状态压缩动态规划

李淳风

长郡中学

2024 年 9 月 24 日

前言

我们知道，动态规划的过程是随着“阶段”的增长，在每个状态维度上不断拓展的。在任意时刻，已经求出最优解的状态与尚未求出最优解的状态在各维度上的分界点组成了 DP 拓展的“轮廓”。而对于某些问题，状态之间的转移需要这个“轮廓”的信息，我们可以在动态规划的“状态”中记录一个集合，保存这个“轮廓”的详细信息。

若集合大小不超过 n，且集合中每个元素都是小于 k 的自然数，则我们可以把这个集合看作一个 n 位 k 进制数，以一个 $[0, k^n - 1]$ 之间的十进制整数的形式作为 DP 状态的一维。这种把集合转化为整数记录在 DP 状态中的一类算法，被称为状态压缩动态规划算法。

例题

最短 Hamilton 路径

给定一张 $n(n \leq 20)$ 个点的带权无向图，点从 $0 \sim n-1$ 编号，求起点 0 到终点 $n-1$ 的最短 Hamilton 路径。
Hamilton 路径的定义是从 0 到 $n-1$ 不重不漏地经过每个点恰好一次。

很容易想到本题的一种朴素做法，就是枚举 n 个点的全排列，计算路径长度取最小值，时间复杂度为 $O(n * n!)$。使用二进制状态压缩可以优化到 $O(n^2 * 2^n)$。

最短 Hamilton 路径

由于每个点都只能被经过一次，所以我们的状态中需要记录哪些点已经被经过了，哪些点还没有。我们可以使用一个 n 位二进制数来记录，若其第 $i(0 \leq i < n)$ 位为 1，则表示第 i 个点已经被经过，反之则未被经过。当然，我们还需要知道当前所处的位置，所以我们可以使用 $f[s][j](0 \leq s < 2^n, 0 \leq j < n)$ 表示“点被经过的状态”对应的二进制数为 s，且目前处于点 j 时的最短路径。

整个状态空间可看作 n 维，每维代表一个节点，只有 0（未被访问）和 1（已访问）两个值。我们可以想象 DP 的“轮廓”从 $(0, 0, \cdots, 0)$ 逐步拓展到 $(1, 1, \cdots, 1)$，也就是 s 从 0 逐渐变成 $2^n - 1$。因此我们在进行 DP 的时候，可以从小到大枚举 s 的值，对应着遍历所有“轮廓”。

在起点时，有 $f[1][0] = 0$，即只经过了点 0（s 只有第 0 位为 1），目前处于起点 0，最短路径的长度为 0。初始时其他值为无穷大，我们的目标是 $f[(1 << n) - 1, n - 1]$，即经过所有点（s 的每个二进制位都是 1），处于终点 $n - 1$ 时的最短路径。

最短 Hamilton 路径

状态转移方程为 $f[s][j]=\min\{f[s\ xor\ (1<<j)][k]+weight(k,j)\}$，其中 $((s>>k)\&1)=1$ 并且 $((s>>j)\&1)=1, k\neq j$。
因为每个点只能被经过一次，我们最后停留在 j，所以 j 一定是最后被经过的，故在这之前“被经过的点的状态”为 $s\ xor\ (1<<j)$。另外，在上一时刻所处的位置可能是 $s\ xor\ (1<<j)$ 中任意一个是 1 的数位 k，从 k 走到 j 需经过 $weight(k,j)$ 的路程，枚举所有这样的 k 并计算最短路径。

```
for(int s=1;s<(1<<n);s++)
    for(int j=0;j<n;j++) if((s>>j)&1)
        for(int k=0;k<n;k++) if(j!=k && ((s>>k)&1))
            f[s][j]=min(f[s][j],f[(s^(1<<j))][k]+weight[k][j]);
```

例题

Mondriaan's Dream

求把 $n * m(1 \leq n, m \leq 11)$ 的棋盘分割成若干个 $1 * 2$ 的长方形，有多少种方案。

例题

Mondriaan's Dream

求把 $n * m(1 \leq n, m \leq 11)$ 的棋盘分割成若干个 $1 * 2$ 的长方形，有多少种方案。

对于第 i 行的第 j 个格子，要么和第 $j-1$ 或第 $j+1$ 个格子组成一个横着的长方形，要么和上一行、下一行的格子组成一个竖着的长方形。对于后面的情况，我们可以定义位于 (i, j) 的格子只能向下和 $(i+1, j)$ 组成竖着的长方形，不能向上和 $(i-1, j)$ 组成，也就是只在上端进行考虑，这样可以做到不重不漏。
这样一来，对于任意一种方案，考虑以某一行为界，把整个棋盘横着分成两半。假设以第 i 行下端为分割线，那么对于第 $i-1$ 行，肯定所有格子都被覆盖了，否则就应该在之前考虑；而第 i 行每个格子被覆盖的情况，对于第 $i+1$ 行的填充方案有着较大的影响。因此我们可以把“行号”作为 DP 的阶段，把“上半部分”不断向下拓展，直至确定这个棋盘的分割方法。在过程中，我们需要把第 i 行的格子被覆盖情况用一个 m 位的二进制数存储下来，其中第 k 位为 1 表示第 k 列是一个竖着长方形的上半部分，为 0 则表示其他情况。

Mondriaan's Dream

设 $f[i][j]$ 表示第 i 行的状态为 j 时，前 i 行的分割方案的总数。j 是用十进制记录的 m 位二进制数。
第 $i-1$ 行的形态 k 能转移到第 i 行的形态 j，当且仅当：

- j 和 k 执行按位与运算的结果是 0。这保证了每个数字 1 的下方必须是数字 0，继续补全竖着的 $2*1$ 的长方形；
- j 和 k 执行按位或运算的结果的二进制表示中，每一段连续的 0 都必须有偶数个。这些 0 代表若干个横着的 $1*2$ 的长方形，奇数个 0 无法分割成这种形态。

我们可以在 DP 前预处理出 $[0, 2^m-1]$ 内所有满足“二进制下每一段连续 0 都有偶数个”的整数，记录在集合 S 中。

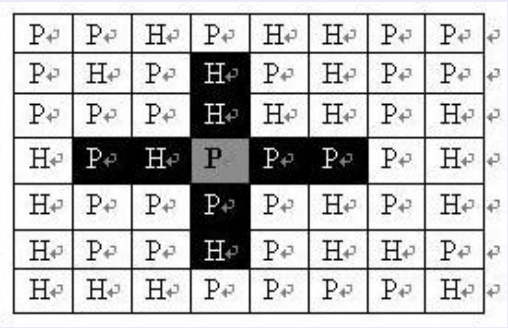
$$f[i][j] = \sum_{j\&k=0且j|k\in S} f[i-1][k]$$

初始时 $f[0][0]=1$，其余均为 0，目标为 $f[n][0]$。时间复杂度为 $O(4^m n)$。

例题

炮兵阵地

司令部的将军们打算在 $N*M$ 的网格地图上部署他们的炮兵部队。
一个 $N*M$ 的地图由 N 行 M 列组成，地图的每一格可能是山地（用 H 表示），也可能是平原（用 P 表示）。
在每一格平原地形上最多可以布置一支炮兵部队（山地上不能够部署炮兵部队）；一支炮兵部队在地图上的攻击范围如图中黑色区域所示：



炮兵的攻击范围为横向两格，纵向两格，且不受地形的影响。
现在，将军们规划如何部署炮兵部队，在任意一支炮兵部队都不在其它炮兵部队的攻击范围内的前提下，最多能部署多少支炮兵部队？
$N \leq 100, M \leq 10$。

炮兵阵地

因为每个位置能否放置炮兵，与它上两行对应位置上是否放置炮兵有关，所以在向第 i 行的状态转移时，需要知道第 $i-1$ 行和第 $i-2$ 行的状态。我们把每一行的状态看作一个 M 位二进制数，其中第 p 位为 1 表示改行第 p 列放置了炮兵，为 0 则表示没有放置炮兵。

我们可以在 DP 前预处理出集合 S，存储“相邻两个 1 的距离不小于 3”的所有 M 位二进制数。另外设 $count(x)$ 表示 M 位二进制数 x 中 1 的个数，$valid(i,x)$ 用于判断 x 属于集合 S，x 中的每个 1 对应在地图第 i 行中的位置是否都是平原。

设 $f[i][j][k]$ 表示第 i 行压缩后的状态为 j，第 $i-1$ 行压缩后的状态为 k 时，前 i 行最多能放置多少炮兵。

若 $valid(i,j), valid(i-1,k)$ 为 1 并且 $j\&k=0$ 时：

$$f[i][j][k]=\max_{j\&l=0}\{f[i-1][k][l]+count(j)\}$$

否则 $f[i][j]$ 为负无穷。目标为

$$\max_{j\&k=0,valid(N,j),valid(N-1,k)}\{f[N][j][k]\}$$

。复杂度看似不对，但实际上 S 集合非常小，只有不到 100 个数。时间复杂度为 $O(N|S|^3)$。

炮兵阵地

根据题意，一个格子放置炮兵以后，该格子上下两行的同一列都不能再放置炮兵。可以用数字 2 表示放置炮兵的格子，同时规定 2 的下面必须是 1，1 的下面必须是 0，0 的下面才可以放置新的炮兵。
把每行的状态压缩为一个 M 位三进制数，设 $f[i][j]$ 表示第 i 行压缩后的状态为 j 时，前 i 行最多能放多少炮兵且不会冲突。
对于每个已经求出的合法的 $f[i][j]$，我们考虑它能转移到哪些状态。这等价于考虑第 $i+1$ 行的每个位置填写什么数字。根据题意，我们可以得到下列要求：

- 第 i 行第 j 列为 2 时，第 $i+1$ 行第 j 列必须为 1；
- 第 i 行第 j 列为 1 时，第 $i+1$ 行第 j 列必须为 0；
- 山地格子不能填 2；
- 一个格子填 2 以后，它右边的两个格子不能填 2。

像这道题这样状态标识复杂、冗余较多的题目，不一定非要写出转移方程，可以通过 DFS 搜索第 $i+1$ 行每个位置填写什么数字，对于搜索出来的合法状态 k，我们从 $f[i][j]$ 转移到 $f[i+1][k]$。
总而言之，整个动态规划算法使用三进制压缩的状态表示，以“行号”为阶段，在相邻两行之间通过 DFS 转移。

例题

宝藏

参与考古挖掘的小明得到了一份藏宝图，藏宝图上标出了 n 个深埋在地下的宝藏屋，也给出了这 n 个宝藏屋之间可供开发的 m 条道路和它们的长度。

小明决心亲自前往挖掘所有宝藏屋中的宝藏。但是，每个宝藏屋距离地面都很远，也就是说，从地面打通一条到某个宝藏屋的道路是很困难的，而开发宝藏屋之间的道路则相对容易很多。

小明的决心感动了考古挖掘的赞助商，赞助商决定免费赞助他打通一条从地面到某个宝藏屋的通道，通往哪个宝藏屋则由小明来决定。

在此基础上，小明还需要考虑如何开凿宝藏屋之间的道路。已经开凿出的道路可以任意通行不消耗代价。每开凿出一条新道路，小明就会与考古队一起挖掘出由该条道路所能到达的宝藏屋的宝藏。另外，小明不想开发无用道路，即两个已经被挖掘过的宝藏屋之间的道路无需再开发。

新开发一条道路的代价是 $L * K$。其中 L 代表这条道路的长度，K 代表从赞助商帮你打通的宝藏屋到这条道路起点的宝藏屋所经过的宝藏屋的数量（包括赞助商帮你打通的宝藏屋和这条道路起点的宝藏屋）。

请你求出工程总代价的最小值。$1 \leq n \leq 12, 0 \leq m \leq 10^3$

宝藏

根据题意，我们需要花费最小的代价让所有宝藏屋连通，那么最终修建的道路必然是一棵树，“赞助商帮你打通的宝藏屋”就是这棵树的根。我们考虑朴素的搜索算法，先确定一个宝藏屋作为树的根节点，每次再枚举一个已经打通的宝藏屋 x，从 x 出发开凿一条新道路，到达一个未被打通的宝藏屋 y。打通 y 的代价为道路 (x, y) 的长度乘以节点 x 的深度。

注意到该算法遍历了相当多的重复状态，我们可以做出一些优化：

- 先打通浅的宝藏屋，再打通深的宝藏屋。因为在合法的前提下，对于同一个树形结构，打通宝藏屋的代价与顺序无关。
- 设已打通的宝藏屋集合为 S。对于相同的集合 S，我们只关心代价最小的那一个。因为打通后续宝藏屋的代价与 S 的内部连接方式无关，只要集合 S 是连通的就够了。

宝藏

我们发现，这么考虑之后，已经可以设计状态了。设 $f[i][j]$ 表示已经打通的宝藏屋最大深度为 i，宝藏屋的打通状态为 n 位二进制数 j（0 表示未被打通，1 表示已被打通）时，花费的最小代价。状态转移方程为：

$$f[i][j] = \min_{valid(k,j)} \{f[i-1][k] + (i-1) * cost(k,j)\}$$

初始时 $f[1][1 << (x-1)] = 0$，其余为正无穷。
目标为 $\min_{1 \le i \le n}\{f[i][(1 << n) - 1]\}$。
这是一个按层 DP 的方程，每次考虑第 i 层比第 $i-1$ 层新增了哪些节点。其中 $valid(k,j)$ 是判断状态 k 能否通过增加若干条道路变为状态 j，且只拓展一层。$cost(k,j)$ 表示增加的道路长度总和的最小值。这两个函数我们都是可以预处理出来的，之后就可以进行 DP 了。
因为我们是按层数 DP 的，所以每次拓展新的一层时，我们是默认 k 中所有节点都可以拓展，并且深度为 $i-1$。这并不影响答案，因为深度小于 $i-1$ 的节点在更早的时候拓展的情况已经在其它状态中计算过了，所以这样默认不会干扰最小值的计算。
整个算法的时间复杂度为 $O(N * 3^N + M * 2^N)$。3^N 是所有 2^N 个状态的子集总数，$M * 2^N$ 是预处理两个函数消耗的时间。

宝藏

我们考虑一种每次拓展一个节点的方法。因为计算代价时需要知道道路起点的深度，所以我们仍然以深度为阶段按层 DP，每次尝试拓展一个前 $i-1$ 层的节点。为了区分已经打通的节点中，哪些是前 $i-1$ 层的，哪些是第 i 层的，可以采用三进制状态压缩动态规划。

设 $f[i][j]$ 表示前 i 层，节点的打通状态为 j 时，已经耗费的最小代价。j 是一个三进制数，它的第 $x-1$ 位为 0 表示节点 x 尚未被打通，为 1 表示 x 是在前 i 层打通的，为 2 表示节点 x 是通过第 i 层开辟一条通完第 $i+1$ 层的道路而打通的。

初值为 $f[1][3^x-1]=0$，其余为正无穷。目标值为 $f[n+1][(111\ldots1)_3]$。

对于每个已经被求出的 $f[i][j]$，考虑它能转移到哪些状态。

- 不再从第 i 层开辟道路，进入下一层。设 s 是把 j 中所有为 2 的位改为 1 得到的三进制数，用 $f[i][j]$ 更新 $f[i+1][s]$。
- 考虑拓展 j 的最高位的 1 对应的节点 x。枚举从 x 出发的所有道路，若到达一个未被打通的节点 y，则用 $f[i][j]+i*length(x,y)$ 更新 $f[i][j+2*3^{y-1}]$。
- 不再拓展 j 的最高位的 1 对应的节点 x。可以把该位赋值为 2 表示不再在这一层拓展，用 $f[i][j]$ 更新 $f[i][j+3^{x-1}]$。

最终时间复杂度为 $O(N^2*3^N)$。