BeIlmanFord算法与SPFA算法

BeIlmanFord算法

贝尔曼-福特算法（Bellman-Ford）是由理查德·贝尔曼（Richard Bellman）和莱斯特·福特创立的，求解单源最短路径问题的一种算法。其优于Dijkstra的方面是边的权值可以为负数、实现简单，缺点是时间复杂度过高。但它也有特别的用处，一般用于实现通过m次迭代求出从起点到终点不超过m条边构成的最短路径。

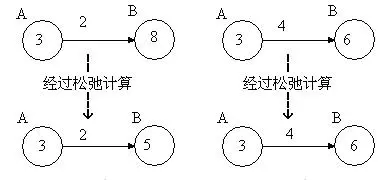
Dijkstra算法不能解决带有负权边的问题，而Bellman-ford算法可以解决带有负权边的问题，是求解带负权边的单源最短路问题的经典算法。时间复杂度是O(nm)，核心思想是”松弛操作”。解决带负权边的单源最短路问题还有一个常用的算法是SPFA算法，SPFA算法的时间复杂度一般是O(m)，最坏是O(nm)。

贝尔曼-福特算法与迪科斯彻算法类似，都以松弛操作为基础，即估计的最短路径值渐渐地被更加准确的值替代，直至得到最优解。在两个算法中，计算时每个边之间的估计距离值都比真实值大，并且被新找到路径的最小长度替代。 然而，迪科斯彻算法以贪心法选取未被处理的具有最小权值的节点，然后对其的出边进行松弛操作；而贝尔曼-福特算法简单地对所有边进行松弛操作，共 |V|-1次，其中|V| 是图的点的数量。在重复地计算中，已计算得到正确的距离的边的数量不断增加，直到所有边都计算得到了正确的路径。这样的策略使得贝尔曼-福特算法比迪科斯彻算法适用于更多种类的输入。

时间复杂度

O(nm)和O(n ^（2）m)就是时间复杂度。符号O表示复杂度， O(nm)可以粗略地理解为运行次数是n ×m.O(n^(2)m)比O(nm)运行时间差不多大n倍。

松弛操作

不断更新最短路径和前驱结点的操作。

如上图所示，松弛计算之前，点B的值是8，但是点A的值加上边上的权重2，得到5，比点B的值（8）小，所以，点B的值减小为5。这个过程的意义是，找到了一条通向B点更短的路线，且该路线是先经过点A，然后通过权重为2的边，到达点B。

当然，如果出现右边这种情况，则不会修改点B的值，因为3＋4>6。

对边集合 E 中任意边，以 w(u,v) 表示顶点 u 出发到顶点 v 的边的权值，以 d[v] 表示当前从起点 s 到顶点 v 的路径权值

若存在边 w(u,v)，使得：

d[v]> d[u]+w(u,v)

则更新 d[v] 值：

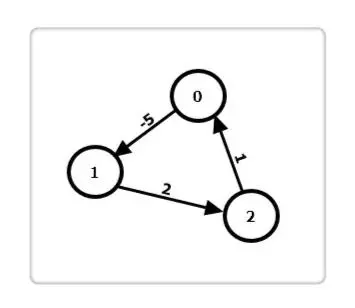
d[v]=d[u]+w(u,v)

所以松弛函数的作用，就是判断是否经过某个顶点，或者说经过某条边，可以缩短起点到终点的路径权值。

为什么将缩短距离的操作称之为“松弛”，不妨理解为，选择某种方式后，到达目的的总代价降低了。什么名字无关紧要，不必纠结。

负权回路

绕一圈绕回来发现到自己的距离从0变成了负数，到各结点的距离无限制的降低，停不下来。在循环n-1次的基础上再次遍历各边，对于所有边，只要存在一条边e(u, v)使得 dis[u] + w(u,v) < dis[v]，则该图存在负权回路。





检测负权回路

为了检测图中是否存在负环路，即权值之和小于0的环路。

检测的方法很简单，只需在求解最短路径的 n-1 次循环基础上，再进行第 n 次循环：对于每一条边e(u, v)，如果存在边使得dis[u] + w(u, v) < dis[v]，则图中存在负环路，即是说改图无法求出单源最短路径。否则数组dis[n]中记录的就是源点s到各顶点的最短路径长度。

小结

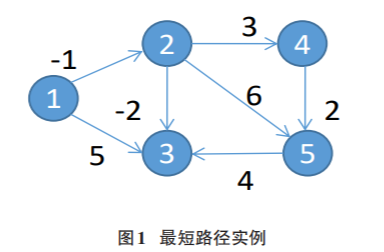
Bellman－Ford算法可以大致分为三个部分：

初始化所有点。每一个点保存一个值，表示从原点到达这个点的距离，将原点的值设为0，其它的点的值设为无穷大（表示不可达）。

进行循环，循环下标为从1到n－1（n等于图中点的个数）。在循环内部，遍历所有的边，进行松弛计算。

遍历途中所有的边，判断是否存在这样情况： d（v） > d (u) + w(u,v)，若存在，则返回false，表示图中存在从源点可达的权为负的回路。

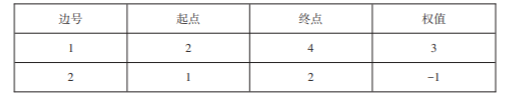
之所以需要第三步的原因，是因为，如果存在从源点可达的权为负的回路，则将因为无法收敛而导致不能求出最短路径。

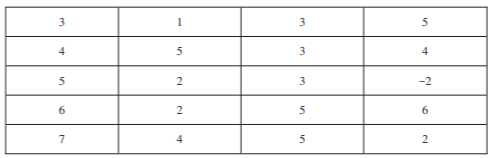


计算实例

例题：如图1所示，求1号顶点到其余各顶点的最短距离。我们用d 数组记录起点到其余各点的最短路径值，用s、e、t三个数组来存储边的信息。例如第 i 条边存储在 s[i]、e[i]、t[i]中，表示从顶点s[i]到e[i]这条边的权值为t[i]。

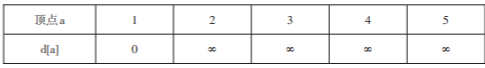
给出边的顺序如下表：





用d数组来存储1号顶点到其余各点的路径值。

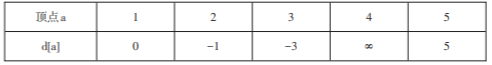
初始化如下表：



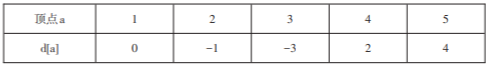
根据边给出的顺序，先处理第一条边“2-4-3”即判断一下

d[4]是否大于d[2]+3，由于此时d[4]和d[2]都是无穷大，因此这条边松弛失败。接下来处理第二条边“1-2-（-1）”，我们发现 d[2] > d[1] + (-1) ，通过这条边可以使d[2]的值从∞变为 -1 ，所以这个点松弛成功。我们可以用同样的方法来处理第三条边到第七条边，对所有的边进行一遍松弛操作后的结果如下：

第一轮对所有边进行松弛以后，结果如下表所示：



第二轮对所有边进行松弛以后，结果如下表所示：

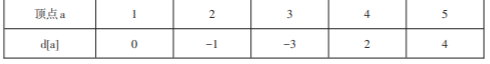


在这轮松弛中，通过“2 4 3”(2→4)这条边，更新了1号顶点到4号顶点的距离(d[4]) 。这条边在第一轮松弛失败，却在第二轮松弛成功。原因是在第一 轮松弛过后，1号顶点到2号顶点的距离(d[2]) 已经发生了变化，这一轮再通过“2 4 3”(2-→4)这条边进行松弛的时候，可以使 1 号顶点到 4 号顶点的距离(d[4]) 的值变小。也就是说，第一轮遍历图中所有边进行松弛操作之后，得到的是起点“经过一条边”到达其余各点的最短路径值。第二轮遍历图中所有边进行松弛操作之后，得到的是从起点“至多经过两条边”到达其余各点的最短路径值。如果进行n-1轮的话，得到的就是起点“至多经过n-1条边”到达其余各顶点的最短路径值。在一个含有n个顶点的图中，由于任意两点之间的最短路径最多经过n-1条边，因此最多松弛n-1轮。

第三轮对所有边进行松弛以后，结果如下表所示：



第四轮对所有边松弛以后，结果如下表所示：

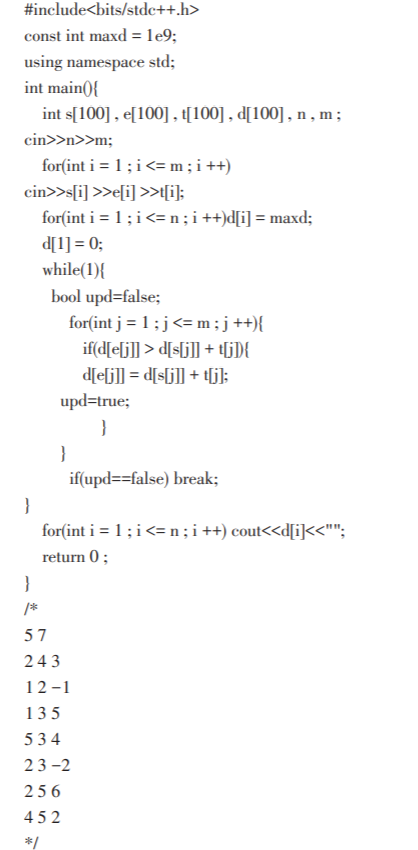


从第三轮开始，对所有边进行松弛操作，发现没有顶点需

要更新，此时便可以提前结束遍历，优化效率。最后表中数据

就是1号顶点到其余各点的最短路径值。

根据以上分析本题完整代码如下：



SPFA算法

队列优化，去掉一些无用的松弛操作，用队列来维护松弛造作的点。继承了Bellman-Ford算法的思想，但时间复杂度相对来说提高了很多。

基本思想

只有一个点在上一轮被松弛成功时，这一轮从这个点连出的点才有可能被成功松弛。

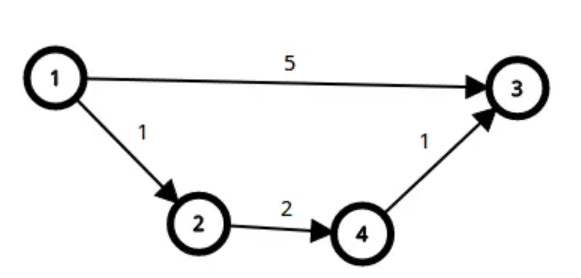
松弛的本质其实是通过一个点中转来缩短距离（如果你看了前置很容易理解）。所以，如果起点到一个点的距离因为某种原因变小了，从起点到这个距离变小的点连出的点的距离也有可能变小（因为可以通过变小的点中转）。

所以，可以在下一轮只用这一轮松弛成功的点进行松弛，这就是SPFA的基本思想。

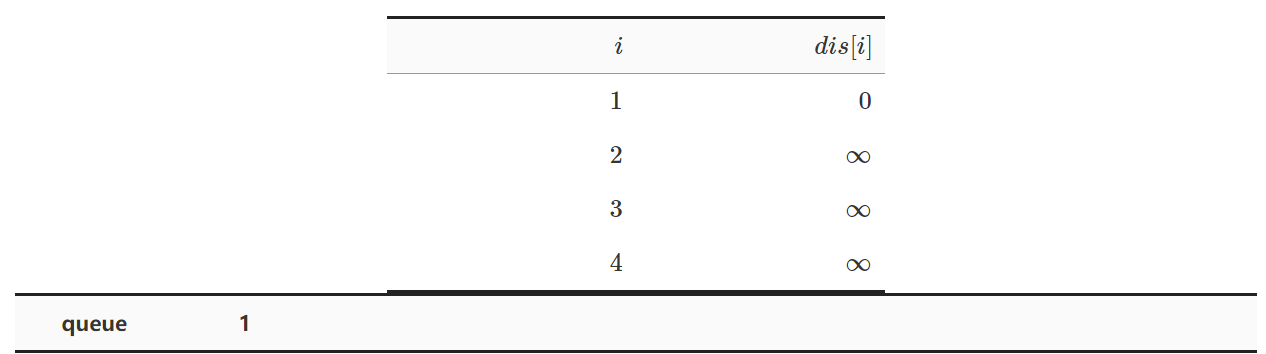
用队列实现

我们知道了在下一轮只用这一轮松弛成功的点进行松弛，就可以把这一轮松弛成功的点放进队列里，下一轮只用从队列里取出的点进行松弛。

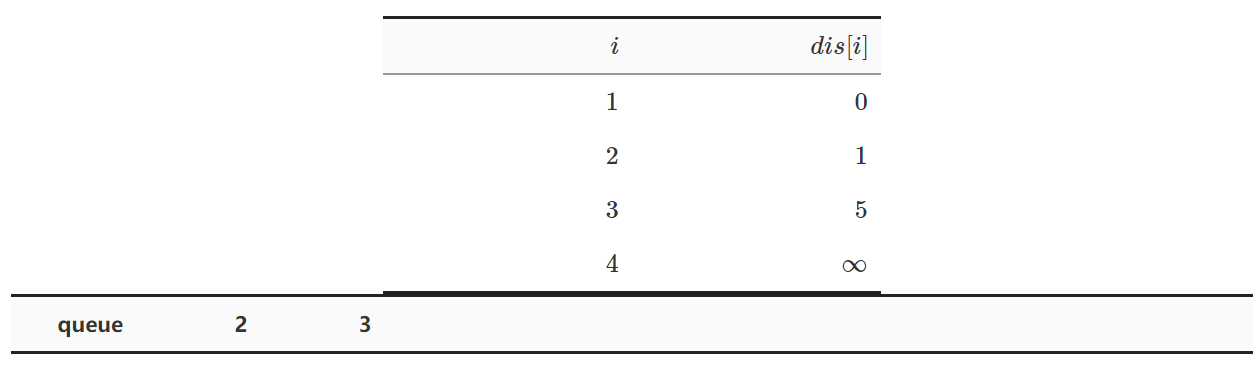
为什么是队列而不是其他的数据结构？因为队列具有“先进先出，后进后出”的特点，可以保证这一轮松弛的点不会在这一轮结束之前取出。



最开始，我们要把1号点放进队列：



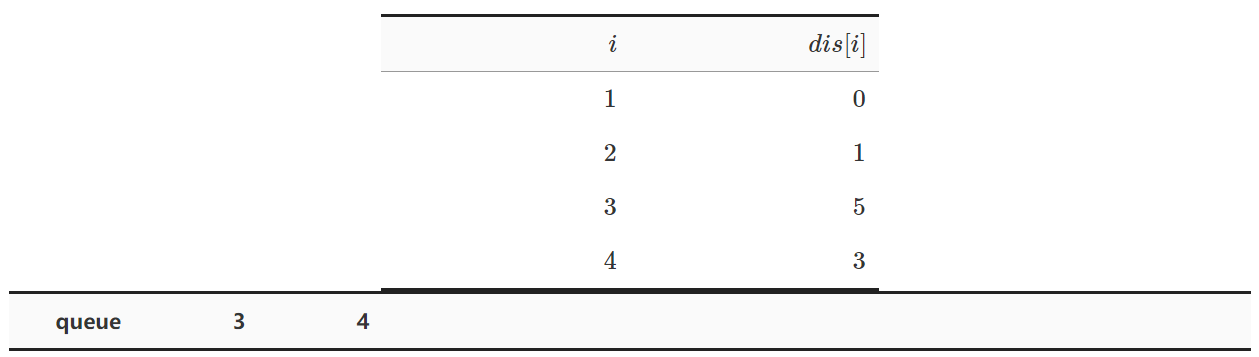
用1号点进行松弛（就是1号到1号再到目标点）：

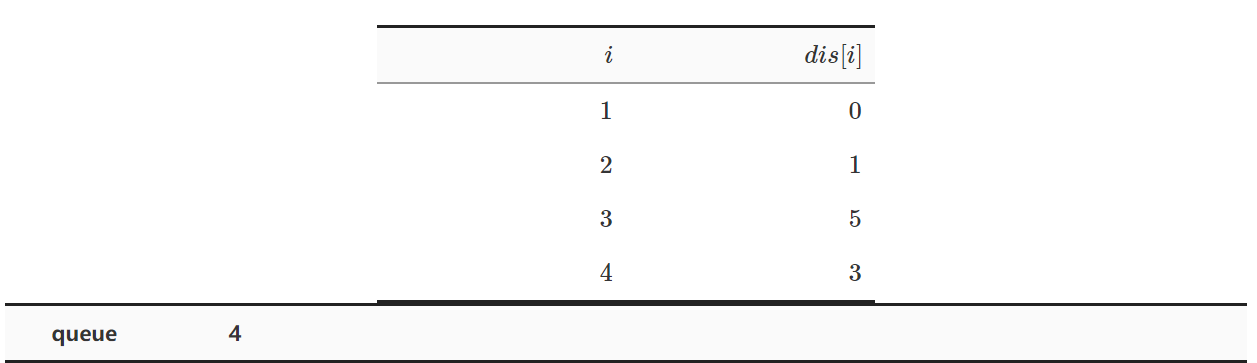


2,3号点被松弛成功了，把它们加入到队列里。

1号点被用过了，把它去掉

用2号点进行松弛：

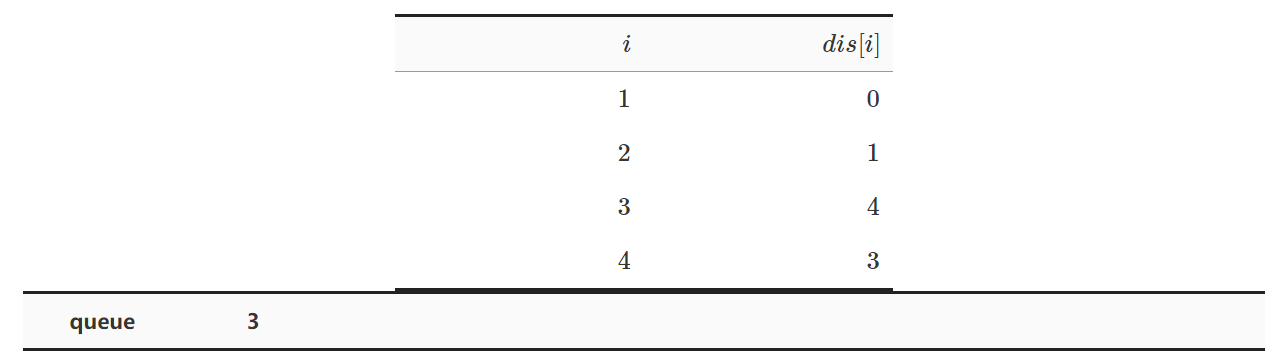




没有点被松弛成功。

3号点被用过了，把它去掉。

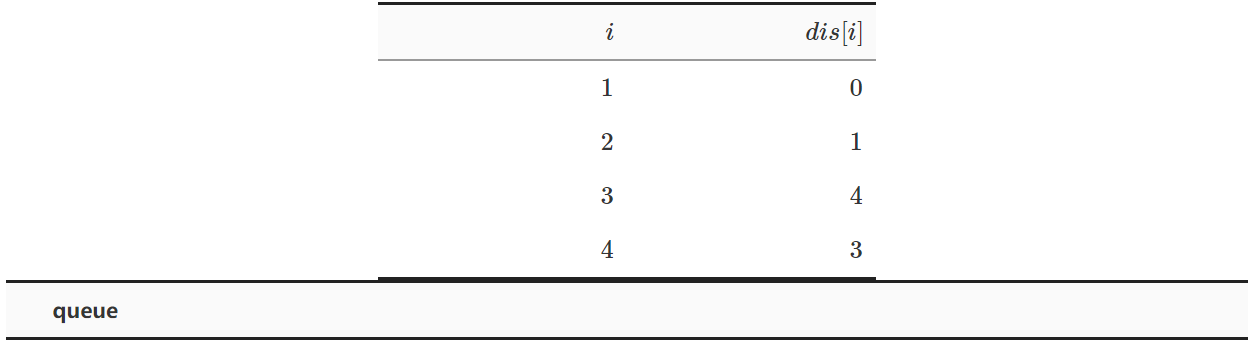
用4号点进行松弛：



3号点被松弛成功了，把它们加入到队列里。

4号点被用过了，把它去掉。

用3号点进行松弛：



没有点被松弛成功。

3号点被用过了，把它扔掉。

现在队列为空（也就是能松弛的都松弛了），算法结束。

小结

Spfa是Bellman-Ford算法的一种队列实现，为了减少了不必要的冗余计算，我们用一个队列来维护。初始时将起始点加入队列，每次从队列中取出队首元素，并对所有与他相邻的点进行松弛，并将松弛成功的顶点入队，直到队列为空时算法结束。

针对本例题的具体实现过程如下：

计算实例

首先建立起始点1到其余各点的最短路径表格，d[i]表示起点1到i点的最短路径值。



首先源点1入队，当队列非空时：队首元素1出队，对以1为起点的所有边的终点（2、3）进行松弛操作，此时路径表格状态为：

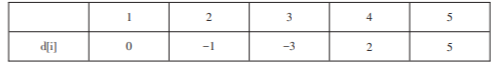


松弛以后，2和3两个顶点的最短路径估值变小，而这两个

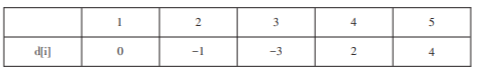
点队列中都没有，因此入队。

队首元素2出队，对以2为起点的所有边的终点（3，4，5）依

次进行松弛操作，此时路径表格状态为：



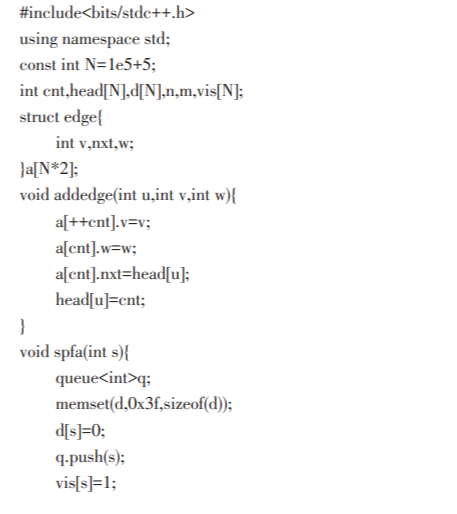
此时，3，4，5 三个顶点的最短路径估值变小，其中3这个点已经在队列中，不用入队，4，5 两个点入队。队首元素3出队，但是以3为起点没有出边，因此此时无松弛操作。队首元素4出队，对以4为起点的所有边的终点（5）进行松弛，此时路径表格状态为：

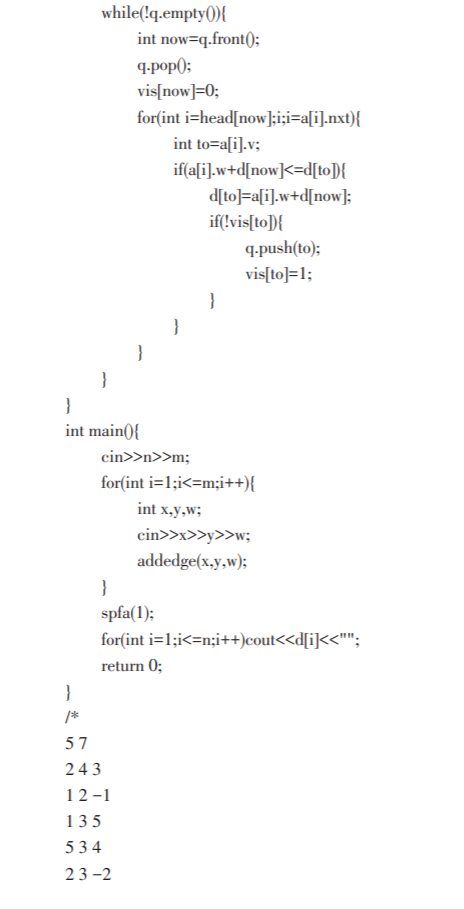


队首元素5出队，但是以5为起点没有出边，此时无松弛操

作，并且队列已空，算法结束，表中数据便是顶点1到其余各点

的最短路径值。







两种算法的联系和区别

Bellman-Ford 算法时间复杂度为 O (nm)，最多需要执行 n1 次循环，如果执行了n次循环，说明图中含有负圈

。Spfa算法是为了改进Bellman-Ford算法的效率而提出来的，在最优的情况下，每个节点只入队一次，这时退化为广度优先搜索，在最坏情况下，每个节点都入队 n-1 次，此时 Spfa 算法退化为 Bell⁃man-Ford 算法，时间复杂度为 O(nm)。如果某个节点的入队次数超过 n 次，说明图中肯定含有负圈。由于这两种算法均采用邻接表的方式存储边的信息，因此他们的空间复杂度均为 O(m)。它们都可以解决负权图问题，并能够判断图中是否含有负圈，都适用于稀疏图。由以上分析可以看出，Bellman-Ford 算法可解决的问题Spfa 算法也都适用。因此，奥赛解题中更常用 Spfa 算法。但是，Bellman-Ford算法思想精妙，是Spfa算法的前置知识，在学习中先理解 Bellman-Ford 算法更有利于对 Spfa 算法的理解和应用。

参考资料

实例解析Bellman-ford和Spfa算法 作者： 周鑫，张晶

Bellman-ford算法详解 CSDN作者：真的没事鸭

bellmanford算法 百度作者：侃侃阳光

最短路径3-Bellman-ford 及其优化SPFA 知乎作者：编程界的郭德纲