Prim-Jarník-Dijkstra贪心算法
–方法、算法、代码和正确性证明

乔海燕
中山大学数据科学与计算机学院

2018 年 1 月 11 日

摘 要

本文介绍Prim算法，包括方法、算法、正确性证明和算法的C++实现。

§1 基本思想

最小生成树来源于求构建一个连通网络的最小费用设计。解决问题的基本思想基于下列定理：

定理1. *设G是一个带权连通图，其结点集划分为两个不相交的结点子集V_1和V_2。如果e两个端点分别在V_1和V_2中具有最小权的边，则e必定包含在G的某个最小生成树中。*

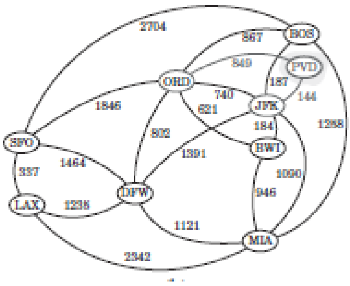
定理的证明见[1]。

§2 求最小生成树的贪心方法

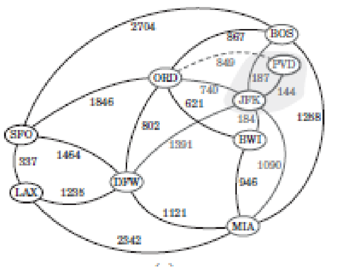
对于给定的带权图$G = (V, E, w)$，为了使得生成树$T = (V, E_T)$的权$w(T) = \sum_{t_i \in E_T} w(t_i)$达到最小，基于以上基本思想，我们使用下面的贪心方法。

求最小生成树的贪心法:

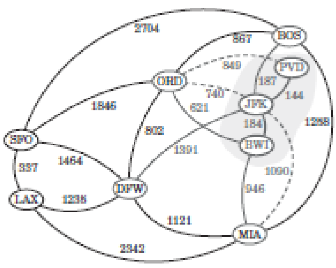
1. 选择一个初始结点，如v_1，令$V_1 = \{v_1\}$，$V_2 = V - V_1$, $E_T = \{\}$；
2. 为了使得生成树的权极小化，选择连接V_1和V_2的最小权边，如$t_1 = (v_1, v_2)$，不妨称v_2为V_1的最近邻居，并将v_2连到v_1，即$V_1 = V_1 \cup \{v_2\}$, $V_2 = V_2 - \{v_2\}$, $E_T = E \cup \{t_1\}$;
3. 继续选择$V_1 = \{v_1, v_2\}$的最近邻居，如v_3 ,并将其连接到最近邻居，更新V_1、V_2和E_T；
4. 重复以上步骤，直至$V_1 = V$，即所有结点连到一起，便得最小生成树。



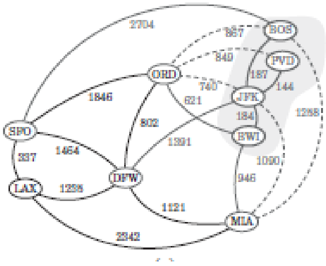
(a) 带权连通图。从结点PVD开始求最小生成树



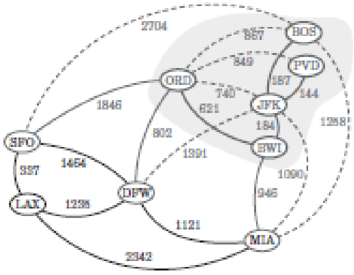
(b) 连接PVD的最近邻居JFK



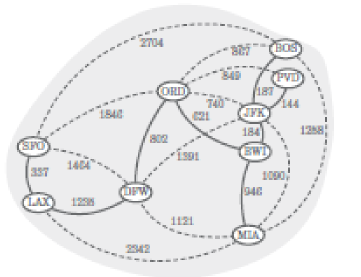
(c) 连接PVD和JFK的最近邻居BWI



(d) 连接{PVD,JFK, BWI}的最近邻居BOS



(e) 连接{PVD,JFK, BWI, BOS}的最近邻居ORD



(f) 最终的生成树

图 1: 求最小生成树的Prim算法，阴影部分表示当前求得的部分生成树，粗线边是树边。

注意，以上方法中，每次完成第3步，结果得到的子图$T=(V_1,E_T)$都是一颗树。当$V_1=V$时，$T=(V,E_T)$是G的生成树，而且是最小生成树。

例如，对图1a[1]，从结点PVD开始，首先将其最近邻居JFK通过最小权边连接，得到图1b阴影部分的树，继续将阴影部分树与其最近邻居BWI 通过最小权边连接，得到图1c所示阴影部分树，重复这个过程，直至得到图1f的生成树。

这种贪心方法就是著名的Prim算法，一种求最小生成树的贪心算法。该算法于1930年由捷克数学家Jarník发明，后来Prim和Dijkstra分别于1957 年和1959 年发现了该算法[4]。

Prim算法（方法）：

设$G=(V,E)$是一个带权连通图$V=\{v_1,v_2,\cdots,v_n\}$。用$T=(V_T,E_T)$表示Prim算法构造的最小生成树。

1. 选择一个结点作为初始结点，如v_1, 令$V_T=\{v_1\}$, $E_T=\{\}$;
2. 选择V_T 的最近邻居，如v_i 是V_T的最近邻居，而且v_i与V_T中结点v_j最近，将结点v_i 添加到V_T, 将边(v_i,v_j)添加到E_T；
3. 重复2直至所有结点都加入V_T .

以上方法的重点是第2步，但是，该步骤操作性不强。下面我们使用标记法细化该步骤，得到更具操作性的Prim算法。

§3 使用标记的Prim算法

我们对图上的每个结点u标记一对信息(N_u,D_u), 表示结点u在T中的最近邻居是N_u, 距离是D_u。

例如，对于图1a的初始树T（阴影部分），结点JFK 和ORD的最近邻居都是PVD, 结点JFK的标记为(PVD, 144)，结点ORD的标记为(PVD, 849)，而其他结点的标记为(-，∞)，表示这些结点与当前T 中结点不连通。因此，此时T的最近邻居就是JFK。再比如，在1b状态，结点JFK的邻接点ORD 的标记变为(JFK, 740), BWI的标记变为(JFK, 184)。如果查看T之外结点标记第二个分量最小者，则该结点便是T的最近邻居。也就是说，前一节方法中第二步，求T的最近邻居可以通过查看T之外所有结点的标记，其中第二个分量最小者即为T的最近邻居。由此得到下面的Prim 标记算法1。

§4 Prim算法的C++代码

定义一些类型：

```
//我们使用 0, 1, 2, ..., n-1表示n个结点
typedef float Weight;
typedef int Vertex;
typedef pair<Vertex, Vertex> Edge;
typedef vector<Edge> Edges; //边集用结点对向量表示
typedef map<Vertex, bool> Vertices; //结点子集用特征函数表
typedef vector<vector<Weight> > GraphMatrix;
```

算法 1 Prim-Jarnik-Dijkstra算法

输入: M是一个带权连通图$G=(V,E)$的矩阵表示，$s\in V$是一个起始结点
输出: $T=(V_T,E_T)$是G的最小生成树

  $V_T \leftarrow \{s\}$
  $E_T \leftarrow \{\}$
  //初始化各结点标记
  $(N_s, D_s) \leftarrow (s, 0)$
  for $v \in V$ **do**
    $(N_v, D_v) \leftarrow (s, M[s][v])$
  end for
  for $i = 1$ to $n-1$ **do**
    //求V_T的最近邻居u
    令u是$V - V_T$中结点标记第二个分量值最小的结点
    $V_T \leftarrow V_T \bigcup \{u\}$
    $E_T \leftarrow E_T \bigcup \{(u, N_u)\}$
    for u的每个邻接点$v \notin V_T$ **do**
      if $M[u][v] < D_v$ **then**
        $(N_v, D_v) \leftarrow (u, M[u][v])$
      end if
    end for
  end for

```
typedef vector<pair<Vertex, Weight> > Labels; //结点标记
float inf = numeric_limits<float> :: max();//无穷, 表示无
```

Prim算法实现：

```
//输入: 一个图g, 一个初始结点s.
//输出: 图g的最小生成树T的边集和权.
pair<Edges, Weight> prim_jarnik(GraphMatrix g, Vertex s){
    int n = g.size();
    Vertices V_T; //  T的结点集
    //初始? V_T = {s}
    for (size_t i = 0; i<n; i++)
        V_T[i] = false;
    V_T[s] = true;
    Edges E_T;   //  E_T ={}
    Weight w = 0.0;// weigth of T
    Labels label(n);
    //初始化各结点标记
    for (size_t i=0;i<n;i++){
       label[i] = make_pair(s, g[s][i]);
    }
    for (int i = 1; i<n; i++){
        int u = nearest_neighbour(V_T, label);
```

```
        //V_T <- V_T + {u}, E_T <- E_T +{(u, n_u)}
        V_T[u] = true;
        E_T.push_back(make_pair(u, label[u].first));
        w += g[u][label[u].first];
        for (size_t i=0;i<n;i++){
             if (!V_T[i] && (g[u][i] < inf) && (g[u][i] < label[i].second))
                 label[i] = make_pair(u, g[u][i]);
        }
    }
    return make_pair(E_T, w);
}
```

通过遍历标记求T的最近邻居函数：

```
//求V的最近邻居，即标记中第二个分量最小的结点
Vertex nearest_neighbour(Vertices V, Labels label){
    Vertex u;
    Weight w = inf;
    int n = V.size();
    for (size_t i=0;i<n;i++){
      if (!V[i] && label[i].second < w){
           w = label[i].second;
           u = i;
      }
    }
   return u;
}
```

§5 Prim算法正确性证明

定理2. *设$G=(V,E)$是连通带权图，$T=V_T,E_T)$是Prim算法1的输出。则*

1. *T是G的生成树；*
2. *T是G的最小生成树。*

证明： 假设带权连通图$G=(V,E)$ 有n个结点，T 是Prim算法输出的图，而且$t_1,t_2,\cdots,t_{n-1}$是Prim算法中顺序添加到T中的边，并用T_i 表示Prim算法中添加了t_i之后的图，即$T_0=\{\}$, $T_1=\{t_1\}$, $T_2=\{t_1,t_2\}$, $\cdots$, $T=T_{n-1}=\{t_1,t_2,\cdots,t_{n-1}\}$。

首先证明T是G的生成树。不妨用归纳法。显然，$T_1=t_1$是树。假设T_{i-1}是树，则因为Prim算法将树T_i的一个结点与T_i之外的一个结点连接，所以结果T_{i+1}仍然是连通且无回路的图，因此也是树。所以，$T=T_{n-1}$ 是包含所有结点的树，即T是G的生成树。

为了证明T是G的最小生成树，我们证明T 一定包含在G 的某个最小生成树T'中，即T'包含T的所有边。仍然用归纳法。显然，$T_0=\{\}$包含在所有最

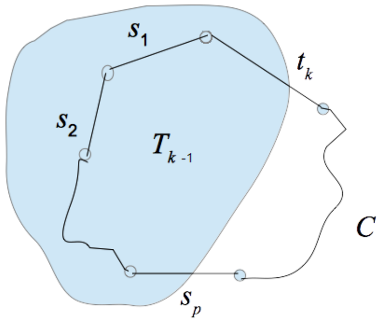


图 2: 在T'中添加一条边t_k出现回路C。

小生成树中。假设T_{k-1}包含在某个最小生成树T'中。现在证T_k也包含在某个最小生成树中。因为$T_k = T_{k-1} \bigcup \{t_k\}$。

分两种情况：如果t_k也包含在T'中，则T_k包含在T'中。

另外一种情况是T'不包含t_k。此时将t_k加入T'中必然出现回路C, 如图2 所示（在树上任意添加一条边将出现回路）。现在沿着回路的边s_1, s_2, $\cdots$ 找一条一端在T_k上（阴影部分）, 但是另一端不在T_k上边s_p。这样的边一定存在，而且显然不能是$t_1, t_2, \cdots, t_{k-1}$ 中的任何边。注意，t_k 和s_p 都是一端在T_k上, 但是另一端不在T_k上边，根据Prim算法，两个边的权满足$w(t_k) \leq w(s_p)$ 。现在考虑树$T'' = T' + t_k - s_p$, T'' 也是G 的生成树，而且$w(T'') = w(T') + w(t_k) - w(s_p) \leq w(T')$，这表明$T''$也是$G$的最小生成树。因为去掉的边$s_p$不是$T_k$的边，所以$T''$是包含$t_1, t_2, \cdots, t_k$ 的最小生成树。

由此证明，$T = T_{n-1}$也必然包含在一棵最小生成树中。 $\square$

鸣谢： 感谢16级软件工程专业同学们给予老师的激励，并指出文中的错漏问题!

参考文献

[1] Michael T. Goodrich, Roberto Tamassia. *Algorithm Design and Applications*, Wiley, 2014.

[2] Bernard Kolman, Robert C. Busby and Sharon Culter Ross. *Discrete Mathematical Structures* (sixth edition), 高等教育出版社，2010年。

[3] Ron Graham and P. Hell. *On the History of the Minimum Spanning Tree Problem*, Annals Hist. of Comp. 7 (1985), 43-57.

[4] Prim's Algorithm: `https://en.wikipedia.org/wiki/Prims_algorithm`.