最短路算法及其应用

广东北江中学　余远铭

【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。本文较详尽地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，最后作出总结。

【关键字】

最短路

【目录】

【正文】

一、基本概念

1．1 定义

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:E→R 为从边到实型权值的映射。路径 P=(v_0, v_1,....., 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

$$\delta(u,v)=\begin{cases}\min\{w(p):u\to v\} & \text{如果存在由}u\text{到}v\text{通路}\\ \infty & \text{如果不存在}\end{cases}$$

从结点 u 到结点 v 的**最短路径**定义为权 $w(p)=\delta(u,v)$ 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。

1．2 简单变体

单目标最短路径问题： 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，我们就可以把这一问题转化为单源最短路径问题。

单对结点间的最短路径问题：对于某给定结点 u 和 v，找出从 u 到 v 的一条最短路径。如果我们解决了源结点为 u 的单源问题，则这一问题也就获得了解决。对于该问题的最坏情况，从渐进意义上看，目前还未发现比最好的单源算法更快的方法。

每对结点间的最短路径问题：对于每对结点 u 和 v，找出从 u 到 v 的最短路径。我们可以用单源算法对每个结点作为源点运行一次就可以解决问题。

1．3 负权边

在某些单源最短路问题中，可能存在权为负的边。如果图 G(V,E)不包含由源 s 可达的负权回路，则对所有 $v \in V_s$，最短路径的权的定义 $\delta(s,v)$ 依然正确。即使它是一个负值也是如此。但如果存在一从 s 可达的负权回路，最短路径的定义就不能成立了。从 s 到该回路上的结点不存在最短路径——因为我们总可以顺着找出的“最短”路径再穿过负权回路从而获得一权值更小的路径，因此如果从 s 到 v 的某路径中存在一负权回路，我们定义 $\delta(s,v) = -\infty$ 。

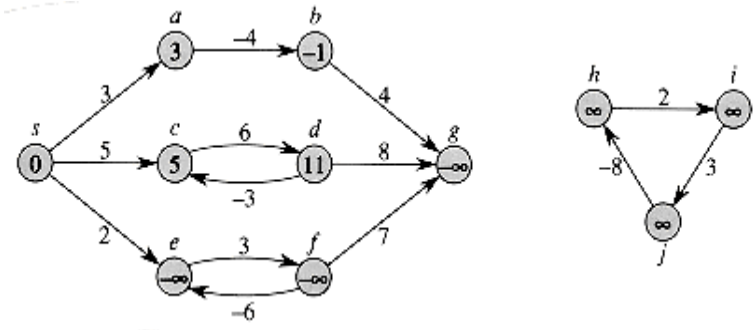


图 1　含有负权和负权回路的图

图 1 说明负的权值对最短路径的权的影响。每个结点内的数字是从源点 s 到该结点的最短路径的权。因为从 s 到 a 只存在一条路径(路径<s,a>)，所以：$\delta(s,a) = w(s,a) = 3$ 。

类似地，从 s 到 b 也只有一条通路，所以：

$\delta(s,b) = w(s,a) + w(a,b) = 3 + (-4) = -1$ 。

从 s 到 c 则存在无数条路径：<*s,c*>,<*s,c,d,c*>,<*s,c,d,c,c,d,c*>等等。因为回路<*c,d,c*>的权为 6+(-3)=3>0，所以从 *s* 到 *c* 的最短路径为<*s,c*>，其权为：

$\delta(s,c)=5$。

类似地，从 s 到 d 的最短路径为<*s,c,d*>，其权为：

$\delta(s,d)=w(s,c)+w(c,d)=11$。

同样，从 s 到 e 存在无数条路径：<*s,e*>,<*s,e,f,e*>,<*s,e,f,e,f,e*>等等.由于回路<*e,f,e*>的权为 3+(-6)=-3<0，所以从 s 到 e 没有最短路径。只要穿越负权回路任意次，我们就可以发现从 s 到 e 的路径可以有任意小的负权值，所以：

$\delta(s,e)=-\infty$

类似地，

$\delta(s,f)=-\infty$

因为 *g* 是从 *f* 可达的结点，我们从 *s* 到 *g* 的路径可以有任意小的负权值，则：

$\delta(s,g)=-\infty$。

结点 h，j，i 也形成一权值为负的回路，但因为它们从 s 不可达，因此

$\delta(s,h)=\delta(s,i)=\delta(s,j)=\infty$。

一些最短路径的算法，例如 *Dijkstra* 算法，都假定输入图中所有边的权取非负数，如公路地图实例。另外一些最短路算法，如 *Bellman-Ford* 算法，允许输入图中存在权为负的边，只要不存在从源点可达的权为负的回路，这些算法都能给出正确的解答。特定地说，如果存在这样一个权为负的回路，这些算法可以检测出这种回路的存在。

1．4 重要性质及松弛技术

本文的算法所运用的主要技术是松弛技术，它反复减小每个结点的实际最短路径的权的上限，直到该上限等于最短路径的权。让我们看看如何运用松

弛技术并正式证明它的一些特性。

定理 1 (最优子结构) 给定有向加权图 G=(V,E)，设 P=<v_1, v_2,…, 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<=j<=k，设 P_{ij}=< v_i, v_{i+1}, …, 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,…,v_i,v_{i+1},…v_j,…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},…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）

下面看定理 1 的一个推论，它给出了最短路径的一个简单而实用的性质：

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)⊂E，有 $\delta(s,v) \le \delta(s,u) + w(u,v)$

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）

下面介绍松弛技术。

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∈ V[G]
2.      Do d[v]← ∞
3.          π[v]← NIL
4.  d[s]← 0
```

经过初始化以后,对所有 $v \in V$，π[v]=NIL，对 $v=s$，d[v]=0，对 $v \in V-\{s\}$，d[v]= ∞ 。

松弛一条边*(u,v)*的过程包括测试我们是否可能通过结点 *u* 对目前找出的到 v 的最短路径进行改进，如果可能则更新 d[v]和π[v]，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π[v]，下面的代码实现了对边*(u,v)*的进一步松弛操作。

```
RELAX(u,v,w)
1.   If d[v]>d[u]+w(u,v)
2.       Then d[v]←d[u]+w(u,v)
3.            π[v]←u
```

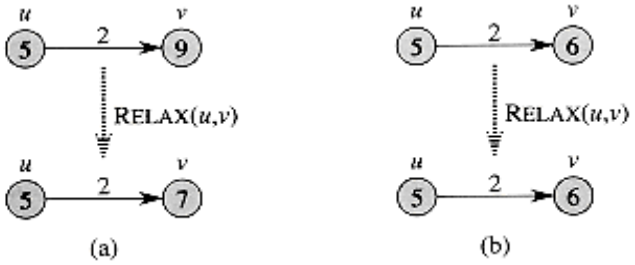


图 2 对边(u,v)进行松弛

图 2 说明了松弛一条边的两个实例，在其中一个例子中最短路径估计减小，而在另一实例中最短路径估计不变。(a)因为在进行松弛以前 d[v]>d[u]+w[u,v]，所以 d[v]的值减小。(b)因为松弛前 d[v]<=d[u]+w[u,v]，所以松弛不改变 d[v]得值。

下文介绍的每个算法都调用 INITIALIZE-SINGLE-SOURCE，然后重复对边进行松弛的过程 RELAX。区别在于对每条边进行松弛操作的次数以及对边执行操作的次序有所不同。需要指出的是，松弛是改变最短路径估计和先辈的唯一方式。

二、常用算法

这一节着重讨论两种常用算法：*Dijkstra* 算法和 *Bellman-Ford* 算法。虽然它们都是建立在松弛技术基础上的算法，但是在实现上有着各自的特点，适用的范围也有所不同。

另外，我们还将介绍一种期望复杂度与边数同阶的高效算法——SPFA 算

法，并对其复杂度作出简要的分析。

2．1 *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*(u,v)*∈ E，w(u,v)>=0。

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*∈ *S*，有 d[v]=δ (s,v)。算法反复挑选出其最短路径估计为最小的结点 *u*∈ *V-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 由临接表表示。

```
Dijkstra(G,w,s)
1.  INITIALIZE-SINGLE-SOURCE(G,S)
2.  S←∅
3.  Q←V[G]
4.  While Q≠∅
5.     Do  u←EXTRACT-MIN(Q)
6.         S←S U{u}
7.         For 每个顶点 v∈ Adj[u]
8.            Do  RELAX(u,v,w)
```

Dijkstra 算法如图 3 所示对边进行松弛操作，最左结点为源结点，每个结点内为其最短路径估计。

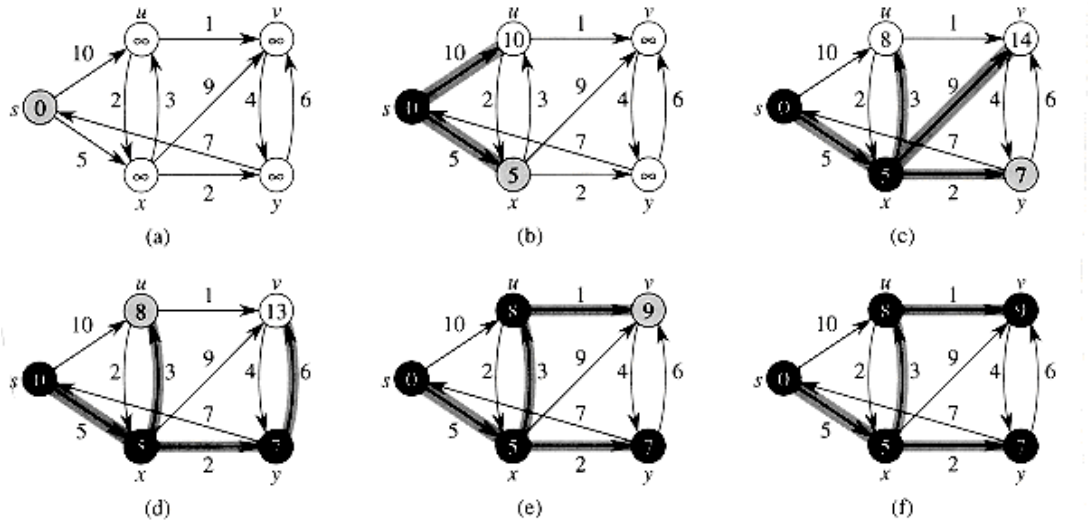


图 3 Dijkstra 算法的执行流程

阴影覆盖的边说明了前驱的值：如果边*(u,v)*为阴影所覆盖，则π[v]=*u*。黑色结点属于集合 S，白色结点属于优先队列 Q=V-S。第 1 行对 d 和π值进行通常的初始化工作。第 2 行置集合 S 为空集。第 3 行对优先队列 Q 进行初始化使其包含 V-S=V-∅=V 中的所有结点。每次执行第 4-8 行的 While 循环时，均从队列 Q=V-S 中压出一结点 *u* 并插入到集合 *S* 中(第一次循环时 *u*=*s*)。因此在 V-S 中的所有结点中，结点 *u* 具有最小的最短路径估计。然后，第 7-8 行对以 *u* 为起点的每条边*(u,v)*进行松弛。如果可以经过 *u* 来改进到结点 v 的最短路径,就对估计值 d[v]以及先辈π[v]进行更新。注意在第 3 行以后就不会再插入结点到 Q 中，并且每个结点只能从 Q 弹出并插入 S 一次，因此第 4 至 8 行的 While 循环的迭代次数为|V|次。

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。但 *Dijkstra* 算法确实计算出了最短路径，关键是要证明：

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

我们最关注的问题是：*Dijkstra* 算法的执行速度如何？首先我们考虑用一维数组来实现优先队列 Q=V-S 的情形。在该算法的实现下，每次 EXTRACT-MIN 操作运行时间为 O(V)，存在|V|次这样的操作，所以 EXTRACT-MIN 的全部运行时间为 $O(V^2)$。因为每个结点 $v\in V$ 仅被插入集合 S 一次，所以在算法的执行过程中邻接边 Adj[v]中的每条边在第 4-8 行的 For 循环中仅被考察一次。因为在所有邻接表中边的总数为|E|，所以在该 For 循环中总共存在|E|次迭代，每次迭代运行时间为 O(1)，因此整个算法的运行时间为 $O(V^2+E)=O(V^2)$。

但在稀疏图的情形下，用二叉堆来实现优先队列 Q 是比较实用的。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 $d[v]\leftarrow d[u]+w(u,v)$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)。通常情况下，边数|E|都不小于结点数|V|，所以运行时间又可以简化为 O(ElgV)。

用 *Fibonacci* 堆来实现优先队列 Q 的话，可以将运行时间改为 O(VlgV+E)。|V|次 EXTRACT-MIN 操作中每次的平摊代价为 O(lgV)，且|E|次 RELAX 中调整结点 v 位置每次的平摊时间仅为 O(1)。

事实上，*Fibonacci* 堆的实现相当繁琐，在竞赛有限的时间里基本上不大

可能写出来并调试好。另外，*Fibonacci* 堆算法中隐含的系数相当大，程序的实际运行效果并不理想。目前该堆的理论价值远大于实用价值。因此，我们不推荐在分秒必争的竞赛中使用。

顺便说一句，*Dijkstra* 算法和宽度优先搜索以及计算最小生成树的 *Prim* 算法都有类似之处。

2．2 *Bellman-Ford* 算法

Bellman-Ford 算法能在更一般的情况下解决单源点最短路径问题，在该算法下边的权可以为负。正如 *Dijkstra* 算法一样，*Bellman-Ford* 算法运用了松弛技术，对每一结点 v∈ 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 δ(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←1 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
```

源结点为 z。每个结点内为该结点的 d 值，阴影覆盖的边说明了 π 值。在该实例中，*Bellman-Ford* 算法返回 TRUE。在进行了通常的初始化后，算法对图的边执行|V|-1 次操作。每次均为第 2-4 行 For 循环的一次迭代，在迭代过程中对图的每条边松弛一次，图 4(b)-(c)说明了全部四次操作的每一次后算法的状态,在进行完|V|-1 次操作后,算法 5-8 行检查是否存在负权的回路并返回正确的布尔值。

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)。

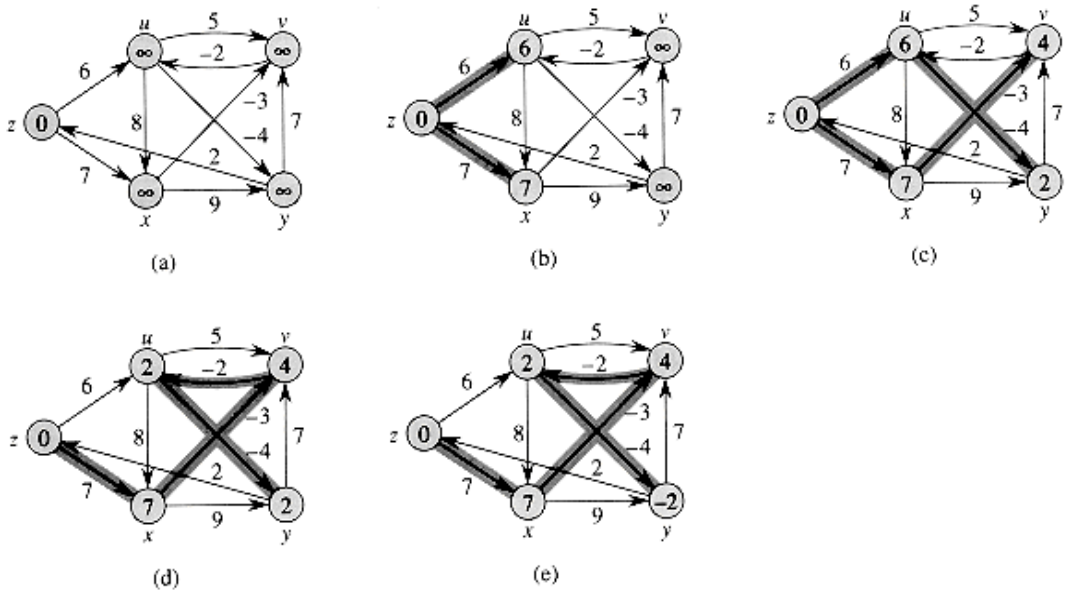


图 4　Bellman-Ford 算法的执行流程

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 时的最短路可以由边数上限为 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。

这里介绍一个优化。很多时候，我们的算法并不需要运行|V|-1 次就能得到最优值了，这之后的运行就是浪费时间。因此，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（因为既然当前这次都不能更新，下一次肯定也不能）。这个简单而显然的优化能大大提高程序的运行速度。虽然优化后的最坏情况依然是 O(VE)，但是对于多数情况而言，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。

目前，对 *Bellman-Ford* 的进一步优化尚在不断研究中。

2．3 *SPFA* 算法

求单源最短路的 *SPFA* 算法的全称是：*Shortest Path Faster Algorithm*。

从名字我们就可以看出，这种算法在效率上一定有过人之处。

很多时候，给定的图存在负权边，这时类似 *Dijkstra* 等算法便没有了用武之地，而 *Bellman-Ford* 算法的复杂度又过高，*SPFA* 算法便派上用场了。

简洁起见，我们约定有向加权图 G 不存在负权回路，即最短路径一定存在。当然，我们可以在执行该算法前做一次拓扑排序，以判断是否存在负权回路，但这不是我们讨论的重点。

和上文一样，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。

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）

```
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←DLQUEUE(Q)
6.          For 每条边(u,v)∈E[G]
7.            Do tmp←d[v]
8.                 Relax(u,v,w)
9.                 If (d[v] < tmp) and (v 不在 Q 中)
10.                   ENQUEUE(Q,v)
```

刚才我们只是笼统地说 *SPFA* 算法在效率上有过人之处，那么到底它的复杂度是怎样的？

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 $T = O\left(h \times \frac{E}{V} \right) = O\left(\frac{h}{V} \times E \right) = O(kE)$。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）

聪明的读者一定发现了，*SPFA* 和经过简单优化的 *Bellman-Ford* 无论在思想上还是在复杂度上都有相似之处。确实如此。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。

三、应用举例

我们从理论上研究问题的最终目的是更好地指导实践。

首先研究模型的建立，简单说，就是构图。

在实际中，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，而在本文中，特指最短路算法。

3．1 例题 1——货币兑换[1]

题意简述：

若干个货币兑换点在我们的城市中工作着。每个兑换点只能进行两种指定货币的兑换。不同兑换点兑换的货币有可能相同。每个兑换点有它自己的兑换汇率，货币 A 到货币 B 的汇率表示要多少单位的货币 B 才能兑换到一个单位的货币 A。当然货币兑换要支付一定量的中转费。例如，如果你想将 100 美元兑换成俄元，汇率是 29.75，中转费为 0.39，那么你会兑换到 (100-0.39)×29.75=2963.3975 俄元。

城市中流通着 N(N<=100)类货币，用数字 1 至 N 标号表示。每个货币兑换点用 6 个数字来描述：整数 A 和 B 是兑换货币的编号，实数

[1]Ural State University Problem Archive 1162

RAB，CAB，RBA，CBA 分别是 A 兑换成 B 和 B 兑换成 A 的汇率和中转费用。

Nick 有一些第 S 类货币，他想在若干次交换后增加他的资金，当然这些资金最终仍是第 S 类货币。请你告诉他该想法能否实现。

分析：

如果我们可以求出，通过兑换每种货币可以得到的最大值，那么问题便迎刃而解了。

因为要是可以得到的第 S 类货币最大值都不比初值大，资金肯定无法增加；否则，得到最大值的过程就是一种解法。

从问题本身来看，求最大值也是与其特性相符的。

因为：根据兑换的公式可以看出，兑换前的货币越多，那么兑换后得到的货币也会越多。就是说，原货币越多越好。

到了这里，我们发现求最大值和我们学过的求最短路很类似，构图用最短路算法做也显得水到渠成了。

具体做法是：

将 N 种货币看成 N 个结点，将每个兑换点转化为两条有向边。根据兑换公式，目前从 A 货币兑换到 B 货币的汇率和中转费用为 RAB，CAB，那么由对应的 A 结点向 B 结点连一条有向边，从 A 点得到的 B 的可能最大值为：(A 目前的最大值-CAB)×RAB。

注意，这里所求的是最大值，为了转化为最短路，我们可以在数字前面加上一个负号。

更简洁的方法是利用求最短路的方法，求最长路。

由于存在负权（求最长路和负权等价），所以在这里 Dijkstra 算法不适用了，我们可以用 Bellman-Ford 算法或者 SPFA 算法做，这里用 Bellman-Ford 就够了。

需要指出一点，无论是求负权最短路还是求最长路，可能遇到的一个问题是：存在环，从而导致货币最大值不存在（因为可以沿环使最大值不断增大）。

解决的办法其实很简单，我们只需要设立一个停止条件就行了。

如果当前没有点的最优值可以更新，又或者第 S 个点的最优值已经优于初始值，就可以退出循环了。

在细节上还有两点要注意。一是两点间的边可能不止一条，所以存图不能用邻接矩阵（从效率的角度也不提倡用），而应该用邻接表；二是涉及比较实

数的大小，判断 A<B 应该写成 A+Zero<B，判断 A<=B 应该写成 A-Zero<B，其中 Zero 是一个很小的数，具体值要根据实际情况设定，本题可以设为 1e-8。

至此，本题解决。

小结：

想到求最大值是个人思维能力的体现；本题能够用求最大值解，却取决于题目本身的性质；采用最短路算法求最大值，由最短路模型的适用范围决定；对于细节的处理，则要靠以往的解题经验和平时的积累。

有时候，已经知道了应该用最短路做，但是选择不同的实现方式，时空效率有较大的差别，编程复杂度也有所不同，我们应该选择在时空复杂度可以接受的情况下，编程复杂度尽可能低的方式。就是说，要找到时空复杂度和编程复杂度的平衡点，写出性价比高的程序。

3．2 例题 2——双调路径[2]

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。

分析：

本题显然是一道求最短路径的题目，不过和一般的问题也有所不同。

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，初看似乎不能用最短路模型做。我们可以采用拆点法（这招在网络流问题上也很有用，比如点流量有限制的问题），让 d[i,c]表示从 s 到 i 费用为 c 时的最短时间。

现在已经可以运用最短路算法做了。

我们面前有两条路：一是用标号设定算法，Dijkstra；二是用标号修正算法，Bellman-Ford 或 SPFA。

[2]Baltic Olympiad in Informatics 2002

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。

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 $O(n^2c)$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(mnclognlogc)，总的时间复杂度为 $O(n^2c+mnclognlogc)$，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 n^2c，程序效率是相当理想的。

算法二：

我们考虑在本题中运用 SPFA 算法的话，情况会是怎样？

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。

根据我们之前的讨论可以知道，SPFA 算法的平均时间复杂度是 O(E)的。但是最坏情况下，复杂度可以达到 O(VE)，对于本题而言，$V=10^6$，$E=3*10^6$，总复杂度将达到 $O(3*10^{12})$之巨！

实际上，即使是用 Bellman-Ford 算法，$O(3*10^{12})$的情况也极少出现（要做出这种数据真的不容易），更何况是 SPFA 了。

所以算法的复杂度应该是 O(kE)，其中的 k 将会比 V 小很多。

还有一个好消息是：刚才对于时间复杂度的估计是基于一般图的，而本题非常特别。首先，费用的最大值，上限 10000 是很难达到的，它要求经过所有

点，并且每条边都是最大的费用 100，一般来说，实际上限会比 10000 小很多。还有，图的构造也很特别，因为它的边是由最多只有 300 条边的原图拆出来的，每个结点拆出的所有点之间是相互独立的。

辅助队列应该采用循环队列，这样空间就不会浪费了。

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。这和我们对时间复杂度在理论上的分析是一致的。

在空间上和算法一是同阶的，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。

建模和选择合适的实现方式是最短路的两个基础问题。

仅仅满足于此是远远不够的。

下面给出两道相对复杂的题目。它们的模型比较隐蔽，要求把题目抽象化，得出些本质的东西。考察的知识面也比较广泛，综合性较强。这里给出笔者思考解答的过程，力求做到：思路清晰，证明严谨，程序简洁高效。希望能抛砖引玉，对读者有所启发。

3．3 例题 3——Layout[3]

题意简述：

当排队等候喂食时，奶牛喜欢和它们的朋友站得靠近些。FJ 有 N（2<=N<=1000）头奶牛，编号从 1 到 N，沿一条直线站着等候喂食。奶牛排在队伍中的顺序和它们的编号是相同的。因为奶牛相当苗条，所以可能有两头或者更多奶牛站在同一位置上。即使说，如果我们想象奶牛是站在一条数轴上的话，允许有两头或更多奶牛拥有相同的横坐标。

一些奶牛相互间存有好感，它们希望两者之间的距离不超过一个给定的数 L。另一方面，一些奶牛相互间非常反感，它们希望两者间的距离不小于一个给定的数 D。给出 ML 条关于两头奶牛间有好感的描述，再给出 MD 条关于两头奶牛间存有反感的描述。（1<=ML,MD<=10000，1<=L,D<=1000000）

[3]Usaco Month Contest December 2005 Gold

你的工作是：如果不存在满足要求的方案，输出-1；如果 1 号奶牛和 N 号奶牛间的距离可以任意大，输出-2；否则，计算出在满足所有要求的情况下，1 号奶牛和 N 号奶牛间可能的最大距离。

分析：

如果当前问题比较复杂，我们应该学会“退一步”思考，由简单到复杂。
求最大值不知从何下手，我们先从容易的开始分析。

我们先研究，如果不要求输出 1 和 N 的最大距离，而只需一个可行的距离，应该如何操作。

我们用 D[i]表示 I 号奶牛和 1 号奶牛间的距离。

因为在队伍中的顺序必须和编号相同，所以对于任意 I 号奶牛，1<=I<N，在距离上应该满足：

D[I+1] - D[I] >= 0

对于每个好感的描述(i，j，k)，假设 i<=j，体现到距离上的要求就是：

D[j] - D[I] <= k

对于每个反感的描述(i，j，k)，假设 i<=j，体现到距离上的要求就是：

D[j] - D[I] >= k

这时的模型有一个名称，叫作：**差分约束系统**。

为了方便起见，我们将每种不等式写成我们约定的形式：

D[I] <= D[I+1]

D[j] <= D[I] + k

D[I] <= D[j] - k

看到这些不等式，你想到了什么？

没错，在求顶点间地最短路问题中，我们有这样的不等式：

若顶点 u 到顶点 v 有边 e=uv，且边权为 w(e)，设 d(u),d(v)为源点到顶点 u 和顶点 v 的最短路长，

则 d(v) <= d(u) + w(e)

这个不等式和前面的条件形式十分相似，这就启发我们用构图用最短路做。

具体步骤是：

作有向图 G=(V,E)，V={ v_1，v_2，v_3，…，v_n}，E={e_1，e_2，e_3，…}，对于

相邻两点 i 和(i+1)，对应的顶点 v_{i+1} 向 v_i 引一条边，费用为 0；对于每组好感描述(a_i，b_i，d_i)，我们假设有 $a_i<b_i$，否则 a_i 和 b_i 交换，则顶点 v_{ai} 向 v_{bi} 引一条边，费用为 d_i；对于每组反感描述(a_i，b_i，d_i)，我们假设有 $a_i<b_i$，否则 a_i 和 b_i 交换，则顶点 v_{bi} 向 v_{ai} 引一条边，费用为 $-d_i$。

于是问题变为在 G 中求 v1 到其它所有顶点的最短路。我们证明若 G 中无负权回路，则问题有解，即存在满足条件的数列，若 G 中有负权回路，则问题无解，即不存在满足条件的数列。

定理 5 问题是否有解等价于图 G 是否没有负权回路。

证明：若 G 中无负权回路，我们可以求出 v_1 其他顶点 u 的最短路长，设为 d(u)。由于是最短路，因此对于任意边 e∈ E，e=uv，有 d(u)+w(e)>=d(v)，从而所有的约束条件都被满足，问题一定有解。若 G 中有负权回路，说明在任何时刻，G 中至少有一个点 v 的最短路长可以更新，因此必须存在一条边 e=uv，使得 d(u)+w(e)<d(v)。所以无论何时，都会有某个约束条件不被满足，问题无解。（证毕）

检测负权回路，可以用 Bellman-Ford 算法。

回到原问题。
先说说考试时我的做法。

将所有点的最短路估计值设为一个充分大的值，v_1 的最短路估计值设为 0。然后运行一次 Bellman-Ford。

如果图 G 中有负权回路，那么输出-1；

否则，如果标号为 N 的顶点的最短路估计仍为一个充分大的值，那么它和标号为 1 的顶点间的距离可以任意大，这时输出-2；

如果以上两种情况都不满足，那么输出标号为 N 的点的最短路径估计值。

什么是充分大呢？
只需要比原图中最大的边权×顶点数还大就行了。

因为只要无圈，每个顶点最多只会被经过一次，所以肯定比这个值小，所以在该图中，我们可以把这个值看作是无穷大。

该方法可以通过竞赛的所有测试数据。

考试的时候，我没有去刻意证明这个方法的正确性，只是感觉它应该可行。

现在让我们从理论上证明它。

定理 6 若运行 Bellman-Ford 后，标号为 N 的顶点的最短路估计值仍为充分大，那么 N 和 1 的距离可以任意大。

证明：从刚才的操作可以看出，到了这一步，已经把含有负权回路的情况排除掉了。在图 G 中，该充分大的值比可能得到的最大距离大，因此，它和任意大的值对于 G 的效果都是一样的（同样大于合法的最大距离）。由于充分大的值在 G 中满足约束，所以任意大的值亦满足约束，从而距离可以任意大。（证毕）

剩下还有一个问题。

定理 7 若运行 Bellman-Ford 后，标号为 N 的顶点的最短路估计值比充分大小，那么它是 N 和 1 可能的最大距离。

证明：设 D[i]是顶点 i 和 1 的最短路径估计值，d[i]是顶点 i 和 1 可能的最大距离。

我们首先证明，d[n]<=D[n]，运用反证法。

假如 d[n]>D[n]，那么在 Bellman-Ford 运行之前，将赋予每个顶点 i 的充分大的值换成对应的 d[i]。由于 d 本身满足所有约束条件，所以运行后，得出 D'=d。由于充分大的值比所有 d[i]都大，而求最短路运用的是逐步松弛操作，我们设立一个更大的初值不可能导致我们的终值反而更小。所以对于任意 i，必定有 D[i]>=D'[i]，即有 D[n]>=d[n]，这与我们的假设矛盾。

然后我们证明，d[n]>=D[n]。

根据 d[i]的定义，它是 i 和 1 的可能最大距离。由于 D[i]是满足题目的所有

约束的，所以 D[i]是顶点 i 和 1 可能的距离。如果 D[i]>d[i]，那么与 d[i]的定义矛盾。

综合上述，有 D[i]=d[i]。从而 D[n]是 n 和 1 可能的最大距离。（证毕）

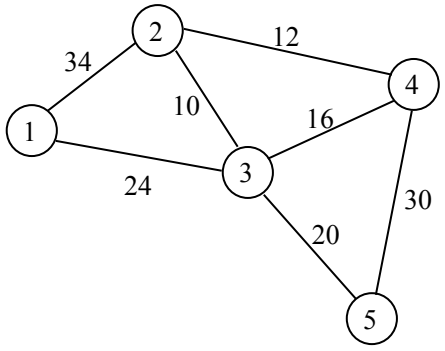
运行 Bellman-Ford 最坏情况下的复杂度是 O((ML+MD)*N)=O(2*10^7)，可以在规定时间内出解。另外，实际运行的速度相当理想，大部分数据运行基本上不需要时间。

3．4 例题 4——网络提速[4]

某学校的校园网由 n(1<=n<=50)台计算机组成，计算机之间由网线相连，如图 5。其中顶点代表计算机，边代表网线。正如你所见，不同网线的传输能力不尽相同，例如计算机 1 与计算机 2 之间传输信息需要 34 秒，而计算机 2 与计算机 3 之间的传输信息只要 10 秒。计算机 1 与计算机 5 之间传输信息需要 44 秒途径为机 1 到机 3 到机 5。

现学校购买了 m(1<=m<=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，即用两台设备，用时为原来的 1/4，用三台设备，用时为原来的 1/8。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。例如图 5，若 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。

图 5



[4]经典问题

输入格式

从文件 network.in 输入。第一行先输入 n,m。以下 n 行，每行有 n 个实数。第 i 行第 j 列的数为计算机 i 与计算机 j 之间网线的传输用时，0 表示它们之间没有网线连接。注意输入数据中，从计算机 1 到计算机 n 至少有一条网路。

输出格式

输出到文件 network.out。输出计算机 1 与计算机 n 之间传输信息的最短时间。同时输出 m 台设备分别用于何处。

样例输入

network.in
5 2
0 34 24 0 0
34 0 10 12 0
24 10 0 16 20
0 12 16 0 30
0 0 20 30 0

样例输出

network.out
22.00
1 3
3 5

分析：

题目中背景过多，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。
一个明显的反例是：

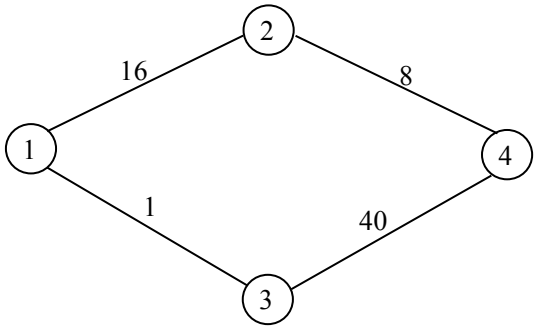


图 6

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。

让我们回忆一番学过的各种算法，看看能否有所帮助。

搜索？无论剪枝条件有多强，算法的复杂度始终是指数级别的……行不通；

数学方法？我们发现方案具有很大的随意性，同构但是权值不同的图答案可能完全不同，加速设备台数的不同也可能导致方案完全不同……数学方法也行不通；

贪心？刚才用的增量法其实应该属于贪心范畴的，很明显可以看出，局部的最优与全局最优有相当大的差别，所以估计其他策略的贪心也起不到什么作用……同样行不通；

动态规划？动态规划的两个基本要素是：状态的表示以及状态的转移。我们会发现在状态的表示上我们就遇到了困难。如果仅将点和设备台数作为元素，边的权值改变了，方程上必须体现出来，这似乎很难操作。如果加上边作为元素，一者空间上不能承受，二者撇开空间，这时候的效率和盲目搜索已经没有太大差别了。

以往学过的知识对于解题似乎毫无裨益。

为什么？

我们刚才犯了一个挺严重的错误，就是脱离了题目来想算法。这好比建空中楼阁，失败是难免的。

回到题目上。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也是解题的捷径。

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

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

对于原图的每一条边，注意是无向的，$e_k=v_iv_j \in E_0$，将它拆成(m+1)(m+2)条有向边 $(e_k)_{0,0}=v_{i,0}v_{j,0}$，$(e_k)_{0,1}=v_{i,0}v_{j,1}$，……，$(e_k)_{m,m}=v_{i,m}v_{j,m}$，$(e_k)'_{0,0}=v_{j,0}v_{i,0}$，$(e_k)'_{0,1}=v_{j,0}v_{i,1}$，……$(e_k)'_{m,m}=v_{j,m}v_{i,m}$。构造 E 使得

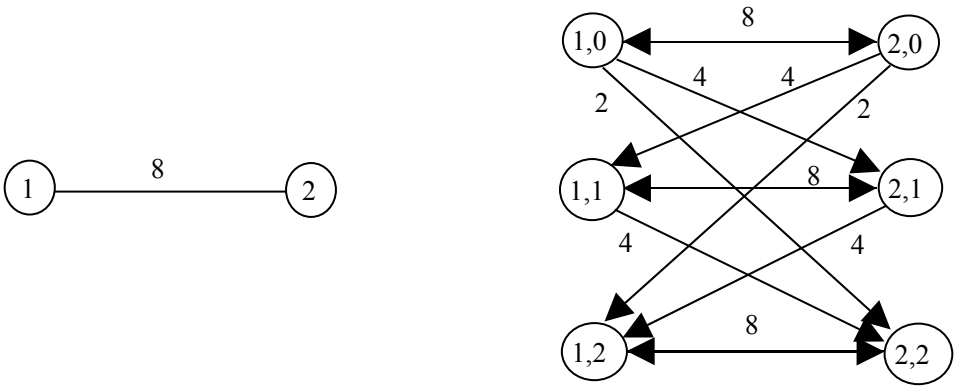
$E=\{(e_k)_{p,q}=v_{i,p}v_{j,q}, (e_k)'_{p,q}=v_{j,p}u_{i,q} | e_k=v_iv_j \in E_0, 0<=p<=q<=m\}$

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

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

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。图 7(a)显示了某个 G_0(n=2,m=2)，按照以上的构造方法得出图 7(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)

图 7

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。

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。

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。

四、总结

本文首先介绍了最短路的一些相关概念，它们是后边介绍的三种算法的基础。其中的某些性质对于建立最短路模型十分有用。

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，是性价比极高的算法。

在不含负权的图中，特别是在边数稠密的图中，我们常常选择 Dijkstra 算法，在稀疏图中，二叉堆实现的 Dijkstra 算法也是不错的选择，SPFA 算法效率极高；图中含有负权，稀疏图随便用后两种算法中的那一种都行，因为它们的效率都可以接受，而且都很容易写，要是稠密图的话，就非 SPFA 莫属了。总之，

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，在考场上用最少的时间拿尽可能多的分数。

对于绝大多数最短路问题，我们只需套用经典算法就行了。所以往往我们的难点是在模型的建立上。

这要求我们对最短路的核心思想有较深入的了解，对题目的本质有较全面的认识，还需要丰富的解题经验，……这里是最考察一个人综合素质的地方。

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。

——把握最短路的核心思想，合理转化模型。

阵而后战，兵法之常；运用之妙，存乎一心。

——因地制宜，根据实际选择最合适的算法。

【感谢】

感谢黄叶亭老师对我的指导，提出宝贵的意见和指出论文的不足之处。

感谢邹雨晗、江弘升同学在写论文的过程中所给予的帮助。

【参考文献】

1、《实用算法分析和程序设计》 吴文虎 王建德 电子工业出版社

2、《算法艺术与信息学竞赛》 刘汝佳 黄亮 清华大学出版社

3、《金牌之路·高中计算机》 王建德 周咏基 陕西师大出版社

4、《Introduction to Algorithm》 Thomas H.Cormen 等 The MIT Press

【附录】

附录一：例题一 货币兑换 英文原题

Currency Exchange
Time Limit: 1.0 second
Memory Limit: 1 000 KB

Several currency exchange points are working in our city. Let us suppose that each point specializes in two particular currencies and performs exchange operations only with these currencies. There can be several points specializing in the same pair of currencies. Each point has its own exchange rates, exchange rate of A to B is the quantity of B you get for 1A. Also each exchange point has some commission, the sum you have to pay for your exchange operation.

Commission is always collected in source currency.

For example, if you want to exchange 100 US Dollars into Russian Rubles at the exchange point, where the exchange rate is 29.75, and the commission is 0.39 you will get (100 - 0.39) * 29.75 = 2963.3975RUR.

You surely know that there are N different currencies you can deal with in our city. Let us assign unique integer number from 1 to N to each currency. Then each exchange point can be described with 6 numbers: integer A and B - numbers of currencies it exchanges, and real RAB, CAB, RBA and CBA - exchange rates and commissions when exchanging A to B and B to A respectively.

Nick has some money in currency S and wonders if he can somehow, after some exchange operations, increase his capital. Of course, he wants to have his money in currency S in the end. Help him to answer this difficult question. Nick must always have non-negative sum of money while making his operations.

Input

The first line of the input file contains four numbers: N - the number of currencies, M - the number of exchange points, S - the number of currency Nick has and V - the quantity of currency units he has. The following M lines contain 6 numbers each - the description of the corresponding exchange point - in specified above order. Numbers are separated by one or more spaces. 1 <= S <= N <= 100, 1 <= M <= 100, V is real number, 0 <= V <= 103.

For each point exchange rates and commissions are real, given with at most two digits after the decimal point, 10-2 <= rate <= 102, 0 <= commission <= 102.

Let us call some sequence of the exchange operations simple if no exchange point is used more than once in this sequence. You may assume that ratio of the numeric values of the sums at the end and at the beginning of any simple sequence of the exchange operations will be less than 104.

Output

If Nick can increase his wealth, output YES, in other case output NO to the output file.

Sample Input

3 2 1 20.0
1 2 1.00 1.00 1.00 1.00
2 3 1.10 1.00 1.10 1.00

Sample Output

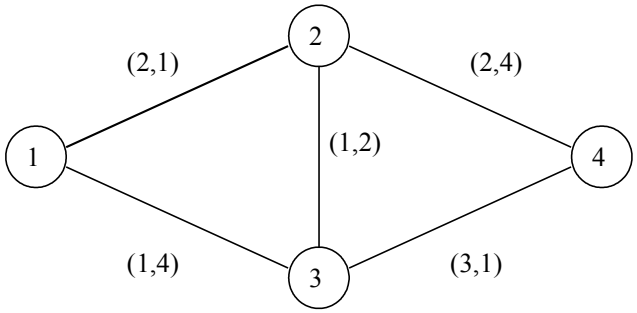
YES

附录二： 例题二 双调路径 英文原题

Bicriterial routing

The network of pay highways in Byteland is growing up very rapidly. It has become so dense, that the choice of the best route is a real problem. The network of highways consists of bidirectional roads connecting cities. Each such road is characterized by the traveling time and the toll to be paid.

The route is composed of consecutive roads to be traveled. The total time needed to travel the route is equal to the sum of traveling times of the roads constituting the route. The total fee for the route is equal to the sum of tolls for the roads of which the route consists. The faster one can travel the route and the lower the fee, the better the route. Strictly speaking, one route is better than the other if one can travel it faster and does not have to pay more, or vice versa: one can pay less and can travel it not slower than the other one. We will call a route connecting two cities minimal if there is no better route connecting these cities. Unfortunately, not always exists one minimal route – there can be several incomparable routes or there can be no route at all.



Example

The picture below presents an example network of highways. Each road is accompanied by a pair of numbers: the toll and the traveling time.

Let us consider four different routes from city 1 to city 4, together with their fees and traveling times: 1-2-4(fee4, time5), 1-3-4(fee4, time5), 1-2-3-4(fee6, time4) and 1-3-2-4(fee4, time10).

Routes 1-3-4 and 1-2-4 are better than 1-3-2-4. There are two minimal pairs fee time: fee4, time5 (roots 1-2-4 and 1-3-4) and fee6, time4 (root 1-2-3-4). When choosing the route we have to decide whether we prefer to travel faster but we must pay more (route 1-2-3-4), or we would rather travel slower but cheaper (route 1-3-4 or 1-2-4).

Task

Your task is to write a program, which:

Reads the description of the highway network and starting and ending cities of the route form the text file bic.in.

Computes the number of different minimal routes connecting the starting and ending city, however all the routes characterized by the same fee and traveling time count as just one route; we are interested just in the number of different minimal pairs fee time.

Writes the result to the output file bic.out.

Input data

There are four integers, separated by single spaces, in the first line of the text file bic.in: number of cities n (they are numbered from 1 to n), $1 \le n \le 100$, number of roads m, $0 \le m \le 300$, starting city s and ending city e of the route, $1 \le s,e \le n$, $s \ne e$. The consecutive m lines describe the

roads, one road per line. Each of these lines contains four integers separated by single spaces: two ends of a road p and r, 1≤ p,r≤ n, p≠r, the toll c, 0≤ c≤ 100, and the traveling time t, 0≤ t≤ 100. Two cities can be connected by more than one road.

Output data

Your program should write one integer, number of different minimal pairs fee time for routes from s to e, in the first and only line of the text file bic.out.

bic.in	**bic.out**	**Comments**
4 5 1 4 2 1 2 1 3 4 3 1 2 3 1 2 3 1 1 4 2 4 2 4	2	This is the case shown on the picture above.

附录三：例题三 layout 英文原题

Problem 3: Layout [Brian Dean, 2004]

Like everyone else, cows like to stand close to their friends when queuing for feed. FJ has N (2 <= N <= 1,000) cows numbered 1..N standing along a straight line waiting for feed. The cows are standing in the same order as they are numbered, and since they can be rather pushy, it is possible that two or more cows can line up at exactly the same location (that is, if we think of each cow as being located at some coordinate on a number line, then it is possible for two or more cows to share the same coordinate).

Some cows like each other and want to be within a certain distance of each other in line. Some really dislike each other and want to be separated by at least a certain distance. A list of ML (1 <= ML<= 10,000) constraints describes which cows like each other and the maximum distance by which they may be separated; a subsequent list of MD constraints (1 <= MD <= 10,000) tells which cows dislike each other and the minimum distance by which they must be separated.

Your job is to compute, if possible, the maximum possible distance between cow 1 and cow N that satisfies the distance constraints.

PROBLEM NAME: layout

INPUT FORMAT:

* Line 1: Three space-separated integers: N, ML, and MD.

* Lines 2..ML+1: Each line contains three space-separated positive integers: A, B, and D, with 1 <= A < B <= N. Cows A and B must be at most D (1 <= D <= 1,000,000) apart.

* Lines ML+2..ML+MD+1: Each line contains three space-separated positive integers: A, B, and D, with 1 <= A < B <= N. Cows A and B must be at least D (1 <= D <= 1,000,000) apart.

SAMPLE INPUT (file layout.in):

4 2 1

1 3 10

2 4 20

2 3 3

INPUT DETAILS:

There are 4 cows. Cows #1 and #3 must be no more than 10 units apart, cows #2 and #4

must be no more than 20 units apart, and cows #2 and #3 dislike each other and must be no fewer than 3 units apart.

OUTPUT FORMAT:

* Line 1: A single integer. If no line-up is possible, output -1. If cows 1 and N can be arbitrarily far apart, output -2. Otherwise output the greatest possible distance between cows 1 and N.

SAMPLE OUTPUT (file layout.out):

27

OUTPUT DETAILS:

The best layout, in terms of coordinates on a number line, is to put cow #1 at 0, cow #2 at 7, cow #3 at 10, and cow #4 at 27.