Prim算法描述

1).输入：一个加权连通图，其中顶点集合为V，边集合为E；

2).初始化：Vnew = {x}，其中x为集合V中的任一节点（起始点），Enew = {},为空；

3).重复下列操作，直到Vnew = V：

a.在集合E中选取权值最小的边<u, v>，其中u为集合Vnew中的元素，而v不在Vnew[集合](http://baike.baidu.com/subview/15216/10703234.htm" \t "_blank)当中，并且v∈V（如果存在有多条满足前述条件即具有相同权值的边，则可任意选取其中之一）；

b.将v加入集合Vnew中，将<u, v>边加入集合Enew中；

4).输出：使用集合Vnew和Enew来描述所得到的[最小生成树](http://baike.baidu.com/view/288214.htm" \t "_blank)。

时间复杂度

|  |  |
| --- | --- |
| **最小边、权的数据结构** | **时间复杂度（总计）** |
| [邻接矩阵](http://baike.baidu.com/view/549589.htm)、搜索 | O(V^2) |
| [二叉堆](http://baike.baidu.com/view/668854.htm)（后文伪代码中使用的数据结构）、[邻接表](http://baike.baidu.com/view/549594.htm" \t "_blank) | O((V + E) log(V)) = O(E log(V)) |
| [斐波那契堆](http://baike.baidu.com/view/2185995.htm)、[邻接表](http://baike.baidu.com/view/549594.htm) | O(E + V log(V)) |

通过邻接矩阵图表示的简易实现中，找到所有最小权边共需O（V）的运行时间。使用简单的二叉堆与邻接表来表示的话，普里姆算法的运行时间则可缩减为O(ElogV)，其中E为连通图的边数，V为顶点数。如果使用较为复杂的斐波那契堆，则可将运行时间进一步缩短为O(E+VlogV)，这在连通图足够密集时（当E满足Ω（VlogV）条件时），可较显著地提高运行速度。

图例描述

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 图例 | 说明 | 不可选 | 可选 | 已选（Vnew） |
| [https://imgsa.baidu.com/baike/s%3D220/sign=73fee7996e224f4a5399741139f69044/9213b07eca806538cdd235ac94dda144ad348251.jpg](http://baike.baidu.com/pic/Prim/10242166/0/9213b07eca806538cdd235ac94dda144ad348251?fr=lemma&ct=single) | 此为原始的加权连通图。每条边一侧的数字代表其权值。 | - | - | - |
| [https://imgsa.baidu.com/baike/s%3D220/sign=d2d0565c542c11dfdad1b82153266255/359b033b5bb5c9eaa5c7925cd639b6003af3b35d.jpg](http://baike.baidu.com/pic/Prim/10242166/0/359b033b5bb5c9eaa5c7925cd639b6003af3b35d?fr=lemma&ct=single) | 顶点**D**被任意选为起始点。顶点**A**、**B**、**E**和**F**通过单条边与**D**相连。**A**是距离**D**最近的顶点，因此将**A**及对应边**AD**以高亮表示。 | C, G | A, B, E, F | D |
| [https://imgsa.baidu.com/baike/s%3D220/sign=16ed179e9013b07eb9bd570a3cd69113/b58f8c5494eef01fd736887ae3fe9925bc317d6b.jpg](http://baike.baidu.com/pic/Prim/10242166/0/b58f8c5494eef01fd736887ae3fe9925bc317d6b?fr=lemma&ct=single) | 下一个顶点为距离**D**或**A**最近的顶点。**B**距**D**为9，距**A**为7，**E**为15，**F**为6。因此，**F**距**D**或**A**最近，因此将顶点**F**与相应边**DF**以高亮表示。 | C, G | B, E, F | A, D |
| [https://imgsa.baidu.com/baike/s%3D220/sign=a85d29db97eef01f49141fc7d0ff99e0/9e3df8dcd100baa1d1c3d7044410b912c9fc2ea5.jpg](http://baike.baidu.com/pic/Prim/10242166/0/9e3df8dcd100baa1d1c3d7044410b912c9fc2ea5?fr=lemma&ct=single) | 算法继续重复上面的步骤。距离**A**为7的顶点**B**被高亮表示。 | C | B, E, G | A, D, F |
| [https://imgsa.baidu.com/baike/s%3D220/sign=040f9c8e223fb80e08d166d506d02ffb/6d81800a19d8bc3eb4d39675818ba61ea9d345b1.jpg](http://baike.baidu.com/pic/Prim/10242166/0/6d81800a19d8bc3eb4d39675818ba61ea9d345b1?fr=lemma&ct=single) | 在当前情况下，可以在**C**、**E**与**G**间进行选择。**C**距**B**为8，**E**距**B**为7，**G**距**F**为11。点**E**最近，因此将顶点**E**与相应边**BE**高亮表示。 | 无 | C, E, G | A, D, F, B |
| [https://imgsa.baidu.com/baike/s%3D220/sign=18e682f09d2f07085b052d02d925b865/f31fbe096b63f624f450a9bc8444ebf81a4ca304.jpg](http://baike.baidu.com/pic/Prim/10242166/0/f31fbe096b63f624f450a9bc8444ebf81a4ca304?fr=lemma&ct=single) | 这里，可供选择的顶点只有**C**和**G**。**C**距**E**为5，**G**距**E**为9，故选取**C**，并与边**EC**一同高亮表示。 | 无 | C, G | A, D, F, B, E |
| [https://imgsa.baidu.com/baike/s%3D220/sign=872e7f76d42a60595610e6181835342d/3801213fb80e7bec82c6905e2c2eb9389b506b75.jpg](http://baike.baidu.com/pic/Prim/10242166/0/3801213fb80e7bec82c6905e2c2eb9389b506b75?fr=lemma&ct=single) | 顶点**G**是唯一剩下的顶点，它距**F**为11，距**E**为9，**E**最近，故高亮表示**G**及相应边**EG**。 | 无 | G | A, D, F, B, E, C |
| [https://imgsa.baidu.com/baike/s%3D220/sign=e4a20adef0deb48fff69a6dcc01e3aef/c83d70cf3bc79f3de184748fb9a1cd11728b2923.jpg](http://baike.baidu.com/pic/Prim/10242166/0/c83d70cf3bc79f3de184748fb9a1cd11728b2923?fr=lemma&ct=single) | 现在，所有顶点均已被选取，图中绿色部分即为连通图的最小生成树。在此例中，最小生成树的权值之和为39。 | 无 | 无 | A, D, F, B, E, C, G |

Kruskal算法定义

[编辑](javascript:;)

克鲁斯卡尔算法

假设 WN=(V,{E}) 是一个含有 n 个顶点的连通网，则按照克鲁斯卡尔算法构造[最小生成树](http://baike.baidu.com/view/288214.htm" \t "_blank)的过程为：先构造一个只含 n 个顶点，而边集为空的子图，若将该子图中各个顶点看成是各棵树上的根结点，则它是一个含有 n 棵树的一个森林。之后，从网的边集 E 中选取一条权值最小的边，若该条边的两个顶点分属不同的树，则将其加入子图，也就是说，将这两个顶点分别所在的两棵树合成一棵树；反之，若该条边的两个顶点已落在同一棵树上，则不可取，而应该取下一条权值最小的边再试之。依次类推，直至森林中只有一棵树，也即子图中含有 n-1条边为止。

举例描述

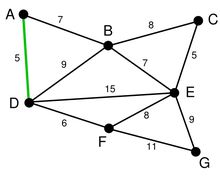
[编辑](javascript:;)

克鲁斯卡尔算法(Kruskal's algorithm)是两个经典的最小生成树算法的较为简单理解的一个。这里面充分体现了贪心算法的精髓。大致的流程可以用一个图来表示。这里的图的选择借用了Wikipedia上的那个。非常清晰且直观。

首先第一步，我们有一张图，有若干点和边

第一步我们要做的事情就是将所有的边的长度排序，用排序的结果作为我们选择边的依据。这里再次体现了贪心算法的思想。资源排序，对局部最优的资源进行选择。

排序完成后，我们率先选择了边AD。这样我们的图就变成了

[](http://baike.baidu.com/pic/kruskal%E7%AE%97%E6%B3%95/4483275/0/48151723f120371293580775?fr=lemma&ct=single)

.

.

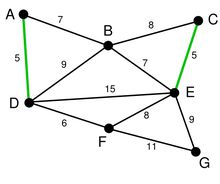
.

.

.

.

第二步，在剩下的边中寻找。我们找到了CE。这里边的权重也是5

[](http://baike.baidu.com/pic/kruskal%E7%AE%97%E6%B3%95/4483275/0/d478a800dc8a5bdbe850cd73?fr=lemma&ct=single)

.

.

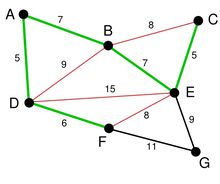
.

.

.

.

依次类推我们找到了6,7,7。完成之后，图变成了这个样子。

[](http://baike.baidu.com/pic/kruskal%E7%AE%97%E6%B3%95/4483275/0/4d497006b7ea6b2703088179?fr=lemma&ct=single)

.

.

.

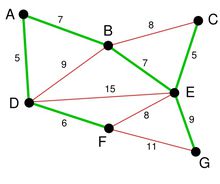
.

.

.

下一步就是关键了。下面选择那条边呢？ BC或者EF吗？都不是，尽管现在长度为8的边是最小的未选择的边。但是他们已经连通了（对于BC可以通过CE,EB来连接，类似的EF可以通过EB,BA,AD,DF来接连）。所以我们不需要选择他们。类似的BD也已经连通了（这里上图的连通线用红色表示了）。

最后就剩下EG和FG了。当然我们选择了EG。最后成功的图就是下图：

[](http://baike.baidu.com/pic/kruskal%E7%AE%97%E6%B3%95/4483275/0/034965f4a44b0385f3d3855c?fr=lemma&ct=single)

.

.

.

.

.

.

到这里所有的边点都已经连通了，一个最小生成树构建完成。

Kruskal算法的时间复杂度由排序算法决定，若采用快排则时间复杂度为O(N log N)。