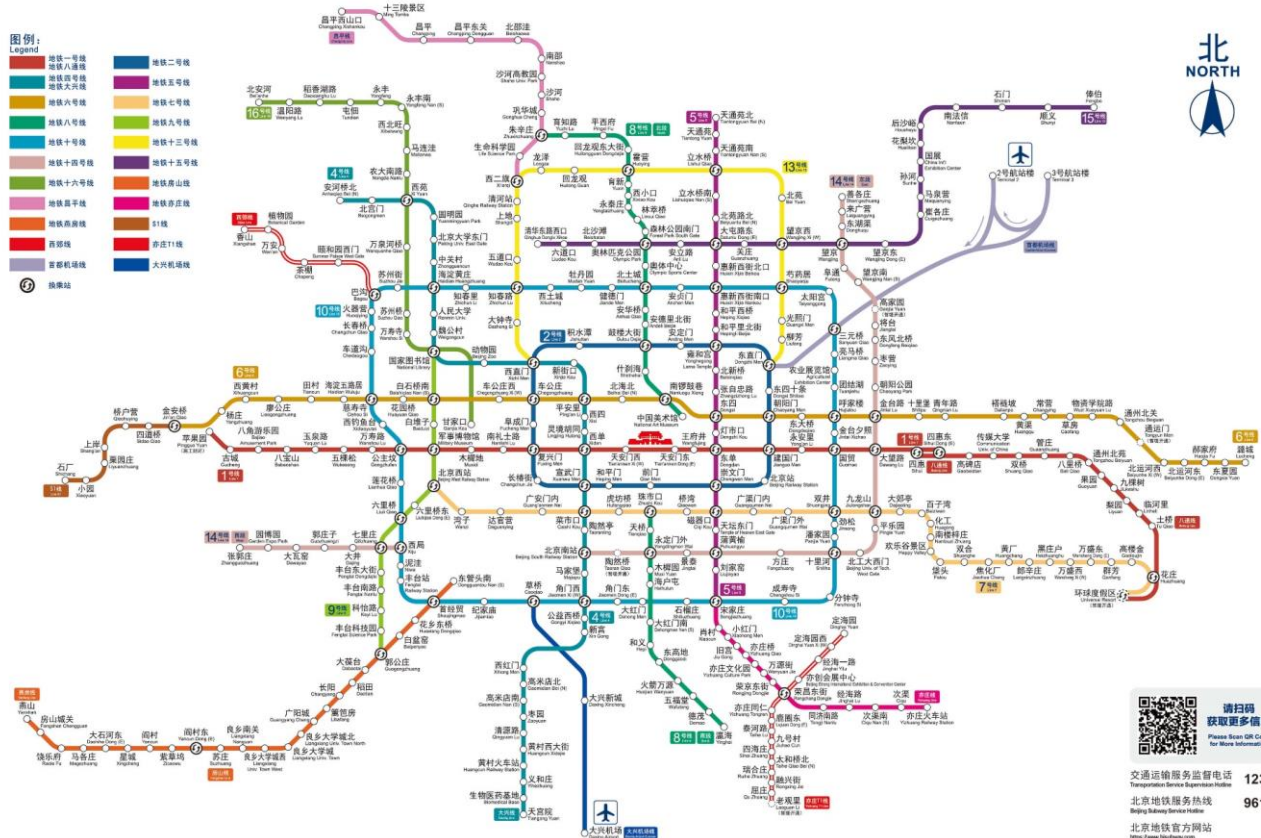
slzhang.jdl@pku.edu.cn

图中的一个重要问题



北京城市轨道交通线网图 Beijing Rail Transit Lines

- 图例:
Legend:
- 地铁一号线
 - 地铁二号线
 - 地铁四号线
 - 地铁五号线
 - 地铁六号线
 - 地铁七号线
 - 地铁八号线
 - 地铁九号线
 - 地铁十号线
 - 地铁十一号线
 - 地铁十三号线
 - 地铁十四号线
 - 地铁十五号线
 - 地铁十六号线
 - 地铁十七号线
 - 地铁十八号线
 - 地铁十九号线
 - 地铁二十号线
 - 地铁二十一号线
 - 地铁二十二号线
 - 地铁二十三号线
 - 地铁二十四号线
 - 地铁二十五号线
 - 地铁二十六号线
 - 地铁二十七号线
 - 地铁二十八号线
 - 地铁二十九号线
 - 地铁三十号线
 - 地铁三十一号线
 - 地铁三十二号线
 - 地铁三十三号线
 - 地铁三十四号线
 - 地铁三十五号线
 - 地铁三十六号线
 - 地铁三十七号线
 - 地铁三十八号线
 - 地铁三十九号线
 - 地铁四十号线
 - 地铁四十一号线
 - 地铁四十二号线
 - 地铁四十三号线
 - 地铁四十四号线
 - 地铁四十五号线
 - 地铁四十六号线
 - 地铁四十七号线
 - 地铁四十八号线
 - 地铁四十九号线
 - 地铁五十号线
 - 地铁五十一号线
 - 地铁五十二号线
 - 地铁五十三号线
 - 地铁五十四号线
 - 地铁五十五号线
 - 地铁五十六号线
 - 地铁五十七号线
 - 地铁五十八号线
 - 地铁五十九号线
 - 地铁六十号线
 - 地铁六十一号线
 - 地铁六十二号线
 - 地铁六十三号线
 - 地铁六十四号线
 - 地铁六十五号线
 - 地铁六十六号线
 - 地铁六十七号线
 - 地铁六十八号线
 - 地铁六十九号线
 - 地铁七十号线
 - 地铁七十一号线
 - 地铁七十二号线
 - 地铁七十三号线
 - 地铁七十四号线
 - 地铁七十五号线
 - 地铁七十六号线
 - 地铁七十七号线
 - 地铁七十八号线
 - 地铁七十九号线
 - 地铁八十号线
 - 地铁八十一号线
 - 地铁八十二号线
 - 地铁八十三号线
 - 地铁八十四号线
 - 地铁八十五号线
 - 地铁八十六号线
 - 地铁八十七号线
 - 地铁八十八号线
 - 地铁八十九号线
 - 地铁九十号线
 - 地铁九十一号线
 - 地铁九十二号线
 - 地铁九十三号线
 - 地铁九十四号线
 - 地铁九十五号线
 - 地铁九十六号线
 - 地铁九十七号线
 - 地铁九十八号线
 - 地铁九十九号线
 - 地铁一百号线
 - 地铁一百零一条线
 - 地铁一百零二号线
 - 地铁一百零三号线
 - 地铁一百零四号线
 - 地铁一百零五号线
 - 地铁一百零六号线
 - 地铁一百零七号线
 - 地铁一百零八号线
 - 地铁一百零九号线
 - 地铁一百一十号线
 - 地铁一百一十一号线
 - 地铁一百一十二号线
 - 地铁一百一十三号线
 - 地铁一百一十四号线
 - 地铁一百一十五号线
 - 地铁一百一十六号线
 - 地铁一百一十七号线
 - 地铁一百一十八号线
 - 地铁一百一十九号线
 - 地铁一百二十号线
 - 地铁一百二十一号线
 - 地铁一百二十二号线
 - 地铁一百二十三号线
 - 地铁一百二十四号线
 - 地铁一百二十五号线
 - 地铁一百二十六号线
 - 地铁一百二十七号线
 - 地铁一百二十八号线
 - 地铁一百二十九号线
 - 地铁一百三十号线
 - 地铁一百三十一号线
 - 地铁一百三十二号线
 - 地铁一百三十三号线
 - 地铁一百三十四号线
 - 地铁一百三十五号线
 - 地铁一百三十六号线
 - 地铁一百三十七号线
 - 地铁一百三十八号线
 - 地铁一百三十九号线
 - 地铁一百四十号线
 - 地铁一百四十一号线
 - 地铁一百四十二号线
 - 地铁一百四十三号线
 - 地铁一百四十四号线
 - 地铁一百四十五号线
 - 地铁一百四十六号线
 - 地铁一百四十七号线
 - 地铁一百四十八号线
 - 地铁一百四十九号线
 - 地铁一百五十号线
 - 地铁一百五十一号线
 - 地铁一百五十二号线
 - 地铁一百五十三号线
 - 地铁一百五十四号线
 - 地铁一百五十五号线
 - 地铁一百五十六号线
 - 地铁一百五十七号线
 - 地铁一百五十八号线
 - 地铁一百五十九号线
 - 地铁一百六十号线
 - 地铁一百六十一号线
 - 地铁一百六十二号线
 - 地铁一百六十三号线
 - 地铁一百六十四号线
 - 地铁一百六十五号线
 - 地铁一百六十六号线
 - 地铁一百六十七号线
 - 地铁一百六十八号线
 - 地铁一百六十九号线
 - 地铁一百七十号线
 - 地铁一百七十一号线
 - 地铁一百七十二号线
 - 地铁一百七十三号线
 - 地铁一百七十四号线
 - 地铁一百七十五号线
 - 地铁一百七十六号线
 - 地铁一百七十七号线
 - 地铁一百七十八号线
 - 地铁一百七十九号线
 - 地铁一百八十号线
 - 地铁一百八十一号线
 - 地铁一百八十二号线
 - 地铁一百八十三号线
 - 地铁一百八十四号线
 - 地铁一百八十五号线
 - 地铁一百八十六号线
 - 地铁一百八十七号线
 - 地铁一百八十八号线
 - 地铁一百八十九号线
 - 地铁一百九十号线
 - 地铁一百九十一号线
 - 地铁一百九十二号线
 - 地铁一百九十三号线
 - 地铁一百九十四号线
 - 地铁一百九十五号线
 - 地铁一百九十六号线
 - 地铁一百九十七号线
 - 地铁一百九十八号线
 - 地铁一百九十九号线
 - 地铁二百号线



如何乘坐地铁，
从昌平到亦庄时
间最短？

内容提要

- 图的基本概念
- 存储表示
- 图的基本运算与周游
- 最小生成树
- 拓扑排序
- 关键路径
- 最短路径

图的生成树

- 对于连通^红的无向图或强连通^红的有向图，从任一顶点出发（或对于有根^红的有向图，从图的根顶点出发）周游^红，可以访问到所有的顶点。
- 周游时经过的边加上所有顶点构成图的一个连通子图^红，称为图的一棵生成树^蓝
 - 连通图的生成树是连通图的一个极小连通子图^红，它包含图中的全部 n 个顶点，但只有足以构成一棵树的 $n-1$ 条边

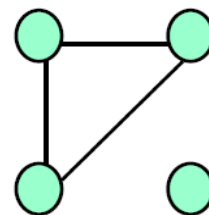
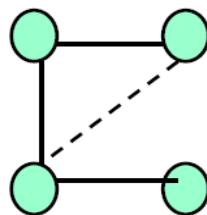
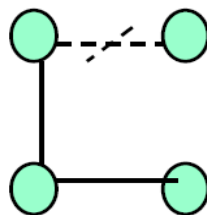
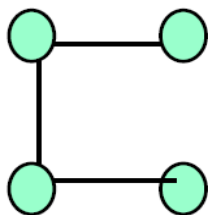
生成树

□ 如果在一颗生成树中

- 增加一条边，则必定构成环；
- 去掉一条边，则连通图变为不连通的。

□ 证明

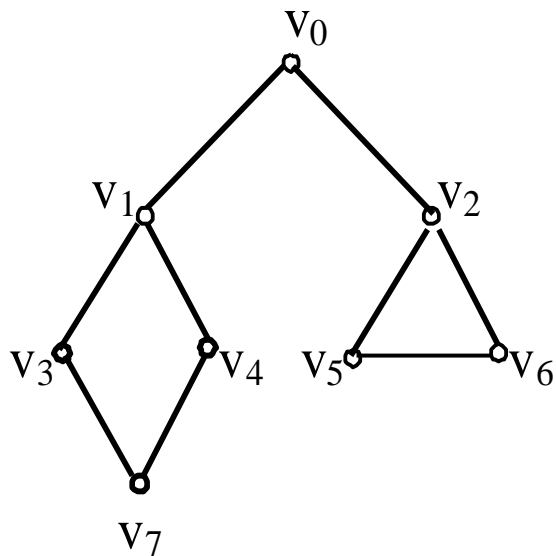
- 顶点数为 n ，边数小于 $n-1$ 的无向图必定是不连通的；
- 顶点数为 n ，边数大于 $n-1$ 的无向图必定存在环；
- 顶点数为 n ，边数为 $n-1$ 的无向图不一定是生成树。



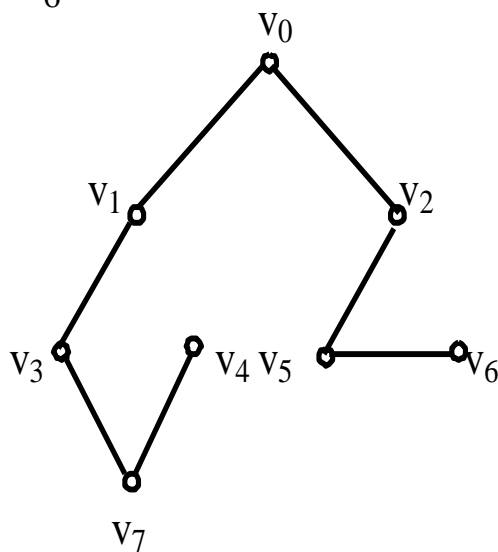
DFS/BFS树

- 连通的无向图或强连通的有向图，定义DFS/BFS生成树：
 - 从连通图的任一顶点出发，进行深度/广度优先周游，
 - 记录周游中访问的所有顶点，以及经过的边，
 - 得到的是深度/广度优先生成树
 - 简称为DFS/BFS生成树

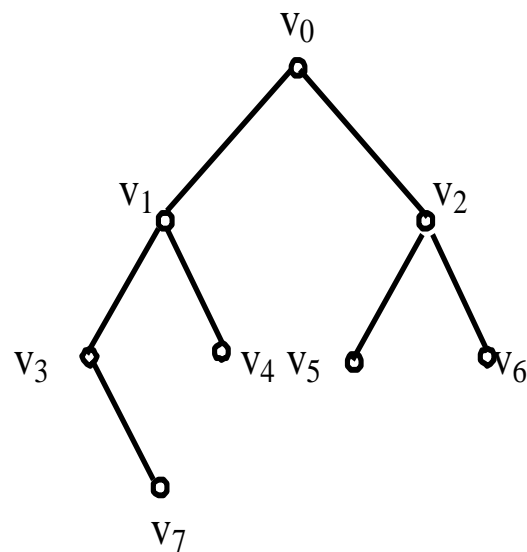
无向图的生成树-例子



例如：从无向图的顶点 v_0 出发分别进行深度优先周游和广度优先周游，得到的DFS及BFS生成树



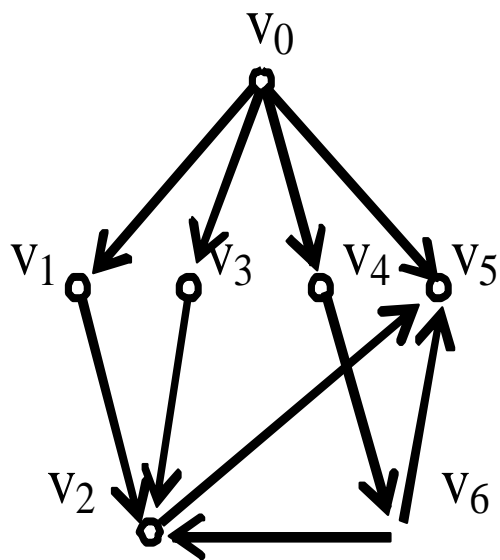
DFS生成树



BFS生成树

有向图的生成树-例子

从有向图的顶点 v_0 出发周游，得到的DFS及BFS生成树如图所示



说明

- 对于非连通的无向图和非强连通的有向图，从任一顶点出发无法访问到所有的顶点，只能得到各连通分量的生成树所组成的生成树林
- 图的生成树不唯一：
从不同顶点出发，或从同一顶点出发，但周游的路径不一样，则得到的生成树都不同

图的最小生成树

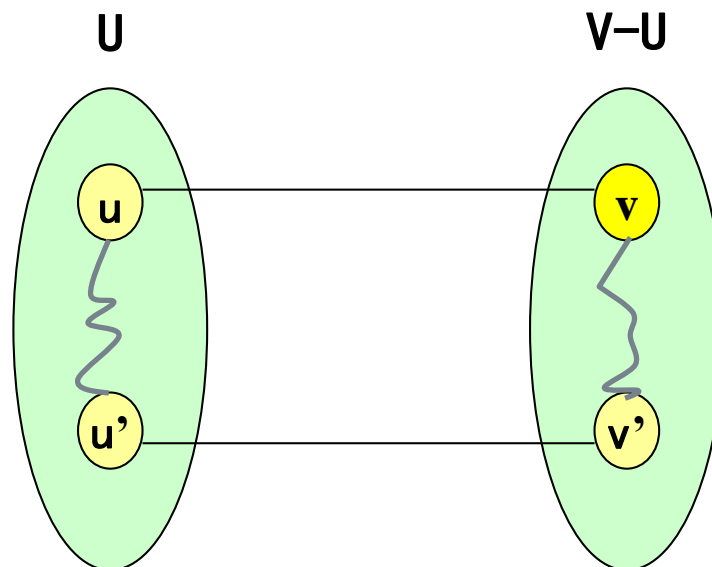
- 对于网络，其生成树中的边也带权，将生成树各边的权值总和称为生成树的权，并把权值最小的生成树称为最小生成树(Minimum Spanning Tree, 简称MST)
- MST应用非常广泛，例如城市中利用最小生成树建立通讯网络花费最小的方案：
 - 城市间通讯线路的布设， n 个城市最多有 $n(n-1)/2$ 条线路连接
 - 但根据架设通讯线路的代价需要，可以在 $n(n-1)/2$ 条可以选择的线路中选择连接 n 个城市并且代价最小的 $n-1$ 条线路，组成这 n 座城市之间的通讯连接。

MST性质

- 设 $G=(V,E)$ 是一个网络， U 是顶点集合 V 的一个真子集
- 如果
 - 边 (u,v) 的顶点 $u \in U$, $v \in V-U$, 且
 - 边 (u,v) 是图 G 中所有有一个端点在 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)

证明:

反证法



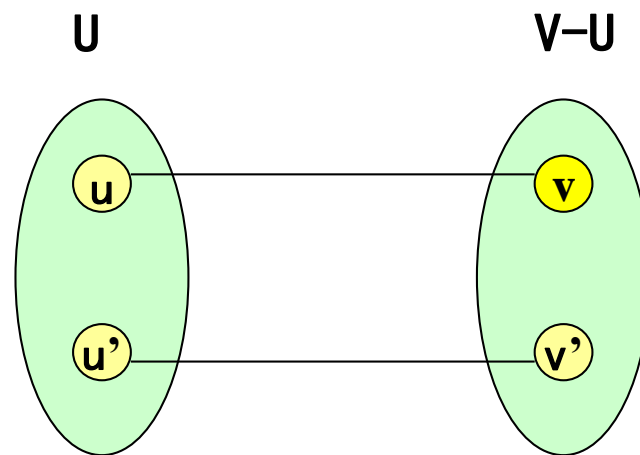
最小生成树的构造

- 最小生成树的构造利用了MST性质，一条一条地选择可以加入的边
- 主要有两种算法：
 - Prim算法（普里姆算法）
 - Kruskal算法（克鲁斯卡算法）

均为贪心算法

Prim算法的基本思想

- 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 $U=\{v_0\}$, 边集合 $TE=NULL$
- 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 $(u,v)(u \in U, v \in V-U)$, 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算法最小生成树的构造

□ 设图采用邻接矩阵表示法表示如下：

```
① #define MAXVEX 100    // 常数1
② typedef char VexType;
③ typedef float AdjType;

④ typedef struct {
⑤     VexType vexs[MAXVEX];           /* 顶点信息 */
⑥     AdjType arcs[MAXVEX][MAXVEX];  /* 邻接矩阵 */
⑦     int n;      /* 图的顶点个数 */
⑧ }Graph;
```

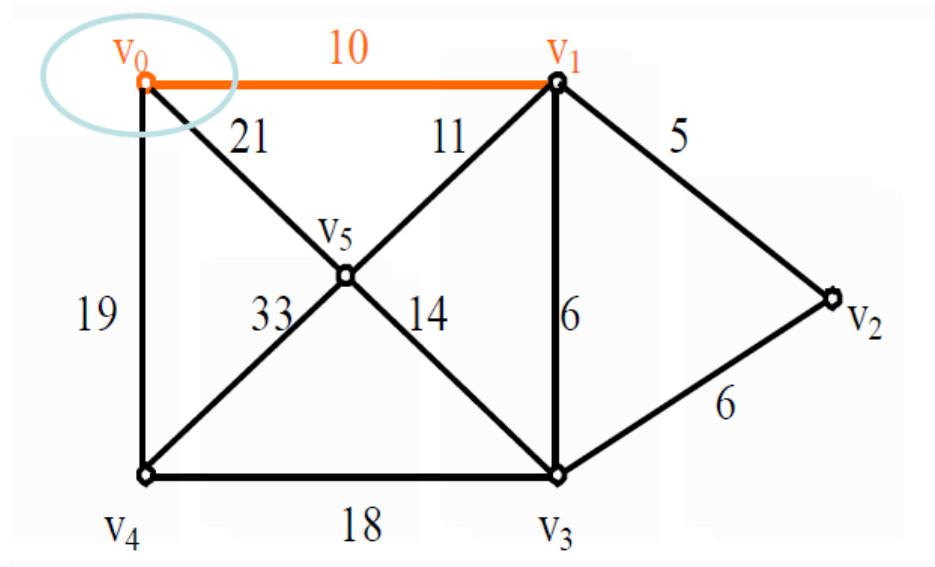
Prim算法最小生成树的构造

□ 在构造过程中定义一个类型为Edge的数组mst[n-1]

```
① typedef struct
② {
③     int start_vex, stop_vex;      /* 边的起点和终点 */
④     AdjType weight;              /* 边的权 */
⑤ }Edge;
⑥ Edge mst[n-1]; /* 算法结束时mst存放最小生成树的n-1条边*/
```

Prim算法最小生成树的构造过程1

最初集合U中只有一个顶点 v_0 ，mst中存放从 v_0 到其它 $n-1$ 个顶点的边；如果 v_j 与顶点 v_0 不相邻，则权值为 ∞ ；



$mst[5] = \{\{0, 1, 10\}, \{0, 2, \infty\}, \{0, 3, \infty\}, \{0, 4, 19\}, \{0, 5, 21\}\}$

$U = \{v_0\}$

Prim算法最小生成树的构造过程2

下面依次求出最小生成树的 $n-1$ 条边

- ① 当前 $mst[0]$ 到 $mst[n-2]$ 中存放的边都是一个顶点在集合 U 中，另一个顶点在集合 $V-U$ 中的边。
- ② 在 $mst[0]$ 到 $mst[n-2]$ 中选出权值最小的边 $mst[\min]$ ，将其加入最小生成树，即将 $mst[\min]$ 与 $mst[0]$ 互换。

$mst[0]$ 中存放的是刚加入最小生成树的边，而 $mst[0].stop_vex$ 是新加入集合 U 的顶点的下标(假设为 x)，则 v_x 即为加入最小生成树中的新结点。

$$U = \{V_0, V_x\}$$

最小生成树的构造过程3

下面依次求出最小生成树的 $n-1$ 条边

$$U = \{V_0, V_x\}$$

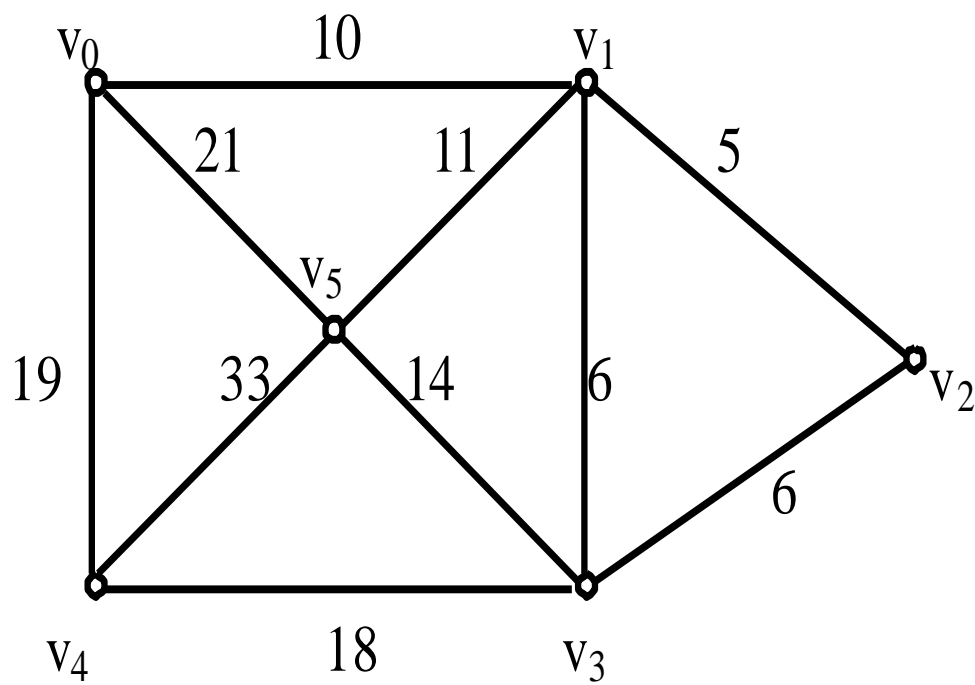
③ 调整mst[1]到mst[n-2]

若集合 $V-U$ 中的顶点 v_y 到新顶点 v_x 的边长度(权)比原来 v_y 到集合 U 中其他旧顶点 v_z 的长度小, 就需要将mst中的边 (v_z, v_y) 调整为边 (v_x, v_y)

④ 从mst[1]到mst[n-2]重复上述(2)、(3)操作, 每次在一个顶点在集合 U 中, 另一个顶点在集合 $V-U$ 中的所有边中, 选出一条最小的边, 将这条边和对应的顶点加入最小生成树, 并调整mst中的边, 直到 $n-1$ 条边都在最小生成树中为止。

例子

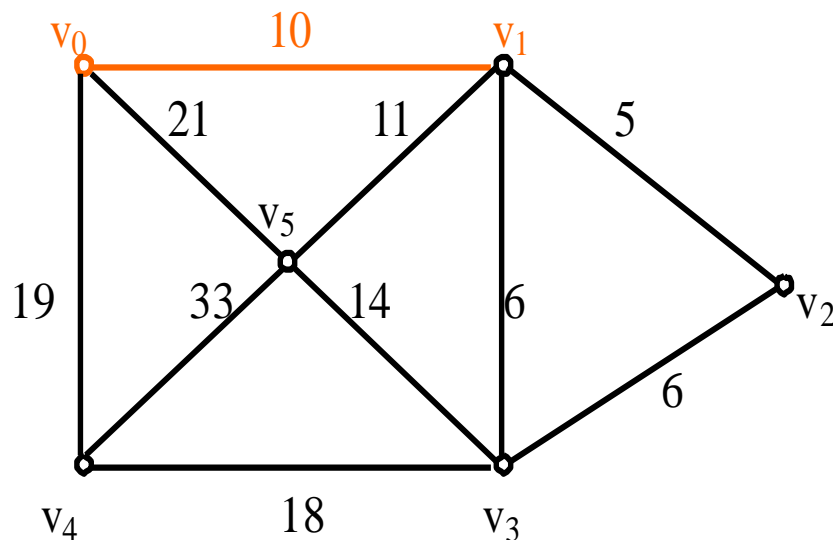
已知带权图G及其邻接矩阵如图所示
请构造该图的最小生成树



0	10	∞	∞	19	21
10	0	5	6	∞	11
∞	5	0	6	∞	∞
∞	6	6	0	18	14
19	∞	∞	18	0	33
21	11	∞	14	33	0

- $n=6$, 只有顶点 v_0 在最小生成树中。
 $mst[5]=\{\{0,1,10\}, \{0,2,\infty\}, \{0,3,\infty\}, \{0,4,19\}, \{0,5,21\}\}$
 $U=\{V_0\}$

- 在 $mst[0]$ 到 $mst[4]$ 中找出权值最小的边 $mst[0]$, 即 (v_0, v_1) , 将顶点 v_1 及边 (v_0, v_1) 加入最小生成树。
 $U=\{V_0, V_1\}$



$mst[5] = \{ \{0,1,10\}, \{0,2,\infty\}, \{0,3,\infty\}, \{0,4,19\}, \{0,5,21\} \}$

➤ 调整 $mst[1]$ 到 $mst[4]$

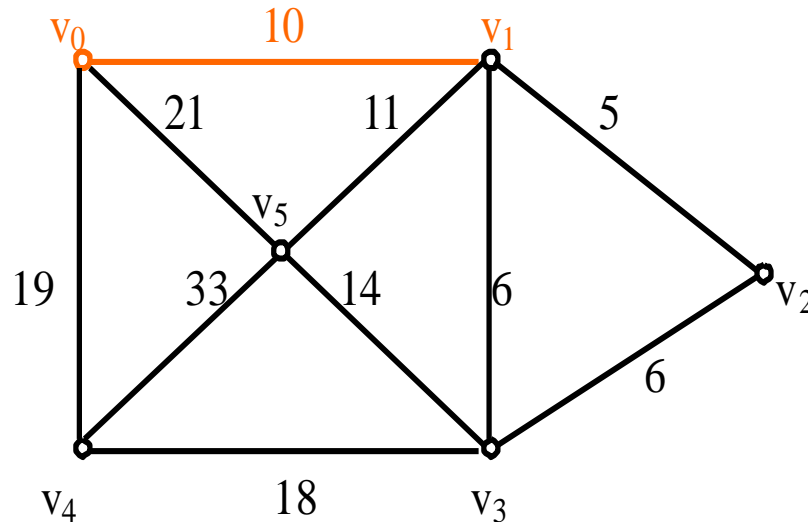
$(v_1, v_2)=5$ 小于 $(v_0, v_2) \{0,2,\infty\}$ 调整

$(v_1, v_3)=6$ 小于 $(v_0, v_3) \{0,3,\infty\}$ 调整

$(v_1, v_4)=\infty$ 大于 $(v_0, v_4) \{0,4,19\}$ 不需要调整

$(v_1, v_5)=11$ 小于 $(v_0, v_5) \{0,5,21\}$ 调整

$mst[5] = \{ \{0,1,10\}, \{1,2,5\}, \{1,3,6\}, \{0,4,19\}, \{1,5,11\} \}$



$mst[5]=\{ \{0,1,10\}, \{1,2,5\}, \{1,3,6\}, \{0,4,19\}, \{1,5,11\} \}$

➤ 在 $mst[1]$ 到 $mst[4]$ 中找出权值最小的边 $mst[1]$, 即 (v_1, v_2) , 将顶点 v_2 及边 (v_1, v_2) 加入最小生成树

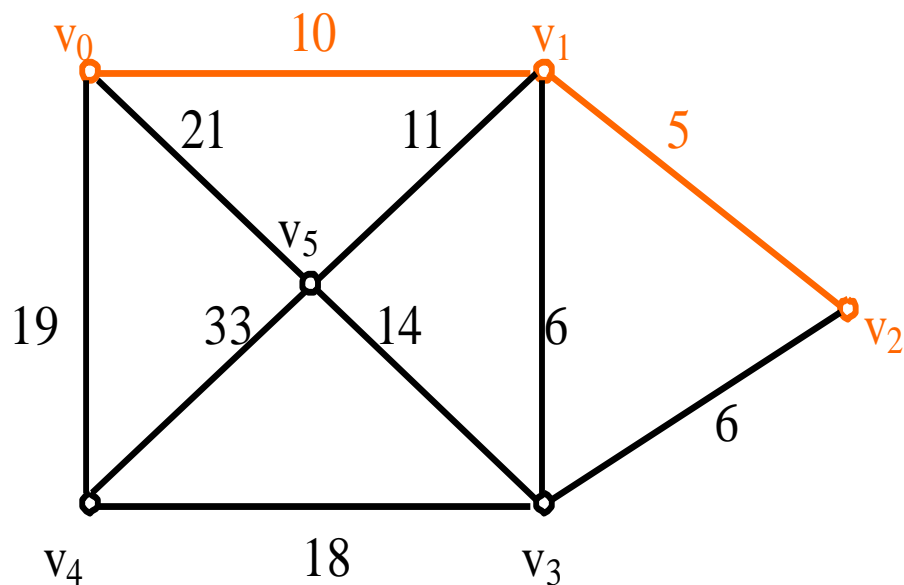
➤ 调整 $mst[2]$ 到 $mst[4]$

$(v_2, v_3)=6$ 不小于 (v_1, v_3) $\{1,3,6\}$ 不需要调整

$(v_2, v_4)=\infty$ 大于 (v_0, v_4) $\{0,4,19\}$ 不需要调整

$(v_2, v_5)=\infty$ 大于 (v_1, v_5) $\{1,5,11\}$ 不需要调整

$mst[5]=\{ \{0,1,10\}, \{1,2,5\}, \{1,3,6\}, \{0,4,19\}, \{1,5,11\} \}$



$U = \{V_0, V_1, V_2\}$

$mst[5] = \{ \{0,1,10\}, \{1,2,5\}, \{1,3,6\}, \{0,4,19\}, \{1,5,11\} \}$

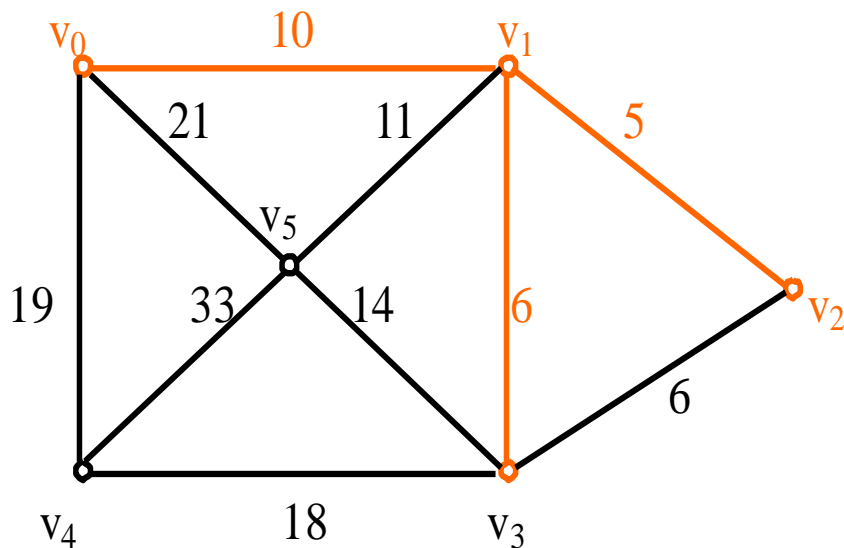
➤ 在 $mst[2]$ 到 $mst[4]$ 中找出权值最小的边 $mst[2]$, 即 (v_1, v_3) , 将顶点 v_3 及边 (v_1, v_3) 加入最小生成树

➤ 调整 $mst[3]$ 到 $mst[4]$

$(v_3, v_4)=18$ 小于 (v_0, v_4) $\{0,4,19\}$ 调整

$(v_3, v_5)=14$ 大于 (v_1, v_5) $\{1,5,11\}$ 不需要调整

$mst[5] = \{ \{0,1,10\}, \{1,2,5\}, \{1,3,6\}, \{3,4,18\}, \{1,5,11\} \}$



$U = \{v_0, v_1, v_2, v_3\}$

$mst[5] = \{ \{0,1,10\}, \{1,2,5\}, \{1,3,6\}, \{3,4,18\}, \{1,5,11\} \}$

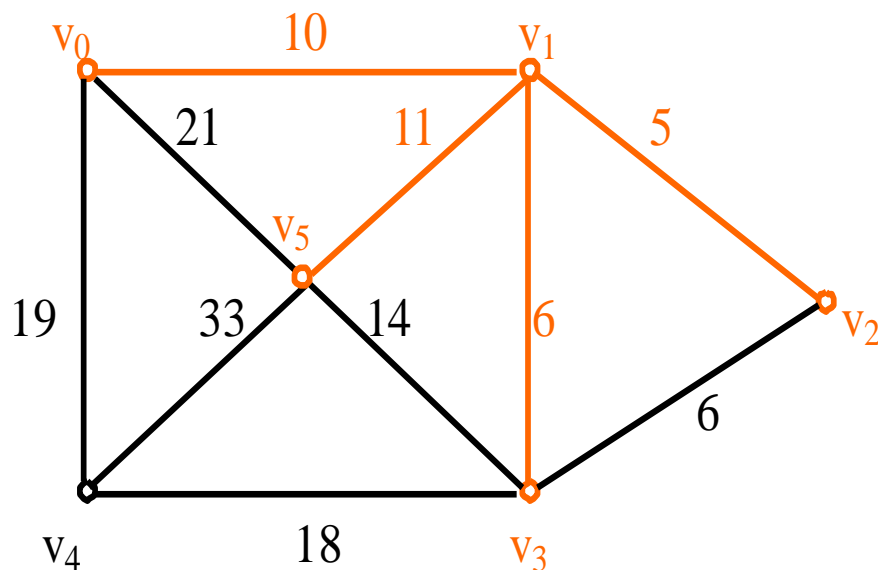
- 在 $mst[3]$ 到 $mst[4]$ 中找出权值最小的边 $mst[4]$ ，即 (v_1, v_5) ，将顶点 v_5 及边 (v_1, v_5) 加入最小生成树。互换 $mst[3]$ 和 $mst[4]$

$mst[5] = \{ \{0,1,10\}, \{1,2,5\}, \{1,3,6\}, \{1,5,11\}, \{3,4,18\} \}$

- 调整 $mst[4]$

$(v_5, v_4) = 33$ 大于 (v_3, v_4) $\{3,4,18\}$ 不需要调整

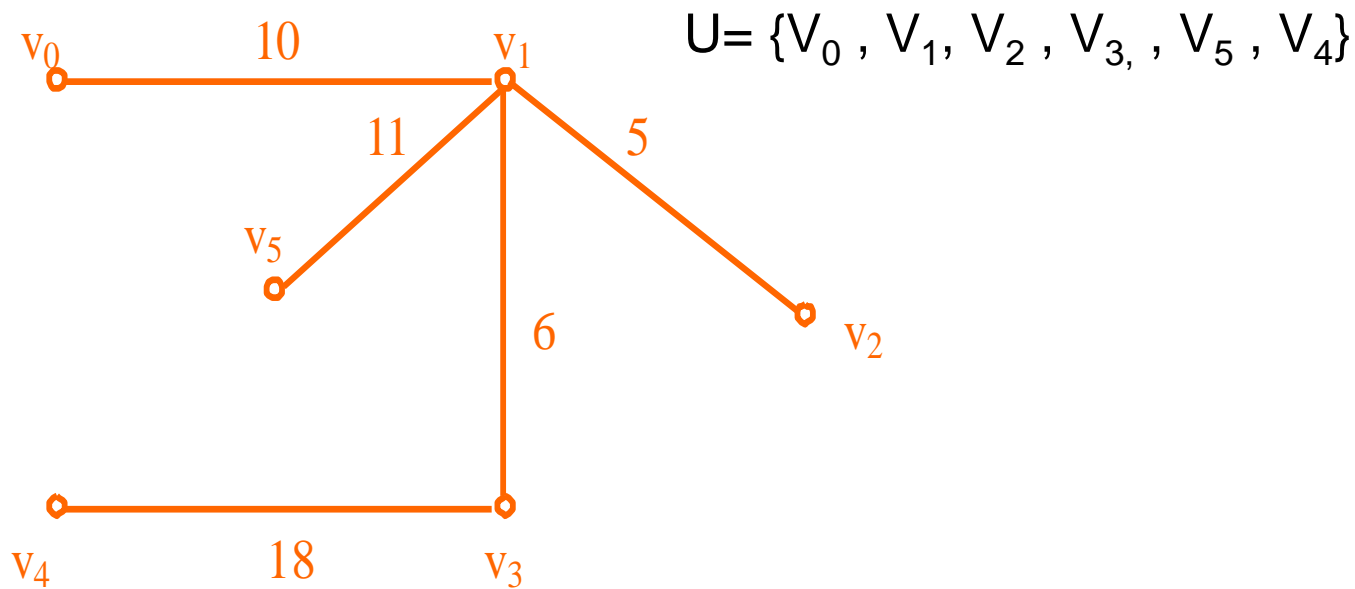
$mst[5] = \{ \{0,1,10\}, \{1,2,5\}, \{1,3,6\}, \{1,5,11\}, \{3,4,18\} \}$



$U = \{V_0, V_1, V_2, V_3, V_5\}$

➤ 将mst[4]加入最小生成树。

➤ 最小生成树如图所示



Prim 算法

```
1. void prim(Graph * pGraph, Edge mst[])
2. {
3.     int i, j, min, vx, vy;
4.     float weight, minweight;
5.     Edge edge;
6.     for(i=0; i<pGraph->n-1; i++) //初始化mst[i]
7.     {
8.         mst[i].start_vex=0;
9.         mst[i].stop_vex=i+1;
10.        mst[i].weight=pGraph->arcs[0][i+1];
11.    }
```

Prim 算法

```
12. for(i=0; i<pGraph->n-1; i++)    /* 共n-1条边 */
13. {
14.     minweight=MAX; min=i;
15.     /* 从所有边(vx,vy)(vx∈U,vy∈V-U)中选出最短的边 */
16.     for(j=i; j<pGraph->n-1; j++)
17.         if(mst[j].weight<minweight)
18.         {
19.             minweight=mst[j].weight;    min = j;
20.         }
21.     /* mst[min]是最短边(vx,vy)(vx∈U, vy∈V-U), 将mst[min]加入 */
22.     edge=mst[min]; mst[min]=mst[i]; mst[i]=edge; //替换 mst[i], mst[min]
23.     vx=mst[i].stop_vex;    /* vx为刚加入最小生成树的顶点下标 */
```

Prim 算法

```
24.     for(j=i+1; j<pGraph->n-1; j++) /* 调整mst[j+1]到mst[n-1] */
25.     {
26.         vy=mst[j].stop_vex; //取V-U中的一个顶点vy
27.         weight=pGraph->arcs[vx][vy]; //U中新顶点vx与vy边的权重
28.         if(weight<mst[j].weight) //如果新权重 小于旧权重
29.         {
30.             mst[j].weight=weight; //替换为权重小的边
31.             mst[j].start_vex=vx;
32.         }
33.     } /* 结束调整*/
34. } /* 结束共n-1条边 */
35. }
```

Prim算法时间复杂度

- Prim算法的时间主要花费在选择最小生成树的 $n-1$ 条边上。外循环执行 $n-1$ 次，内循环两个，时间耗费为：

$$\sum_{i=0}^{n-2} \left(\sum_{j=i}^{n-2} O(1) + \sum_{j=i+1}^{n-2} O(1) \right)$$
$$\approx 2 \sum_{i=0}^{n-2} \sum_{j=i}^{n-2} O(1)$$

- 整个算法的时间复杂度为 $O(n^2)$

最小生成树的构造

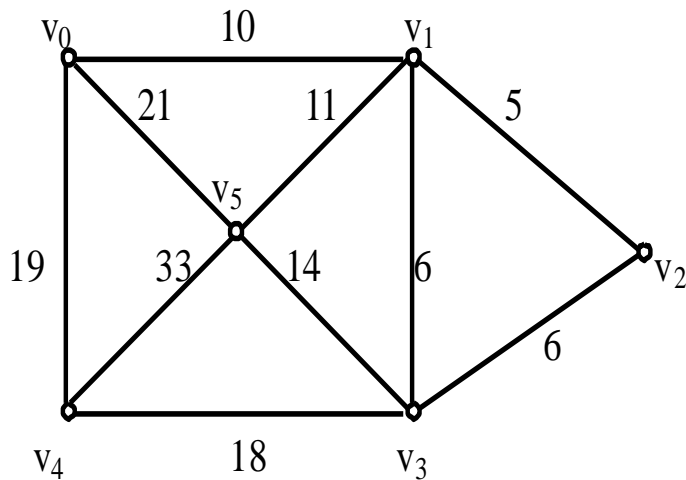
- 最小生成树的构造利用了MST性质，一条一条地选择将要加入的边
- 主要有两种算法：
 - Prim算法（普里姆算法）
 - Kruskal算法（克鲁斯卡算法）

Kruskal算法的基本思想

- ① 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 $T=(V, \varphi)$ ， T 中每个顶点自成为一个连通分量
- ② 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
- ③ 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 - 例子

□ 用Kruskal方法构造下图的最小生成树



0	10	∞	∞	19	21
10	0	5	6	∞	11
∞	5	0	6	∞	∞
∞	6	6	0	18	14
19	∞	∞	18	0	33
21	11	∞	14	33	0

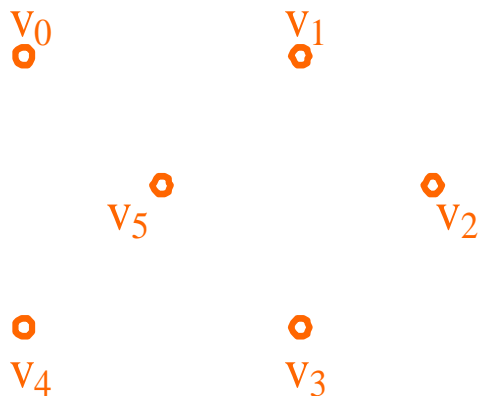
□ 集合E中的边按递增顺序排列为：

$(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_0, v_1), (v_1, v_5), (v_3, v_5), (v_3, v_4),$
 $(v_0, v_4), (v_0, v_5), (v_4, v_5)$

Kruskal算法 - 例子

$(v1, v2), (v1, v3), (v2, v3), (v0, v1), (v1, v5),$
 $(v3, v5), (v3, v4), (v0, v4), (v0, v5), (v4, v5)$

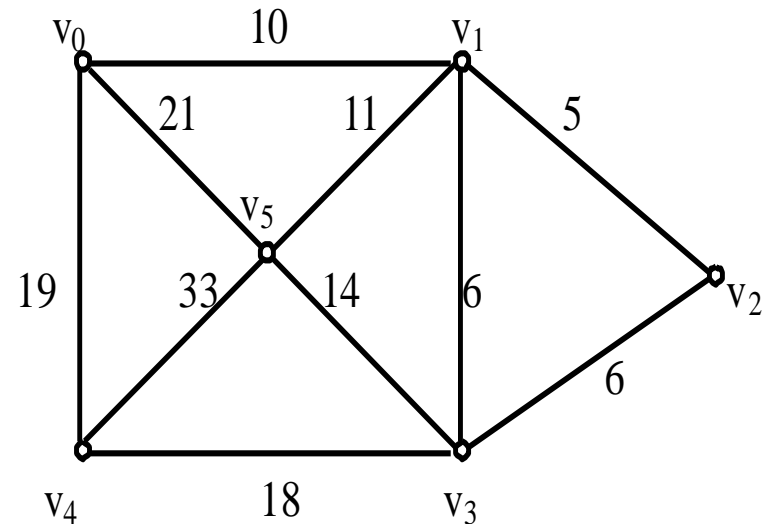
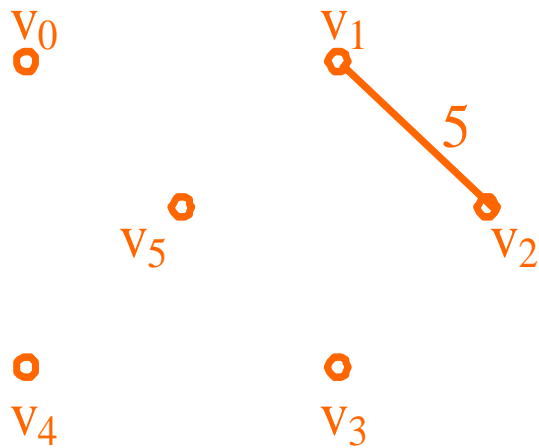
①. 初始时, T为只有6个顶点的非连通图



Kruskal算法 - 例子

(v_1, v_2) , (v_1, v_3) , (v_2, v_3) , (v_0, v_1) , (v_1, v_5) ,
 (v_3, v_5) , (v_3, v_4) , (v_0, v_4) , (v_0, v_5) , (v_4, v_5)

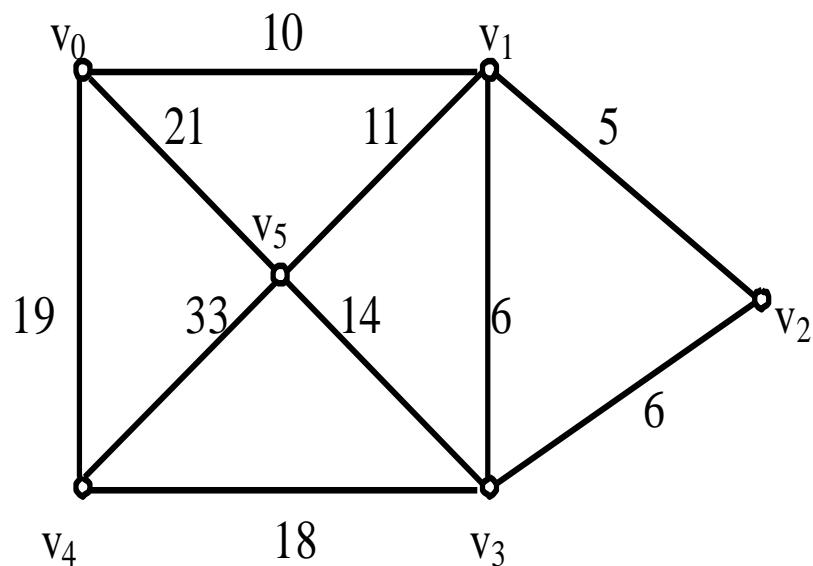
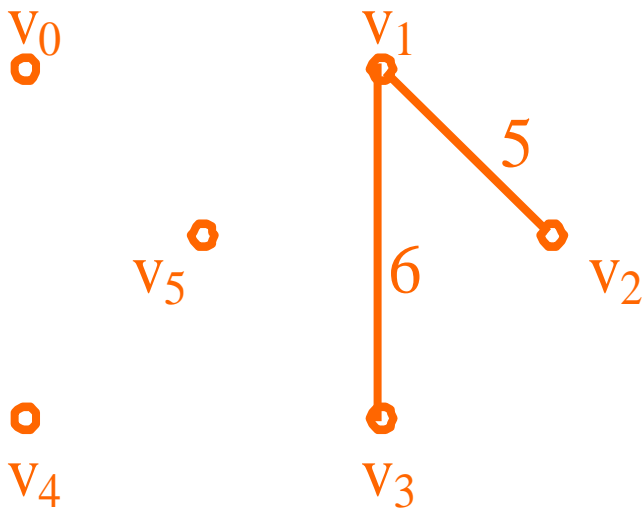
②. 边 (v_1, v_2) 的两个顶点 v_1 , v_2 分别属于两个连通分量, 将边 (v_1, v_2) 加入T



Kruskal算法 - 例子

$(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_0, v_1), (v_1, v_5),$
 $(v_3, v_5), (v_3, v_4), (v_0, v_4), (v_0, v_5), (v_4, v_5)$

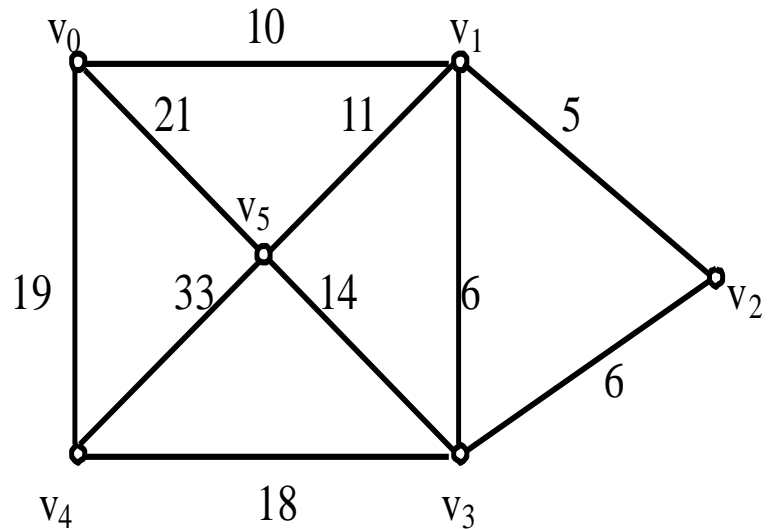
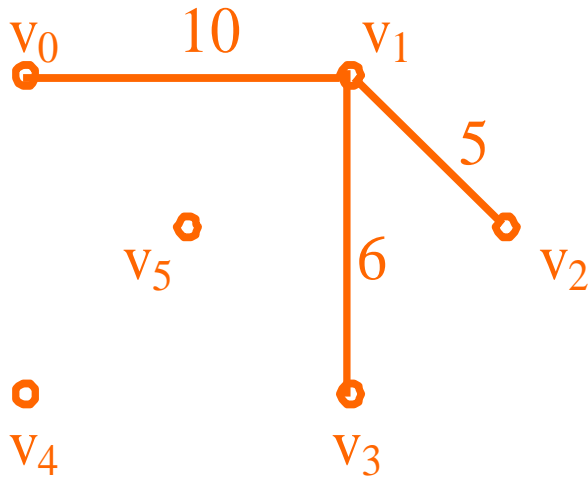
③. 同理，将边 (v_1, v_3) 加入T



Kruskal算法 - 例子

$(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_0, v_1), (v_1, v_5),$
 $(v_3, v_5), (v_3, v_4), (v_0, v_4), (v_0, v_5), (v_4, v_5)$

- ④ 由于边 (v_2, v_3) 的两个顶点 v_2, v_3 属于同一个连通分量，因此，舍去这条边，将 (v_0, v_1) 加入

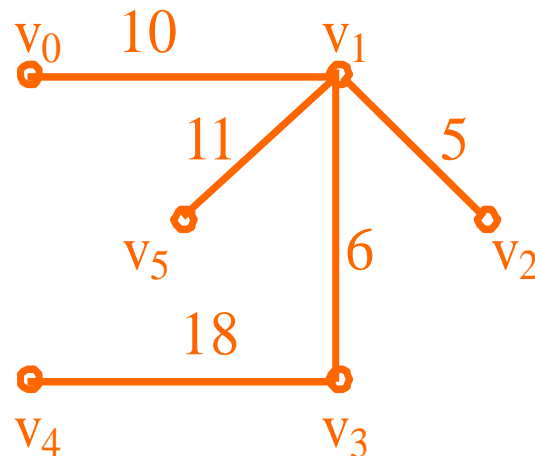
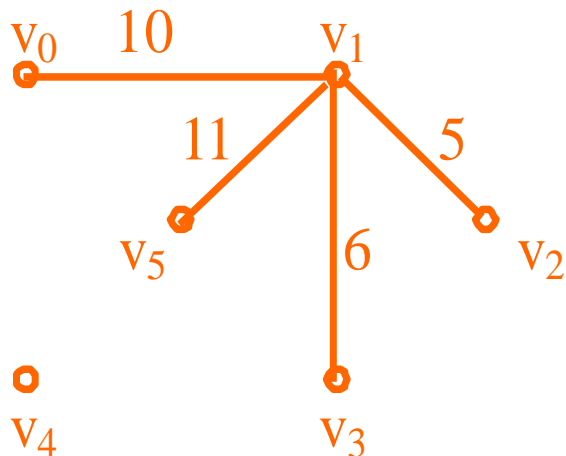


Kruskal算法 - 例子

$(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_0, v_1), (v_1, v_5),$
 $(v_3, v_5), (v_3, v_4), (v_0, v_4), (v_0, v_5), (v_4, v_5)$

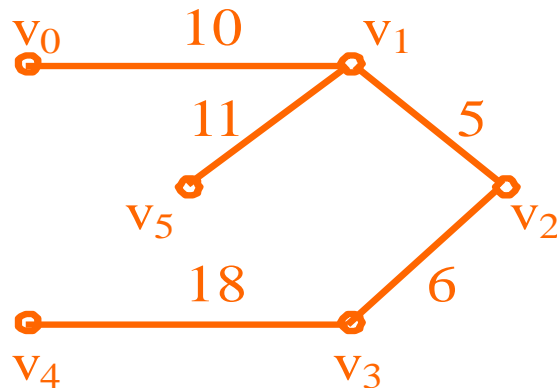
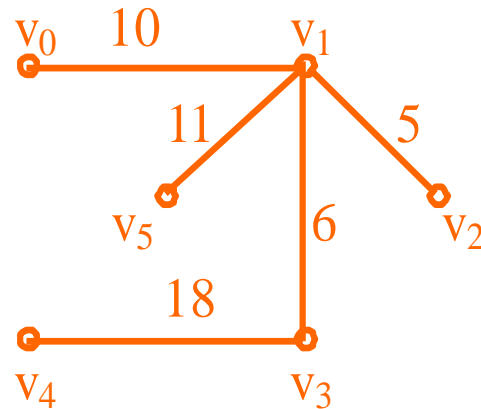
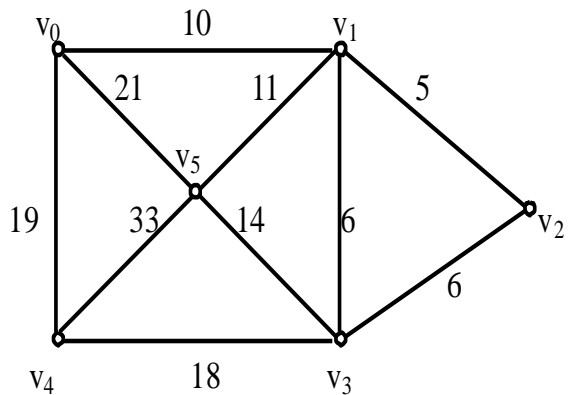
⑤同理将边 (v_1, v_5) 加入T, 边 (v_3, v_5) 舍去, 边 (v_3, v_4) 加入T

这时T中含的边数为5条, 成为一个连通分量, T就是G的一棵最小生成树



说明

图的最小生成树并不一定是唯一的，例如，上图中边 (v_1, v_3) 和 (v_2, v_3) 的长度相同，该图的另一棵最小生成树如下图所示



Kruskal 算法框架

$T=(V,\varphi)$

While (T中所含边数 $<n-1$)

{

 从E中选取当前最短边 (u,v) ; /* 需要先按权值排序 */

 从E中删去（或者标记）边 (u,v) ;

 if((u,v) 加入T中后不产生回路)

 将边 (u,v) 加入T中;

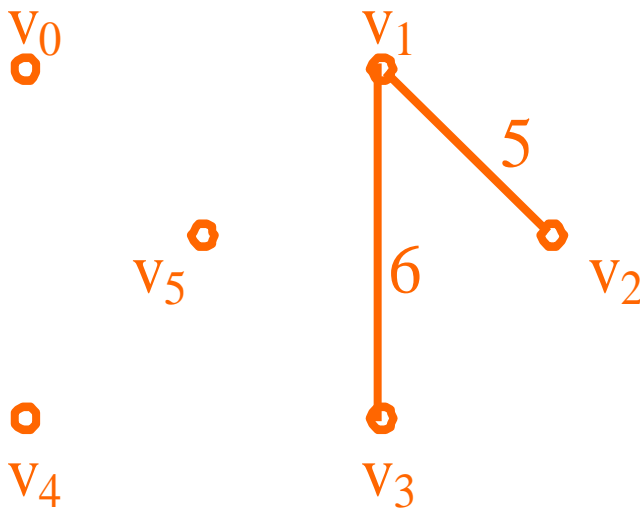
}

Kruskal 算法的关键

- 取权值最小的边
 - 首先对边进行完全排序？
 - 使用最小值堆来实现！一次取一条边。
- 实际上在完成MST 前仅需访问一小部分边。

Kruskal 算法的关键

- 确定加入一条边是否会产生回路
 - 可以记录每个连通分量中每个节点的“父节点”
 - 如果加入的边连接的两个顶点“父节点”不同。一定不会产生回路



```

int kruskal(Graph * pGraph, Edge mst[]) {
    int i, j, num=0, start, stop;
    float minweight;
    int status[pGraph->n];
    for(i=0; i<pGraph->n; i++)
        status[i]=i; /* 记录每个顶点的“根节点”序号，初始时每个顶点就是自己的“根节点” */
    while(num<pGraph->n-1) { /* 共n-1条边 */
        minweight=MAX;
        for(i=0; j<pGraph->n-1; ++i)
            for(j=i+1; j<pGraph->n; ++j)
                if(pGraph->arcs[i][j] < minweight) {
                    minweight= pGraph->arcs[i][j];
                    start = i; stop = j;
                } /* start和stop是找到的最短边的起点和终点 */
        if(minweight == MAX) return FALSE; /* 无MST */
        if(status[start] != status[stop]) { /* start stop顶点的根节点不同，不属于同一连通分量 */
            mst[num].start_vex= start;
            mst[num].end_vex= stop;
            mst[num].weight = pGraph->arcs[start][stop];
            ++num; /* 原以status[stop]为根节点的顶点，现在的根节点更新为status[start] */
            for( j=status[stop], i=0; i < pGraph->n; i++)
                if(status[i] == j).
                    status[i] = status[start];
        }
        pGraph->arcs[start][stop] = MAX;
    }
    return TRUE;
}

```

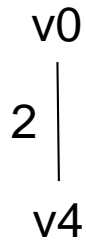
内容提要

- 图的基本概念
- 存储表示
- 图的基本运算与周游
- 最小生成树
- 拓扑排序
- 关键路径
- 最短路径

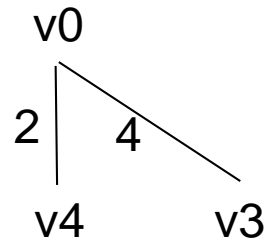
(1)

v0

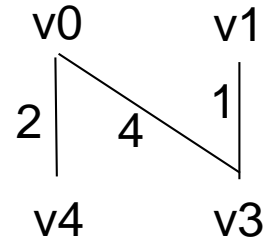
(2)



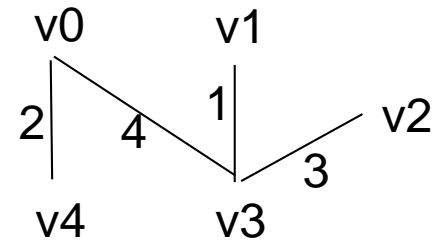
(3)



(4)



(5)



(1) $\{\{0,1,7\}, \{0,2, \infty\}, \{0,3,4\}, \{0,4,2\}\}$

(2) $\{\{0,4,2\}, \{0,2, \infty\}, \{0,3,4\}, \{4,1,5\}\}$

(3) $\{\{0,4,2\}, \{0,3,4\}, \{3,2,3\}, \{3,1,1\}\}$

(4) $\{\{0,4,2\}, \{0,3,4\}, \{3,1,1\}, \{3,2,3\}\}$

(5) $\{\{0,4,2\}, \{0,3,4\}, \{3,1,1\}, \{3,2,3\}\}$