MST 性质与 Prim 算法

马欢飞*

2016年11月

最小生成树 MST (minimum spanning tree) 是指连通带权图中各边权重之和最小的生成树。

性质 (MST性质). 设 $G = (V, E)$ 是一个连通网络，U 是顶点集 V 的一个真子集。则 U 和 $V - U$ 把图 G 的所有顶点分为互不相交的两部分。 考虑所有一端在 U 中，另一端在 $V - U$ 中的边，设 (u, v) 是其中权重最小的一条边，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，若 G 的任何一棵最小生成树中都不含边 (u, v)，任取 G 的一棵最小生成树 T，则 T 不包含 (u, v)。 由于 T 是生成树，由连通性知，必有一条连接 u 和 v 的路径，由于 u 在 U 中而 v 在 $V - U$ 中，所以该路径上必有一条边跨集合 U 和 $U - V$，记这条边为 (u', v')，其中 u' 在 U 中， v' 在 V 中，则由 (u, v) 的定义知，权重 $\omega(u', v') \geqslant \omega(u, v)$。 现在在 T 中删去边 (u', v') 同时加入边 (u, v)。由树的连通性可知，新得到的图依然是一棵生成树 T' （为什么？[1]），且 T' 的权重之和不大于 T 的权重之和，所以 T' 也是最小生成树且 T' 包含 (u, v)，矛盾。

由此性质，可以得到寻找最小生成树的 Prim 算法：

算法 (Prim). 初始化：建立顶点集合 $U = \{v\}$，其中 v 是图 $G = (V, E)$ 中任意一个顶点。构造 $TE = \{\}$ 为一个空的边集。 重复以下步骤直至 $U = V$:

- 边集 E 中所有跨 U 和 $V - U$ 的边组成的集合中挑选权值最小的边 (u, v), 其中 $u \in U, v \in V - U$ （若满足条件的边有多条，则可以任意选取其中之一）；
- 将顶点 v 加入集合 U， 将边 (u, v) 加入集合 TE，得到新的顶点集 U 和边集 TE

以上步骤结束后得到的图 $T = (U, TE)$ 即为原图 G 的一棵最小生成树。 ■

下面说明 Prim 算法得到的一定是最小生成树。若不然，设另有最小生成树 T'，依照 Prim 算法加入边的次序，必存在某一步划分 U 和 $V - U$，使得 T 中跨该两集合的边为 (u, v)，而由 T' 的连通性知，T' 中有连接 u 和 v 的路径，该路径跨 U 和 $U - V$ 的边记为 (u', v'), 且 $(u', v') \neq (u, v)$。 类似于MST 性质的证明可知这不可能。

*hfma@suda.edu.cn

[1] T 中任意两点之间的连通路径若不包含 (u', v')，则经过修改后在 T' 中连通路径不变；否则在原连通路径上用路径 $(u', \ldots, u, v, \ldots, v')$ 来代替原来的边 (u', v') 依然得到连通路径。所以新得到的 T' 依然是连通的。同时，n 个顶点 $n - 1$ 条边形成连通图，必然没有回路，所以 T' 依然是树。

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

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

分析整个算法，困难之处在于每次要在所有跨 U 和 $V-U$ 的边集中寻找权值最小的边，如果要搜索整个图的所有的边来实现这一点，则每一步的复杂度都要达到 $O(n^2)$，整个算法的复杂度会达到 $O(n^3)$, 显然这是复杂度很高的，而且重复搜索很多。所以需要借助一种新的办法来降低复杂度。为此，可以引入两个辅助数组 closest 和 lowcost，其中 lowcost[j] 表示顶点 j 到顶点集 U 中的最短边的权重，closest[j] 表示该最短边的另一端顶点编号。 具体来说，若 lowcost[j] $= 0$, 则表示 j 在 U 中； 若 $0 <$ lowcost[j] $< \infty$，则表示 j 在 $V-U$ 中且顶点 j 和 U 中的顶点 closest[j] 构成的边(closest[j], j) 为 j 到 U 中权重最小的边，其权重为 lowcost[j]；若 lowcost[j] $= \infty$ 则表示顶点 j 在 $V-U$ 中且与 U 中无边相连。这样，算法的每一步只需要扫描 lowcost 数组找到跨 U 和 $V-U$ 的最短边，并通过修改 lowcost 和 closest 的值的方式来实现点加入 U 和修改跨 U 和 $V-U$ 的边值的过程。具体代码如下：

```
#define INF 32767                          //INF表示无穷大
void Prim(MGraph g,int v)
{  int lowcost[MAXV];
   int min;
   int closest[MAXV],i,j,k;

   for (i=0;i<g.n;i++)                     //给lowcost[]和closest[]置初值
   { lowcost[i]=g.edges[v][i];            //初始情况下U只有一个点v，
                                           //所以lowcost初值即为所有v和i之间的边值
    closest[i]=v;                         //i的closest点初始情况下为v
   }

   for (i=1;i<g.n;i++)                     //(n-1)个顶点依次加入集合U
   { min=INF;
      for (j=0;j<g.n;j++)                  //在(V-U)中找出离U最近的顶点k
       if (lowcost[j]!=0 && lowcost[j]<min)
       {min=lowcost[j];
        k=j;                              //k记录最近顶点的编号
       }
   printf(" 边(%d,%d)权为:%d\n",closest[k],k,min);
   lowcost[k]=0;                          //标记k已经加入U

   for (j=0;j<g.n;j++)                     //修改数组lowcost和closest
   if (g.edges[k][j]!=0 && g.edges[k][j]<lowcost[j])
   { lowcost[j]=g.edges[k][j];
```

```
    closest[j]=k;
   }
   }
}
```

显然以上代码的复杂度为 $O(n^2)$.