本节主题：

最小生成树的克鲁斯卡尔算法

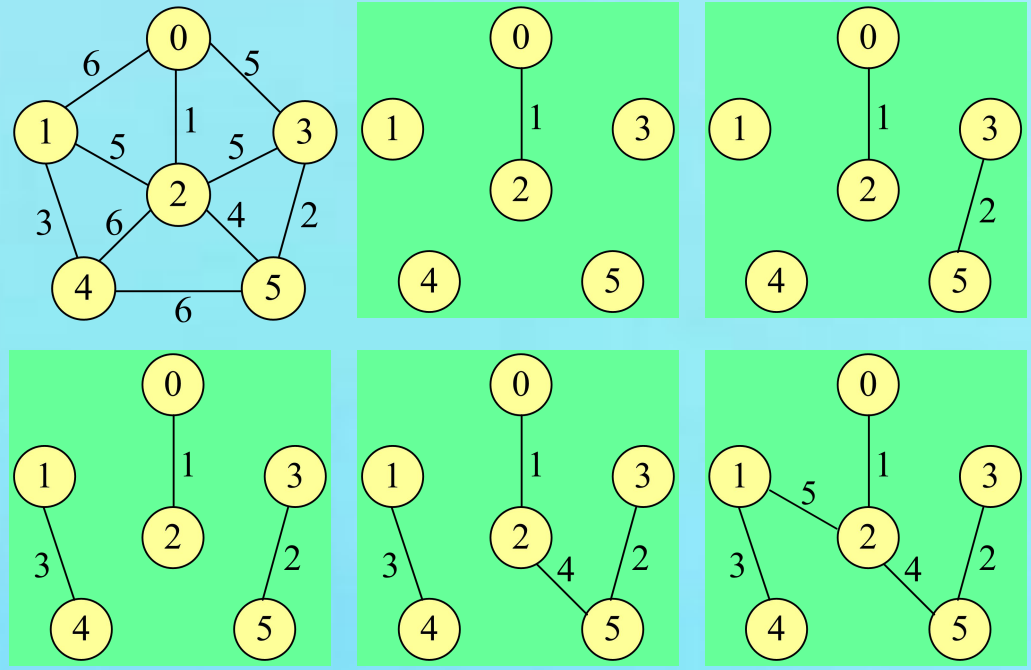
克鲁斯卡尔(Kruskal)算法

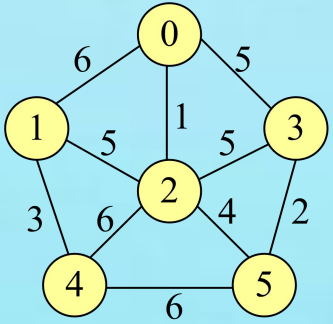
J.B. Kruskal

- 最小生成树算法思想
  - prim算法：逐个加入顶点的方法
  - Kruskal算法：逐个加入边的方法——按权值的递增次序选择合适的边
- 问题
  - G=(V,E)是一个具有n个顶点的带权连通无向图，T=(U,TE)是G的最小生成树
- 策略

（1）置U的初值等于V（即包含有G中的全部顶点），TE的初值为空集（即图T中每一个顶点都构成一个分量）。

（2）将图G中的边按权值从小到大的顺序依次选取：若选取的边未使生成树T形成回路，则加入TE；否则舍弃，直到TE中包含(n-1)条边为止。



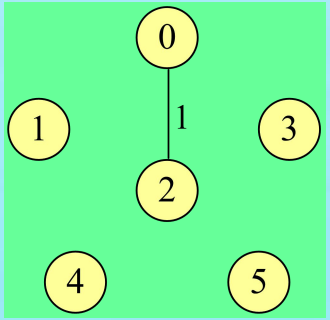


$$\begin{bmatrix} 0 & 6 & 1 & 5 & \infty & \infty \\ 6 & 0 & 5 & \infty & 3 & \infty \\ 1 & 5 & 0 & 5 & 6 & 4 \\ 5 & \infty & 5 & 0 & \infty & 2 \\ \infty & 3 & 6 & \infty & 0 & 6 \\ \infty & \infty & 4 & 2 & 6 & 0 \end{bmatrix}$$

数组E（排序后）

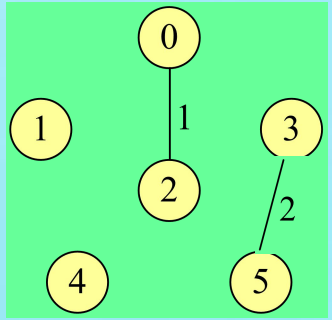
	u	v	w
[0]	0	2	1
[1]	2	0	1
[2]	3	5	2
[3]	5	3	2
[...]	1	4	3
	4	1	3
	2	5	4
	5	2	4
	1	2	5
	2	1	5
	0	3	5
	3	0	5
	2	3	5
	3	2	5
	0	1	6
	1	0	6
	2	4	6
	4	2	6
	4	5	6
	5	4	6

```
//按权值升序排列的边
typedef struct
{
    int u;
    int v;
    int w;
} Edge;
```



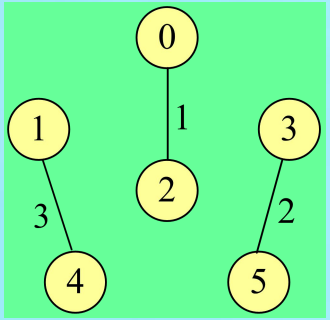
vset

[0]	[1]	[2]	[3]	[4]	[5]
0	1	**0**	3	4	5



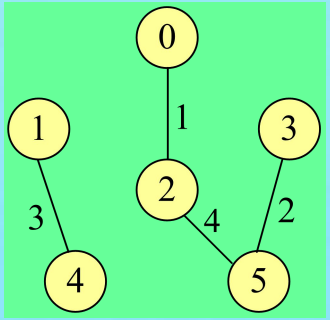
vset

[0]	[1]	[2]	[3]	[4]	[5]
0	1	0	3	4	**3**



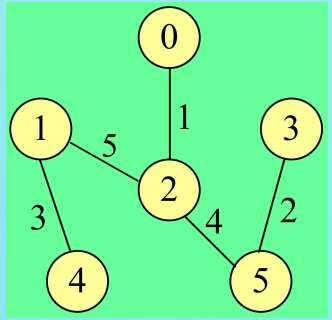
vset

[0]	[1]	[2]	[3]	[4]	[5]
0	1	0	3	**1**	3



vset

[0]	[1]	[2]	[3]	[4]	[5]
0	1	0	**0**	1	**0**



vset

[0]	[1]	[2]	[3]	[4]	[5]
1	1	**1**	**1**	1	**1**

vset

[0]	[1]	[2]	[3]	[4]	[5]
0	1	2	3	4	5

标识子图是否属于同一连通分量

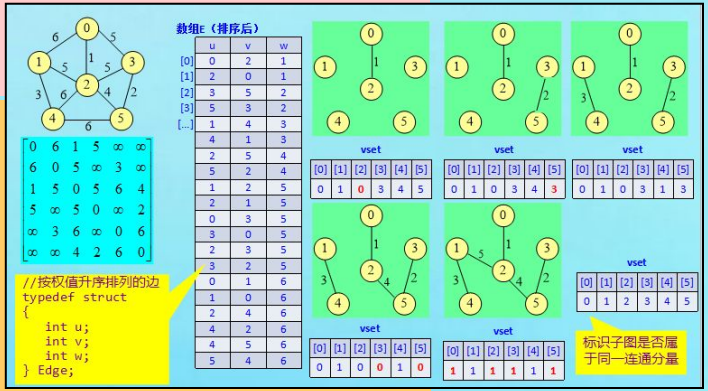
```
void Kruskal(MGraph g)
{
    int i,j,u1,v1,sn1,sn2,k;
    int vset[MAXV];
    Edge E[MaxSize];
    //构造E数组并排序
    //初始化辅助数组vset
    for (i=0; i<g.n; i++)
      vset[i]=i;
```

[0]	[1]	[2]	[3]	[4]	[5]
0	1	2	3	4	5

```
    //选出n-1条边
}
```

```
k=0;
for (i=0; i<g.n; i++)
   for (j=0; j<g.n; j++)
      if (g.edges[i][j]!=0 && g.edges[i][j]!=INF)
      {
         E[k].u=i;
         E[k].v=j;
         E[k].w=g.edges[i][j];
         k++;
      }
InsertSort(E,g.e);
```

```
k=1; j=0;
while (k<g.n)
{
   u1=E[j].u;
   v1=E[j].v;
   sn1=vset[u1];
   sn2=vset[v1];
   if (sn1!=sn2)
   {
      printf("  (%d,%d):%d\n",u1,v1,E[j].w);  k++;
      for (i=0; i<g.n; i++)
         if (vset[i]==sn2)  vset[i]=sn1;
   }
   j++;
}
```



数组E（排序后）

	u	v	w
[0]	0	2	1
[1]	2	0	1
[2]	3	5	2
[3]	5	3	2
[...]	1	4	3
	4	1	3
	2	5	4
	5	2	4
	1	2	5
	2	1	5
	0	3	5
	3	0	5
	2	3	5
	3	2	5
	0	1	6
	1	0	6
	2	4	6
	4	2	6
	4	5	6
	5	4	6

对边的插入排序

```
void InsertSort(Edge E[],int n)
{
   int i,j;
   Edge temp;
   for (i=1; i<n; i++)
   {
      temp=E[i];
      j=i-1;
      while (j>=0 && temp.w<E[j].w)
      {
         E[j+1]=E[j];
         j--;
      }
      E[j+1]=temp;
   }
}
```

两种最小生成树算法的性能

- 空间复杂度
  - 都需要辅助存储空间

int lowcost[MAXV]; //记录从U到U-V的边的最小权值
int closest[MAXV]; //记录最小权值的边对应的顶点

Edge E[maxsize]; //按权值升序排列的边
int vset[MAXV]; //标识子图是否属于同一连通分量

- 时间复杂度

$O(n^2)$
——边多也不怕

$O(e^2)$
——适合边稀疏的图

很多改进，
很多变种。

思考题

- 有n台计算机，已知它们的位置，现连成一个网络，采用什么算法求解所花最少网线。