第八章 图

张史梁
slzhang.jdl@pku.edu.cn

图中的一个重要问题

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

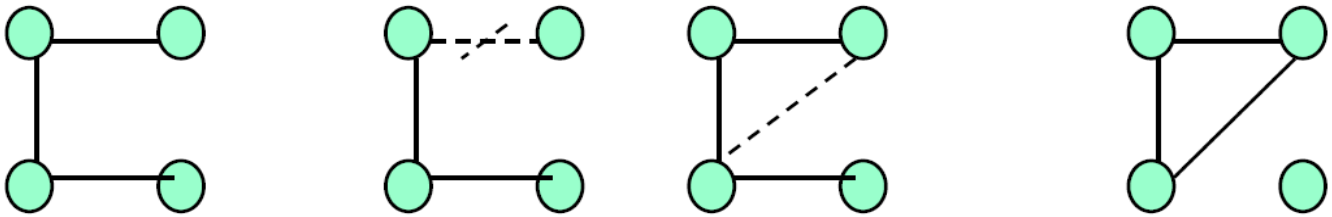
图的生成树

- 对于连通的无向图或强连通的有向图，从任一顶点出发（或对于有根的有向图，从图的根顶点出发）周游，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，称为图的一棵生成树
  - 连通图的生成树是连通图的一个极小连通子图，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

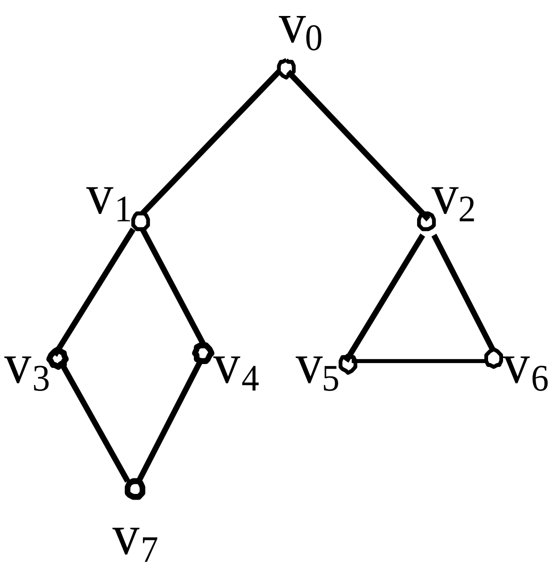
- 如果在一颗生成树中
  - 增加一条边，则必定构成环；
  - 去掉一条边，则连通图变为不连通的。
- 证明
  - 顶点数为n，边数小于n-1的无向图必定是不连通的；
  - 顶点数为n，边数大于n-1的无向图必定存在环；
  - 顶点数为n，边数为n-1的无向图不一定是生成树。



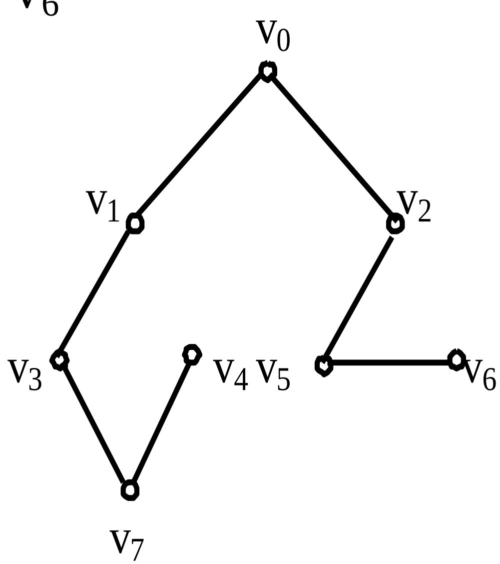
DFS/BFS树

- 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

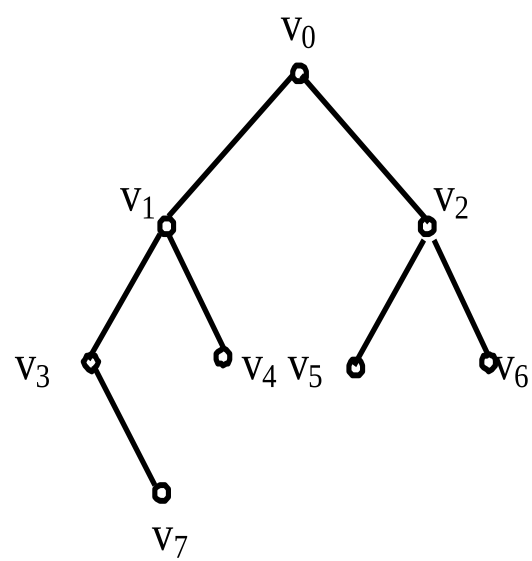
无向图的生成树-例子



例如：从无向图的顶点v0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



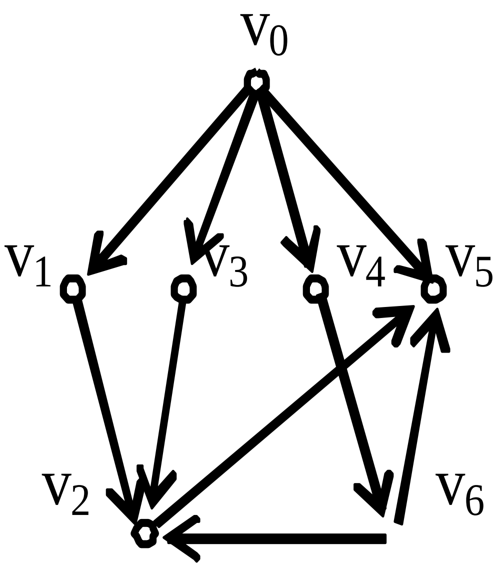
DFS生成树



BFS生成树

有向图的生成树-例子

从有向图的顶点v0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)

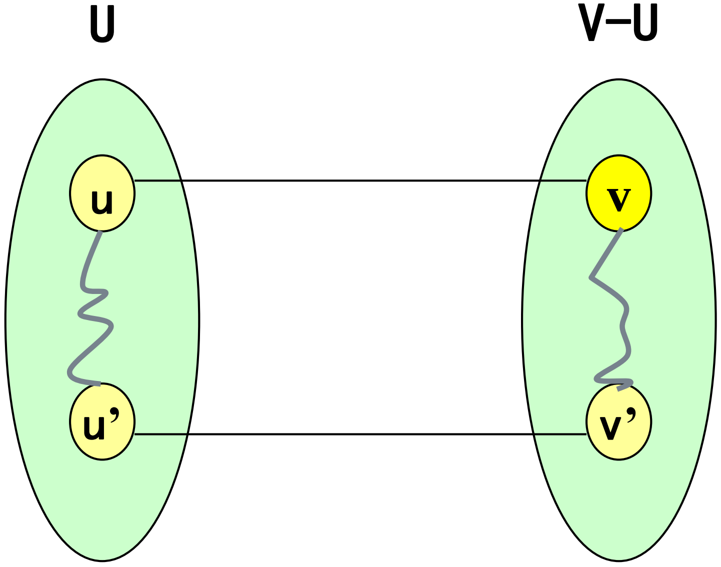
- MST应用非常广泛，例如城市中利用最小生成树建立通讯网络花费最小的方案：
  - 城市间通讯线路的布设，n个城市最多有n(n-1)/2条线路连接
  - 但根据架设通讯线路的代价需要，可以在n(n-1)/2条可以选择的线路中选择连接n个城市并且代价最小的n-1条线路，组成这n座城市之间的通讯连接。

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∈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)

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）

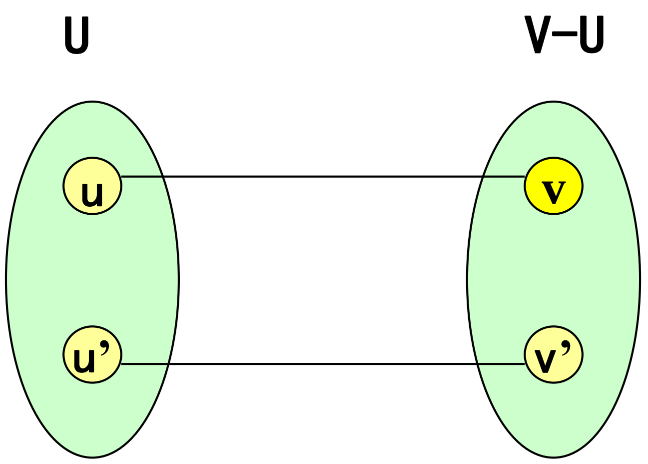
均为贪心算法

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∈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;
```

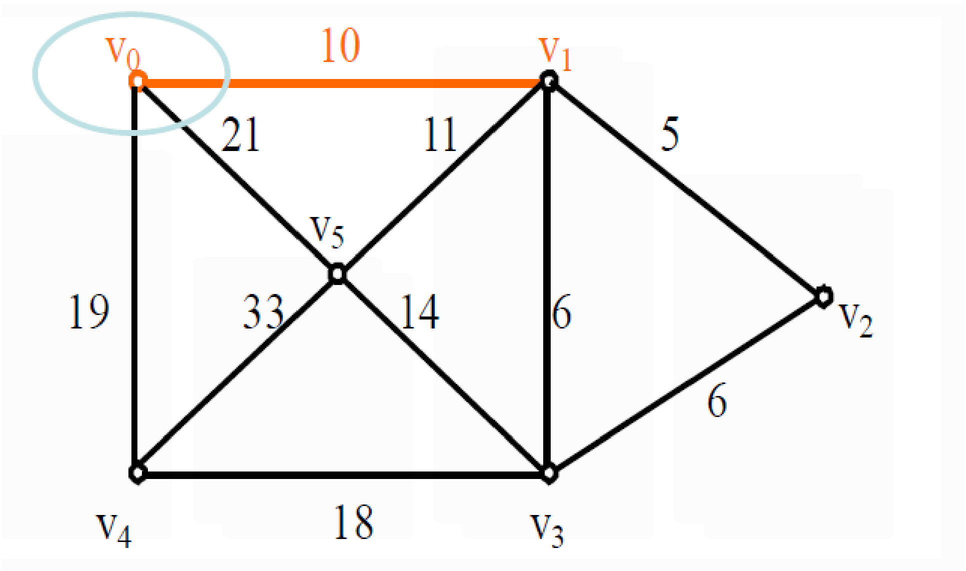
Prim算法最小生成树的构造

- 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,∞}, {0,3,∞}, {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

$U=\{V_0, V_x\}$

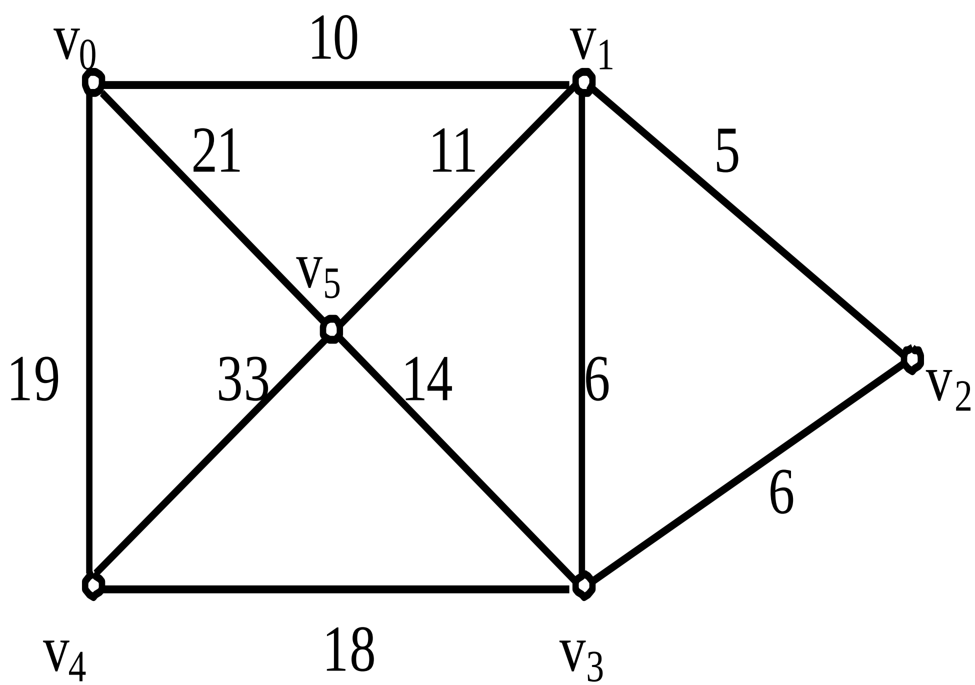
③ 调整mst[1]到mst[n-2]
若集合V-U中的顶点v_y到新顶点v_x的边长度(权)比原来v_y到集合U中其他旧顶点v_z的长度小，就需要将mst中的边(v_z, v_y)调整为边(v_x, v_y)

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

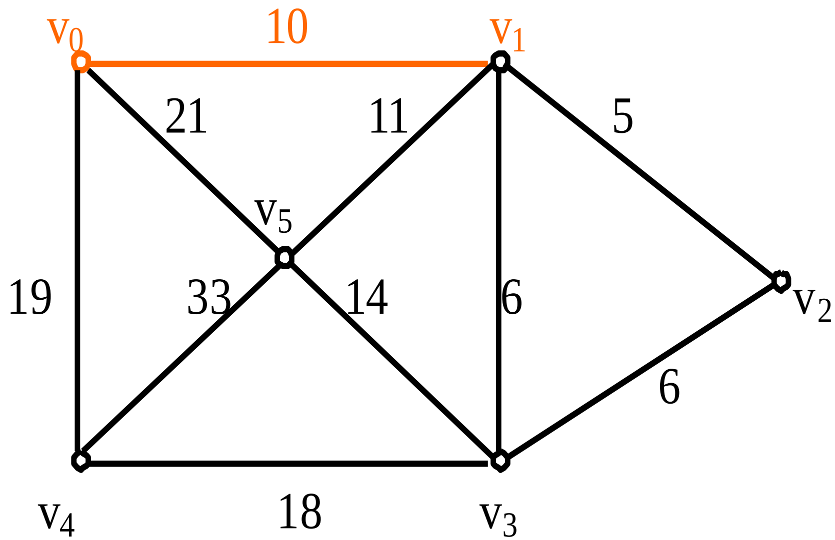


$$\begin{bmatrix} 0 & 10 & \infty & \infty & 19 & 21 \\ 10 & 0 & 5 & 6 & \infty & 11 \\ \infty & 5 & 0 & 6 & \infty & \infty \\ \infty & 6 & 6 & 0 & 18 & 14 \\ 19 & \infty & \infty & 18 & 0 & 33 \\ 21 & 11 & \infty & 14 & 33 & 0 \end{bmatrix}$$

- n=6，只有顶点v_0在最小生成树中。
  mst[5]={{0,1,10}, {0,2,∞}, {0,3,∞}, {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(v0, v1)，将顶点v1及边(v0, v1)加入最小生成树。

  U= {V_0 , V_1}



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

- 调整mst[1]到mst[4]

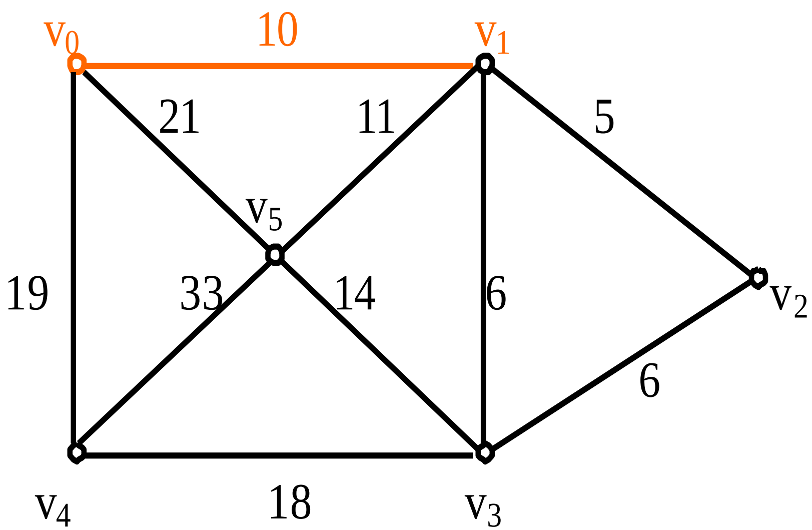
  (v_1, v_2)=5 小于(v_0, v_2) {0,2,∞} 调整

  (v_1, v_3)=6 小于(v_0, v_3) {0,3,∞} 调整

  (v_1, v_4)=∞ 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} }

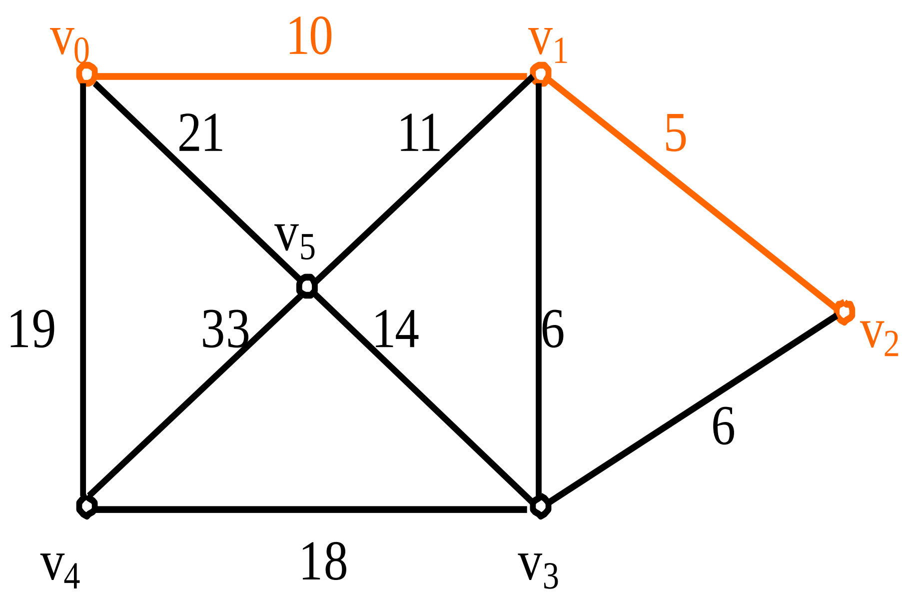


$mst[5]=\{\{0,1,10\}, \{1,2,5\}, \{1,3,6\}, \{0,4,19\}, \{1,5,11\}\}$

- 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

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



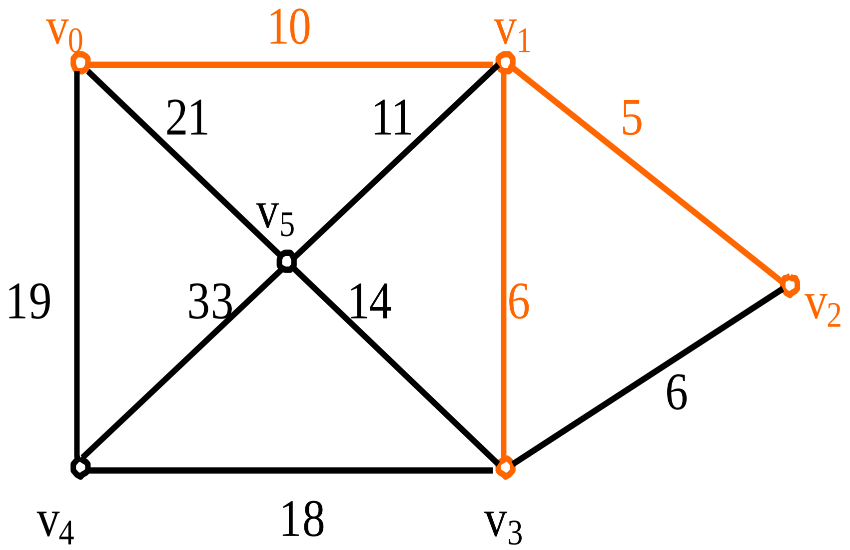
$U=\{V_0, V_1, V_2\}$

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

- 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} }

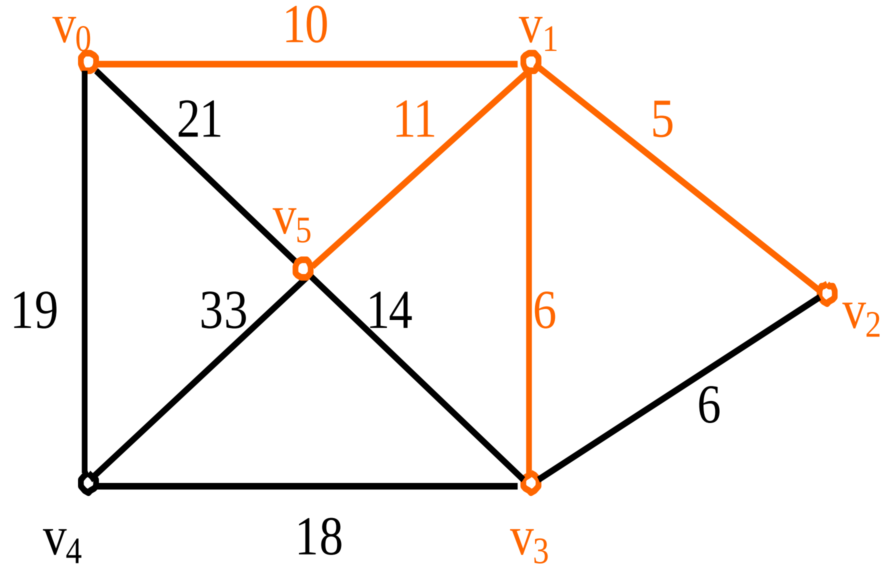
- 在mst[3]到mst[4]中找出权值最小的边mst[4]，即(v_1, v_5)，将顶点v_5及边(v_1, v_5)加入最小生成树。互换mst[3]和mst[4]

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

- 调整mst[4]

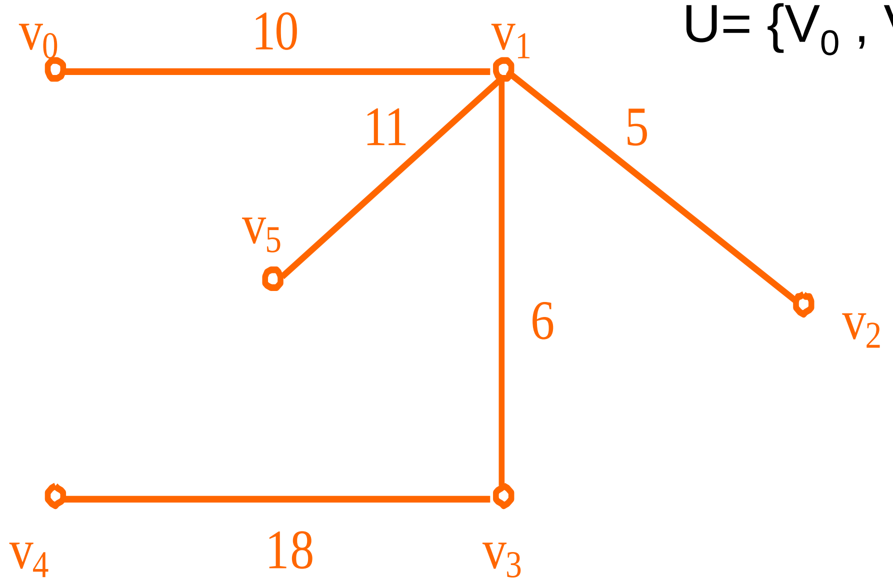
(v_5, v_4)=33 大于(v_3, v_4) {3,4,18} 不需要调整

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



U= {V_0 , V_1, V_2 , $V_{3,}$, V_5 , V_4}

Prim 算法

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

Prim 算法

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

Prim 算法

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

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

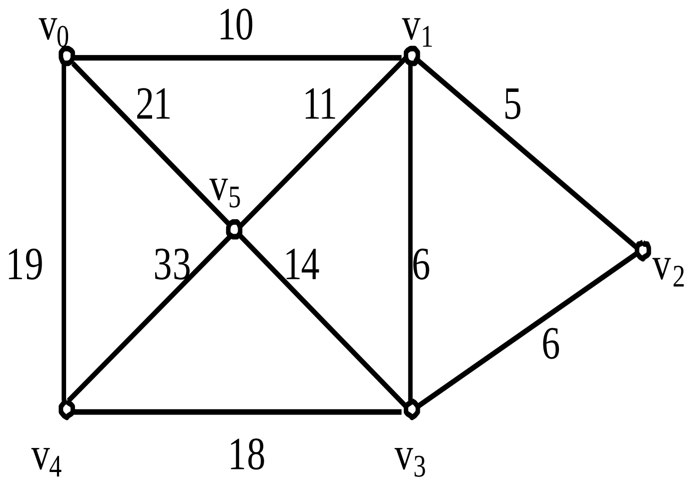
① 设G=(V, E)是网络，最小生成树的初始状态为只有n个顶点而无边的非连通图T=(V，φ)，T中每个顶点自成为一个连通分量

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



$$\begin{bmatrix} 0 & 10 & \infty & \infty & 19 & 21 \\ 10 & 0 & 5 & 6 & \infty & 11 \\ \infty & 5 & 0 & 6 & \infty & \infty \\ \infty & 6 & 6 & 0 & 18 & 14 \\ 19 & \infty & \infty & 18 & 0 & 33 \\ 21 & 11 & \infty & 14 & 33 & 0 \end{bmatrix}$$

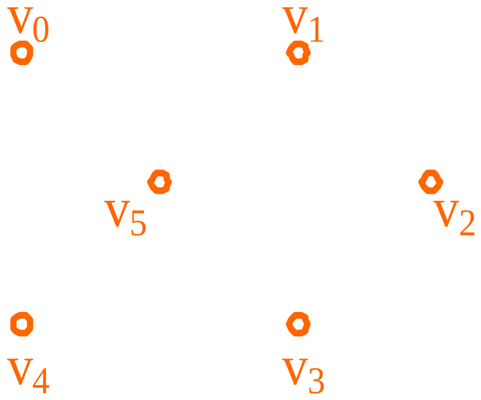
- 集合E中的边按递增顺序排列为：

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

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)

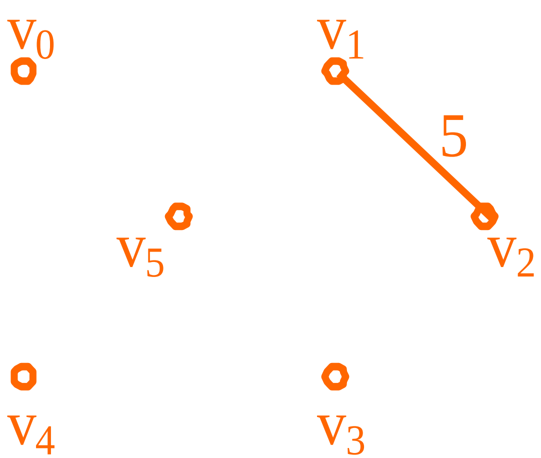
①．初始时，T为只有6个顶点的非连通图

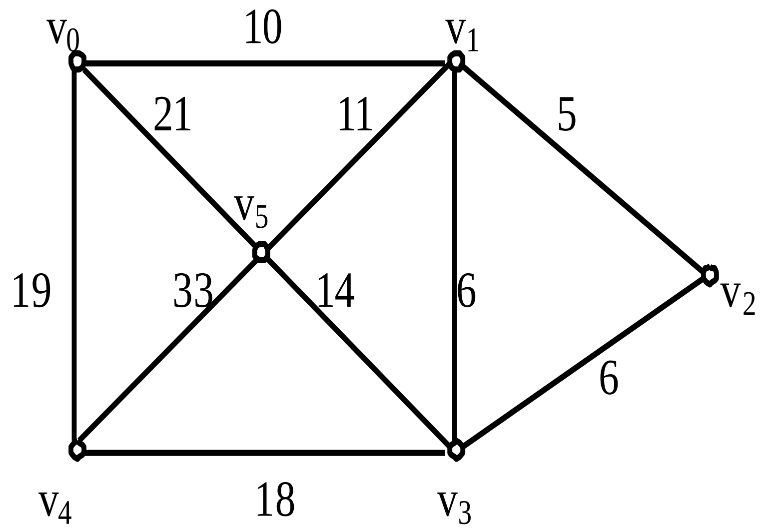


Kruskal算法 - 例子

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

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

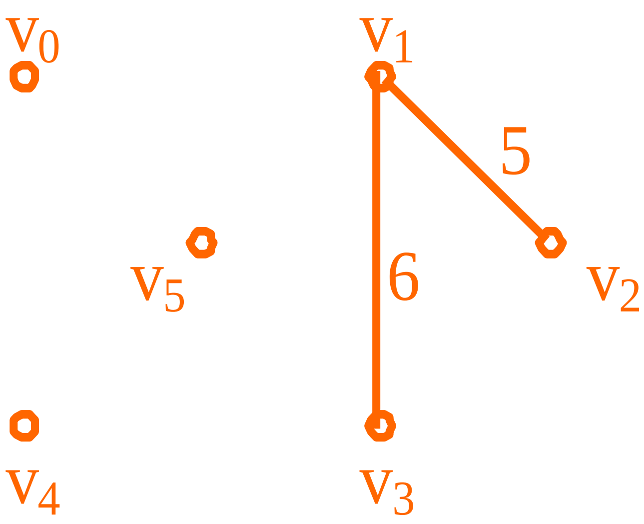


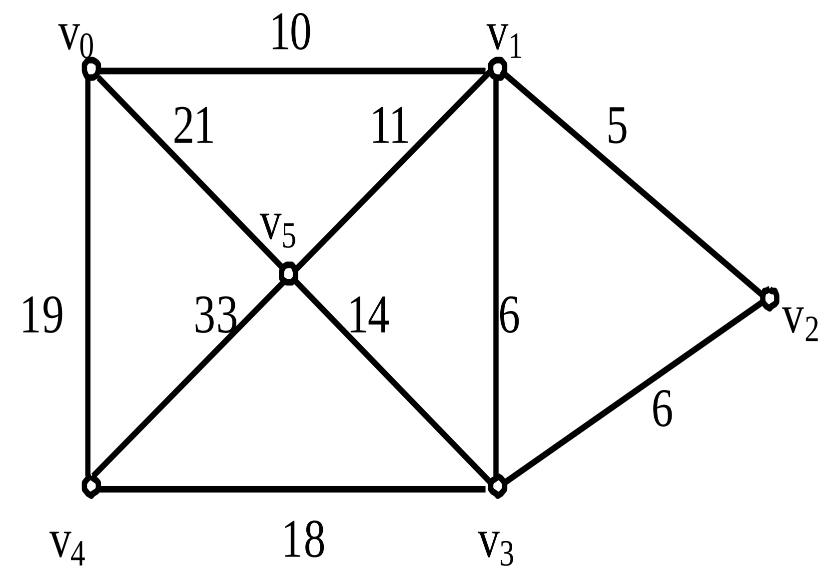


Kruskal算法 - 例子

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

③．同理，将边(v_1, v_3)加入T

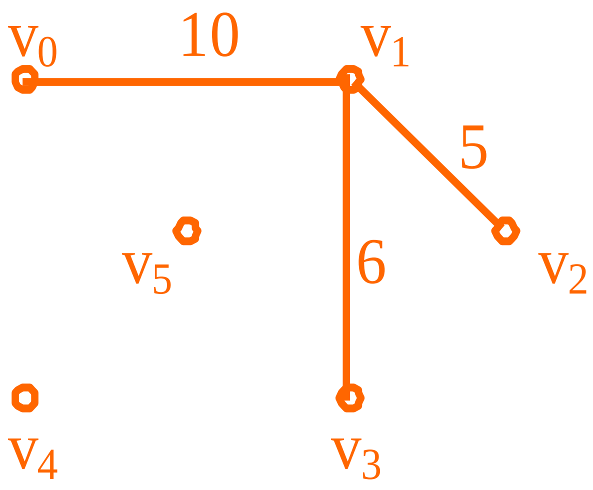


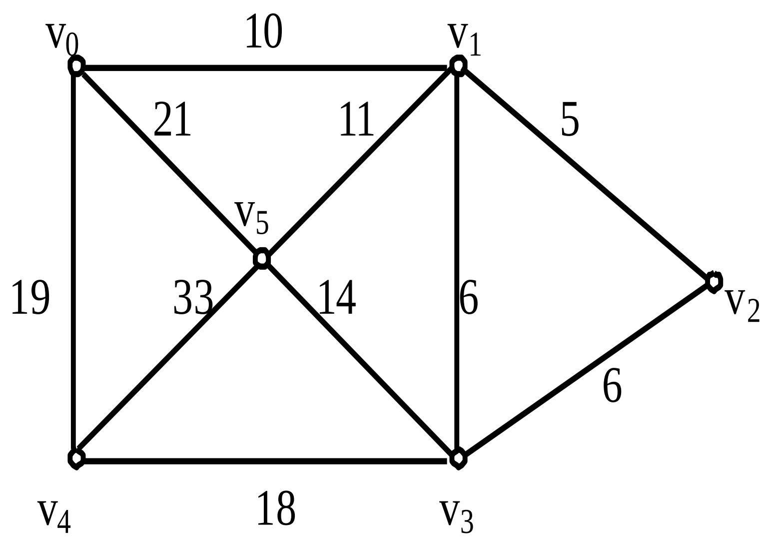


Kruskal算法 - 例子

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

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





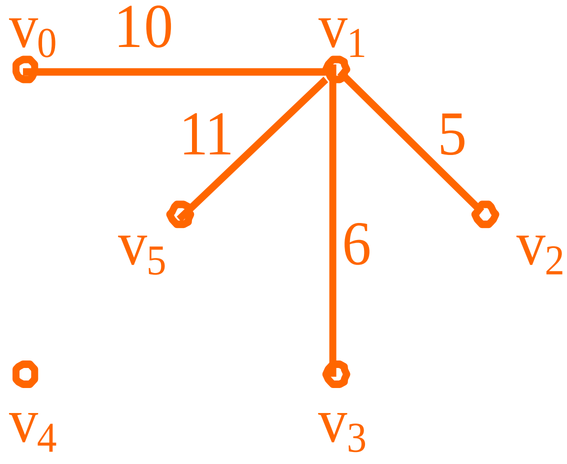
Kruskal算法 - 例子

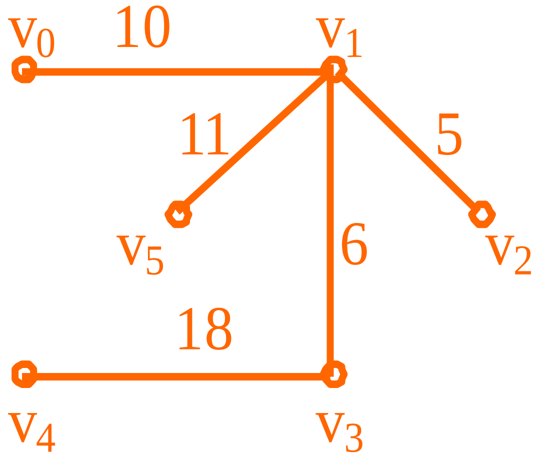
(v1, v2), (v1, v3), (v2, v3), (v0, v1), (v1, v5),

(v3, v5), (v3, v4), (v0, v4), (v0, v5), (v4, v5)

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

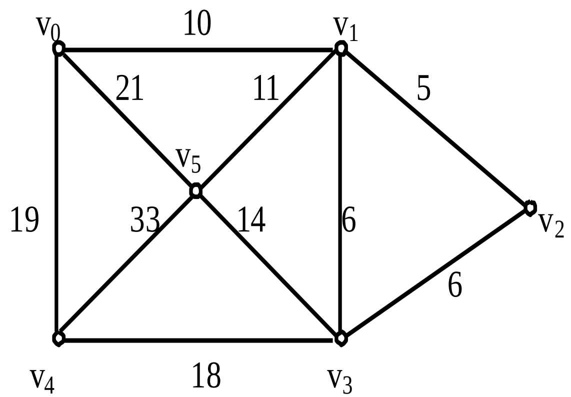
这时T中含的边数为5条，成为一个连通分量，T就是G的一棵最小生成树

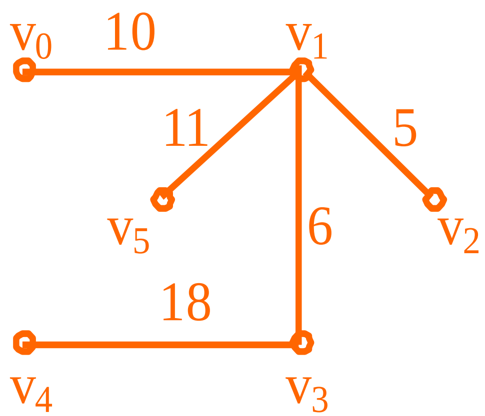


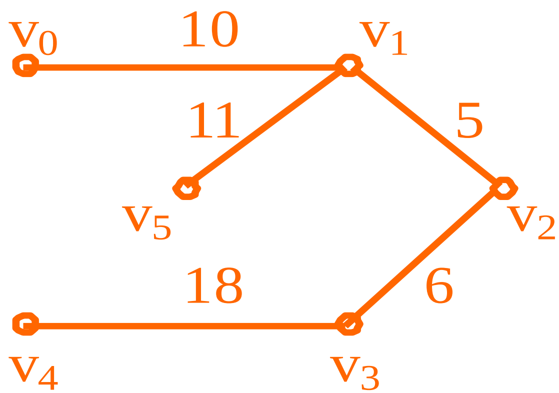


说明

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,φ)
While (T中所含边数<n-1)
{
    从E中选取当前最短边(u,v)；/* 需要先安权值排序 */
    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；
}
```

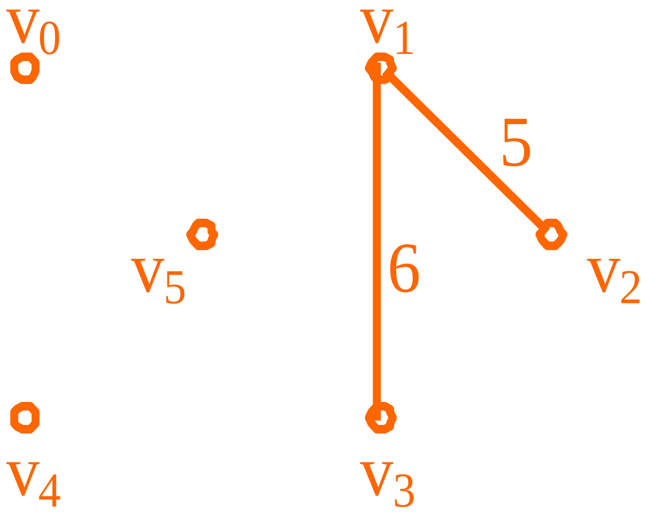
Kruskal 算法的关键

- 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

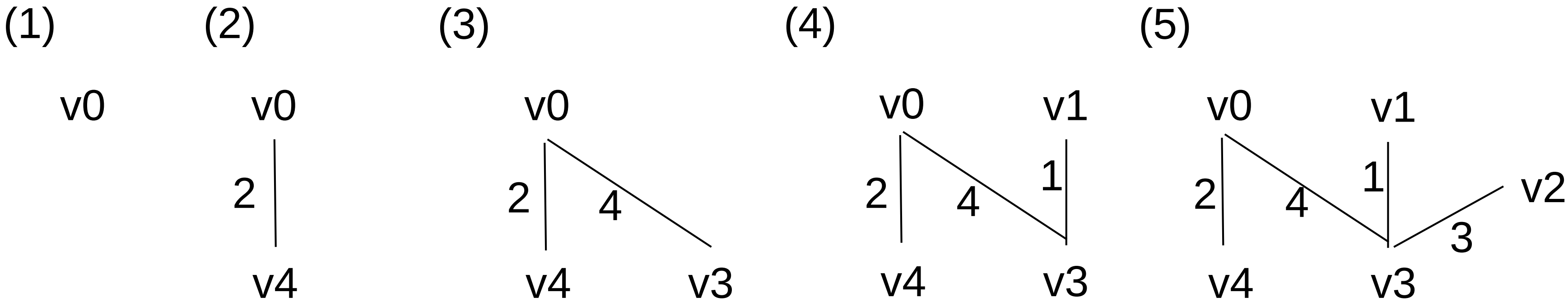
- 确定加入一条边是否会产生回路
  - 可以记录每个连通分量中每个节点的“父节点”
  - 如果加入的边连接的两个顶点“父节点”不同。一定不会产生回路



```
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) {{0,1,7}, {0,2, ∞}, {0,3,4}, {0,4,2}}
(2) {{0,4,2}, {0,2, ∞}, {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}}