SHOPPING 解题报告

谷晟

1 题目大意

某国有 n 种钞票，最小面值为 1。你手里有一些钱，需要处理若干的以下几种事件。

1. Pay 事件。你需要付 x 元钱。可以用手里的钞票任意组合凑钱。商店收银员能提供无限的所有种类钞票，按面值从大到小贪心找零。
2. Receive 事件。收到 x 元钱，需要找一个 ATM 取出来。ATM 也是能提供无限数量的钞票，贪心出钞，但能提供的面值类型有限制（所有 ATM 都能提供 1 元的钞票）。必须任意选择一台 ATM 把收到的钱全部取出来。
3. Banknote 事件。发行了一种新的面值的纸币。
4. ATM 事件。建造了一台新的 ATM，并得知其能提供的面值列表。

目标是最小化收银员找零的钞票张数总和。

所有询问强制在线。

事件数不超过 10^4，面值不超过 10^6，每种面值初始钱数不超过 10^4，面值和 ATM 机数量不超过 61，每次交易金额不超过 10^9。

2 算法分析

2.1 关于本题交互方式和特殊优化方法

本题询问之间具有较强的相关性，又是强制在线，选手的程序几乎不可能达到最优解，难以使用常见的全局最优算法，如动态规划、搜索等。因此，对于本题来说，一个优秀的算法应该在处理各种形态（随机）数据上具有较好的平均表现。

而正式比赛时，获得高分的程序基本上是“预知数据，针对数据优化”，即比赛时多次提交，通过反馈信息推断数据特征，针对数据调整策略，淡化“强制在线”的条件。这样做确实比较容易在比赛中获得较高的分数，但通用性差，不适用于新生成的数据。本文接下来不讨论这种方法。

2.2 Receive 策略

对于“取钱”事件，一般有两种策略。

一种想法是，如果手上各种面值的钞票比较均匀，则支付各种金额的准确度会比较高（全是大钞，避免不了找零；小钞很多但缺少中等面值的钞票，容易出现可凑到的金额存在“断层”的情况）。设取款前各种面值的数量为 n_i，取款后为 m_i，则选择 ATM 时最大化

$$\sum \frac{(m_i+4)}{(n_i+4)}$$

可以获得一组分布较为均匀的钞票（常数 4 用于防止除以 0）。

另一种想法更简单，因为每次收款和付款的金额是固定的，在每个事件点手里的钞票的总额也是一定的，不同的只有面值的分布。一般来说，小面值钞票的自由度更高，而面值越小张数越多，这时只需最大化

$$\sum m_i$$

，由于 1 元的钞票通用性非常强，多一些总是有好处的，所以在运用这种策略时可以给 1 元钞票的数量乘上一个权值。

经过实验，如果付款策略比较简单，第一种策略具有显著的优化效果，但在付款策略高度优化时，效果反而不如第二种更简单的策略。

2.3 Pay 策略

对于“付款”事件，可以优化的余地更大。有以下几种思考方向。

2.3.1 简单贪心

最简单的策略。从大到小贪心地选用手里的钞票，对于最后剩下的一些，再补上一张大面值的钞票。使用这种策略，结合上面的第一种取钱策略，在 CodeChef 上的分数约为 307000(0.025pts)。（CodeChef 上反馈的数值是各个测试点的平均找零张数，越小越好；最终分数的计算为反比例函数；截至本文写完，本题 CodeChef 上 Practice 最优解和正式比赛中最优解分数几乎相同，所以以上 pt 值可以看作是与比赛时最优解的比值）

2.3.2 进阶贪心

上面的贪心算法还有很大的优化余地。比如先从大到小选，再补上一张大的，很可能会浪费一些 1 元钞票；再是不一定要补上一张大面值的钞票，也可以补上若干张中等面值的钞票。这里可以有许多种优化方案，比如在一定范围内枚举补上的钞票的张数，再用预处理 +RMQ 的方式计算收回几张 1 元钞票时找零张数最少。贪心算法优化后，在 CodeChef 上的分数约为 115600(0.066pts)。

2.3.3 精确付款

在某些情形下，比如手上的小面值钞票充裕，或者剩余的事件不多的情况下，最直接的减少找零的方法是精确付款，不找零。即寻找一组钞票（用 C_i 代表使用第 i 种面值钞票的张数，V_i 为对应面值，X 为应付金额），要求 $\sum V_i \times C_i = X$。

由于 1 元钞票通用性强，我们把它分开考虑。则上述精确付款问题变为，寻找一组钞票（不包括 1 元面值），在 $S = \sum V_i \times C_i \leq X$ 的限制下最大化这组钞票的总金额 S，$X - S$ 为所需的 1 元钞票张数，不仅更容易判断有无解，还顺便保留了更多有用的 1 元钞票。

这是个背包问题，数据范围小时可以 DP 求解，数据范围大时可以使用多次随机选取或模拟退火近似求解。

使用“模拟退火计算精确付款较优解 $\rightarrow$ 无解则优化贪心”的策略，在 CodeChef 上的分数约为 55100(0.138pts)。

2.3.4 近似精确付款

精确付款的可应用范围还是比较窄的，因为其条件十分苛刻，即使成功实现，也可能会大量消耗小面值钞票使接下来的付款陷入困境。考虑放宽限制，允许

1 张钞票（或更宽松的情况，比较复杂）的找零，适用面更广，1 张钞票的限制也不会大幅增加找零张数，还可以通过每次 1 张的找零调节手中钞票的数量和分布。

由于这本身是比较宽松的一种策略，不必使用背包算法，直接枚举找哪一张再贪心即有较显著的优化效果。

经过实验，使用“近似精确 $\rightarrow$ 无解则尝试精确付款 $\rightarrow$ 无解则使用优化后的贪心算法”策略，在 CodeChef 上最高分为 13444.15(0.564pt)，截至本文写完，是 CodeChef 上所有不使用“预知数据，针对数据优化”的程序中的最高分（CodeChef 上正式比赛时超过 0.7pt 的程序均是“针对数据优化”，经过实验，这些程序在随机数据下表现明显不如上述策略）。

2.4 总结

这道题具有“不能预知未来”的特性，无法触及“理论最优解”，策略多样，但即使手中有很多优秀的策略，也要弄清它们的适用范围，灵活运用才能获得高分。十分具有挑战性。