朱刘算法

Author:Cyf

主要作用：求解最小有向生成树。

最小有向生成树：给定一个有向带权图G和其中一个点u，找出一个以u为根结点，权和最小的有向生成树。有向生成树也叫树型图，是指一个类似树的有向图，满足以下条件：   
 1.恰好有一个入度为0的点，称为根结点   
 2.其他结点的入度均为1   
 3.可以从根结点到达其他结点

[**算法**](http://lib.csdn.net/base/datastructure)的主过程如下:   
 1.找到除了root以为其他点的权值最小的入边。用In[i]记录   
 2.如果出现除了root以为存在其他孤立的点，则不存在最小树形图。   
 3.找到图中所有的环，并对环进行缩点，重新编号。   
 4.更新其他点到环上的点的距离   
 5.重复3，4直到没有环为止。

6.判断有无收缩点，展开收缩点。

详细过程：

算法一开始先判断从固定根开始是否可达所有原图中的点，若不可，则一定不存在最小树形图。这一步是一个很随便的搜索，写多搓都行，不加废话。第二步，遍历所有的边，从中找出除根结点外各点的最小入边，累加权值，构成新图。接着判断该图是否存在环。若不存在，则该图便是所求最小树型图，当前权为最小权。否则对环缩点，然后回到第二步继续判断。直到无环，然后展开收缩点。

下面是Sasuke\_SCUT的blog，解释了细节实现：

最小树形图，就是给有向带权图中指定一个特殊的点root，求一棵以root为根的有向生成树T，并且T中所有边的总权值最小。最小树形图的第一个算法是1965年朱永津和刘振宏提出的复杂度为O(VE)的算法。

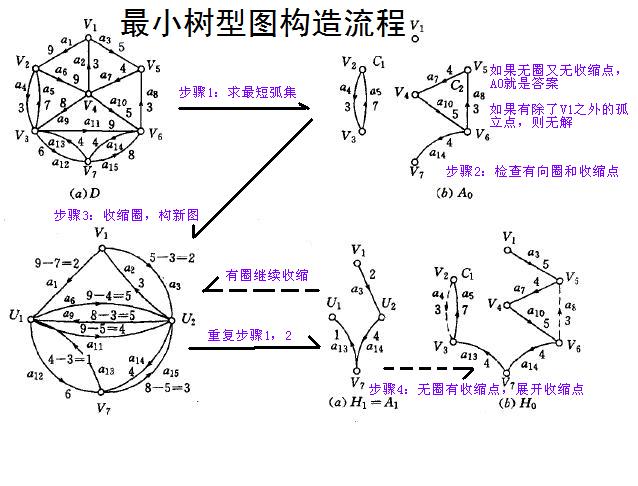
判断是否存在树形图的方法很简单，只需要以v为根作一次图的遍历就可以了，所以下面的算法中不再考虑树形图不存在的情况。

在所有操作开始之前，我们需要把图中所有的自环全都清除。很明显，自环是不可能在任何一个树形图上的。只有进行了这步操作，总算法复杂度才真正能保证是O(VE)。下面的说明可以参考下方图片，以点V2为例。

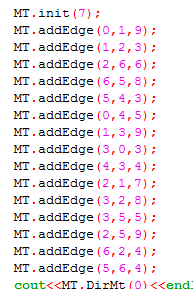
首先为除根V1之外的每个点选定一条入边，这条入边一定要是所有入边中最小的。现在所有的最小入边都选择出来了，如果这个入边集不存在有向环的话，我们可以证明这个集合就是该图的最小树形图。这个证明并不是很难。

下面主要针对存在环情况分析

下图最小生成树长度5+4+5+4+4+7=29即v1-v5-v4-v6-v7-v3-v2 也就是v1-u2-v7-u1，代码测试后也是29。

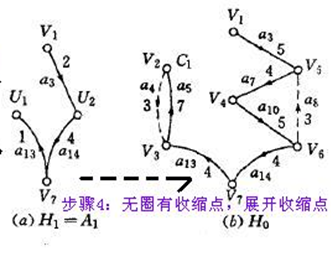


**注意：如果存在有向环的话，以上图为例，在上图(a)中我们先筛选出所有点的最小边集如(b)(根节点V1虽然是3，但是程序中会把它置为0不做运算)，这时出现了两个环v2v3和v4v5v6，我们就要将这两个有向环缩成两个人工顶点U1,U2，同时改变图中边的权，注意，图中及接下来的黑色文字说明了仅仅需要改变进入人工节点U1,U2的边的权，做减法，实际上，我在测试程序时，发现是新形成的图的所有节点即本图步骤3中的四个点，U1,U2,V3,V7全部都要按下述规则更改，即环内的边的权重不变，而所有进入这些新图节点的边的权重都要更新为当前权重减去与该点之前相连的点的最小入度计算出新的最小入度，如a6，在压缩后是U1-U2的连线，而以前是V2-V4的连线，所以就是说，此时对U2来说a6是入边，所以更新为9-V4的最小入度4得出5，也就是说，上图步骤3中，U1->V7边应变为6-V7的初始最小入度4，所以结果为2，U2->V7也要这么算，即4-4=0这样得到的，那可能有人问U2->V1的权重为什么不是3-3=0？这是因为V1是根，在程序计算时，已经把根的入度变为0了，所以实际还是3-0=3，这样在程序迭代时才会得出正确结果，即步骤4中(a)应该是U2->V7=0，这样才不会把加过的边多算一次，这也是程序代码中每次求ans时就把所有的in[i]不分好坏加在一起的原因，也是后面几行字原因的程序解释，即为毛要剪掉in[u]，下面两个程序截图验证了我的说法，第一行是步骤1中边的权重，顺序可以看右图，第二行是压缩后的边的权重。**假设某点u在该环上，并设这个环中指向u的边权是in[u]，那么对于每条从u出发的边(u, i, w)，在新图中连接(new, i, w)的边，其中new为新加的人工顶点; 对于每条进入u的边(i, u, w)，在新图中建立边(i, new, w-in[u])的边。为什么入边的权要减去in[u]，这个后面会解释，在这里先给出算法的步骤。然后可以证明，新图中最小树形图的权加上旧图中被收缩 的那个环的权和，就是原图中最小树形图的权。



上面结论也不做证明了。现在依据上面的结论，说明一下为什么出边的权不变，入边的权要减去in [u]。对于新图中的最小树形图T，设指向人工节点的边为e。将人工节点展开以后，e指向了一个环。假设原先e是指向u的，这个时候我们将环上指向u的边 in[u]删除(去环)，这样就得到了原图中的一个树形图。我们会发现，如果新图中e的权w'(e)是原图中e的权w(e)减去in[u]权的话，那么在我们删除掉in[u]，并且将e恢复为原图状态的时候，**这个树形图的权仍然是新图树形图的权加环的权**，而这个权值正是最小树形图的权值。所以在展开节点之后，我们得到的仍然是最小树形图。逐步展开所有的人工节点，就会得到初始图的最小树形图了。是一种保证了展开环后，路径总权值不变的去环手段。也就是说这么做是为了保证压缩成点时得到一个权值和最小的最终图和该最终图环展开后的路径权值和不变！！因为虽然有环被压缩为点，但是这个环也是当初取得顶点的最小入度得到的，所以，相当于最终求得压缩图的最小路径还是最小路径，所以展开的最小路径环总权值+压缩环后生成的图的最小路径总权值得到的就是最终的最小有向路径。(出边不变权值是因为我们是以考虑入边最小来建树的，所以只需要考虑一种情况而不用考虑出边，实际上，考虑入边对于本点也相当于考虑了出边对于其他点。)

可以以下图为例：



比如U1实际上是环v2，v3，所以打开环U1后V7-U1的路径和为4-3+3+7=11；这和最终右侧展开图中V7~V2的值4+7=11是一样的......即压缩的V7-U1和展开的V7-V3-V2路径和不变，也就是V7-U1权值+U1环的权值之和就是最终右侧的V7-V3-V2最短路径权值的和。就是考虑了所有点的最小入度情况下求出的最短路径，只是有的点的入度构成了环所以没有被选入。

**总的来说，我认为这种方法应该是一种在最优路径被隐藏(由于环)的条件下通过求一个其他次优路径相对于最优路径的偏差值作为产生的新的图的新的路径代价然后重复迭代这种思想找出最终最短路径的方法。体会下这种“折叠”再展开的思想。具体看代码，有详解，更清晰**

如果实现得很聪明的话，可以达到找最小入边O(E)，找环 O(V)，收缩O(E)，其中在找环O(V)这里需要一点技巧。这样每次收缩的复杂度是O(E)，然后最多会收缩几次呢？由于我们一开始已经拿掉了所有的 自环，我门可以知道每个环至少包含2个点，收缩成1个点之后，总点数减少了至少1。当整个图收缩到只有1个点的时候，最小树形图就不不用求了。所以我们最 多只会进行V-1次的收缩，所以总得复杂度自然是O(VE)了。由此可见，如果一开始不除去自环的话，理论复杂度会和自环的数目有关。

扩展：以上为定根最小树型图，对于无固定根最小树型图，只要虚拟一个根连所有的点的权为边权总和+1，最后的结果减去(边权+1)即可，对于本次实现的图来说，就是使V1不再是根了，也要参与运算，这时虚拟一个新的点作为根连入图中，再计算即可，最后结果剪掉这一虚拟根的边就行了。另外对于求所定的根标号，只要搜索被选中的虚边就可以判断了，注意灵活变化。

详细代码见 朱刘算法详细代码.cpp