环形与后效性处理

李淳风

长郡中学

2024 年 9 月 24 日

前言

在许多环形结构的问题中，我们都能通过枚举法，选择一个位置把环断开，变成线性结构进行计算，最后根据每次枚举的结果求出答案。我们把能通过上述枚举方式求解的环形问题称为“可拆解的环形问题”。我们的目标是采取适当的策略避免枚举，从而使时间复杂度得到降低。

例题

Naptime

在某个星球上，一天由 n 个小时构成，我们称 0 点到 1 点为第 1 个小时、1 点到 2 点为第 2 个小时，依此类推。在第 i 个小时睡觉能够恢复 U_i 点体力。在这个星球上住着一头牛，它每天要休息 B 个小时。它休息的这 B 个小时不一定连续，可以分成若干段，但在每段的第 1 个小时，它需要从清醒逐渐入睡，不能恢复体力，从下一个小时开始才能睡着。
为了身体健康，这头牛虚妄遵循生物钟，每天采用相同的睡觉计划。另外，因为时间是连续的，即每天的第 n 个小时和下一天的第 1 个小时是相连的（n 点等于 0 点），这头牛只需要在每 n 个小时内休息够 B 个小时就可以了。
请你帮忙给这头牛安排一个睡觉计划，使它每天恢复的体力最多。
$1 \leq n \leq 3830$。

Naptime

先来考虑一个简化版的问题，即每天的第 n 个小时和下一天的第 1 个小时不是相连的。这样星球上每天的时间就是线性的，可以用一个长度为 n 的序列 $\{U_i\}$ 来描述一天内每小时的体力恢复情况。我们需要从这个序列中选出 B 项，让一天里恢复的体力尽量多。

运用前面的知识，设 $f[i][j][1]$ 表示前 i 个小时休息了 j 个小时，并且第 i 个小时正在休息，累计恢复体力的最大值；$f[i][j][0]$ 表示前 i 个小时休息了 j 个小时，并且第 i 个小时没有在休息，累计恢复体力的最大值。

$$f[i][j][0] = \max(f[i-1][j][0], f[i-1][j][1])$$

$$f[i][j][1] = \max(f[i-1][j-1][0], f[i-1][j-1][1] + U_i)$$

在这个简化的问题中，第 1 个小时是每天的开始，不可能回复体力，因此初值为 $f[1][0][0] = 0, f[1][1][1] = 0$，其余均为负无穷。

我们的目标值为 $\max(f[n][B][0], f[n][B][1])$。

Naptime

到目前为止，我们解决的“线性问题”仅仅比原来的“环形问题”少了一种情况，即在第 1 个小时熟睡并获得 U_1 点体力。如果在原问题的最优解中，第 n 个小时与第 1 个小时不都在休息，那么简化问题的答案即为所求。为了补充这个缺少的情况，可以强制令第 n 个小时和第 1 个小时都在休息，使第 1 个小时能够恢复体力。我们仍然采用之前线性 DP 的方法，只不过需要把初始值修改为 $f[1][1][1] = U_1$，其余为负无穷，目标值变为 $f[n][B][1]$。
两次动态规划中的较优者就是原问题的答案。

Naptime

到目前为止，我们解决的“线性问题”仅仅比原来的“环形问题”少了一种情况，即在第 1 个小时熟睡并获得 U_1 点体力。如果在原问题的最优解中，第 n 个小时与第 1 个小时不都在休息，那么简化问题的答案即为所求。为了补充这个缺少的情况，可以强制令第 n 个小时和第 1 个小时都在休息，使第 1 个小时能够恢复体力。我们仍然采用之前线性 DP 的方法，只不过需要把初始值修改为 $f[1][1][1] = U_1$，其余为负无穷，目标值变为 $f[n][B][1]$。

两次动态规划中的较优者就是原问题的答案。

本题的解法本质上是把问题拆成了两部分，第一部分不可能在第 1 小时恢复体力 U_1，而第二部分强制在第 1 小时获得 U_1 点体力。这两部分合起来就能确定整个问题。无论哪一部分，因为第 n 小时和第 1 小时之间的特殊关系被确定，所以我们就能把环拆开，用线性 DP 计算。

这是环上问题的常见策略——执行两次 DP，第一次在任意位置把环断开成链，按照线性问题求解；第二次通过适当的条件与赋值，保证计算出的状态等价于把断开的位置强制相连。

例题

环路运输

在一条环形公路旁边均匀分布着 n 座仓库，编号为 $1 \sim n$，编号为 i 的仓库与编号为 j 的仓库之间的距离为 $dist(i,j) = \min(|i-j|, n-|i-j|)$，就是逆时针走和顺时针走两种情况中较近的一种。每座仓库都存有货物，其中编号为 i 的仓库库存量为 A_i。在 i 和 j 两座仓库之间运送货物需要的代价为 $A_i + A_j + dist(i,j)$。求在哪两座仓库之间运送货物需要的代价最大。$1 \le n \le 10^6$。

环路运输

我们在任意位置（例如仓库 1 和 n 之间）把环断开，复制一遍接在末尾，形成长度为 $2n$ 的直线公路。在转化之后的模型当中，公路旁均匀分布着 $2n$ 座仓库，其中 $A_i = A_{i+n}(1 \le i \le n)$。
对于原来环形公路上的任意两座仓库 i 和 $j(1 \le j < i \le n)$，如果 $i - j \le n/2$，那么在新的直线公路上，仍然可以对应成在 i 和 j 之间运送货物，代价为 $A_i + A_j + i - j$。
如果 $i - j > n/2$，那么可以对影成在 i 和 $j + n$ 之间运送货物，代价为 $A_i + A_{j+n} + j + n - i$，其中 $j + n - i = n - (i - j) \le n/2$。
综上所述，原问题转化为：长度为 $2n$ 的直线公路上，在满足 $1 \le j < i \le 2n$ 且 $i - j \le n/2$ 的哪两座仓库 i 和 j 之间运送货物，运送代价 $A_i + A_j + i - j$ 最大？

我们可以枚举 i，对于每个 i，需要找到一个 $j \in [i - n/2, i - 1]$，使 $A_j - j$ 尽量大。这是一个滑动窗口求最大值问题，使用单调队列维护，可以在均摊 $O(1)$ 的时间复杂度内求解。总复杂度为 $O(n)$。

有后效性的转移方程

从最初学习 DP 开始，我们就知道，“无后效性”使应用动态规划算法的三前提之一。事实上，在一些题目中，当我们根据题目的关键点抽象出“状态维度”，并设计出状态标识和转移方程之后，却发现不满足“无后效性”这一基本条件——部分状态之间互相转移，互相影响，构成了环形，无法确定出一个合适的 DP “阶段”，从而沿着某个方向递推。

但这样的问题也并非无解。我们可以把动态规划的各个状态看作未知量，状态的转移看作若干个方程。如果仅仅是“无后效性”这一条前提不能满足，并且状态转移方程都是一次方程，我们可以不进行线性递推，而是用高斯消元直接求出状态转移方程的解。

在更多的题目中，动态规划的状态转移“分阶段带环”——我们需要把 DP 和高斯消元相结合，在整体层面采用动态规划框架，而在局部使用高斯消元解出互相影响的状态。

例题

Broken Robot

给定一张 $N * M$ 的棋盘，有一个机器人位于 (x, y) 位置。这个机器人可以进行很多轮行动，每次等概率地随机选择停在原地、向左移动一格、向右移动一格或向下移动一格。当然机器人不能移动出棋盘。求机器人从起点走到最后一行地任意一个位置上，所需行动次数的数学期望值。

$1 \leq n, m \leq 1000$

例题

Broken Robot

给定一张 $N * M$ 的棋盘，有一个机器人位于 (x, y) 位置。这个机器人可以进行很多轮行动，每次等概率地随机选择停在原地、向左移动一格、向右移动一格或向下移动一格。当然机器人不能移动出棋盘。求机器人从起点走到最后一行地任意一个位置上，所需行动次数的数学期望值。
$1 \le n, m \le 1000$

在本题中，我们可以用 $f[i][j]$ 来表示从 (i, j) 走到最后一行的期望步数，也可以表示从 (x, y) 走到 (i, j) 的期望步数。但对于后者，我们还需要进行额外的计算才能得到答案，较为复杂。因此我们设 $f[i][j]$ 来表示从 (i, j) 走到最后一行的期望步数，通过倒推的方式来进行计算。
机器人在第一列时：$f[i][1] = \frac{1}{3}(f[i][1] + f[i][2] + f[i+1][1]) + 1$；
同理，在第 M 列时：$f[i][M] = \frac{1}{3}(f[i][M] + f[i][M-1] + f[i+1][M]) + 1$；
对于 $1 < j < M$：$f[i][j] = \frac{1}{4}(f[i][j] + f[i][j-1] + f[i][j+1] + f[i+1][j]) + 1$
初值为 $f[N][j] = 0$，目标为 $f[x][y]$。

Broken Robot

观察上面的状态转移方程，我们可以发现因为第 i 行的状态只能转移到第 $i+1$ 行；但在同一行内，各个状态之间互相转移，无法确定一个递推顺序。故我们可以采用 DP 套高斯消元的算法来求解本题。

我们先以行号为阶段，从 M 到 x 来倒序扫描每一行，依次计算以该行的每个位置为起点走到最后一行，所需行动次数的数学期望值。在计算第 i 行的状态时，由于第 $i+1$ 行的状态已经计算完毕，我们可以把 $f[i+1][j]$ 看作已知数，状态转移方程中就只剩下了 M 个未知量，并且共有 M 个方程，可以使用高斯消元求解。

我们列出高斯消元的系数矩阵并仔细观察，就发现除了主对角线和对角线两侧的斜线之外，其余部分均为 0。下面是 $M=5$ 时的系数矩阵：

$$\begin{matrix} -2/3 & 1/3 & 0 & 0 & 0 \\ 1/4 & -3/4 & 1/4 & 0 & 0 \\ 0 & 1/4 & -3/4 & 1/4 & 0 \\ 0 & 0 & 1/4 & -3/4 & 1/4 \\ 0 & 0 & 0 & -2/3 & 1/3 \end{matrix}$$

对这样的矩阵进行高斯消元，每一行中实际只有 $2\sim3$ 个位置需要相减，只需 $O(M)$ 的时间即可求出各个未知量的解。整个算法的时间复杂度为 $O(NM)$。注意 $M=1$ 时需要特判。

Accumulation Degree

用“二次扫描与换根法”代替源点的枚举，可以在 $O(n)$ 的时间内解决整个问题。首先，我们任选一个源点作为根节点，记为 $root$，然后采用上面的代码进行一次树形 DP，求出 D_{root} 数组，简单记为 D 数组。
设 $f[x]$ 表示把 x 作为源点时的最大流量，显然有 $f[root] = D[root]$。
假设 $f[x]$ 已经被正确求出，考虑其子节点 y，$f[y]$ 尚未计算。而 $f[y]$ 由两部分组成。

- 从 y 流向以 y 为根的子树的流量，已经计算并被保存在了 $D[y]$ 中。
- 从 y 沿着到父节点 x 的河道，进而流向水系中其它部分的流量。

因为把 x 作为源点的总流量为 $f[x]$，从 x 流向 y 的流量为 $\min(D[y], c(x, y))$，所以 x 流向 y 以外其他部分的流量就是二者之差。
于是，把 y 作为源点时，先流到 x，再流向其他部分的流量，就是把这个“差”再与 $c(x, y)$ 取最小值后的结果。
当 x 的度数大于 1 时，有：

$$f[y] = D[y] + \begin{cases} \min(f[x] - \min(D[y], c(x, y)), c(x, y)) & y \text{ 的度数大于 } 1 \\ \min(f[x] - c(x, y), c(x, y)) & y \text{ 的度数等于 } 1 \end{cases}$$

x 度数为 1 时：

$$f[y] = D[y] + c(x, y)$$