各类最短路径问题算法的分析比较研究

信息42 王浩成 2140502043

西安交通大学电信学院信息与通信工程系 710049 西安市

摘要：最短路问题是图论中的核心问题之一，它是许多更深层算法的基础。同时，该问题有着大量的生产实际的背景。不少问题从表面上看与最短路问题没有什么关系，却也可以归结为最短路问题。本文介绍了相关的基本概念、常用算法及其适用范围，并对其应用做出了举例说明，侧重于模型的建立、思考和证明的过程，最后做出总结。

关键词：最短路径 Dijkstra Bellman-Ford SPFA 效率比较 模型应用

1 基本概念

1.1最短路径

乘汽车旅行的人总希望找出到目的地尽可能短的行程。如果有一张地图并在地图上标出了每对十字路口之间的距离，如何找出这一最短行程？

一种可能的方法是枚举出所有路径，并计算出每条路径的长度，然后选择最短的一条。然而我们很容易看到，即使不考虑含回路的路径，依然存在数以百万计的行车路线，而其中绝大多数是没必要考虑的。

下面我们将阐明如何有效地解决这类问题。在最短路问题中，给出的是一有向加权图G=(V,E)，在其上定义的加权函数W:ER为从边到实型权值的映射。路径P=(v0, v1,……, vk)的权是指其组成边的所有权值之和：

定义u到v间最短路径的权为

从结点*u*到结点*v*的**最短路径**定义为权的任何路径。

在乘车旅行的例子中，我们可以把公路地图模型化为一个图：结点表示路口，边表示连接两个路口的公路，边权表示公路的长度。我们的目标是从起点出发找一条到达目的地的最短路径。

在其他的实际应用当中，边的权常被解释为一种度量方法，而不仅仅是距离。边权常常被用来表示时间、金钱、罚款、损失或任何其他沿路径线性积累的数量形式。

1.2相关概念的说明

单源最短路问题：找出从某指定结点v到每一结点u的一条最短路径。多数将要讨论的最短路算法基于解决这一类问题。多数其他类型的最短路问题可以通过适当的变形为此类问题得以解决。

单目标最短路问题：找出从每一结点v到某指定结点u的一条最短路径。把图中的每条边反向（在图论当中称作构建反图），我们就可以把这一问题转化为单源最短路径问题。

每对节点间的最短路问题：对于每对结点u和v,找出从u到v的最短路径。这类问题可以转化为对每一结点的单源最短路问题，也可以使用更有效率的Floyd算法来解决。受限于此算法所要求的模型特殊性，在本文中不予讨论。

负权边：在模型转化的过程当中，可能出现边的权值为负的特殊情况。在这种情况下，一些特定的最短路算法，如Dijkstra算法将无法得到正确的答案。另一些算法可以略作修改以保证正确性。

2 常用算法

* 1. Dijkstra算法

Dijkstra算法解决了有向加权图的最短路径问题，该算法的条件是该图所有边的权值非负，因此在本小节我们约定：对于每条边

Dijkstra算法中设置了一结点集合S，从源结点s到集合S中结点的最终最短路径的权均已确定，即对所有结点，有。算法反复挑选出其最短路径估计为最小的结点uV-S，把u插入集合S中，并对离开u的所有边进行松弛。在下列算法实现中设置了优先队列Q，该队列包含所有属于V-S的结点，且队列中各结点都有相应的d值。算法假定图G由邻接表表示。

**Dijkstra(G,w,s)**

1. INITIALIZE-SINGLE-SOURCE(G,S)
2. S
3. QV[G]
4. While Q
5. Do uEXTRACT-MIN(Q)
6. SS U{u}
7. For 每个顶点vAdj[u]
8. Do RELAX(u,v,w)

因为*Dijkstra*算法总是在集合V-S中选择“最轻”或“最近”的结点插入集合S中，因此我们说它使用了贪心策略。需要指出的是，贪心策略并非总能获得全局意义上的最理想结果。

* 1. Bellman-Ford算法

Bellman-Ford算法能在一般的情况（存在负权边的情况）下，解决单源最短路问题。对该图运行Bellman-Ford算法可返回一个布尔值，表明图中是否存在着一个从源点可达的权为负的回路。若存在这样的回路的话，算法说明该问题无解；若不存在这样的回路，算法将产生最短路径及其权值。

**Bellman-Ford(G,w,s)**

1. INITIALIZE-SINGLE-SOURCE(G,s)
2. For i1 to |V[G]|-1
3. Do For 每条边(u,v)E[G]
4. Do RELAX(u,v,w)
5. For每条边(u,v)E[G]
6. Do If d[v] > d[u] + w(u, v)
7. Then Return FALSE
8. Return TRUE

1.3 SPFA

SPFA算法的全称是：*Shortest Path Faster Algorithm*。其本质是基于Bellman-Ford算法的一种特殊的改进。

SPFA采取数组d记录每个结点的最短路径估计值，而且用邻接表来存储图G。采取的方法是动态逼近法：设立一个先进先出的队列用来保存待优化的结点，优化时每次取出队首结点u*，*并且用u点当前的最短路径估计值对离开u点所指向的结点v进行松弛操作，如果v点的最短路径估计值有所调整，且v点不在当前的队列中，就将v点放入队尾。这样不断从队列中取出结点来进行松弛操作，直至队列空为止。

**SPFA(G,w,s)**

1. INITIALIZE-SINGLE-SOURCE(G,s)
2. INITIALIZE-QUEUE(Q)
3. ENQUEUE(Q,s)
4. While Not EMPTY(Q)
5. Do uDLQUEUE(Q)
6. For 每条边(u,v)E[G]
7. Do tmpd[v]
8. Relax(u,v,w)
9. If (d[v] < tmp) and (v不在Q中)
10. ENQUEUE(Q,v)
11. 效率比较

本文采用C++语言实现上述算法。源代码、测试数据、测试环境见附录。

|  |  |  |  |
| --- | --- | --- | --- |
| 算法  数据 | Dijkstra | Bellman-Ford | SPFA |
| 第一组 | 2ms | 5ms | 3ms |
| 第二组 | 9ms | 16ms | 4ms |
| 第三组 | 23ms | 12ms | 7ms |

第一组：10^3个点的完全图

第二组：10^5个点的一条链

第三组：5\*10^4个点的稀疏图（每个点10条边）

可以看出，在稠密的图（完全图）中Dijkstra算法表现较好，这是因为在该算法实现下，每个EXTRACT-MIN操作需要时间为O(lgV)，存在|V|次这样的操作。建立二叉堆需要时间为O(V)。在RELAX中的赋值语句是通过调整结点v在二叉堆中的位置来完成的，运行时间为O(lgV)，并且至多存在|E|次这样的操作。因此算法的全部运行时间为O((V+E)lgV)。除了与边数相关的部分以外，算法复杂度与点的数量关系较为密切，因此在稠密图中效率较高；而在稀疏的图当中Bellman-Ford以及SPFA表现更好。对于后两者进行分析可得Bellman-Ford算法的运行时间为O(VE)。因为第1行的初始化占用时间为O(V)，第2-4行对边进行的|V|-1次操作的每一次运行时间为O(E)，第5-7行的For循环的运行时间为O(E)。对于SPFA算法，每次取出队首结点u，并访问u的所有临结点的复杂度为O(d)，其中d为点u的出度。运用均摊分析的思想，对于V个点E条边的图，点的平均出度为，所以每处理一个点的复杂度为。假设结点入队的次数h，显然h随图的不同而不同。但它仅与边的权值分布有关。我们设h=kV，则算法SPFA的时间复杂度为



在平均的情况下，可以将k看成一个比较小的常数，所以SPFA算法在一般情况下的时间复杂度为O(E)。SPFA与Bellman-ford两者的思想都属于标号修正的范畴。算法是迭代式的，最短路径的估计值都是临时的。算法思想是不断地逼近最优解，只在最后一步才确定想要的结果。但是他们实现的方式上存在差异。正因为如此，它们的时间复杂度其实有较大差异的。在Bellman-Ford算法中，要是某个点的最短路径估计值更新了，那么我们必须对所有边指向的终点再做一次松弛操作；在SPFA算法中，某个点的最短路径估计值更新，只有以该点为起点的边指向的终点需要再做一次松弛操作。在极端情况下，后者的效率将是前者的n倍，一般情况下，后者的效率也比前者高出不少。基于两者在思想上的相似，可以这样说，SPFA算法其实是Bellman-Ford算法的一个进一步优化的版本。

1. 应用举例

4.1差分约束系统

差分约束系统是一个线性程序设计中特殊的一种，线性程序设计中矩阵A的每一行包含一个1和一个-1，A的所有其他元素均为0。由给出的约束条件形成m个差分约束的集合，其中包含n个未知单元。每个约束条件均可构成简单的不等式如下：

简单举例：

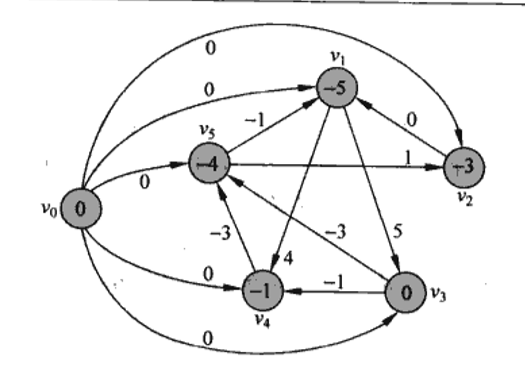


找出未知量并且满足以下差分约束条件：

4.2约束图

用图形理论观点来解释差分约束系统是很有益的。在一理想的差分约束系统中，的线性规划矩阵A可被看作是n顶点、m条边的图的邻接矩阵的转置。对于，图中的每一个顶点对应着n个未知量中的一个。图中的每个有向边对应着关于两个未知量的m个不等式的其中一个。

引入附加顶点保证其他每个顶点均从可达。因此，顶点集合V由对应于每个未知量的顶点和附加的顶点构成。边的集合E由对应于每个差分约束条件的边与对应于每个未知量的边所构成。如果是一个差分约束，则边的权。从出发的每条边的权值均为0。上述例子所对应的约束图如下。



我们可以证明，能够通过在相应的约束图中找出最短路径的方法，来求得对应差分约束系统的解。

具体的，给定一差分约束系统,设为其相应的约束图。如果G不包含负权回路，那么

是此系统的一个可行解。如果G包含负权回路，那么此系统不存在可行解。

考虑到约束图中会出现负权边，Dijkstra算法不适用于求解差分约束系统；可以采用SPFA或者Bellman-Ford算法对差分约束系统求解。

4.3实例

**zju1420 Cashier Employment出纳员问题**

问题描述：

一家每天24小时营业的超市，需要一批出纳员来满足它的需要。超市经理雇佣你来帮他解决他的问题——超市在每天的不同时段需要不同数目的出纳员（例如：午夜时只需一小批，而下午则需要很多）来为顾客提供优质服务。他希望雇佣最少数目的出纳员。

经理已经提供你一天的每一小时需要出纳员的最少数量——。表示从午夜到上午1：00需要出纳员的最少数目，表示上午1：00到2：00之间需要的，等等。每一天，这些数据都是相同的。有人申请这项工作，每个申请者在每24小时中，从一个特定的时刻开始连续工作恰好8小时，定义为上面提到的开始时刻。也就是说，如果第个申请者被录取，他（她）将从时刻开始连续工作8小时。

编写一个程序，输入和，它们都是非负整数，计算为满足上述限制需要雇佣的最少出纳员数目。在每一时刻可以有比对应的更多的出纳员在工作。

问题分析：

考虑如下的不等式模型：

设为时刻能够开始工作的人数，为实际雇佣的人数，那么  
设为时刻至少需要工作的人数，于是有如下关系：   
   
设，得到   
  
 对于以上的几组不等式，我们采用一种非常笨拙的办法处理这一系列的不等式(其实也是让零乱的式子变得更加整齐，易于处理)。首先我们要明白差分约束系统的应用对象（它通常针对多个二项相减的不等式）于是我们将上面的所有式子都转化成两项未知项在左边，另外的常数项在右边，且中间用连接的式子，即:

这里出现了小的困难，我们发现以上式子并不是标准的差分约束系统，因为在最后一个式子中出现了三个未知单位。但是注意到其中跟随变化的只有两个，于是就变得特殊起来，看来是需要我们单独处理，于是我们把当作已知量放在右边。

经过这样的整理，整个图就很容易创建了。对于所有形如 A-BC 的式子，我们从节点B 引出一条有向边指向 A ；边的权值为C，图就建成了。  
 最后枚举所有的可能值，对于每一个，我们都进行一次常规差分约束系统问题的求解，判断这种情况是否可行，如果可行求出需要的最优值，记录到Ans中，最后的Ans的值即为所求。

1. 参考文献

[专著] Thomas H. Cormen, Charles E. Leiseison, Ronald L. Rivest and Clifford Stein. Introduction to Algorithm, Second Edition[M].机械工业出版社，2001:357-370