MOPEC-2010-001离散数学习题解析

图论(2) 最短路径Dijkstra Algorithm

魏恒峰 2011 年 5 月 23 日

本文档用以介绍与解释求最短路径的 Dijkstra Algorithm.

图 1: Dijkstra at the blackboard during a conference at ETH Zurich in 1994

1 问题描述

Problem 1.1. 最短路径问题

最短路径问题描述如下:

给定带权图G,求某固定源点$s(source)$到其它所有顶点的最短路径.

Remark 1.2. 最短路径问题

最短路径问题至少有以下几种不同的版本:

1. 无向图 *vs.* 有向图
2. 非负权值边 *vs.* 允许边的权值为负值的情况

3. 单源点多目标点 *vs.* 多源点多目标点

我们这里所考虑的是无向图,非负权值边情况下的单源点多目标点最短路径问题.下文中所说的最短路径问题均为该类.

Remark 1.3. 最短路径问题的应用

在实际生活和应用中，我们经常会考虑最短路径的问题.比如,假设你有你所在乡村城镇的设计图,这张设计图标注了每一户人家、每一座工厂、每一所学校还有所有的重要设施的位置.不仅如此，街道也都标注了长度. 现在你所关心的问题是，以我家为源点，到该城镇的其它所有位置的最短路径分别是多少.

最短路径算法的另一个重要应用就是网络路由.

2 算法设计

Dijkstra Algorithm — shortest path from one vertex:

输入: 无向非负带权图G以及单源点u.($w(xy)$表示边xy的权值.)

算法思想: 维护一个顶点集合S,对该集合中的所有顶点v均已经求出从u到v的最短路径.最初的时候,$S = \{ u \}$, 该算法的过程就是逐步扩大该集合,每次迭代使得给集合元素增加一个(永不删除),最终使得$S = V(G)$.为了达到该目的,我们维护另外一组变量$t(z)$(t表示 tentative ,“试探的”; $z \notin S$),表示目前(算法迭代了多次，运行到该时刻)所发现的$u-z$最短路径的长度.我们在迭代的每一步，就是通过$t(z)$来决定将哪一点加入到S中来的.

初始化: $S = \{ u \} ; t(u) = 0; t(z) = w(uz), \forall z \neq u.$

算法迭代过程: 选取顶点v,$t(v) = \min_{z \notin S} t(z)$,也就是不属于$S$集合的顶点中$t(z)$最小的顶点.将$v$加入$S$.

考察v所邻接的顶点(当然也就考察了v所邻接的边),需要注意的是,此时我们只需要考虑与v邻接但是不属于S的那些顶点(对于属于S的顶点,最短路径已经求出来了，关于该点，我们会在后面加以证明).$\forall vz \in E(G) \wedge z \notin S, t(z) = \min \{ t(z), t(v) + w(vz) \}$.

算法终止: S不断扩大，直到$S = V(G)$ 或者$\forall z \notin S, t(z) = \infty$. 算法结束时，$\forall v, d(u,v) = t(v)$.

3 算法分析

我们需要证明该算法的正确性.

在算法设计2的思想描述中，大家已经看到，该算法维护了两个变量，一个是集合S，另一个是$t(z)$.在算法的迭代过程中，这两个集合相互依赖.也就是说，在扩展集合S时，我们选取的是$t(z)$最小的点;在扩展了集合S之后，我们又更新了$t(z)$的值. 所以，在算法正确性证明中，我们需要同时考虑这两个集合的性质.同样，在算法设计2中，我们已经描述了这两个变量的性质:对于S来说，它的元素是已经求出u到v的最短路径的那些点;对于$t(z)$来说，它是目前所发现的$u-z$最短路径的长度.我们可以把这两个性质看作循环的不变式.所谓循环不变式，是指在循环(迭代)过程中保持不变的那个量.寻求这种量对于算法的正确性证明至关重要.

我们所要证明也就是这个循环不变式:

在每一次迭代开始时:

1. $\forall z \in S, t(z) = d(u,z)$,
2. $\forall z \notin S, t(z)$ is the least length of a $u-z$ path reaching z directly from S.

Proof. 循环不变式

使用数学归纳法.对$|S| = k$做归纳.

基础步: 初始时:

1. $S = \{u\}, d(u,u) = t(u) = 0$;
2. 从u到其他顶点z的最短路径长度为$t(z) = w(u,z)$.

归纳假设: 假设$|S| = k$时,两个循环不变式都成立.(注意，我们同时考虑两个循环不变式.)

归纳证明: v是$t(z)(z \notin S)$中最小的.按照算法，我们将v加入到S中,并且按照算法过程更新$t(v)$.此时，$|S| = k + 1$.我们需要证明循环不变式仍然成立.

1. 先证明第一个关于S的循环不变式成立.

   只需证明对于新加入的点v,$d(u, v) = t(v)$成立即可.

   显然，任何$u - v$路径都必须是先“穿过”集合S中的顶点，然后再直接或者间接地到达v.

   对于穿过S集合然后直接与v相连的情况,我们借助于$|S| = k$的情况下对于第二个循环不变式的归纳假设可知,穿过S集合直接到达v的最短路径的长度为$t(v)$;

   对于第二种情况,假设$u - v$路径在穿过S后先到达$z(z \notin S)$,然后再与v相连(见图).根据我们对v的选择，我们知道$t(z) \geq t(v)$，所以该间接路径的长度必定不小于$t(v)$.

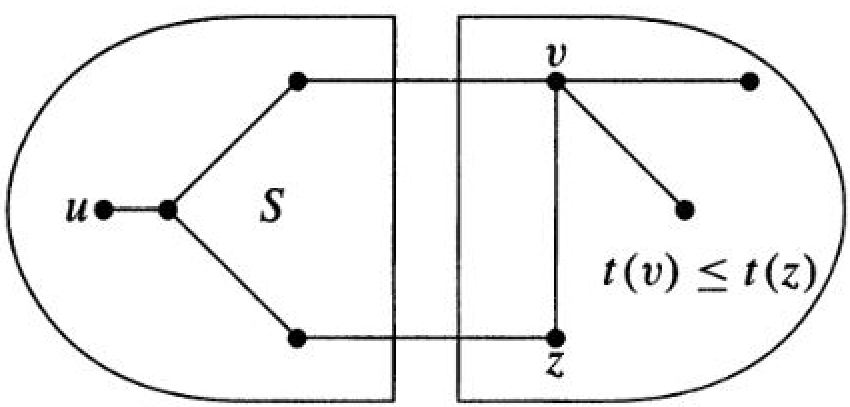

图 2: 算法正确性证明用图

2. 再证明第二个关于$t(z)$的循环不变式成立.

   根据归纳假设,在没有将v加入S时，也就是$|S| = k$的情况下,从S直接到达$z(z \notin S)$的最短的$u - z(z \notin S)$路径的长度为$t(z)$.

   现在，我们加入了新的v点.v点可能与z相连，它可能使得从S(v现在是S的一员)直接到z的最短路径的长度变小(也就是$t(z)$需要更新)(注意，由归纳假设$t(v) = d(u, v)$为最短).如果这种情况发生了，根据算法，我们

更新了$t(z)$，而且取的是最小值.当然，在$|S| = k + 1$的情况下,从S直接到z的最短路径的长度为$\min\{ t(z), t(v) + w(vz) \}$.

□

Remark 3.1. 正确性证明

在证明过程中，我们用到了两个循环不变式，而且这两个循环不变式相互依赖.在证明第一个循环不变式时，用到了第二个循环不变式的归纳假设.同样地,在证明第二个循环不变式时，也用到了第一个循环不变式的归纳假设.这种技巧我们称为*"co-induction"*.

大家对此可能会有疑惑,两个相互依赖的话，证明A需要用到B，证明B又要用到A，这不成了无限循环了吗?其实不然，这和普通递归的情况类似.实际上，证明A时，我们用的是规模小一些的B.所以，只要基础步有保证，该类递归就是可行的.

大家可以找一个图的例子，然后对照着证明过程，从初始条件开始依次验证该过程，大家会体会到*"co-induction"*的意思，并且发现它并没有造成死循环.

Remark 3.2. 跳出算法

我们站在算法之外来观察该算法过程给问题带来了什么特殊的性质.

1. $\forall x \in S, z \notin S, d(u, x) \le t(z)$.
2. 该算法以$d(uv)$非递减的顺序选取并计算了源点u与其他所有顶点的最短距离.

4 扩展

关于该算法的更多信息，请参见图3. 该网页的最后给出了很多有用的链接.

图 3: Wiki for dijkstra algorithm