寻寻觅觅　道阻且长
动态规划与寻路问题
v0.3 preview

崔添翼 (Tianyi Cui)*

2011-09-25†

本文题为《寻寻觅觅　道阻且长》，副题为《动态规划与寻路问题》，从属于《动态规划的思考艺术》系列。

本文的修订历史及最新版本请访问 https://github.com/tianyicui/DP-Book 查阅。

本文的这个版本并未公开发布，只由作者直接发布给若干试读者，试读者没有传播给别人的权利。若你并非作者指定的试读者，而通过其它途径读到本文，则说明存在至少一位试读者未尽保密义务。

Contents

*a.k.a. dd_engi

†Build 20110925221400

1 寻路问题导引

本文所探讨的寻路问题是指：给定某个可抽象为图论中的模型的图，要求找到图中满足某种性质的一条路径。除判别存在性之外，一般均要求此条路径具备某种最优性，亦即在某种计算方法下最短或最长。对于路径的起点和终点也常常有着限制。

寻路问题一般都是最优化问题，这一点上与动态规划是一致的。所以说，很多寻路类型的题目都可用动态规划解答，图论中的 Dijkstra 算法（见3.1.1）、Floyd 算法（见3.2）都是动态规划在图论中的经典应用。

另一方面，很多原本不存在图论模型的动态规划题目，也可转化成或建立起图论模型，并利用图论中的寻路算法进行解答。例如完全背包问题（TODO：引用《背包问题九讲》的相应章节），就可转化为图论模型：容量为 V 的背包转化为编号从 0 到 V 的点，每个费用为 v_i、价值为 w_i 的物品转化为 $V-v_i+1$ 条从 $k+v_i$ 到 k 的长度为 w_i 的边（$0 \le k \le V-v_i$）。而找到一组最优解则对应为找到这个有向无环图中点 V 到点 0 的一条最长路（具体算法见2.2）。

综上，寻路问题与动态规划有着千丝万缕的联系。本文后面的部分便着重探讨这种联系衍生出的千变万化的动态规划题目与解法。

2 特殊图中的寻路

2.1 矩阵中的寻路

2.1.1 基础问题

问题是这样的：设有一 $M \times N$ 的矩阵 P，矩阵单元格中的元素皆是数字。从矩阵的左上角走到右下角，只能向右或向下走，怎样能使覆盖到的数字之和最大？（TODO：配图？）

$M \times N$ 的矩阵从左上走到右下，需要走 $M+N-2$ 步，其中向下走 $M-1$ 步，向右走 $N-1$ 步，故可能的方案数共有 $\begin{pmatrix} M-1 \\ M+N-2 \end{pmatrix}$ 种。用搜索穷举所有方案的时间复杂度太高，考虑使用动态规划算法。

动态规划算法的一个典型思考方式是“只想一步”。以这道题目为例来说明，尽管要求做出的是 $M+N-2$ 个向右或者向下的指令，我们不妨想想：知道了什么信息，就可以得出第一步应该怎么走。

第一步有两种走法，向右走从 $(0,0)$ 走到 $(0,1)$，或者向下走从 $(0,0)$ 走到 $(1,0)$，二者的分野不仅存在于这一步覆盖到的数字不同，也体现在今后可选择的策略不同。例如若第一步向右走则左边第一列的数字再也无法拿到。

如何才能判别从 $(0,0)$ 到 $(M-1,N-1)$ 的路途中第一步怎么走呢？怎样比较第一步的两种走法的优劣呢？很简单，这一方面取决于 (0,1) 和 (1,0) 上面的数字，另一方面也取决于，从这两点出发，走到终点的过程中还会覆盖哪些数字，可能覆盖的最大的数字之和是多少。

于是这样抽象出动态规划的子问题：

设 $F[i,j]$ 表示从 (i,j) 出发，走到 $(M-1,N-1)$，可以覆盖到的数字之和的最大值，需求解的原问题既是 $F[0,0]$，边界条件是 $F[M-1,N-1]=P_{M-1,N-1}$，状态转移方程是：

$$F[i,j]=P_{i,j}+\max\left\{\begin{array}{l}F[i,j+1]\ \textbf{if}\ j+1<N\\F[i+1,j]\ \textbf{if}\ i+1<M\end{array}\right\}$$

若将此方程以记忆化递归的方法实现，伪代码如下：

```
def MatrixWalkRec(F,i,j)
    if i ≥ M or j ≥ N
        return 0
    if F[i,j] = undefined
        F[i,j] ←
P_{i,j} + max(MatrixWalkRec(F,i,j+1),MatrixWalkRec(F,i+1,j))
    return F[i,j]
```

刚才应用了“只想一步”的方法，成功地解决了这道题目如何划分子问题、如何写出状态转移方程的问题。美中不足的是，上面给出的自然的实现是递归的实现。怎样将递归的状态转移方程实现成非递归的程序呢?

这就需要对我们的状态转移方程所决定的状态之间的相互关系进行分析。我们看到，每个状态 (i,j) 依赖于它右方和下方的 $(i+1,j)$ 和 $(i,j+1)$，非递归的实现的实质是要确定一种合适的状态的求解顺序，保证任何状态的求解都在其依赖的状态之求解完成之后才开始。一种合适的顺序是：从下到上，从右至左。伪代码如下：

```
def MatrixWalk
    F[0...M,N] ← 0
    F[M,0...N] ← 0
    for i ← M-1 to 0
        for j ← N-1 to 0
            F[i,j] ← P_{i,j} + max(F[i,j+1],F[i+1,j]
```

至此，本问题的动态规划解法分析完毕。留一个问题让读者思考：前面利用“只想一步”的思考方式时，想的是“第一步应该怎么走”，如果考虑的角度变成“最后一步应该怎么走”，会导出怎样的子问题定义、状态转移方程以及伪代码实现呢?

2.1.2 变形问题

第一种变形问题是：改变计算权值的方法。

例如，将求最大值改成最小值，将求和的加法运算改成乘法，将矩阵单元格中的值由实数改成向量，等等

一般而言，这种变形问题只需对状态转移方程进行一些微调就可以。

但要注意的是，子问题的局部最优性是否还代表着全局最优性。例如，将问题中的“数字之和最大”改成“数字之积最大”之后，还按照上文中“如何踏出第一步”的方法思考时，假如说 $(0,0)$ 中的数字是个负数，那么接下来向右走及向下走之后要解决的子问题就不是求“数字之积最大”而是希望求得一个“积是负数且绝对值最大”的子路径。

也就是说，将“和”改成“积”后，对于每个子问题 (i,j)，需要运算及记录的是两个值：一个是路径上数字之积的可能的最大值 $F[i,j]$，一个是可能的最小值 $G[i,j]$（或称绝对值最大的负值）。

这个变形问题的伪代码及其实现强烈建议读者尝试一下。

（TODO：是否在这里给出伪代码呢？）

第二种变形问题是：改变路径的可行性。

在原问题中，所有 $\begin{pmatrix} M-1 \\ M+N-2 \end{pmatrix}$ 种路径都是可行的，但我们可以人为设置一些限制条件。例如，设置一些单元格为禁止通行的单元格。或设置一些单元格之间的边界为禁止通过的边界。或设置通过单元格或单元格之间的边界时必须满足某种条件。这样的限制条件并不会更改状态转移方程的形式，只不过可以转移到的状态要在程序中根据题义进行判断，这种判断可以设计得相当复杂。

给一个这种变形的例题：单元格之间有可能存在锁着的门，打开一扇门必须耗费一把钥匙，不同的门需要的钥匙的种类不同，单元格中可能会捡到钥匙；问题仍然是求解最优的路径。

这道例题的求解，需要在状态转移方程中增加一维来表示手中现有的钥匙种类和数量。然后在计算过程中根据捡到和消耗的钥匙进行状态之间的转移。

（TODO：加个示意图？）

第三种变形问题是：增加路径的条数。

典型问题如，要求两条从矩阵左上到右下的路径，这两条路径不可以相交（或相交后同一单元格的数值只算一次），最大化路径经过的权值和。

这种题目可以以两条路径所经过的单元格距左上单元格的距离划分阶段，以两条路径所经过的单元格坐标做为状态，列出状态转移方程。

（TODO：方程其实写出来很难看，不如写程序代码附录进去。）

以上说明了矩阵中寻路问题的三种变形问题的方向，在实际中，将这道基础的动态规划题目经由多个方向同时变形，就可以构造出看似很复杂的题目。

（TODO：还有什么变形问题的种类吗?）

2.2 有向无环图中的寻路

2.2.1 从矩阵寻路到有向无环图

在上一节2.1中，我们给出的基础问题及其所有变形问题都有一个共同点：要求找到从左上角到右下角的一条路径，且只能向相邻的右方、下方的单元格走。

这个限制条件给所有的单元格排了个顺序，亦即，单元格 $(i+p,j+q)$ 和 (i,j) 若均在路径中出现，$(i+p,j+q)$ 一定出现在 (i,j) 之后（$p \geq 0, q \geq 0, p+q > 0$）。这个顺序决定了我们把动态规划的状态的表示确定为单元格的坐标时，状态之间的依赖关系是很明晰的，状态的求解顺序也不是个难题。

我们来看一道没有这个只能向右、下走的限制条件的例题[1]：

仍然是 $M \times N$ 的数字矩阵 P，要求寻找的路径可以从任意一点出发，可以向上、下、左、右任意方向延伸，但路径经过的数字必须越来越小、单调递减。求矩阵中存在的最长的满足要求的路径。

去掉了仅能向右、下走及固定起点、终点的限制条件，加上了一个路径中数字单调递减的限制条件。这里还是利用"只想一步"的方法来思考。路径的起点在哪里？不知道，设为 (i, j)。路径的第一步往哪儿走？往哪里走能走得更远就往哪里走。于是状态转移方程是：

$$F[i,j] = 1 + \max \left\{ \begin{array}{c} 0 \\ F[i,j-1] \text{ if } P_{i,j-1} < P_{i,j} \\ F[i,j+1] \text{ if } P_{i,j+1} < P_{i,j} \\ F[i-1,j] \text{ if } P_{i-1,j} < P_{i,j} \\ F[i+1,j] \text{ if } P_{i+1,j} < P_{i,j} \end{array} \right\}$$

考察这个状态转移方程所隐含的状态之间的依赖关系：一个状态的计算有可能依赖于它上下左右的任意一个状态，这看上去很可能存在状态间的循环依赖以至于状态转移方程无法被求解。但实际上，一个状态只依赖于其上下左右的单元格中元素数值比自身小的那些。换句话说，若将所有的单元格按照其中元素从小到大的顺序排个序，并按这个顺序求解状态的话，则只会有后面的状态依赖于前面的状态，不会出现求解某一状态时它所依赖的状态还未解出的可能。下面给出伪代码：

```
def MatrixDecPath
    F[0...M+1, 0...N+1] ← 0
    Q ← [(P_{i,j}, i, j) | for i, j in (1...M, 1...N)]
    sort Q
    for p, i, j in Q
        d ← [(0,−1), (0,+1), (−1,0), (+1,0)]
        for k ← 0 to 3
            if P_{i+d[k].x, j+d[k].y} < P_{i,j}
                F[i,j] ← max{F[i,j], 1 + F[i+d[k].x, j+d[k].y]}
```

最后的答案就是 $F[1 \dots M, 1 \dots N]$ 中的最大值。

值得指出的是，以上给出的非递归的伪代码在理解和实现上均略有些麻烦。在看清了状态的求解不存在循环依赖的时候，就可以不用排序，而直接用记忆化递归的方法（TODO：引用 MatrixWalkRec）实现。

2.2.2 有向无环图的拓扑排序

上一节（2.2.1）引入的新问题中，在矩阵中寻找的路径可以是任意的，不仅仅局限于从左上到右下的不准绕远路的那一种路径。命题者需要寻找的最佳路径，可以根据数据具有任意的形状。但有一点值得注意，不管路径走成何种形状，都不可能存在一条数字单调递减的路径与自身交叉。换句话说，这个"单调递减"的条件，保证了作为寻路

[1] PKU 1088

结果的路径中不可能存在环，导致了动态规划的状态转移中不可能出现循环依赖，决定了这道题目可以用动态规划来解。

用图论的术语来说明，不管是只能向右、向下移动的矩阵，还是经过数字单调递减的矩阵，抽象成图论中图的模型之后，都是有向无环图。有向无环图的最重要性质就是，其中的定点可以进行拓扑排序。

下面从数学中“关系”的角度诠释一下有向无环图的拓扑排序。

数学中定义的二元关系 R，是指任意的有序对的集合。若 $R \subseteq X \times Y$，则用 xRy 表明 x 与 y 之间存在关系 R，有 x∈X,y∈Y,(x,y)∈R。常见的一种二元关系即是“函数”，函数的特点是，定义域中的一个值，对应于值域中的唯一一个值。而在一般的关系中则没有这样的限制。

有一类特殊的关系叫做严格偏序关系，其定义是：满足反自反、反对称、传递的关系。反自反的意思是说，任何元素和自己都不存在关系，$\forall x\ (x,x) \notin R$；反对称的意思是说如果 x 和 y 有关系，反过来 y 和 x 肯定没关系，$(x,y) \in R \Rightarrow (y,x) \notin R$；传递的意思是说，如果 x 跟 y 有关系，y 跟 z 有关系，则 x 和 z 肯定也有关系，$xRy, yRz \Rightarrow xRz$。

可以举一个严格偏序关系的例子，集合之间的“真子集”关系。任两个集合 P 和 Q，有可能 P 是 Q 的真子集，有可能 Q 是 P 的真子集，也有可能二者谁都不是谁的真子集。可以验证这个关系满足反自反、反对称和传递关系。

如果将图的顶点的集合看做关系中的定义域与值域，同时有向边看做关系集合中的有序对的话，则图论中的“有向无环图”的概念对应到组合数学中，就是“严格偏序关系的子集”。

有向无环图满足“反自反”的性质，因为没有自环；还满足“反对称”的性质，否则就有长度为 2 的环。但有向无环图不满足传递性，a 和 b 之间以及 b 和 c 之间都有边，并不能保证 a 和 c 之间就一定有边。所以，有向无环图是严格偏序关系的子集。

从另一个角度讲，严格偏序关系中肯定没有环。因为如果有环，根据传递性，必然会存在自环，违反了反自反性。所以，严格偏序关系的子集一定是有向无环图。

偏序关系之所以“偏”，是因为其中并不是任意两个元素都是可以比较出它们之间的“序”的。如果任意两个元素都能够确定顺序，且此顺序具有一致性，那么这种关系就叫做全序关系。也就是说，集合中的所有元素能够按照这种关系排一下顺序。例如自然数之间的大于、小于关系，单词之间的字典序关系，都是全序关系的例子。

有向无环图的拓扑排序的实质是，给这种“严格偏序关系的子集”指定一个全序，保证所有的边的方向都是从此排序的前面的顶点指向后面的顶点，不存在反向的边。

怎么求出有向无环图的一个拓扑序呢？

（TODO：UNFINISHED）

有了拓扑序的“不存在反向边”的性质，就可以利用这种排序更好地解决有向无环图中的寻路问题。

（TODO：UNFINISHED）

3 一般图中的寻路

在有向无环图中，确定了拓扑序以后，不管是寻找最短路还是最长路的问题，都变得易于解决了。但在一般的并不存在拓扑序的图中，如何寻找两点之间的最短路、从一

点出发到其它所有点的最短路、所有点对之间的最短路的算法并非显而易见。值得一提的是，下面解决这些问题的经典算法都属于动态规划的范畴。

3.1 单起点寻路的三种算法

问题：已知图 $G=(V,M)$，其中 V 是顶点集合，M 是表示边的邻接矩阵，求顶点 S 到其它所有顶点的最短路径。

3.1.1 Dijkstra 算法

用动态规划的眼光来看，需要求解的是 $|V|-1$ 个形式完全相同的问题，且这些问题之间明显存在着相互的关联和依赖。很容易想到，设 $F[x]$ 为 S 到 x 的最短路径的长度这样的状态表示。

状态的表示很简单，但状态之间的关系却很复杂，实际上可以是任意的。状态之间关系的复杂造成了确定求解状态顺序时的复杂。如果是有向无环图，只需要简单地按照拓扑顺序即可求解。但一般图中又按照何种顺序求解这 $|V|-1$ 个最短路问题呢？

Edsger Dijkstra 在上世纪 50 年代提出的算法适用于顶点间边的长度均为正数的情景。在边权为正的前提下，可以根据各个状态的答案的大小来划分状态求解的顺序。

例如，在算法刚刚开始的时候，在 $F[1\dots|V|]$ 这 $|V|$ 个状态中，我们已知的状态只有一个：$F[S]=0$，从 S 到自身的最短路径长度为零。下面根据这个状态的值去试图求解其它的状态。顶点 S 可能有若干条指向其它顶点的出边，在这些边里，肯定有最短的一条，不妨设为 $M_{S,S_1}=k$，如果我们知道了 S 到 S_1 的边是 S 所有出边中最短的，我们就可以断定：$F[S_1]=k$。原因是：边权都是正的，其它任何一条从 S 迂回走向 S_1 的路径，例如 $S-S_2-\dots-S_1$ 一定不会更优，因为 M_{S,S_1} 是同起点的边中最小的，$M_{S,S_2}\geq M_{S,S_1}$。

就这样，我们通过找到从起点 S 出发的长度最小的一条边的方法，将状态集合中已解出状态的数目从 1 个提高到了 2 个。我们希望能够利用类似的方法，将已解出的状态的数目不断增加，直至 S 到所有顶点的最短路径都被确定。

从另一个角度来看状态集合 F，$F[1\dots|V|]$ 中，状态之间的相互关系是什么呢？有边 $M_{i,j}$ 相连的两个节点 i 和 j 的相互关系可以用一个不等式来刻画：$F[i]+M_{i,j}\geq F[j]$，其中不等式取等号当且仅当从 S 到 j 的某条最短路中，i 是 j 的前驱顶点。从这个不等式可以推出：当 $F[i]$ 的值已解出时，可以用 $F[j]\leftarrow\max\{F[j],F[i]+M_{i,j}\}$ 的方法来更新状态 $F[j]$，这也就是在最短路问题中经常用到的放松 (relax) 操作。

上面说了两方面：第一，我们可以将所有的顶点分成已解出和未解出两个集合；第二，放松操作可以根据已解出的顶点的状态值更新未解出集合中的值。那么，在什么样的条件下，可以将一个顶点由未解出集合放入已解出集合中呢？答案是，选择未解出集合中状态值最小的那个放入已解出的集合中。这样选择的原因是，因为这个状态的值小于等于未解出集合中所有其它状态的值，在边权为正的前提下，其最短路的前驱节点必然是已解出集合中的节点，它的值不可能受任何未解出集合中状态的影响。

Dijkstra 算法的伪代码如下：

$Solved[1\dots|V|]\leftarrow$`false`
$F[1\dots|V|]\leftarrow\infty$
$F[S]\leftarrow 0$
`for` $|V|$ `times`

$v \leftarrow undefined$
for $i \leftarrow 1$ to $|V|$
    if not $Solved[i]$
        if $v = undefined$ or $F[i] < F[v]$
            $v \leftarrow i$
$Solved[i] \leftarrow$true
for $j \leftarrow 1$ to $|V|$
    if $M_{i,j}$ exists and not $Solved[j]$
        $F[j] \leftarrow \max\{F[j], F[i] + M_{i,j}\}$

以上代码中主循环共循环 $|V|$ 次，按照路径长度由小到大的顺序，每次解出一个状态。主循环中寻找到下一个放入已解出集合中节点以及利用已解出节点放松相邻的未解出节点的过程的实现都是 $O(V)$ 的，故以上伪代码的时间复杂度是 $O(V^2)$。实际上，如果使用二叉堆来保存 F 集合中未解出节点的信息，并用邻接表而非邻接矩阵来存储图，可以将复杂度优化到 $O(ElogV)$。

3.1.2 Bellman-Ford 算法

Dijkstra 算法是按照路径的长度从小到大的顺序划分阶段、决定状态求解的顺序，但它只适用于边权为正的场合。当边权可以是负数时，在同一条最短路径中后面顶点的距离长度不一定大于前面的顶点，此时 Dijkstra 算法以及其划分阶段的方式就不再适用了。

在 Bellman-Ford 算法中，采用最短路径包含的边的数量从小到大的顺序划分阶段。在初始阶段时，和 Dijkstra 算法一样，只有 $F[S]$ 是已知的，其它都设为 $+\infty$。每个阶段中都用所有的边尝试进行放松操作。在第一次迭代时，最短路径只包含一条边的状态肯定已经求解完毕了；尽管我们此时并不能具体地确定求解完毕的是哪些状态。在第二次迭代时，最短路径包含两条边的状态会由前阶段中求解完毕的状态加上一次放松操作而得到。 在第 k 次迭代时，最短路径包含 k 条边的状态值会经由 k 次成功的放松操作而被求解完毕。

如果在第 k 阶段中所有的放松操作都无效，没有更新任何一个状态的值，那么可以断定不存在包含 k 条边及以上的最短路径，可以不再继续迭代求解了。

从另一个角度将，如果在第 $|V|$ 阶段仍然有状态的值被更新，意味着存在包含边数不小于 $|V|$ 的最短路径。然而若最短路径是不包含环的简单路的话，则包含的边数在最多（所有的定点都在这条简单路中）也只应该有 $|V| - 1$ 条。所以，如果状态转移进行到第 $|V|$ 阶段仍未收敛停止，则说明图中存在负环。换言之，说明图中某些顶点（从任一负环中任意顶点出发能到达的顶点）的最短路径不存在。

下面给出 Bellman-Ford 算法的伪代码。

$F[1 \ldots |V|] \leftarrow \infty$
$F[S] \leftarrow 0$
for $|V|$ times
    $relaxed \leftarrow false$
    for $M_{i,j}$ in M
        if $F[i] + M_{i,j} < F[j]$

```
            relaxed ← true
            F[j] ← F[i] + M_{i,j}
    if not relaxed
        break
    if |V|th time
        output 存在负权环
```

3.1.3 最短路树

（TODO：通过引入最短路树来讲解动态规划中“最优性和历史无关”“无后效性”等概念。Markovian?）

3.2 全图寻路的 Floyd 算法

上面给出了有向图中求单源最短路的两种算法，分别适用于无负权和有负权的两种情况。

在有些情况下，我们需要求得整个图中所有顶点对之间两两的最短路。将图中的每一个点设为起点，用 Dijkstra 之类的算法求一次单源最短路是一种可以考虑的方案。但是否有更简便的做法呢？

我们还是试图用动态规划的方法来解决问题。设 $F[i,j]$ 表示顶点 i 和 j 之间的最短路。

（TODO：UNFINISHED）

3.3 Hamiltonian 回路

3.3.1 简单的动态规划算法

3.3.2 优化的动态规划算法

4 扩展的寻路问题

4.1 扩展的 Dijkstra 算法