动态规划

算法设计与分析小班课

2022 年 3 月

注意: 以下内容主要是给课本作一些注解并补充习题, 可能不能准确地代表课程内容和考核的方向, 也不能替代阅读课本、听课、完成作业题、做往年题等活动; 其中的内容没有经过课程主管老师的审核, 也可能存在错误. 不过出现的所有错误都由作者本人负责.

1 务虚: 设计思想

动态规划是一种游走于暴力搜索边缘的算法. 通过聪明地分解子问题 (的状态) 并记忆和合并子问题的解, 可以为一些解空间很大的问题设计出多项式时间或者是伪多项式时间的算法. 因为它实质上是采用了一种系统的方法搜索整个空间, 这种 "系统的方法" 有时候比较巧妙, 所以我们往往需要一些练习才能彻底熟悉它.

$$\text{DP} = \text{子问题 DAG} + \text{猜测部分解} + \text{记忆化} \approx \text{最短路}.$$

以下给出了一个 "三步骤法".

1. 确定合适的子问题. 子问题中最大的不一定就是目标问题, 最终的解也可以是全部子问题的某种合并. 子问题的数目应当是多项式. 可以通过直接分析问题结构来得出子问题, 也可以先猜一些自然的, 在下一步修正.
2. 确定在每一个状态出发应该如何猜测解, 并利用已有的子问题递归; 注意不仅要化归子问题, 还需要在此过程中将状态的结构联系在一起. 这个利用过程必须是无环的 (比如说, Dijkstra 算法中如果有负回路, 则 DP DAG 中有环). 对于比较复杂的问题, 此时还可以修正之前的子问题设计.
3. 写出状态转移方程, 规定初始解, 导出追踪解的方法.

比如说, 对于字符串或者序列的问题, 子问题往往是前缀 $x[..i]$, 后缀 $x[i..]$ 或者 $x[i..j]$, 在此基础上可以添加新的状态变量来辅助递推.

追踪解的方法是机械的, 不需要思考. 方法就是在计算 max, min 的时候顺便记录一下 argmin, argmax. 比如说, 在 DP[...] 中增加最后一维, 其中一个分量指向上一个状态.

通过分析问题结构来设计子问题可能会更舒服: 我们考虑在贪心法中已经看到的活动选择问题的推广.

问题 1 有 n 项活动同时申请使用同一个礼堂, 每个活动 i 都有一个开始时间 s_i 和结束时间 f_i, 任何两个活动不能同时举行 (端点除外). 举办活动 i 的收益是 $v_i \in \mathbb{R}_{\geq 0}$. 设计一个算法选择一些活动, 使得收益最大.

解 对于一个活动, 只有选和不选两个可能. 沿用不带权的问题的方法, 我们将活动按结束时间递增的顺序排序 $f_1 \leq f_2 \leq \cdots \leq f_n$. 利用 DP 的重要观察是:

断言 如果选择活动 j, 那么最优解一定包含子问题——活动 $1, 2, \ldots, p(j)$ 的最优解, 其中 $p(j)$ 表示结束时间最晚的和 j 不重叠的活动 (不存在的话则规定为 0); 如果不选择活动 j, 那么最优解一定包含子问题——活动 $1, 2, \ldots, j-1$ 的最优解.

事实上, 如果选择 j 而解不包含活动 $1, 2, \ldots, p(j)$ 的最优解, 则可以换上一个最优解, 仍然没有冲突; 不选择 j 的论证也是同理. 所以

$$\mathsf{OPT}[..j] = \max\{v_j + \mathsf{OPT}[..p(j)] + \mathsf{OPT}[..j-1]\}.$$

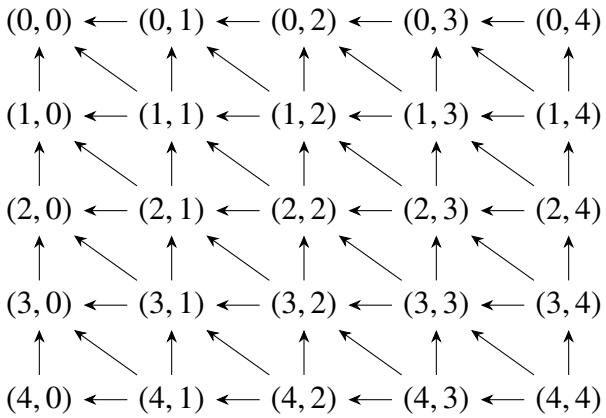
规定 $\mathsf{OPT}[..0] = 0$. 这样直接从 0~j 进行迭代即可. 追踪解的方法: 记录 $\mathsf{OPT}[..j]$ 是否选择 j 即可——父指针指向 $p(j)$ 或者 $j-1$. □

以上问题只有一个状态变量. 课本上讲到的背包问题是增加一个状态变量的典型例子, 不过需要注意的是, DP 给出的算法仍然是指数级的, 因为背包容量 B 是以二进制编码输入的.

在多个变量的 DP 中, 依赖关系 DAG 的形状则变化很大. 举一个经典问题的例子: 给出字符串 x, y, 每次可以删除一个字符, 插入一个字符或者修改一个字符, 分别产生一定的花费, 问如何花费最小地把 x 修改为 y. 利用 x, y 的前缀作为状态变量, 此问题的递推方程明显是:

$$d(i,j) = \min\{d(i-1,j) + c(\text{插入}), d(i,j-1) + c(\text{删除}), d(i-1,j-1) + c(\text{替换}(x[i], y[j]))\}.$$

DAG 像下面这样:



使用父指针的方式来追踪解时, 就是在上面的 DAG 中找了一条最短路.

问题 2 在一个空的, 宽度为 $w = O(1)$ 的俄罗斯方块的版上玩俄罗斯方块, 但排成整行的方块不会自动消除. 假设已经顺次给定了 n 块将要落下的方块的形状, 设计算法判定: 最后累计的方块高度是否可以不超过 $h = O(1)$.

解 如果采用暴力求解的方式, 就是要决定每块放在什么地方, 这样会得到 $n^{O(w)}$ 的算法. 以下给出一个 (实际上也没有太大改进的) 动态规划算法. 动态规划当然也要猜测方块放在什么地方. 但是如果只设置一个状态变量是不足以确定如何递推的. 子问题为当 $x = 0, 1, \dots, w-1$ 时的方块高度分别为 $h_0, h_1, \dots, h_{w-1}$ 时, 是否可能完成任务. 则子问题的个数为 $O(nh^w)$, 递推关系为

$$\mathsf{DP}[i.., h_0, \dots, h_{w-1}] = \bigvee_{j=0}^{w-1} \bigvee_{\text{方向}} \mathsf{DP}[i+1.., h'_0, \dots, h'_{w-1} \mid \text{把第 } i \text{ 块适当旋转后放在位置 } j \text{ 得到的状态}].$$

所需时间为 $O(nwh^w)$. 追踪解的方法仍然是用父指针. □

动态规划也可以用来解一些其他的小游戏, 比如超级玛丽奥, 等等. 不过今后会看到, 如果 w 不是常数, 而且一开始的版不一定是空的, 则确定是否有使得高度不超过 h 的问题是 NP-完全的.

以下例子的对比有关动态规划中状态维数的选择问题, 可以思考一下.

例 3 考虑如下问题. 一些天文学家打算进行 n 个天象的观测活动. 第 i 个天象会在第 i 分钟, 一维坐标 $d_i \in \mathbb{Z}$ 处发生. 假设望远镜一开始在 0 处, 只有在望远镜移动到 d_i 时才能看到 i; 天象瞬间发生, 错过的天象就看不到了. 望远镜的移动速度是每分钟 1 个单位. 问这些天文学家最多能观测到几个天象? 设计算法解决之. 假如要求最后一个天象必须观测到, 算法是否可以更快? ◇

2 动态规划的补充例题

例题大多来源于 [KT06, 第 6 章]. 许多问题的抽象模型都是一样的, 而且都比较简单, 但是都选进来了; 不太熟悉的同学可以看一看. 以下均省略初始条件和追踪解的方法, 它们都可以机械地给出 (应考时必须要写出); 但如果要追踪解, 时间复杂度可能会稍微上升 (解答里都没有考虑).

2.1 一维问题

一维问题的结构通常可以用首步或者末步分析法找出. 也可以直接猜测状态变量.

问题 4 在一般的图中找最大独立集是困难的问题. 考虑在一个 n 个顶点的路径 P_n 上找最大独立集: 其中每个顶点都有一个权, 要求挑选出的独立集顶点权值最大.

解 非常简单. 用 DP[..i] 表示结果, 那么最优解如果选了 i, 则一定包含 DP[..$i-2$] 的最优解, 否则一定包含 DP[..$i-1$] 的最优解, 即

$$\mathsf{DP}[..i] = \max\{\mathsf{DP}[..i-2] + w_i, \mathsf{DP}[..i-1]\}.$$

□

问题 5 期末季有 n 天, 你有一些复习任务要完成. 任务分为两种, 一种是水课, 一种是虐课. 假设在第 i 天, 进行水课复习的收益是 $\ell_i > 0$, 进行虐课复习的收益是 $h_i > 0$, 摸鱼的收益是 0. 假设复习虐课的那天的前一天必须是摸鱼的, 以便于敦促自己复习; 但水课可以连续地复习. 给定 $n; h_1, \ldots, h_n; \ell_1, \ldots, \ell_n$, 设计一个算法确定收益最高的工作方案.

解 这和前一题是几乎是完全一样的, 只需注意到每个前缀子问题中, 最后一天一定不可能摸鱼, 所以

$$\mathsf{DP}[..i] = \max\{\mathsf{DP}[..i-2] + h_i, \mathsf{DP}[..i-1] + \ell_i\}.$$

□

注 6 类似问题: 假设你经营的公司在接下来的 n 周中, 第 i 周生产 w_i 吨货物. 这些货物可以交给 A 公司或者 B 公司承运. A 公司每吨货物收取 r_i 的费用, 而 B 公司则采用包月服务: 必须连续以连续使用四周为单位, 每次签订合同需要收费常数 C. 设计一个算法安排货运公司使得费用最低. ♤

问题 7 小明玩塔防游戏, 在长度为 n 秒的时间段中, 第 i 秒有 x_i 个机器人对其塔发起攻击. 游戏中他控制着一个电磁炮, 电磁炮的威力取决于它空闲了的时间的长度. 当空闲了 j 秒时, 它可以消灭 $f(j)$ 个机器人 (第 i 秒所开的炮不能消灭在第 i 秒之前出现的机器人, 即消灭的机器人数是 $\min\{x_i, f(j)\}$). 问: 小明应该在哪些时刻开炮才能尽可能多地消灭机器人?

解 这和前一题所用到的观察也一样. 我们注意到, 第 n 秒一定要开炮, 因此只需要确定上一次开炮的时间, 便化归为子问题, 递推方程为

$$\mathsf{OPT}[j] = \max_{1 \le i < j} \{\min\{f(j-i), x_i\} + \mathsf{OPT}[i]\}.$$

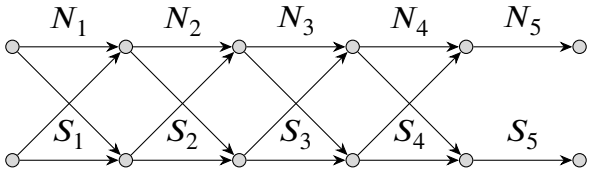
计算时间为 $O(n^2)$. □

注 8 类似问题: 考虑在服务器 $S_1, \ldots, S_n$ 上放置同一文件的副本, 放在 S_i 上的存储代价是 c_i. 当在 S_i 上检索这个文件时, 如果上面有文件, 则检索代价为零, 否则会在 $S_{i+1}, S_{i+2}, \ldots$ 顺序搜索直到 S_j 中找到, 检索代价为 $j - i$. 我们要求 S_n 上必须放置一个副本. 设计算法给出一个放置方案, 使得当对 $S_1, \ldots, S_n$ 都检索一次时, 总的存储和检索代价最小. ♤

问题 9 假设你的公司在北京和上海各有一个办事处. 每个月你在北京的花费是 N_i, 在

上海的花费是 S_i, 在上海和北京之间的一次交通花费是 M. 设计在 n 个月内的办事地点方案, 使得花销最小.

解 这和我们在课上讲到的钢琴指法问题一样, 就是在下面的图上求一个最短路 (交叉的边的花费除了 N/S_i 外还要补上 M). 同样, 状态变量还需要多一维, 那就是在第 i 个月的办公地点.



因此递推方程为

$$\begin{cases} \mathsf{DP}[i][\mathrm{N}] = N_i + \max\{M + \mathsf{DP}[i-1][\mathrm{S}], \mathsf{DP}[i-1][\mathrm{N}]\}, \\ \mathsf{DP}[i][\mathrm{S}] = S_i + \max\{M + \mathsf{DP}[i-1][\mathrm{N}], \mathsf{DP}[i-1][\mathrm{S}]\}. \end{cases}$$

□

问题 10 假设你有一个需要进行计算的任务, 这个任务两个运算核心 A, B 都可以完成. 由于负载变化, 在 n 分钟内的每一分钟, A, B 分别能执行的运算步数分别由 a_i, b_i 给出. 运算任务在两个核心之间的一次转移需要 1 分钟. 设计一个调度方案, 使得 n 分钟内执行的步数最多.

解 这个问题有点像上面一题的对偶, 但其实还是一样的. 观察还是: 因为最后一分钟必然是在运算, 所以递推方程为

$$\begin{cases} \mathsf{DP}[i][A] = a_i + \max\{\mathsf{DP}[i-1][A], \mathsf{DP}[i-2][B]\}, \\ \mathsf{DP}[i][B] = b_i + \max\{\mathsf{DP}[i-1][B], \mathsf{DP}[i-2][A]\}. \end{cases}$$

□

问题 11 因为中文书写时词语没有空格, 所以分词问题是自然语言处理的一个重要话题. 比如我们希望计算机在分析时把 “保卫部通知今天下午进行消防演练” 拆分为 “保卫部 通知 今天 下午 进行 消防 演练”.

假设现在有一个评估算法 $\mathscr{A}$, 它能在 $O(1)$ 时间内判断一个片段是否像一个正常的词 (比如从词典里找得到的就是正常词), 给出一个信心指数 $\mathscr{A}(w) \in \mathbb{R}$ (可以是负数). 现在给出一个长 n 的汉字序列, 如何将它分为词语 $w_1, \ldots, w_k$, 使得 $\sum_{i=1}^{k} \mathscr{A}(w_i)$ 最大[1]?

[1] 自然语言处理中的实际技术比本题复杂许多, 还要考虑很多其他因素, 特别是分出来的词语是否能组成合理的短语和句子. 例如 “研究生命的起源”, 虽然 “研究生 命 的 起源” 都是合理的词语, 但是合起来不是自然的短语.

解 这和矩阵乘法加括号是同一类问题, 不过切分可能性少一些, 所以平方时间就能完成. 假设 $\mathsf{OPT}[..i]$ 表示前 i 个字的最佳分词, 那么

$$\mathsf{OPT}[..i] = \max_{j \le i} \{\mathsf{OPT}[..j-1] + \mathscr{A}(j..i)\}.$$

注意到最优解中去掉最后一个词之后, 剩余部分仍然是最佳分词就可得出上式. □

问题 12 在终端中显示的文本最好整齐一些. 对于分别由 $w_1, \dots, w_n$ 个字母组成的英文单词序列, 我们想把它们排成适当的行, 要求每一行的单词之间有一个空格 (最后一个单词后没有空格), 而且每一行的长度不超过 L ($L \ge \max\{w_1, \dots, w_n\}$). 在此基础上, 希望最小化每行长度和 L 的差值的和. 设计一个算法做这件事.

解 这和上一题的做法完全一样, 区别在于 $\mathscr{A}$ 需要预处理给出 ($\mathscr{A}(i,j) = L - (w_i + \cdots + w_j)$), 其中长度超出 L 的行的权值为 ∞, 最大化改为最小化. □

问题 13 在移动网络中, 运营商往往要确定两部移动设备之间的通信路径. 由于设备在移动, 因此这些路径是会发生变化的. 一方面, 我们希望维护最短路径; 另一方面, 我们不希望这个路径变化太频繁, 以至于给路由算法带来太大负担.

对于这个问题采用如下模型. 给定顶点集 V 和随时间变化为边集 $E_1, \dots, E_t$ 并令 $G_i = (V, E_i)$. 假设这些图都是连通的. 给定 $s, t \in V$ 和实数 $K > 0$, 求 s,t-路径族 $P_1 \in G_1, \dots, P_n \in G_t$, 最小化以下目标函数:

$$\sum_{i=1}^{t} \ell(P_i) + K \cdot \sum_{i=1}^{t-1} \mathbf{1}_{P_i \ne P_{i+1}},$$

其中 $\ell(\cdot)$ 表示路径长度.

解 对于新的边的变化, 我们可以尝试使用旧有的最短路, 也可以重新计算最短路, 因此需要猜测的是什么时候更换新路径. 若记 $\ell(i,j)$ 为 $G_i, \dots, G_j$ 中共同存在的路径中最短的长度 (不存在则为 ∞), 则我们有

$$\mathsf{OPT}[..j] = \min\{j\ell(1,j), \min_{1 \le i < j} \mathsf{OPT}[..i] + K + (j-i)\ell(i+1,j)\}.$$

主要的时间是花费在预处理 $\ell(i,j)$ 上. 最简单的方法是用 $O(|V|^2 t)$ 的时间算出公共子图, 然后在这 $O(t^2)$ 对中计算最短路. □

2.2 超过一维的问题

背包问题是这类问题的经典模型.

问题 14 期末季你有 n 门考试, 有 $T \in \mathbb{Z}_{\ge 0}$ 小时的复习时间. 对于课程 i, 复习 $t \in \mathbb{Z}_{\ge 0}$ 小时的预期绩点由函数 (可以不是递增的) $f_i : \mathbb{Z}_{\ge 0} \longrightarrow \{0\} \cup [1,4]$ 给出. 设计一个算法给出一个时间投入安排, 使得最后的平均绩点最高.

解 我们只需要猜测每门课复习多少时间, 辅助状态变量为剩下多少小时, 因此若以 $\mathsf{OPT}[i,t]$ 表示在 t 小时内复习前 i 门课得到的最高累积绩点, 那么

$$\mathsf{OPT}[i,t] = \max_{0\le k\le t} f_i(k) + \mathsf{OPT}[i-1, t-k].$$

所需时间为 $O(nH^2)$. □

问题 15 某人经营一家售卖汽车的公司. 在未来的 n 个月中, 第 i 个月 (的月初) 会收到 d_i 辆车的订单. 一方面, 每次进货需要花费 K 的资金 (不论进多少辆); 另一方面, 有一个能存储 S 辆卡车的仓库, 该仓库每个月存储一辆卡车需消耗资金 C. 问: 如何进货使得订单都被满足, 而且花费最少?

解 我们只需要猜测每个月月初需要订多少货. 用 $\mathsf{OPT}[i,s]$ 表示在第 $..i$ 个月满足所有需求, 剩余 s 辆车在仓库中的方案. 假设之前那个月的状态是 $[i,z]$, 那么除了储存费用 Cz 之外, 如果 $z-d_i<s$, 还需要进货费用 K. 所以,

$$\begin{aligned}\mathsf{OPT}[i,s] &= \min\begin{cases}\mathsf{OPT}[i-1, s+d_i] + C(s+d_i) & (\text{如果 } s+d_i\le S),\\ \min_{z<\min\{s+d_i,S\}} \mathsf{OPT}[i-1,z] + zC + K & (\text{否则}).\end{cases}\\ &= \min\begin{cases}\mathsf{OPT}[i-1, s+d_i] + C(s+d_i) & (\text{如果 } s+d_i\le S),\\ \mathsf{OPT}[i-1,0] + K & (\text{否则}).\end{cases}\end{aligned}$$

以上第二个等式是因为 $\mathsf{OPT}[i,z]$ 对 z 是单调递增的, 所以可以取 $z=0$ (如果要重新订货的话, 剩下多出来的车也没必要花钱存储). 这样在 $O(nS)$ 的时间内便得到答案. □

注 16 类似问题: 某人经营一家石油公司. 在未来的 n 个月中, 第 i 个月 (的月初) 会收到 d_i 吨汽油的订单. 一方面, 每次进货需要花费 K 运费以及进汽油的费用 (进得越多则花费越多); 另一方面, 有一个能存储 L 吨汽油的仓库, 该仓库每个月存储每吨汽油需消耗资金 C. 问: 如何进货使得订单都被满足, 而且花费最少 (要求始末态仓库都是空的)? ♤

问题 17 考虑调度问题, 但这次优化目标不同. 有 n 项任务, 它们分别需要 t_i 的时间完成, 而且有一个截止时间 ddl_i. 现在希望选择它们中的一部分并安排在互不重叠的时间内完成之, 目标函数为完成任务的数目.

(1) 证明: 存在一个最优解, 选择出来的任务的调度顺序是按截止时间递增排列的.

(2) 设计算法完成优化.

解 对于 (1), 我们在最优解的基础上要求最小化延迟最大的任务即可, 通过标准的贪心法论证 (见课本例 4.3 的交换技巧) 就得到结论.

利用 (1) 的观察便可以容易地做出 (2). 先将任务按 ddl 递增排序, 再用 $\mathsf{OPT}[i,d]$ 表示在 d 之前安排前 i 个任务中的某些的最优解, 则我们要猜测的是是否要安排第 i 个任

务. 如果不安排, 问题变为 $\mathsf{OPT}[i-1,d]$, 否则变为 $\mathsf{OPT}[i-1,d-t_i]+1$. 算法需要 $O(nD)$ 时间. □

问题 18 某人通过模型预测了 n 天内, 加里敦大学的股价分别为 $p_1,\dots,p_n$. 假设预测准确, 他打算在这 n 天内买卖 k 轮 (k 需要优化): 挑选时间 $1 \le b_1 < s_1 < \cdots < b_k < s_k \le n$, 分别在 b_t 时买入, 在 s_t 时卖出相应股份. 求一个 k 和相应的交易时间安排, 最大化收益

$$\sum_{i=1}^{k} p_{s_i} - p_{b_i}.$$

解 我们需要猜测的是最后一个套利交易是在哪个时间段发生的, 也就是说, 令 $\mathsf{OPT}[m,d]$ 表示前 d 天进行 m 轮买卖操作的最大收益 (初始化时, 容不下 m 轮的 d 均对应 $-\infty$), 那么

$$\mathsf{OPT}[m,d] = \max_{1\le i<j\le d} Q[i,j] + \mathsf{OPT}[m-1,i-1].$$

这里 $Q[i,j]$ 表示在 i,j 之间买卖一次的最大收益. 问题化归为求解之. 这个预处理总之可以在 $\mathsf{poly}(n)$ 的时间内做出来, 不过仔细一点的话, 注意

$$Q[i,j] = \max\{p_j - p_i, Q[i+1,j], Q[i,j-1]\},$$

则 $O(n^2)$ 可以完成. 总之, 需要 $O(kn^2)$ 的时间. □

问题 19 我们称字符串 x' 是 x 的**重复**, 如果 x' 的一个前缀是 $xxx\cdots$. 称字符串 s 是 x,y 的**交叠**, 如果 s 能划分成两个子序列, 二者分别是 x,y 的重复. 某人在监听公海上两艘船之间的通讯序列 s, 假设两艘船想要传送的信号都反复发送, 他希望设计算法判断 s 是不是某两个给定字符串 x,y 的交叠. 例如, $s=100010101, x=101, y=00$ 满足条件, 因为 s 的 1, 2, 5, 7, 8, 9 位构成 101101, 剩余位构成 000.

解 这和编辑距离有一些相像. 不妨设 $|x|=|y|=|s|$, 否则我们把 x,y 都不断重复加长到和 s 一样长, 只需要判断 s 是否位 x,y 的前缀的交叠就可以, 不用考虑重复问题.

现在, 若 $s[..j]$ 是 x,y 的交叠, 那么 $s[j]$ 一定来自于 x,y, 去掉这个字符之后得到结构完全一样的子问题 (针对 s,x,y 的前缀). 若令 $\mathsf{DP}[i,j]$ 表示 $s[..i+j]$ 是否为 $x[..i]$ 和 $y[..j]$ 的交叠, 那么

$$\mathsf{DP}[i,j] = (\mathsf{DP}[i-1,j] \wedge x[i-1] = s[i+j]) \vee (\mathsf{DP}[i,j-1] \wedge y[j-1] = s[i+j]).$$

递推时, $\mathsf{DP}[0,0]=1$, 以 $k=i+j$ 为最外层变量向外递推. □

问题 20 某股市大亨持有 y 股的加里敦大学股票. 近期, 加里敦大学丑闻连连, 股价大跌, 他希望将这些股票在 n 天内全部抛售. 已知如果他不抛售, 则未来 n 天股价为

$p_1, \dots, p_n$. 然而, 鉴于 y 很大, 每抛售 x 股, 股价就会下跌 $f(x)$; 下跌值会带来长期影响. 例如, 如果第一天抛出 y_1 股, 则收回 $y_1(p_1 - f(y_1))$ 的资金, 而第二天抛出 y_2 股则收回 $y_2(p_2 - f(y_1) - f(y_2))$, 等等. 设计算法找出一个最优的抛售方案, 使得回笼资金最多.

解 仔细考察一下使用动态规划的条件可以发现子问题应该取后缀 (这样某日卖出股票产生的损失在后面一直生效). 记 $\mathsf{OPT}[i,k]$ 表示 i 日及之后, 卖出剩余 k 股的最大回笼资金, 则

$$\mathsf{OPT}[i,k] = \min_{x \le k} p_i x - k f(x) + \mathsf{OPT}[i+1, k-x].$$

注意我们在递推式中将卖出后每天的损失统一归入当天中, 这样方便计算. 需要的时间为 $O(y^2 n)$. 这是多项式时间的算法 (而不是伪多项式), 因为 $f(\cdot)$ 的输入规模为 $O(y)$. □

问题 21 在民主社会中, 选区划分是一个非常重要的问题 (参看维基百科条目 "格里蝾螈" 和 [Ell21] 中更数学的介绍). 事实证明, 可以通过一些特别的方式划分选区来操纵选举, 从而使得选举结果有利于某个政党.

考虑如下简化模型. 有 n 个村落 $P_1, \dots, P_n$, 每个地区都有 m 位选民. 已经通过民调知道了每个村落中支持 C 党和支持 K 党的人数 (没有无党派人士). 我们希望把这 n 个村落划分为两个选区, 每边各有 $n/2$ 个村落. 称这 n 个村落是 "可能被操纵的", 如果存在一种划分方法, 使得两个选取支持某一个党的选民数都占多数. 比方说, 在 $n = 4$ 和如下民调数据下

村落	1	2	3	4
C 党支持率	55%	43%	60%	47%
K 党支持率	45%	57%	40%	53%

我们可以把 1, 4 和 2, 3 分别划分为两个选区, 这样尽管 C 党的总支持率为 51.25%, 并不算明显, 但在两个选区却都能获胜. 设计一个算法判断 C 党是否可能出现选举操纵.

解 我们需要猜测每个村落的分配. 问题的解就是选区划分, 这个方案的一个限制是需要平衡两边选区的村落数, 因此需要一个状态变量. 另外, 为了判定 C 党是否可能操纵选举, 还需要用到两个选区的支持率, 所以考虑如下状态设计: $\mathsf{DP}[i,p,x,y]$ 表示前 i 个村落中, 把 p 个村落划给选区 1, 使得 C 党在选区 1 中的支持人数为 x, 在选区 2 中的支持人数为 y 是否可能, 所以

$$\mathsf{DP}[i+1,p,x,y] = \{\mathsf{DP}[i,p-1,x-z,y] \vee \mathsf{DP}[i,p,x,y-z].$$

这个表需要 $O(n^2 m^2)$ 时间建立. 最后扫描整个表, 检查是否有 $\mathsf{DP}[n, n/2, x, y]$ 使得 $x, y \ge mn/4$ 即可. □

参考文献

动态规划是上世纪五十年代由 Bellman 发明的, 比较原始的版本出现在最优控制问题中 (是一个连续数学的问题). Bellman 曾经说他发明 “动态规划” 这个词的原因是他不想让人知道他用国防部的经费做着数学研究——“...to hide the fact that he was doing mathematical research at RAND under a Secretary of Defense who ‘had a pathological fear and hatred of the term, research’. ...It would be difficult to give a ‘pejorative meaning’ and because ‘it was something not even a Congressman could object to’.”

有一个比较经典的组合优化问题, 即 Steiner 树, 其中图上的版本可以用动态规划的方法解决. 详细内容可以看 [DW71; Fuc+07].

[DW71] S. E. Dreyfus and R. A. Wagner. “The Steiner problem in graphs”. 刊于: *Networks* 1.3 (1971), pp. 195–207.

[Ell21] J. Ellenberg. “Geometry, inference, complexity, and democracy”. 刊于: *Bulletin of the American Mathematical Society* 58.1 (2021), pp. 57–77.

[Fuc+07] B. Fuchs et al. “Dynamic programming for minimum Steiner trees”. 刊于: *Theory of Computing Systems* 41.3 (2007), pp. 493–500.

[KT06] J. Kleinberg and É. Tardos. *Algorithm Design*. Pearson Education, 2006. ISBN: 978-0-32-129535-4.

编写: WC
E-mail: wchang@pku.edu.cn