**软件设计与实践**

**——综合部分结题报告**

**-** Hop Doubling Label Indexing for Point-to-Point Distance Querying On Scale-Free Networks

|  |  |
| --- | --- |
| **学号：** | **113030128** |
| **姓名：** | **杨尚斌** |

计算机科学与技术学院

**背景**

在大尺度、无标度的图中如何求出两点之间的最短距离一直是一个比较热门的问题。一般是这样的一个问题：给出一个有向图或者无向图，然后我们去为“求两点之间最短距离”这个问题建立一个索引，然后我们应该尽量控制索引的大小和相关复杂度。

《Hop Doubling Label Indexing for Point-to-Point Distance Querying On Scale-Free Networks》这篇文章就是关于求最短路径的这个问题提出了一个叫做“Hop Doubling Label”的技术，并且通过一个系列的证明以及测试，论述了这个方法目前在处理这种大图最短路径问题上的优势。

**算法描述**

**输入：**

|  |  |  |
| --- | --- | --- |
| **N** | **E** |  |
| **V1** | **V2** | **D** |
| **……** | **……** | **……** |
| **n1** | **n1** |  |

(默认输入的D都是1)

**输出：** 建立好的索引

迭代的次数

两点之间的距离

**一些基本概念：**

**Privot :**

对于两个点s和t，每个label就是label集合中的一个元素，并且每个label都是由（v, d）组成的，（v, d）代表v是属于V的，d就是之间的距离，我们称v是一个privot。简单说，就和他的词一样，代表中心。

**“s, t is covered by u”:**

如果对于一个有向图（V, E），我们对每一个点v产生两个label，即L\_in(v)和L\_out(v)，对于d(s, t )来说，如果他的值是无穷大，我们就能找到一个中心点u，对于这个中心点u来说，有两个label，属于L\_out(s) 的 (u, d1) 和属于L\_in(t)的（u, d2）。这样如果在s和t之间不存在比d1+d2更小的距离（即不再存在这样的一点，使得加起来的距离变得更小），我们就称他两之间的最小距离就是d1+d2。我们称为 ”s,t is covered by u””

**2-hop cover:**

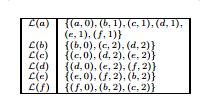
所有点的label的集合就称之为2-hop cover

**一个例子：**

对待这样一个无向图：



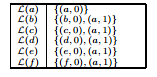
我们对待他，则会有下图：



但是我们发现在上面的表格图中无法再对这个表格的内容进行相关的压缩处理，即上面的表格已经是最小的，没有继续优化的余地。比如说：我们删掉L(b)中的(c, 2)这个普通的label，我们现在去求b到c的最短距离的话，我们就会得出一个不正确或者说效果不好的结果，即4(b到d，然后d到c)。

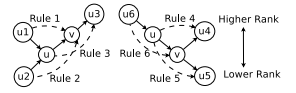
经过相关的研究发现，如果能比较好的确定一个privot的话，就会得出一个比较好的结果，即 能够得到一个比较优化的Indexing。

在这个图中，我们清楚的发现a的度是最大的，如果按照度的倒序排个rank的话，a的rank就是最小的. A的rank是最小的，也就是说他的度是最大的，这样一来，他所能hit到的点也就是最多的。我们把他作为privot，这样的话我们就能够得到下面的这个表：

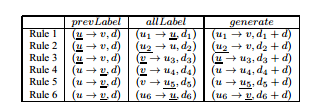


显然，上面的这个表是比较小的，除此之外他还能够清楚的表达和上面的图一样的信息。我们按照这个思路，对每次产生的结果进行相关的迭代，于是就有了这个算法。

在这个算法中，我们做的工作就是对rank的排序以及如何才能比较好的保证这个rank起到的作用，经过一系列的推导以及证明，有下面的图：



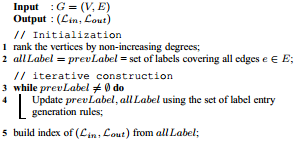
对上图做如下的解释：u和v是pivot，rank是把他们的度按照倒序进行排列的。在上图中，约定rank(u(i)) = I，在没两个label进行合并的时候需要按照上面图中的6条规则才能够最好的得出我们要的Index。在后序的研究中，发现1245这四条规则就可以完全得到上面中6条规则所得到的结果。也就是上，只需要1245这四条规则就可以产生比较好的indexing。



像上面的表中所列举的那样，我们对点之间取最小距离时的合并做一个定义：首先初始化所有的label，并且把他们放入到prevLabel和allLabel中，此时，preLabel和allLabel是完全相等的，然后开始正式的合并去找两点之间的最小值。我们用preLabel是不是为空去判断我们要进行的合并是不是已经结束，即再没有需要合并的label。

对于preLabel与allLabel，做如下的定义，preLabel 是用来存储每次比较新的Label的，allLabel用来储存所有的label。当preLabel里面的元素为空的时候，迭代结束，此刻allLabel里面的元素也正好就是最后的Indexing。

用伪代码表示的话整个流程如下：



**实验配置**

**操作系统：**Ubuntu 15.04 i386

**编译器：**GCC **4.9.2**

**电脑配置：**

**处理器：**Inter(R) Core(TM) i5-3230M CPU @ 2.60GHz

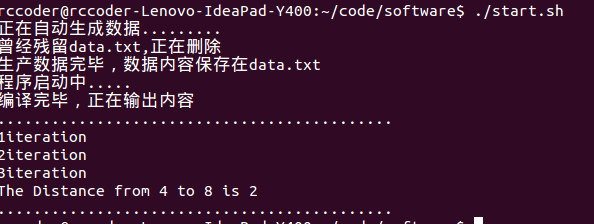
RAM：4GB

**实验结果**

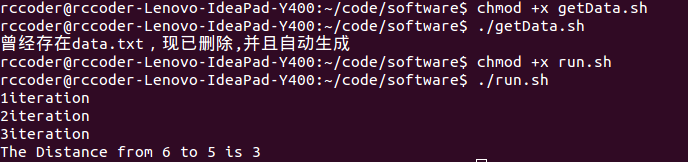
能够良好的建议正确的索引，并且利用索引一次或者是两次（至多两次）求出要求两点之间的最短距离。

**软件界面设计与演示说明**

**直接运行：**

****

**两步运行：**

****

程序运行由简单的**脚本文件**进行控制，可分为直接运行和分两步运行（先生成数据，然后在运行）。

上图中只给出了程序运行的界面，正如上面所显示的那样，会出现程序运行时迭代的次数，一次最后按照相关的要求求出两点之间的额最小距离。

关于程序的输入与输出的详细说明见README.md。

**结论**

通过阅读这篇文章，我了解了在大的无尺度的图中处理两点之间最短距离的问题的一些解决方案。

由于在实际生活中，我们遇到的图一般是比较大的图，如果按照小数据那样在需要的时候再进行查找一般是比较耗费时间的，或者说是不可行的。因此我们需要按照相关的要求与目的给图建立一个适合的索引，然后利用索引就能比较方便的求出最后的结果。

在这篇文章中提到的就是2 hop label。这个方法是在之前的is-label的基础上改进的，即为每个点按照他们度的大小进行了排序，然后依照相关的证明显示要求出最短距离必须在他们按照度排的大小顺序上有一定的规则，通过推理验证吗，得出了六条规则，然后又通过推理，去掉了两条繁杂的规则（可以由其他规则在迭代后推出）。最后建立了索引，以致在最后求两点之间最短距离的时候是非常方便的，只需对索引进行极为简单的操作即可（基本可以去除运算量）。

在编码过程中，我初步了解了面对一篇12页左右的论文应该如何去读与如何去理解。同时，也在很大的意义上加强了cpp以及一些数据结构与基础算法的联系。