最小生成树

wcz

April 3, 2018

Contents

1 前言

图论中的很多东西都是有联系的, 学习的时候要试着发现它们之间的联系. 关于图论中的最小生成树的算法, 大概已经成为一种大家都会一点的算法了, 但是它仍然是非常有用的.

在这里我主要是介绍一下最小生成树的两种算法, Prim 算法和 Kruskal 算法. 它们都有一定的缺陷, 但是通常情况下是都有较好的效率的.

1.1 生成树

图 G 的一个极小联通子图称为生成树. 从图的一个顶点出发不重复的遍历每个顶点形成一棵生成树 G_1, 它包含图中所有的顶点, 图中所有的边 E

被分为两个集合 B 和 T, 其中 T 为生成树上的边, $G_1 = V, T$. 生成树满足树的所有性质, 即有 $|V| - 1$ 条边, 树上两个结点之间有唯一路径.

- 利用深度优先遍历形成的生成树称为深度优先生成树/森林.
- 利用宽度优先遍历形成的生成树称为宽度优先生成树/森林.

2 Prim 算法

2.1 由来

该算法于 1930 年由捷克数学家 Vojtěch Jarník 发现, 并在 1957 年由美国计算机科学家 Robert C. Prim 独立发现,1959 年, 荷兰计算机科学家 Edsger Wybe Dijkstra 再次发现了该算法. 因此, 普里姆算法又被称为 DJP 算法、亚尔尼克算法或普里姆 - 亚尔尼克算法.

2.2 步骤

- 首先确立一个集合 U. 表示目前已经加入生成树中的点, 它开始时是一个空集.$V - textU$ 是未加入生成树的点的集合.
- 任选一个点加入 U, 表示目前生成树中存在一个点.
- 将属于 $V - U$ 且与 U 中点距离最短的点 v 加入 U.
- 当 $|U| = |V|$ 时结束, 最小生成树构造完成.

有没有发现算法和 Kruskal 算法的相似之处?

思考: 为什么表现出这种相似之处?

2.3 正确性证明

Prim 算法本质上是一种贪心算法. 我们需要证明这种贪心策略是正确的.

每次将一条边 $u \leftrightarrow v$ 加入生成树时, 是用边 $u \leftrightarrow v$ 将 U 与 $V - U$ 连接了起来, 而这条边并不会影响 U 与 $V - U$ 之间点的连通性, U 与 $V - textU$ 可以看成是两个互不相联系的强联通分量, 假设加入的边 $u_1 \leftrightarrow v_1$ 并不是最小的, 那么完全可以将其替换成更小的一条边而不会影响结果. 因此这种贪心策略是正确的.

2.4 算法实现

- 使用邻接矩阵存图, 算法复杂度大概是 $O(\mathrm{V}^2)$.
- 使用二叉堆优化寻找最小边, 复杂度大概是 $O(\mathrm{E}\log\mathrm{V})$.
- 使用斐波那契堆优化, 复杂度大概是 $O(\mathrm{E}+\mathrm{V}\log\mathrm{V})$.

未经堆优化的 Prim 算法复杂度远远高于 Kruskal 算法. 因此只推荐第两种形式, 最后一种是在是很难写, 而且对于一些对复杂度要求不是很大的题目来说是在是很鸡肋.

第一种形式的代码我在 luogu 的题解中见过.

3 Kruskal 算法

Kruskal 算法的优点是好实现, 当然在效率方面并不一定会比 Prim 算法优秀, 看具体的图. 它和 Prim 算法最大的不同之处在于它是将边加入生成树而 Prim 算法是将点加入最小生成树, 因此 Prim 算法的效率并不太取决于边的数量.

在比较稠密的图中使用 Prim 算法理论上效率比 Kruskal 算法效率高, 我遇到过一道这种题.

3.1 步骤

- 将所有的边按边权大小进行排序.
- 将连接处于不同强联通分量的权值最小边加入生成树.
- 当生成树中有 $|\mathrm{V}|-1$ 条边时生成树构造完成.

可以利用并查集来维护点之间的联通性.

3.2 正确性证明

Kruskal 算法的证明可以借鉴 Prim 算法的思路, 每次往最小生成树中加入一条边时是连接了两个强联通分量, 而这两个强联通分量的所有边中最小的必定存在于最小生成树上.

3.3 算法实现

这个东西大家应该都会. 确实是很好写.

算法复杂度大概是 $O(\mathrm{E}\log\mathrm{E})$ 或者可以表述为 $O(\mathrm{E}\log\mathrm{V})$

4 联系

可以看出 Prim 算法和 Kruskal 算法都是利用连接两个不相干点集的最短边来构建最小生成树. 这是它们的一致性, 类比 Dijkstra 算法和 SPFA 算法, 发现 Prim 算法和 Kruskal 算法之间的关系与 Dijkstra 算法和 SPFA 算法相似, 后者更好实现, 前者在很多情况下比后者更优秀, 后者的复杂度依赖于边数, 前者能用堆结构优化.

在一些对算法复杂度要求不是很高的题目推荐使用 Kruskal 算法, 一些边数非常多而且时间约束大的题目推荐使用堆优化的 Prim 算法.