



最短路算法及其应用

广东北江中学 余远铭
yyming@hotmail.com



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。

一个在生活中常见的例子是：

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！

实际上，其中绝大多数路线我们是没必要考虑的。

这时候，我们应该用一种系统的方法来解决问题，而不是通常人们所用的凑的方法和凭经验的方法。

定义

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 $w:E \rightarrow R$ 为从边到实型权值的映射。路径 $P=(v_0, v_1, \dots, v_k)$ 的权是指其组成边的所有权值之和：

$$w(P) = \sum_{i=1}^k w(v_{i-1}, v_i)$$

定义 u 到 v 间最短路径的权为：

$$\delta(u, v) = \left\{ \begin{array}{ll} \min\{w(P): u \rightarrow v\} & \text{如果存在由 } u \text{ 到 } v \text{ 的通路} \\ \infty & \text{如果不存在} \end{array} \right\}$$

从结点 u 到结点 v 的最短路径定义为 $w(P) = \delta(u, v)$ 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。

边的权常被解释为一种度量方法，而不仅仅是距离。它们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。

重要性质

定理 1 (最优子结构) 给定有向加权图 $G=(V,E)$, 设 $P=<v_1, v_2, \dots, v_k>$ 为从结点 v_1 到结点 v_k 的一条最短路径, 对任意 i,j 有 $i \leq j \leq k$, 设 $P_{ij}=<v_i, v_{i+1}, \dots, v_j>$ 为从 v_i 到 v_j 的 P 的子路径, 则 P_{ij} 是从 v_i 到 v_j 的一条最短路径。

证明: 我们把路径 P 分解为 $<v_1, v_2, \dots, v_i, v_{i+1}, \dots, v_j, \dots, v_k>$ 。则 $w(P)=w(P_{1i})+w(P_{ij})+w(P_{jk})$ 。现在假设从 v_i 到 v_j 存在一路径 P'_{ij} , 且 $w(P'_{ij}) < w(P_{ij})$, 则将 P 中的路径 $P_{ij}=(v_i, v_{i+1}, \dots, v_j)$ 替换成 P'_{ij} , 依然是从 v_1 到 v_k 的一条路径, 且其权值 $w(P_{1i})+w(P'_{ij})+w(P_{jk})$ 小于 $w(P)$, 这与前提 P 是从 v_1 到 v_k 的最短路径矛盾。
(证毕)

推论

推论 1.1 给定有向加权图 $G=(V,E)$, 源点为 s , 则对于所有边 $(u,v) \in E$, 有:

$$\delta(s,v) \leq \delta(s,u) + w(u,v)$$

证明: 从源点 s 到结点 v 的最短路径 P 的权不大于从 s 到 v 的其它路径的权。特别地, 路径 P 的权也不大于某特定路径的权, 该特定路径为从 s 到 u 的最短路径加上边 (u,v) 。(证毕)

松弛技术

对每个结点 $v \in V$ ，我们设置一属性 $d[v]$ 来描述从源 s 到 v 的最短路径的权的上界，称之为最短路径估计。我们通过下面的过程对最短路径估计和先辈初始化。

INITIALIZE-SINGLE-SOURCE(G, s)

1. For 每个结点 $v \in V[G]$
2. Do $d[v] \leftarrow \infty$
3. $\pi[v] \leftarrow \text{NIL}$
4. $d[s] \leftarrow 0$

一次松弛操作可以减小最短路径的估计值 $d[v]$ 并更新 v 的先辈域 $\pi[v]$

RELAX(u, v, w)

1. If $d[v] > d[u] + w(u, v)$
2. Then $d[v] \leftarrow d[u] + w(u, v)$
3. $\pi[v] \leftarrow u$

常用算法

一、Dijkstra 算法

二、Bellman-Ford 算法

三、SPFA 算法

Dijkstra 算法

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 $v \in S$ ，有 $\delta[v] = \delta(s,v)$ 。算法反复挑选出其最短路径估计为最小的结点 $u \in V-S$ ，把 u 插入集合 S 中，并对离开 u 的所有边进行松弛。

```
Dijkstra(G,w,s)
1. INITIALIZE-SINGLE-SOURCE(G,S)
2.  $S \leftarrow \emptyset$ 
3.  $Q \leftarrow V[G]$ 
4. While  $Q \neq \emptyset$ 
5.   Do  $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6.      $S \leftarrow S \cup \{u\}$ 
7.     For 每个顶点  $v \in \text{Adj}[u]$ 
8.       Do RELAX( $u,v,w$ )
```

Dijkstra 算法

适用条件：所有边的权值非负

定理 2 每当结点 u 插入集合 S 时，有 $d[u] = d(s, u)$ 成立。

简证：我们每次选择在集合 $V-S$ 中具有最小最短路径估计的结点 u ，因为我们约定所有的边权值非负，所以有可能对结点 u 进行松弛操作的结点必不在集合 $V-S$ 中（否则与结点 u 的定义矛盾），因此只会在集合 S 中。又由于我们选取结点进入 S 时， S 中的结点已全部进行过松弛操作了，所以 $d[u]$ 的值不会再发生改变。因此 $d[u] = d(s, u)$ 。（证毕）

效率：

- ① 用一维数组来实现优先队列 Q ， $O(V^2)$ ，适用于中等规模的稠密图
- ② 二叉堆来实现优先队列 Q ， $O((E+V)\log V)$ ，适用于稀疏图
- ③ 用 Fibonacci 堆来实现优先队列 Q 的话， $O(V\log V)$ ，可惜编程复杂度过高，理论价值远大于实用价值

Bellman-Ford 算法

Bellman-Ford 算法运用了松弛技术，对每一结点 $v \in V$ ，逐步减小从源 s 到 v 的最短路径的估计值 $d[v]$ 直至其达到实际最短路径的权 $\delta(s,v)$ ，如果图中存在负权回路，算法将会报告最短路不存在。

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

Bellman-Ford 算法

适用条件：任意边权为实数的图

Bellman-Ford 算法的思想基于以下事实：“两点间如果有最短路，那么每个结点最多经过一次。也就是说，这条路不超过 $n-1$ 条边。”（如果一个结点经过了两次，那么我们走了一个圈。如果这个圈的权为正，显然不划算；如果是负圈，那么最短路不存在；如果是零圈，去掉不影响最优值）

根据最短路的最优子结构（定理 1），路径边数上限为 k 时的最短路由边数上限为 $k-1$ 时的最短路“加一条边”来求，而根据刚才的结论，最多只需要迭代 $n-1$ 次就可以求出最短路。

效率：

Bellman-Ford 算法的运行时间为 $O(VE)$ 。很多时候，我们的算法并不需要运行 $|V|-1$ 次就能得到最优值。对于一次完整的第 3-4 行操作，要是一个结点的最短路径估计值也没能更新，就可以退出了。

经过优化后，对于多数情况而言，程序的实际运行效率将远离 $O(VE)$ 而变为 $O(kE)$ ，其中 k 是一个比 $|V|$ 小很多的数。

SPFA 算法

我们用数组 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 。我们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 $u \leftarrow$ DLQUEUE(Q)
6. For 每条边 $(u, v) \in E[G]$
7. Do $tmp \leftarrow d[v]$
8. Relax(u, v, w)
9. If $(d[v] < tmp)$ and (v 不在 Q 中)
10. ENQUEUE(Q, v)

SPFA 算法

适用条件：任意边权为实数的图

定理 3 只要最短路径存在，上述 SPFA 算法必定能求出最小值。

证明：每次将点放入队尾，都是经过松弛操作达到的。换言之，每次的优化将会有某个点 v 的最短路径估计值 $d[v]$ 变小。所以算法的执行会使 d 越来越小。由于我们假定图中不存在负权回路，所以每个结点都有最短路径值。因此，算法不会无限执行下去，随着 d 值的逐渐变小，直到到达最短路径值时，算法结束，这时的最短路径估计值就是对应结点的最短路径值。（证毕）

定理 4 在平均情况下，SPFA 算法的期望时间复杂度为 $O(E)$ 。

证明：上述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 $\frac{E}{V}$ ，所以每处理一个点的复杂度为 $O(\frac{E}{V})$ 。假设结点入队次数为 h ，显然 h 随图的不同而不同。但它仅与边权值分布有关。我们设 $h=kV$ ，则算法 SPFA 的时间复杂度为 $O(\frac{h}{V} \times E) = O(kE)$ 。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 $O(E)$ 。（证毕）

小结

Dijkstra 算法的效率高，但是也有局限性，就是对于含负权的图无能为力。

Bellman-Ford 算法对于所有最短路长存在的图都适用，但是效率常常不尽人意。

SPFA 算法可以说是综合了上述两者的优点。它的效率同样很不错，而且对于最短路长存在的图都适用，无论是否存在负权。它的编程复杂度也很低，是性价比高的算法。

算法	时间复杂度	空间复杂度	编程复杂度	适用范围
Dijkstra	$O(V^3)$ 或 $O((E+V)\log V)$	$O(V^2)$ 或 $O(E+V)$	简单或相对复杂	不含负权的图 (窄)
Bellman-Ford	$O(VE)$	$O(E+V)$	简单	实数图 (广)
SPFA	$O(E)$	$O(E+V)$	简单	实数图 (广)

我们应该根据实际需要，找到时空复杂度和编程复杂度的平衡点，在考场上用最少的拿尽可能多的分数。

例一 双调路径 (BOI2002)

题意简述:

- 如今的道路密度越来越大，收费也越来越多，因此选择最佳路径是很现实的问题。城市的道路是双向的，每条道路有固定的旅行时间以及需要支付的费用。路径由连续的道路组成。总时间是各条道路旅行时间的和，总费用是各条道路所支付费用的总和。同样的出发地和目的地，如果路径 **A** 比路径 **B** 所需时间少且费用低，那么我们说路径 **A** 比路径 **B** 好。对于某条路径，如果没有其他路径比它好，那么该路径被称为最优双调路径。这样的路径可能不止一条，或者说根本不存在。
- 给出城市交通网的描述信息，起始点和终点城市，求最优双条路径的条数。城市不超过 100 个，边数不超过 300，每条边上的费用和时间都不超过 100。

例一 双调路径 (BOI2002)

分析：

- 这道题棘手的地方在于标号已经不是一维，而是二维，因此不再有全序关系。我们可以采用拆点法，让 $d[i,c]$ 表示从 s 到 i 费用为 c 时的最短时间。

算法一：

标号设定算法是根据拓扑顺序不断把临时标号变为永久标号的。

在这题中，其实，我们并不需要严格的拓扑顺序，而只需要一个让标号永久化的理由。拓扑顺序能保证标号的永久化，但是还有其他方式。在本题中，标号永久化的条件是：从其他永久标号得不到费用不大于 c 且时间不大于 t 的临时标号（这里利用了费用和时间的非负性），即：所有的“极小临时标号”都可以永久化。这样，一个附加的好处是一次把多个临时标号同时变成永久的。

例一 双调路径 (BOI2002)

假设时间上限为 t ，费用上限为 c ，城市数为 n ，边数为 m ，则每个点上的标号不超过 $O(nc)$ 个，标号总数为 $O(n*n*c)$ 个，每条边考虑 $O(nc)$ 次。如果不同顶点在同一费用的临时标号用堆来维护，不同费用的堆又组成一个堆的话，那么建立（或更新）临时标号的时间为 $O(mnc\log n\log c)$ ，总的时间复杂度为 $O(nnc+mnc\log n\log c)$ ，本题的规模是完全可以承受的。实际上由于标号的次数往往远小于 nnc ，程序效率是相当理想的

算法二：

在本题中构图直接用 SPFA 算法。

在最坏情况下，费用最大值 c 为 $100 * 100 = 10000$ ，那么每个点将被拆成 10000 个点，由于原图边数不超过 300，所以我们构造的新图中边数不会超过 3000000。

因此算法的时间复杂度是 $O(k*3000000)$ 。写出来的程序运行的实际效果非常好，每个数据的速度都比官方参考程序（算法一）快，有几个甚至比官方程序快 3~4 倍！完全通过这题简直是轻而易举。这和我们时间复杂度在理论上的分析是一致的。

例一 双调路径 (BOI2002)

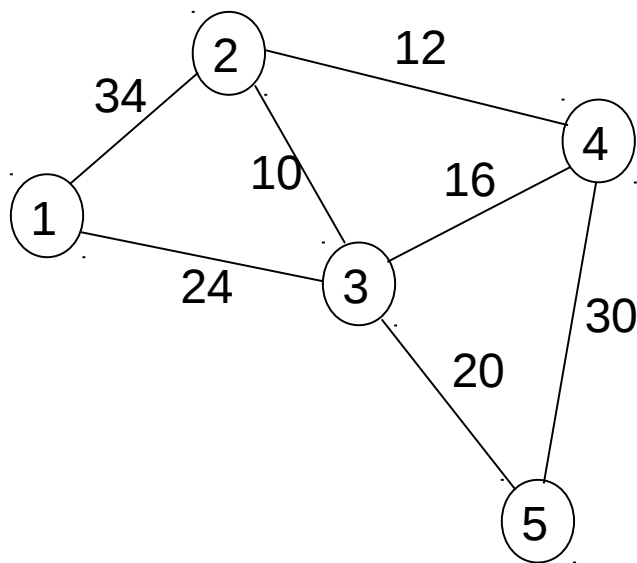
- 两个算法在空间上是同阶的，一样是 $O(E)$ 。虽然算法一仅用到二叉堆，并不是特别复杂，但是因为要用两个堆，建立更新删除写起来还是有一定的工作量的。**SPFA** 算法写起来极其简单，效率又高，而且适用范围广（可以处理含有负权的图），在很多情况下，是最短路问题上好的选择。

例二 网络提速（经典问题）

题意简述：

- 某学校的校园网由 $n(1 \leq n \leq 50)$ 台计算机组成，计算机之间由网线相连，其中顶点代表计算机，边代表网线。不同网线的传输能力不尽相同。
- 现学校购买了 $m(1 \leq m \leq 10)$ 台加速设备，每台设备可作用于一条网线，使网线上传输信息用时减半。多台设备可用于同一条网线，其效果叠加。
- 如何合理使用这些设备，使计算机 1 到计算机 n 传输用时最少，这个问题急需解决。校方请你编程解决这个问题。

例二 网络提速（经典问题）



如图，若 $n=5$ ， $m=2$ ，则将两台设备分别用于 1-3，3-5 的线路，传输用时可减少为 22 秒，这是最佳解。

例二 网络提速（经典问题）

分析：

🕒 让我们重新描述一下问题：

给定含有 n 个顶点的带权无向图，一共可以进行 m 次操作，每次操作将一条边的权值除以 2。问每次应该对哪条边进行操作，使得 1 到 n 的最短路径权和最小。

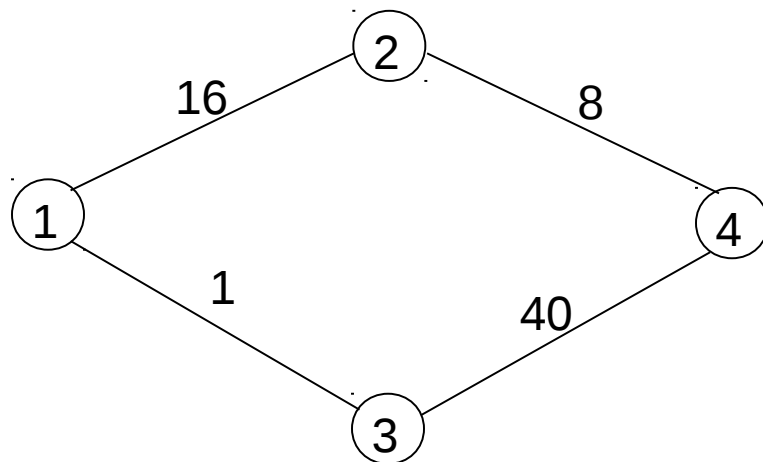
如果我们把 **Dijkstra** 算法直接用在原图上，得到的是没有使用任何加速设备顶点 1 到顶点 n 的最短路长，不是我们想要的结果。

能否用增量法做这题呢？就是，我们先求出使用前 $m-1$ 台加速设备的最短路长，然后通过枚举之类算出第 m 台设备用在那条边上，行不行呢？

经过简单的举例后发现行不通。

例二 网络提速（经典问题）

一个明显的反例是：



如果我们只有一个加速设备的话，显然将它加在 **1-2** 上，那么最短路长 **16** 是最佳的。

但是，我们有两个的话，将两个都加在 **3-4** 上，则最短路长 **11** 是最优的。

所以要是我们的加速设备增多一台的话，可能导致前面的所有放置方案都不是最优值。

例二 网络提速（经典问题）

我们注意到一点，就是 n 和 m 都很小，特别是 m ，最大只有 10。直觉和经验告诉我们，应该从数据规模小上面作文章。

从简单情形入手往往是解题的捷径。

没有 m 值，或者说 $m=0$ 时，问题的解就是最简单的最短路径问题。 m 值的出现导致了最短路算法的失败。

关键是我们不知道应该在哪儿几条边用加速设备，而且每条边用多少次也不知道。方案与权值的分布还有 m 值的大小都有莫大的关联。

我们想想，有无 m 值的差异在哪，能够消除吗？

可以的。

例二 网络提速（经典问题）

构造图 $G=(V,E)$ 。设原图 $V_0 = \{v_1, v_2, \dots, v_n\}$ ，将原图中的每个顶点 v_i 拆成 $m+1$ 个顶点 $v_{i,0}, v_{i,1}, v_{i,2}, \dots, v_{i,m}$ ，构造 V 使得：

$$V = \{v_{i,0}, v_{i,1}, v_{i,2}, \dots, v_{i,m} \mid v_i \in V_0\}$$

对于原图的每一条边，注意是无向的 $e_k = v_i v_j \in E_0$ ，将它拆成 $(m+1)(m+2)$ 条有向边：

$$(e_k)_{0,0} = v_{i,0} v_{j,0} \quad (e_k)_{0,1} = v_{i,0} v_{j,1} \quad \dots \quad (e_k)_{m,m} = v_{i,m} v_{j,m}$$

$$(e_k)'_{0,0} = v_{j,0} v_{i,0} \quad (e_k)'_{0,1} = v_{j,0} v_{i,1} \quad \dots \quad (e_k)'_{m,m} = v_{j,m} v_{i,m}$$

构造 E 使得：

$$E = \{(e_k)_{p,q} = v_{i,p} v_{j,q}, (e_k)'_{p,q} = v_{j,p} v_{i,q} \mid e_k = v_i v_j \in E_0, 0 \leq p \leq q \leq m\}$$

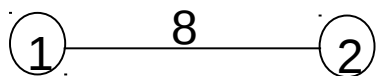
最后设定权函数 w 。对 $v_{i,p} v_{j,q} \in E$ ， w 满足：

$$w(v_{i,p} v_{j,q}) = w(v_i, v_j) \times 2^{q-p}$$

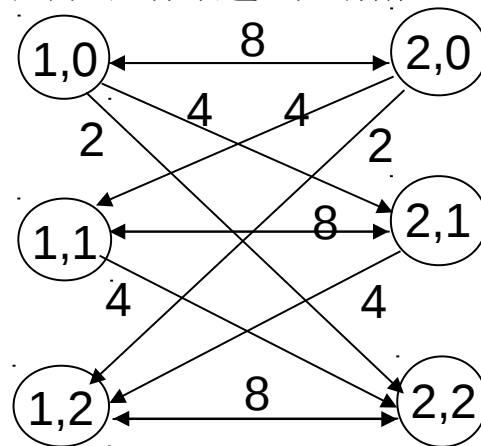
例二 网络提速（经典问题）

解释一下图 G 的含义。

为了更清楚地说明问题，我们举出一个例子。下图 (a) 显示了某个 $G_0(n=2, m=2)$ ，按照以上的构造方法得出 (b) 中的 G 。将图 G 分成三层，第 0 层由顶点 $v_{1,0}$ ， $v_{2,0}$ 构成，第 1 层由顶点 $v_{1,1}$ ， $v_{2,1}$ 构成，第 2 层由顶点 $v_{1,2}$ ， $v_{2,2}$ 构成。可以看出，若两个顶点间有边相连，两个顶点在同一层的，则顶点之间是互连的，若不在同一层，则层号小的顶点为边的起始点，层号大的为边的终点。



(a)



(b)

例二 网络提速（经典问题）

层号代表已用的加速设备台数，例如从 $v_{1,0}$ 到 $v_{2,1}$ 需要且恰好要用一台加速设备。我们无法从层号大的顶点到达层号小的顶点，这符合同一个设备不能使用多次的规定。

顶点 $v_{1,0}$ 到 $v_{n,m}$ 的最短路长就是添置 m 台加速设备后计算机 **1** 到计算机 **n** 的最少传输时间。

剩下的工作就是对图 G 使用 **Dijkstra** 算法求 $v_{1,0}$ 到 $v_{n,m}$ 的最短路长。相信大家都会，我就不多说了。

结语

- 我们从理论上研究问题的最终目的是更好地指导实践。
- 在题目中，往往不会直接告诉你，什么元素应该作为结点，什么元素应该作为边，应该如何构图。这就需要你根据题目的自身的特点，借鉴以往的解题经验，运用所学的相关知识，抽象出合适的模型，利用效率已有公论的算法高效求解。
- 实际中遇到的题目可能千奇百怪，模型的转化千变万化，从而导致解法也多种多样，不拘一格，本文实在难以涵盖万一，只要能对读者有所启发，那么我的目的也就达到了。

最后送上两句话：

万变不离其宗。

阵而后战，兵法之常；运用之妙，存乎一心。

把握最短路的相关性质，
合理转化。

因地制宜，根据实际
选择最合适的算法。

谢谢大家