Kruskal 算法

马欢飞*

2016年11月

寻找带权连通图 $G=\{V,E\}$ 的最小生成树的 Kruskal 算法：

算法 (Kruskal). 初始化： 建立图 T 包含 G 的所有顶点，同时建立集合 S 包含 G 的所有边。重复以下步骤直至 T 已包含 $n-1$ 条边：

- 从边集 S 中挑选权重最小的一条边移出边集 S；
- 若该边加入图 T 不会形成回路，则将该边加入 T，否则舍弃；

以上步骤结束后得到的图 T 即为原图 G 的一棵最小生成树。 ■

下面说明 Kruskal 算法得到的一定是最小生成树。

证明：要证明 T 为最小生成树，只要证明 T 为生成树且权重值和最小。 首先，根据算法可知，T 不含回路，同时，算法的每一步将两个连通分量合并为一个，所以最后得到的一定是连通图。所以 T 是一个连通无回路的图，且包含 G 的所有顶点，所以是一棵生成树。

下面用归纳法说明以下命题成立：对于算法中的每一步得到的 T 所包含的边集，都包含在 G 的某一棵最小生成树中。

初始状态下，T 的边集为空集，显然命题成立。 现假设算法某一步获得的 T 的边集包含在某一棵最小生成树 P 中。若下一步算法挑选的边 e 仍然属于该树 P 则命题对下一步 $T+e$[1] 也成立；如 e 不属于 P，则 $P+e$ 必定包含回路 C 且 C 中必定包含不属于 T 的边 f（不然算法这一步加入 e 会引起回路）。这样把边 f 从 P 中删除同时加入边 e 得到的 $P'=P-f+e$ 仍然是生成树（必定连通，且包含 $n-1$ 条边），同时 f 的权重值不小于 e（不然在这一步算法会选择 f 而不是 e），由 P 的最小性知，新构造的 P' 必然也是最小生成树，且包含 $T+e$。综合上述两种情况知，对下一步得到的 $T+e$ 命题仍然成立。

由此命题知，当算法结束时，得到的 T 依然包含在某一个最小生成树中，显然这棵最小生成树即 T 本身。 ■

代码实现. 现在以邻接矩阵存储方式为例来看 Kruskal 算法的代码实现。其中邻接矩阵edges[n][n]的定义为：

$$\text{edges}[i][j]=\begin{cases}\omega_{ij}, & i\neq j \ \text{ and } \ (i,j)\in E,\\ 0, & i=j,\\ \infty, & \text{otherwise}\end{cases} \tag{1}$$

*hfma@suda.edu.cn

[1]这里 $T+e$ 表示在图 T 中加入边 e 得到的新图，以下 $P-f$ 同。

分析整个算法，困难之处在于每次加入边的时候要保证加入后不形成回路，如果每次都要调用回路检测算法的话开销是非常大的，而且显然没有必要。所以可以考察整个算法，借助于连通分量的思想来编写算法代码。在算法开始之初，所有 n 个顶点两两之间都没有边相连，所以可以认为是 n 个连通分量，之后每次加入一条边的时候，加入的边的两端一定是处在两个不同的连通分量中的（不然一定会形成回路），所以每次加入的边都把两个连通分量合并为一个连通分量。这样，在代码实现的时候，可以把顶点用类别号码来对所在连通分量编号，只要保证加入的边的两端所处的连通分量号码不一致即可。加入边之后，需要把合并的两个连通分量中的顶点类别编号合并以代表连通分量的合并。具体代码如下：

```
typedef struct
{   int u;      //边的起始顶点
    int v;      //边的终止顶点
    int w;      //边的权值
} Edge;         //用于存储所有边的信息以方便排序和使用

void Kruskal(MGraph g)
{  int i,j,u1,v1,sn1,sn2,k;
   int vset[MAXV];
   Edge E[MaxSize]; //存放所有边
   k=0; //E数组的下标从0开始计
for (i=0;i<g.n;i++) //由g产生边集E
    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);//采用直接插入排序对E数组按权值递增排序
for (i=0;i<g.n;i++) //初始化辅助数组用于表示连通分量类别号
   vset[i]=i;        //初始时n个顶点分处于n个连通分量中
k=1; //k表示当前构造生成树的第几条边,初值为1
j=0; //E中边的下标,初值为0
while (k<g.n) //生成的边数小于n时循环
{  u1=E[j].u;v1=E[j].v; //取一条边的头尾顶点
   sn1=vset[u1];
   sn2=vset[v1]; //分别得到两个顶点所属的集合编号
    if (sn1!=sn2)  //两顶点属于不同的集合
    {printf("  (%d,%d):%d\n",u1,v1,E[j].w);
     k++;    //生成边数增1
     for (i=0;i<g.n;i++)  //两个集合统一编号
     if (vset[i]==sn2) //集合编号为sn2的改为sn1
     vset[i]=sn1;
    }
```

```
    j++;     //扫描下一条边
     }
}
```

对于一个有 n 个顶点，e 条边的带权连通无向图 G 而言，把邻接矩阵转化为边集 E 的复杂度为 $O(n^2)$, 而对边集 E 采用直接插入排序的时间复杂度为 $O(e^2)$。 while 循环最坏情况下要执行 e 次, 其中合并编号需要循环 n 次，所以 while 循环的时间复杂度为 $O(en)$。 考虑到连通无向图具有性质 $e \geqslant (n-1)$, 以上代码的复杂度为 $O(e^2)$。