“穿越沙漠”游戏最佳策略模型

摘要

本文旨在研究“穿越沙漠”小游戏的最佳策略问题。

在第一问中，我们分析了玩家决策的优劣情形，证明了使用***dijskra*算法**寻找并仅保留特殊点间最短路径而简化地图的合理性，将关卡的地图转化为图模型；同时给出了描述玩家在游戏中状态的方式，并分析了本游戏求解的特性：1、**玩家决策具有无后效性。2、游戏的最优解包含所有子问题的最优解。**在此基础上我们选用**动态规划**求解最优策略，最终第一关的结果为耗时 24 天，资金 10470 元，第二关的结果为耗时 30 天，资金 12730 元。

在第二问中，我们用与第一问相同的方式得到简化图模型。考虑到天气未知，在第一问的基础上我们首先对天气情况进行模拟，以第一二关天气出现的频率为基准，并且基于查阅的资料，建立一个天气关联机制，在此机制下随机生成 100 组天气数据作为测试数据。同时我们对第三关、第四关的地图进行具体分析，缩小路线选择的范围，并统计利用第一问中的动态规划模型在 100 组天气数据中得出的各条线路的情况，构建**统计评价模型**。利用该模型首先**确定线路**。之后在限定该线路的情形下，我们再综合考量不同初始物资购买策略，**确定购买策略**，计算他们的收益期望与失败率期望，选出最优策略。第三关卡期望最优收益为 9357.5 元，第四关卡期望收益为 10941.5 元。

在第三问中，我们使用**静态博弈模型**考虑多玩家策略互相影响时的情况，第五关卡双方仅同时进行一次决策，为单段静态博弈，使用混合博弈模型求解；第六关卡双方在每一节点都同时进行一次决策，为多段静态博弈。我们首先构建玩家可行的最优策略组，再依据纳什均衡点是否存在判断两玩家使用异种策略时的最优解，若没有则在数种策略组中为玩家选择失败率最低的纯策略。

关键词：动态规划、统计评价模型、静态博弈模型

一、问题重述

1.1 游戏设定

玩家在给出的地图中，利用初始资金购买一定数量的水和食物，从起点出发，在沙漠中行走。途中会遇到不同的天气，也可在矿山、村庄补充资金或资源，目标是在规定时间内到达终点，并保留尽可能多的资金。

规则细节：

（1）以天为基本时间单位，游戏的开始时间为第 0 天，玩家位于起点。玩家必须在截止日期或之前到达终点，到达终点后该玩家的游戏结束。

（2）穿越沙漠需水和食物两种资源，它们的最小计量单位均为箱。每天玩家拥有的水和食物质量之和不能超过负重上限。若未到达终点而水或食物已耗尽，视为游戏失败。

（3）每天的天气为“晴朗”、“高温”、“沙暴”三种状况之一，沙漠中所有区域的天气相同。

（4）每天玩家可从地图中的某个区域到达与之相邻的另一个区域，也可在原地停留。沙暴日必须在原地停留。

（5）玩家在原地停留一天消耗的资源数量称为基础消耗量，行走一天消耗的资源数量为基础消耗量的2倍。

（6）玩家第 0 天可在起点处用初始资金以基准价格购买水和食物。玩家可在起点停留或回到起点，但不能多次在起点购买资源。玩家到达终点后可退回剩余的水和食物，每箱退回价格为基准价格的一半。

（7）玩家在矿山停留时，可通过挖矿获得资金，挖矿一天获得的资金量称为基础收益。如果挖矿，消耗的资源数量为基础消耗量的3倍；如果不挖矿，消耗的资源数量为基础消耗量。到达矿山当天不能挖矿。沙暴日也可挖矿。

（8）玩家经过或在村庄停留时可用剩余的初始资金或挖矿获得的资金随时购买水和食物，每箱价格为基准价格的 2 倍。

1.2 问题解决

问题一：只有一名玩家，在整个游戏时段内每天天气状况事先全部已知，给出第一关、第二关中玩家的最优策略。

问题二：只有一名玩家，玩家仅能知道当天的天气状况，并依据天气决定当天的行动，给出最佳策略，在第三关、第四关上具体讨论

问题三：有 n 名玩家，他们有相同的初始资金，且同时从起点出发。若某天其中的任意 $k\left(2\le k\le n\right)$ 名玩家均从区域 A 行走到区域 B（$\mathrm{B}\neq\mathrm{A}$），则他们中的任一位消耗的资源数量均为基础消耗量的 $2k$ 倍；若某天其中的任意 $k\left(2\le k\le n\right)$ 名玩家在同一矿山挖矿，则他们中的任一位消耗的资源数量均为基础消耗量的3倍，且每名玩家一天可通过挖矿获得的资金是基础收益的 $\frac{1}{k}$；若某天其中的任意

$k\left(2\le k\le n\right)$名玩家在同一村庄购买资源，每箱价格均为基准价格的4倍。其他情况下消耗资源数量与资源价格与单人游戏相同。

1、游戏全程天气已知，且每个玩家的行动方案需要在第 0 天确定且不可更改，给出最优策略，在第五关进行讨论。

2、假设所有玩家仅知道当天的天气状况，从第1天起，每名玩家在当天行动结束后均知道其余玩家当天的行动方案和剩余的资源数量，随后确定各自第二天的行动方案。试给出一般情况下玩家应采取的策略，并对附件中的“第六关”进行具体讨论。

二、模型假设

1、在每一个关卡中，各个天气出现的基础概率是恒定的，并且以第一、二关卡的具体情况为基准。

2、在多人游戏中，每一个玩家都是理性人，即完全为自己的利益考虑，该利益的衡量标准为在保证存活的前提下获得尽量多的资金利益。

三、符号说明

符号	说明
F	最优资金数
i	游戏天数
j	玩家位置
k	玩家水储备
l	玩家食物储备
p_w	水基础价格
p_e	食物基础价格
m_w	水每箱质量
m_e	食物每箱质量
c_w	水基础消耗量
c_e	食物基础消耗量
$init$	初始资金

四、问题分析

4.1 问题一、二的图构造

为了方便讨论与计算，我们首先要将附件中所给出的地图转化为图$\mathrm{G}=(\mathrm{V},\mathrm{E})$，地图中每一个区域为一个顶点，若两块区域$i$，$j$毗邻，则顶点$i$，$j$之间存在边。

4.1.1 图模型的简化与建立

在之前的基础上，我们需要对上述地图进行简化以降低运算量。

我们提取地图中带有特殊意义的点，称为特殊点，即矿山、村庄以及终点，**我们首先猜想：在最优解中，玩家必定在每一时刻都在向某一特殊点更近的方向移动。**

证明：在游戏设定中，整个沙漠区域的天气是统一的，因而无论采取何种路线规划，不同计划因天气而引起的行走/停留是统一的，在这种情形下向着离某特殊点更近的方向移动是耗时最少的策略，耗时少则生存资源消耗少，这更符合我们尽量减少资源消耗，增加最终存款的择优指标，因而玩家必定在每一时刻都在向某一特殊点更近的方向移动。

我们其次猜想：在最优解中，玩家只会在矿山、村庄、终点之间以最短距离（即图中两顶点间的最短路）移动。

证明：在本游戏中，每个玩家都寻找最优策略，因而在其规划中只有特殊点可能成为某一日的目标地点。如果某两特殊点之间的一条或多条最短路不经过任何另一特殊点，那么显然玩家会选择一条最短路，并且选择不同的最短路不会影响结果。在此情况下，每两个特殊点之间的最短路都会被记录。如果两特殊点i，j之间的一条最短路经过了特殊点 k，那么组成该最短路的边集$(i，j),(k，j)$必定分别为两两特殊点间的最短路，已经被记录。

综上，我们可以对 4.1 中构造的图进行简化，仅保留特殊点以及特殊点间最短路所构成的边集，新图记为G'，我们通过$dijskra$算法求出特殊点间的最短路以及最短路经过的点，并删除所有非特殊点，即可求得每一关地图所对应的地图图模型 G’。

4.2 问题一分析

对于一个玩家沙漠穿越进行策略规划的问题有如下特点：

1、该玩家在穿越沙漠过程中每一天的情况都可以用一组参数描述，从题设来看他们分别是：天数t、金钱数M、水量i、食物量j、位置k（顶点编号），作为状态参数。

2、该玩家在某一天之前的决策方式对该天之后的策略规划无影响，该天之后穿越行程的发展变化仅与后续决策以及该天的上述状态参数相关，具有无后效性。

3、在达成最优解，即最终剩余物资最多的决策中，无论玩家在第i天做出什么样的决策，其前$i-1$天的策略达成的剩余物资一定是最多的，即该策略对其每一个子问题都达成最优解。

综合上述分析，本问适宜使用动态规划的方法进行求解，我们设定函数$M(\mathrm{t,i,j,k})$为天数为第i天，水量为k、食物量为l、位置为j的状态下金钱数的最大值，很显然，在钱数相同的情形下，我们希望食物水量的剩余量越多越好，因而动态规划的目标可以表示为：

$$\max_{i,j} \quad F(i,j,k,l) \tag{1}$$

4.3 问题二分析

与第一问不同的是，我们首先需要对未知的天气情况进行讨论，考虑到我们需要尽量考虑不同天气组合下的情形，我们可以依据第一、二关中不同天气出现的频率为基准，并根据我们查阅的资料，构建了天气关联机制，以使得生成的天气数据更合理，通过该方式随机生成多组数据进行模拟，并寻找这些模拟中收益较高的情形下通关率也较高的策略。

同时，我们也要具体考虑地图的特点，第三关的地图中没有村庄，仅有矿山，而关卡 4 的地图十分对称，村庄以及矿山都与同一节点相连，且恰好都在该节点与终点的最短路径上。通过这样的特点我们可以通过问题分析一中的两个结论直接分析出最佳路径的数种可能，大大减少计算量；通过动态规划在多组数据上不同线路的结果统计并选择合适的线路。

4.3 问题三分析

在本问中，玩家数量多于一个人，并且每个人决策的效果会受到其他人决策的影响，在这种情形下，每个玩家想要实现自己的最优方案必须考虑其他玩家的行动策略，因而我们使用博弈模型来解决这一问题。第五关卡的地图与第三关相同，因而我们仅用考虑两个玩家在从起点直接前往终点的行程安排，双方仅同时进行一次决策，为单段静态博弈，使用混合博弈模型求解；第六关卡双方在每一节点都同时进行一次决策，为多段静态博弈。我们首先构建玩家可行的最优策略组，再依据纳什均衡点是否存在判断两玩家使用异种策略时的最优解，若没有则在数种策略组中为玩家选择失败率最低的纯策略。

五、问题求解

5.1 问题一图模型简化

由问题分析，我们首先利用$dijskra$算法寻找起点、终点、村庄、矿山两两之间的最短路，仅保留这些最短路所包含的边集与点集，删去其余部分，得到第一关第二关的简化图：

图 1 第一关简化图

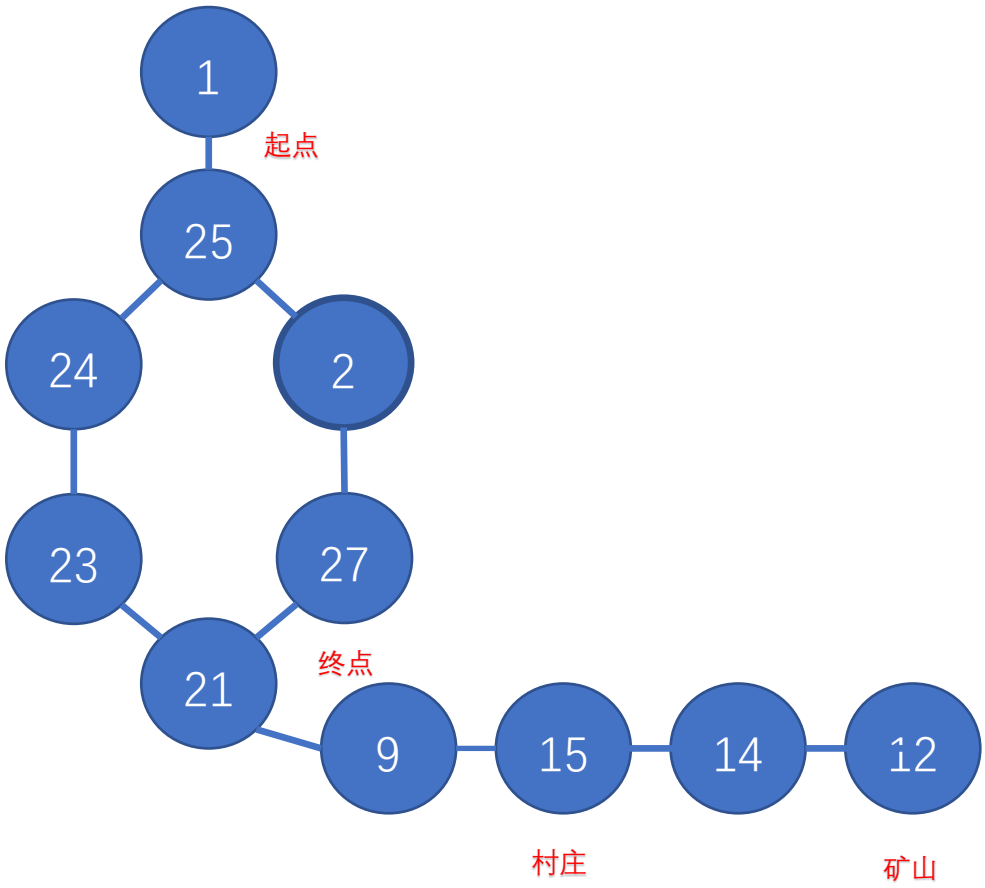
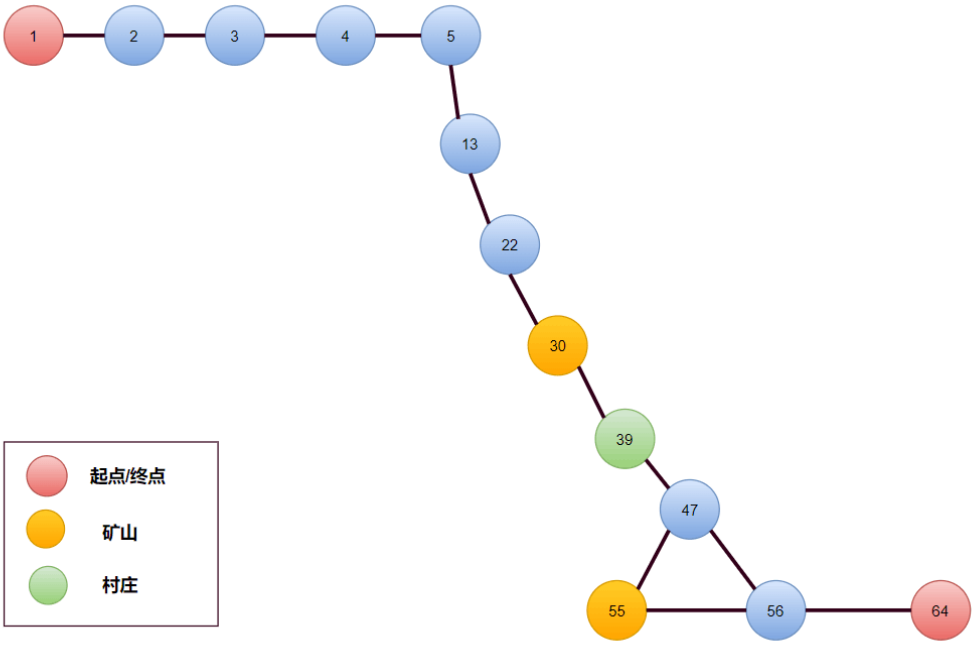


图 2 第二关简化图



5.2 状态转移方程的确定

由题目设定，玩家可以进行的不同种类决策有：移动、停留、挖矿、购买资源、以及在起点处准备物资这 5 类操作，我们分别对五类操作的状态转移方程进行分析。

5.2.1 准备物资

我们规定玩家的初始资金数量为$init$，食物每箱基础价格、质量、购买量为p_e、m_e、l；水每箱基础价格、质量、购买量为p_w、m_w、k，则状态初值为：

$$F(1,0,k,l) = init - p_e l - p_w k \tag{2}$$

同时必须满足：

$$init - p_e l - p_w k \geq 0, 2k + 3l \leq 1200$$

5.2.2 移动

设玩家某日从位置t移动到t', 依据题意，我们规定玩家的基础水、食物消耗

量分别为c_w、c_e，同时在移动时消耗量为基础消耗量的两倍,对状态进行更新：

$$F(i+1,t^{'},k-2c_w,l-2c_e)=\max\{F(i+1,t^{'},k-2c_w,l-2c_e),F(i,t,k,l)\} \tag{3}$$

其中需要满足：

$$k\geq 2c_w,l\geq 2c_e$$

5.2.3 矿山挖矿

依据题意，挖矿时食物与水消耗量为基础消耗量的三倍，设基础收入为I，同时位置不变，对状态进行更新：

$$F(i+1,j,k-3c_w,l-3c_e)=\max\{F(i+1,j,k-3c_w,l-3c_e),F(i,t,k,l)+I\} \tag{4}$$

其中需要满足：

$$k\geq 3c_w,l\geq 3c_e$$

5.2.4 停留

依据题意，停留时位置不变，天数增加，水量消耗食物消耗为基础消耗量，对状态进行更新：

$$F(i+1,j,k-c_w,l-c_e)=\max\{F(i+1,j,k-c_w,l-c_e),F(i,t,k,l)\} \tag{5}$$

需要满足：

$$k\geq c_w,l\geq c_e$$

5.2.5 购买资源

依据题意，到达村庄购买资源，买水$k^{'}$箱，买食物$l^{'}$箱，则可对状态进行更新：

$$F(i,j,k+k^{'},l+l^{'})=\max\{F(i,j,k+k^{'},l+l^{'}),F(i,t,k,l)-k^{'}p_w-l^{'}p_e\} \tag{6}$$

同时需要满足：

$$2(k+k')+3(l+l')\leq 1200,F(i,t,k,l)\geq k'p_w+l'p_e$$

5.2.6 复杂度分析

对于状态变量(i，j，k，l),四个变量的约束条件分别为:游戏天数少于 30 天：$0<i<30$;简化后地图上的节点数约为 10-20 个，因而位置变量J:$1<j<20$;对于食物量与水量，其与最大负重量挂钩,其中食物每箱质量为 2，水每箱质量为 3，最大负重质量为 1200：$0\leq 3k+2l\leq 1200$，我们按照水、食物各自买满进

行计算。对于状态空间：

$$S = \{(i, j, k, l) \mid i \leq 30, j \leq 20, 3k + 2l \leq 1200\}$$

其大小：

$$|S| \approx 30 \times 20 \times 400 \times 600 = 1.44 \times 10^8$$

对于我们在每一步遍历取最优其空间复杂度也是可接受的。

同时对于$i = 30$天，$j =$ 20个点，携带食物、水各$n = 300$箱的大体情形，循环最外层天数的时间复杂度为$O(i)$, 其次对于节点的循环复杂度为$O(j)$, 对于水、食物各自遍历时移动、移动、挖矿等复杂度皆为$O(n^2)$, 但在村庄购买的情形下购买食物、水的量又需各自遍历，其复杂度为$O(n^2 \times n^2)$, 算法整体复杂度$O(ijn^4)$, 耗时大约 10 分钟左右。

；

5.3 动态规划求最优策略

根据题目给出的天气数据，通过动态规划算法，我们可以求得第一关、第二关的最佳策略：

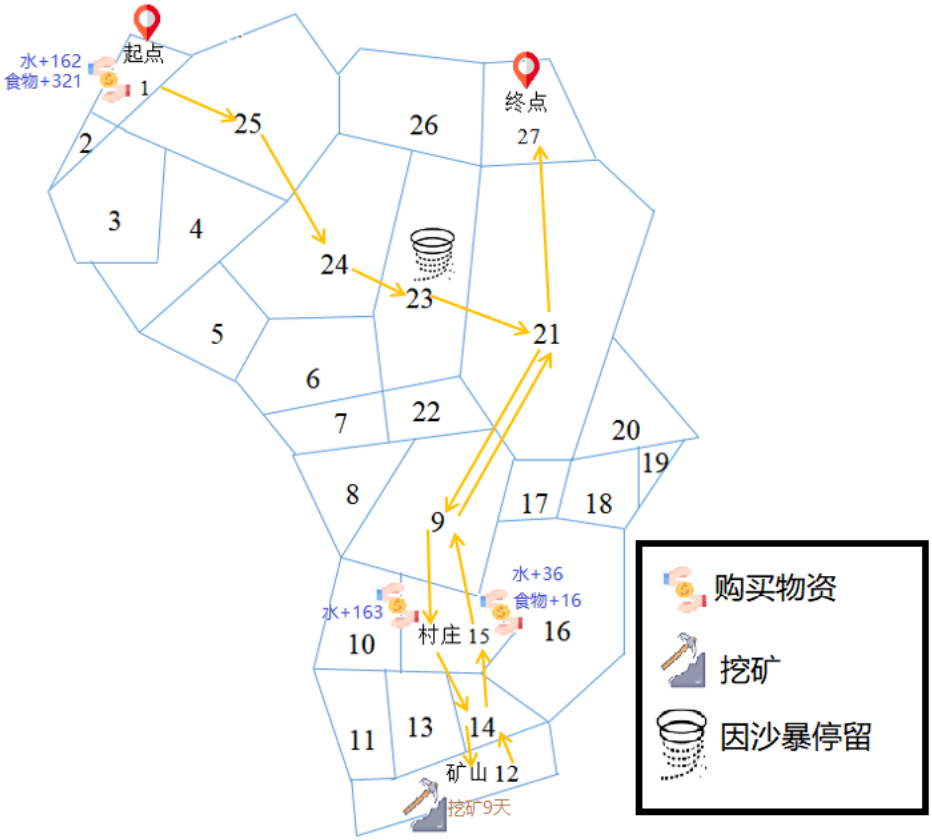
第一关：用时 23 天，最终剩余资金 10470

表 1 第一关每日状态表

日期	所在区域	剩余资金数	剩余水量	剩余食物量
0	1	10000	0	0
1	25	5780	162	321
2	24	5780	146	309
3	23	5780	136	295
4	23	5780	126	285
5	21	5780	116	271
6	9	5780	100	259
7	9	5780	90	249
8	15	4150	243	235
9	14	4150	227	223
10	12	4150	211	211
11	12	5150	181	181
12	12	6150	157	163
13	12	7150	142	142
14	12	8150	118	124
15	12	9150	94	106
16	12	10150	70	88
17	12	10150	60	78
18	12	10150	50	68
19	12	11150	26	50
20	14	11150	10	38

21	15	10470	36	40
22	9	10470	26	26
23	21	10470	10	14
24	27	10470	0	0

图 2 第一关策略示意图



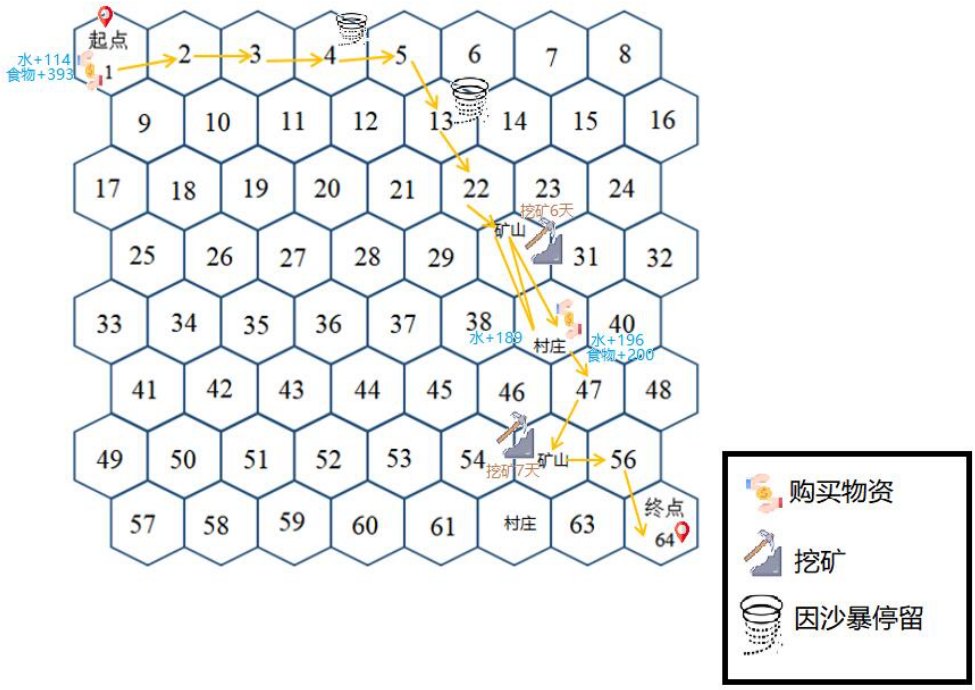
第二关：耗时 30 天，剩余资金 12730

表 2 第二关每日状态表

日期	所在区域	剩余资金数	剩余水量	剩余食物量
0	1	10000	0	0
1	2	5300	114	393
2	3	5300	98	381
3	4	5300	88	367
4	4	5300	78	357
5	5	5300	68	343
6	13	5300	52	331

7	13	5300	42	321
8	22	5300	32	307
9	30	5300	16	295
10	39	3410	189	283
11	39	3410	179	273
12	30	3410	163	261
13	30	4410	148	240
14	30	5410	124	222
15	30	6410	100	204
16	30	7410	76	186
17	30	8410	46	156
18	30	9410	16	126
19	39	5730	196	200
20	47	5730	180	188
21	55	5730	170	174
22	55	6730	155	153
23	55	7730	131	135
24	55	8730	116	114
25	55	9730	86	84
26	55	10730	62	66
27	55	11730	47	45
28	55	12730	32	24
29	56	12730	16	12
30	64	12730	0	0

图 3 第二关策略示意图



5.4 单玩家天气已知情形下最优策略概述

综合第一问二个关卡的问题最优策略并分析他们共有的特点，我们也可以分析出单人游戏情形下决策的较优方式：

1、天气已知情形下，最优解的资源需要恰好满足生存需求而无剩余，这一点是显然的。

2、在起点处尽量购买满足需求的水与食物，因为由题目设定，村庄的购买价格为起点的两倍，同时，在起点购买时优先满足食物需求，这是有因可循的：我们定义资源携带效率：

$$\eta = \frac{prize}{mass}$$

即我们携带的单位质量所等效的价值，该指标的意义在于：

在 30 日限时下通关所需生存资源质量是超过单次可携带的最大质量的，而为了尽量减少补给次数，节省更多时间挖矿或者快速通关减少资源消耗，我们在开局时需要携带尽量有价值的物资，即最大化资源携带效率。

在本题中，水的资源携带效率为 1.67 元/kg，而食物的资源携带效率为 5 元/kg，显然食物的资源携带效率更高，因而最优解中初始资源准备要尽量满足食物储备需求。

5.5 第二问求解

依据分析，第二问在第一问的基础上增加了难度，没有给出天气状况，因而

我们首先通过随机生成的方式获得 100 组可能的天气数据，并利用第一问的动态规划方法进行多次求解分析，首先通过统计结果确定线路，再确定最优物资准备方案。

5.5.1 地图简化

与第一问相同，我们首先得出关卡 3、关卡 4 的简化图模型：

图 4 关卡 3 简化图

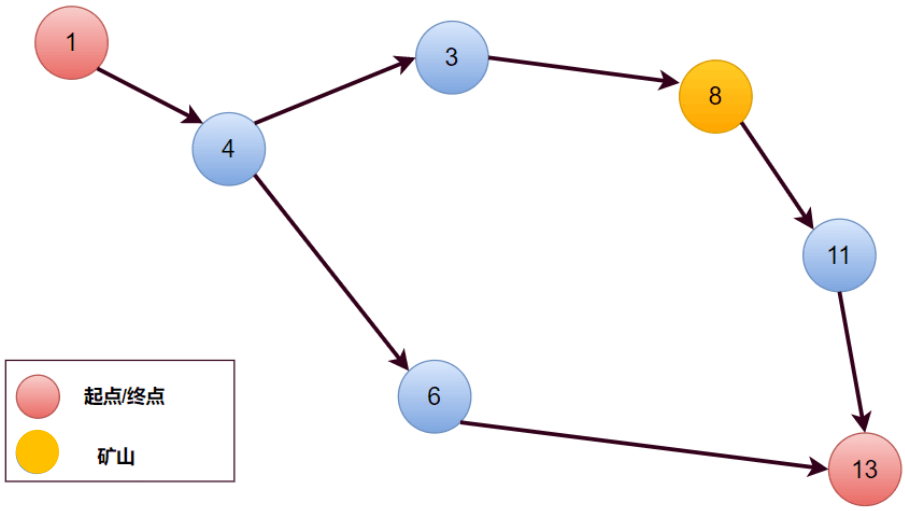
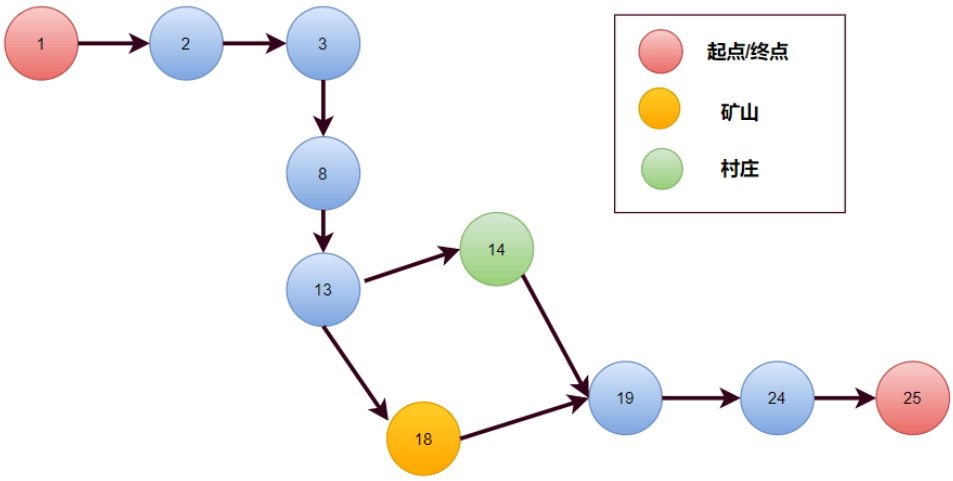


图 5 关卡 4 简化图



5.5.2 天气模拟

首先，较为简单的方法是利用第一问已知数据中各天气所占的比例进行预测。但这种方法也有一定的问题：一是第一问数据较少，容易出现大的偏差；二是经过后几题可以看出，该地区的天气也有不出现沙暴、较少出现沙暴这种情况。我们查阅了一些简单的气象知识，结合第一问的比例，对于之后的天气进行简单的预测。

沙尘暴的形成要有不稳定的地表热层结构。在沙尘暴发生的前几天往往有持续高温天气,从而促进上升气流的活动,增强气流对地面的扰动,所以在冷暖交替的春季最易形成沙尘暴。因此高温天气会导致沙暴更大概率的发生；连续的常温晴天会导致高温天气更大可能的出现；沙暴过后的气温回归正常。

但自然界的情况不可能是一成不变地遵循一个规律的，所以在对天气的预测中，我们加入了不确定性：在预测下一天的天气时，有 15%的概率使得明天的天气与之前几天无关，而是只依靠第一问的比例得出；85%的比例遵循以下规律：

设晴天、高温、沙暴的基础概率P_s, P_h, P_m分别为 0.3，0.5，0.2.

1、若第i天为晴天，则第$i+1$天三种天气概率满足：

$$P_s' = P_s \cdot 0.85, P_h' = P_h + P_s \cdot 0.15 \tag{7}$$

2、若第i天为高温，则第$i+1$天三种天气概率满足：

$$P_h' = P_h \cdot 0.85, P_m' = P_m + P_h \cdot 0.15 \tag{8}$$

3、若第i天为沙暴，则第$i+1$天三种天气概率回归基准概率

基于上述机制，我们随机生成 100 组天气数据作为模拟数据。

5.5.3 特殊路线确定

与第一关、第二关相比，第三、四关的地图有特殊之处，在求解前可先对其线路进行分析，减少求解难度。

对于第三关，本关的地图中没有村庄，仅有起点、终点与矿山，即依照我们在问题一分析中得出的“玩家一定在向某一特殊点延最短路径移动”的结论，本文可行的路线仅有两条，即**直接前往终点**与**前往矿山挖矿后去往终点**，我们仅需考虑在我们生成的 100 组天气数据下两种路线的表现来选择线路。

对于第四关，从起点到节点 13 间的路径已经可以确定，而村庄与矿山恰巧都与节点 13 相邻，并且分别在节点 13 到终点的两条最短路径上，因而我们的路线选择仅需考虑到达节点 13 后在村庄、矿山、终点之间的策略安排。

5.5.4 收益风险平衡模型

在天气未知的情形下，我们需要考虑动态规划得出的解答在实际游戏中的风险问题,因为其在模拟 100 组天气情况已知的数据时给出的最优物资购买策略必定是在玩家到达时正好用完的,而我们的初始采购策略应该能最好的平衡游戏失败的风险与收益。

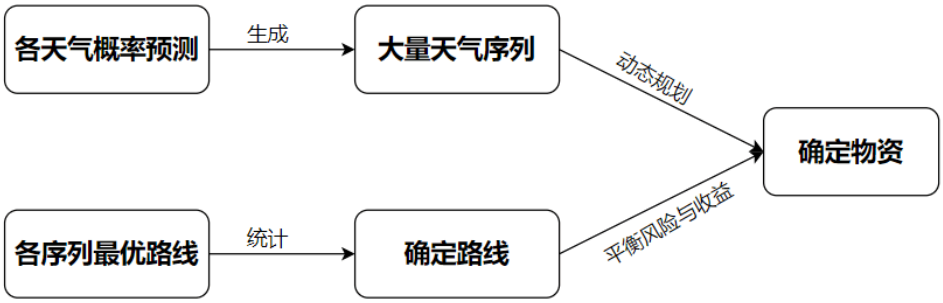
我们选择通过较简单的计算期望的方式来判断上述指标，即计算 100 组数据的平均通关时长以及初始物资购买方案，以该理想的水、食物购买方案为基准增加一定购买量构造可能的最优策略，同时保障通关率。

在获得可能的最优策略后，我们再生成 100 组数据对该策略进行测试，综合其通关率与收益对其进行评价。

5.5.5 关卡三的策略规划

根据上述分析，我们第三关的求解思路如下图所示：

图 6 第三关求解流程图



我们首先对 100 组生成的天气数据都进行了最优策略模拟，所有的 100 组数据都是直接前往终点这条线路最佳，因而本关可以直接排除前往矿山的方案，仅考虑初始资源购置问题。

100 组数据的具体通关天数统计情况如下，其影响因素为天气的随机性：

表 3 100 次模拟通关天数统计

天数	1	2	3	4	5	6
次数	0	0	55	34	9	2

依照 5.5.4 中我们提到的方案设计方法，我们找出的最优初始资源购买策略为水 45 箱、食物 50 箱。在测试中该方案期望收益为 9357 元，通关成功率为 93.5。

5.5.6 关卡四的策略规划

第四关可以视为几个第三问的组合，其中，第四关最关键的两个位置是点 13 和 19----既可以选择去村庄，也可以选择去矿场；同时，也需要考虑可不可以在村庄和矿场之间来回补给挖矿。

依照第三问得到的期望，我们大致可以估计出起点到 13 节点；村庄与矿场往返；村庄/矿场到终点所需的物资，由此可以确定行为策略：

1、玩家首先到达 13 节点

2、第一次到达 13 点时判断所剩物资：

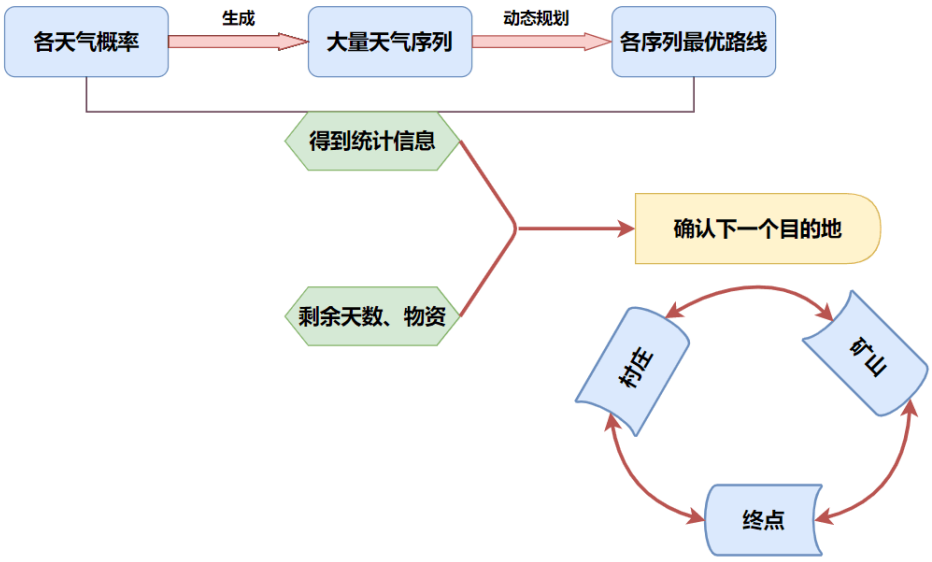
1）水量$k > 80$时决策先去矿山，其余情况先去村庄补给

2）挖矿进行直到物资开始短缺：当水量$k < 30$或食物量$l < 30$时立刻返回村庄补给；$30 < k < 50$且$30 < l < 50$时直接走向终点。

其他策略与第三问相同。

同样，我们得出的最优策略在测试中的数据为期望收益 10941.5 元，通关成功率 53.9，考虑到从起点到节点时间的 5 天里再模拟中会经常出现高温持续天气，这样的通关率可以接受。

图 7 关卡 4 求解流程图



5.6 第三问求解

5.6.1 第五关博弈设定

依照题目分析，我们规定两名玩家 A、B 都为理性人，并且可以充分了解对方的策略规划，在此基础上设计符合对方利益的同时让自己的收益期望最高的最优策略，在博弈中 A、B 的地位是对称的，因而两者的可选策略集合也是相同的。

5.6.2 可选策略集合

依据第六关的地图，我们可以直接分析出可以纳入策略组的数种行动方案：

S_1：第一天由节点 1 移动到节点 5，第二天在节点 2 停留，第三天第四天从 5 走到终点，最终剩余收益 9535 元，消耗物资 465 元。

S_2：第一天由节点 1 移动到节点 5，第二天移动到节点 6，第三天移动到终

点，最终剩余收益 9510 元，消耗物资 490 元。

S_3：在使用S_2策略时对方也使用S_2策略，需要耗费 980 元的物资，收益 9020 元。

S_4：在使用S_1策略时对方也使用S_1策略，此时需要耗费 795 元的物资，收益 9205 元。

S_5：在使用在使用S_2策略时对方使用S_1策略，此时需要消耗 685 元物资，收益为 9400 元。

S_6：在使用在使用S_1策略时对方使用S_2策略，此时需要消耗 800 元物资，收益为 9400 元。

为了方便计算，我们以消耗量作为每一种策略的指标。

5.6.3 最优策略

对于最优纯策略，在天气已知的情形下我们可以保证成功率，选择收益最高的情形即可，即双方都采用S_3策略，消耗 795 元，收益 9205 元。

同时我们考虑采用混合策略时的不同可能，对于S_1、S_2两种个人策略，两人不同选择的四种组合即为S_3~S_6的四种情况，制表如下：

表 3 玩家博弈策略成本表

成本 玩家 1 / 玩家 2	S_3	S_4
S_3	$(980, 980)$	$(800, 685)$
S_4	$(685, 800)$	$(795, 795)$

令两位玩家采用S_3、S_4的概率分别为P_1、P_2，求纳什均衡点：

$$980P_1 + 800P_2 = 685P_1 + 795P_2 \tag{9}$$

解得$P_1 = -0.02$，$P_2 = 1.02$，所以本策略不存在纳什均衡点，两个人都会倾向选择S_4策略。

在寻找其他组合的混合策略时我们遇到了同样的情况，不存在纳什均衡点使得两人趋向选择同一单人行动策略，数据说明本关卡最适合的方案就是使用纯策略，即两个人都使用S_4策略，收益为 9205 元。

六、模型评价

优点：

1、在求解第一、二问时首先简化所要寻找的路线，分为直接到达终点、先到达关键点两种行为方式。同时利用动态规划和图论的方法，使得结果取到最优。

2、后几问求解时都以第一问的模型为基础，但又以各题的特殊情况增加了决策依据，使得求解更为整体

3、对于天气的预测参考了文中的数据和其他方面的知识，比直接依靠概率更加接近真实

缺点：

1、依靠自主随机生成的 100 组天气序列中仍然会受偶然因素的影响。

2、天气预测中的系数 0.15 只是常用的阻尼系数，并不准确

附件

Dij 简化图

```
#include <bits/stdc++.h>
using namespace std;

const int N=1e5+50,M=2e5+50;

int head[N],etop;
struct Edge
{
    int v,w,nxt;
}e[M];
void add(int u,int v,int w)
{
    e[++etop].v=v;
    e[etop].w=w;
    e[etop].nxt=head[u];
    head[u]=etop;
}

struct Data
{
    int u,w;
    Data (int _u,int _w)
    {
```

```
      u=_u; w=_w;
    }
    bool operator <(const Data &t) const
    {
      return w>t.w;
    }
};

priority_queue <Data> q;
int vis[N],dis[N],n,m,S;
int main()
{
    scanf("%d%d%d",&n,&m,&S);
    for (int u,v,w;m--;)
    {
      scanf("%d%d%d",&u,&v,&w);
      add(u,v,w);
    }
    memset(dis,0x3f,sizeof(dis));
    q.push(Data(S,dis[S]=0));
    while (!q.empty())
    {
      int u=q.top().u; q.pop();
      if (vis[u]) continue;
      vis[u]=1;
      for (int v,i=head[u];i;i=e[i].nxt)
      {
         v=e[i].v;
         if (dis[v]>dis[u]+e[i].w)
         {
            dis[v]=dis[u]+e[i].w;
            q.push(Data(v,dis[v]));
         }
      }
    }
    for (int i=1;i<=n;i++)
      printf("%d ",dis[i]);
    puts("");
    return 0;
}
```

第一问动态规划（第一关为例）

```cpp
#include <bits/stdc++.h>
using namespace std;

const int P = 70; //最大总点数

int f[2][P][605][605]; //滚动一维

int n = 27;

int pv[P], pm[P]; //是否为村庄、矿山


int etop, head[P];
struct Edge
{
    int v, nxt;
}e[P * P];
void add(int u, int v)
{
    e[++etop].v = v;
    e[etop].nxt = head[u];
    head[u] = etop;
}
void link(int u, int v)
{
    add(u, v); add(v, u);
}

void link_G()
{
    for (int i = 1; i < 27; i++)
        link(i, i + 1);
    link(1, 25);
    link(3, 25);
    link(4, 24);
    link(4, 25);
    link(5, 24);
    link(6, 23);
    link(6, 24);
    link(7, 22);
    link(8, 22);
    link(9, 15);
    link(9, 16);
    link(9, 17);
    link(9, 21);
    link(9, 22);
```

```
    link(10, 13);
    link(10, 15);
    link(11, 13);
    link(12, 14);
    link(13, 15);
    link(14, 16);
    link(16, 18);
    link(17, 21);
    link(18, 20);
    link(21, 23);
    link(21, 27);
    link(23, 26);
    link(24, 26);
}
void key_G()
{
    pv[15] = 1;
    pm[12] = 1;
}

//晴朗0 高温1 沙暴2

int wt[P] = {0, 1, 1, 0, 2, 0, 1, 2, 0, 1, 1,
             2, 1, 0, 1, 1, 1, 2, 2, 1, 1,
             0, 0, 1, 0, 2, 1, 0, 0, 1, 1};

int r1[] = {5, 8, 10}, r2[] = {7, 6, 10}; // 基础消耗量

int mny = 1000; //挖矿收益

void mx(int &a, int b) {a = max(a, b);}
int main()
{

    link_G(); key_G();//建图

    int M = 1200;  //负重上限

    int day = 30; //30天

    int c0 = 10000;  //初始资金

    int c1 = 5, c2 = 10;  //水和食物的基准价格

    int m1 = 3, m2 = 2;  //水和食物的每箱质量

    int v1 = M / m1, v2 = M / m2; //最能带多少箱水或食物
```

```
    memset(f[0], -0x3f, sizeof(f[0])); //不可达状态设为 -inf
    for (int k = 0; k <= v1; k++) //带多少箱水
    {
        int lim = (M - k * m1) / m2;  //最多还能带多少箱食物
        for (int l = 0; l <= lim; l++)
            f[0][1][k][l] = c0 - k*c1 - l*c2; //初值
    }

    for (int w, d = 0; d < day; d++) //天
    {
        cout<<d<<" "<<endl;
        int cur = d & 1, nxt = !(d & 1); //滚动
        w = wt[d]; //当前天气
        memset(f[nxt], -0x3f, sizeof(f[nxt])); //不可达状态
设为 -inf
        for (int u = 1; u <= n; u++) //当前点
            for (int k = 0; k <= v1; k++)
            {
                int lim = (M - k * m1) / m2;
                for (int l = 0; l <= lim; l++)
                {
                    if (f[cur][u][k][l] < 0) continue; //当前
状态不合法，跳过
                    cout<<d<<" "<<u<<" "<<k<<" "<<l<<endl;
                    if (wt[d] != 2) //不是沙暴
                        for (int v, i = head[u]; i; i =
e[i].nxt) //1. 移动
                        {
                            v = e[i].v; //下一个点
```

```
                        if (k - (r1[w]<<1) >= 0 && l -
(r2[w]<<1) >= 0) //水食物未耗尽
                            mx(f[nxt][v][k - (r1[w]<<1)][l
- (r2[w]<<1)], f[cur][u][k][l]); //两倍消耗
                    }
                    if (k - r1[w] >= 0 && l - r2[w] >= 0)
                        mx(f[nxt][u][k - r1[w]][l - r2[w]],
f[cur][u][k][l]); //2. 停留

                    if (pm[u]) //当前位于矿场
                    {
                        if (k - 3 * r1[w] >= 0 && l - 3 *
r2[w] >= 0)
                            mx(f[nxt][u][k - 3 * r1[w]][l - 3
* r2[w]], f[cur][u][k][l] + mny);
                    }
                    if (pv[u]) //当前位于村庄
                    {
                        for (int ak = 0;; ak++) //买ak箱水
                        {
                            int tmp1 = (k + ak) * m1;
                            if (tmp1 > M) break; //超出负重上限

                            for (int al = 0;; al++) //买al箱食
物
                            {
                                int tmp2 = (l + al) * m2;
                                if (tmp1 + tmp2 > M ||
f[cur][u][k][l] - 2*ak*c1 - 2*al*c2 < 0) //钱不够用或超出负
重
                                    break;
                                mx(f[cur][u][k + ak][l + al],
f[cur][u][k][l] - 2*ak*c1 - 2*al*c2); //当前层的更新
                            }
```

```
                    }
                }
            }
        }
    }
    return 0;
}
```

第二问第三关动态规划

```
#include <bits/stdc++.h>
using namespace std;

const int n = 7; //简化后节点数

const int M = 1200;  //负重上限

const int day = 10; //10天

const int c0 = 10000;  //初始资金

const int c1 = 5, c2 = 10;  //水和食物的基准价格

const int m1 = 3, m2 = 2;  //水和食物的每箱质量


int etop, head[n + 1];
struct Edge
{
    int v, nxt;
}e[n * n];
void add(int u, int v)
{
    e[++etop].v = v;
    e[etop].nxt = head[u];
    head[u] = etop;
}
void adds(int u, int v) {add(u, v); add(v, u);}

int pv[n+1], pm[n+1]; //是否为村庄、矿山

int ed = 7; //7为终点

void build_G()
{
    for (int i = 1; i <= 4; i++)
        adds(i, i + 1);
```

```
    adds(5, 7);
    adds(2, 6);
    adds(6, 7);
    pm[4] = 1;
}

//晴朗0 高温1 沙暴2

int wt[day];

int r1[] = {3, 9, 10}, r2[] = {4, 9, 10}; // 基础消耗量

int mny = 200; //挖矿收益


struct Data
{
    char i, j;
    short k, l;
    Data () {}
    Data (char _i, char _j, short _k, short _l)
    {
        i = _i, j = _j, k = _k, l = _l;
    }
}p[day + 1][n + 1][M/m1 + 1][M/m2 + 1]; //path

short f[day + 1][n + 1][M/m1 + 1][M/m2 + 1];

void mx(int ni, int nj, int nk, int nl, int i, int j, int
k, int l, int val=0)
{
    if (nk < 0 || nl < 0) return ;
    if (f[ni][nj][nk][nl] < f[i][j][k][l] + val)
    {
        f[ni][nj][nk][nl] = f[i][j][k][l] + val;
        p[ni][nj][nk][nl] = Data(i, j, k, l);
    }
}
int v1 = M / m1;
void buy(int d, int u, int k, int l)
{

    for (int ak = 0;; ak++) //买ak箱水

    {
        int tmp1 = (k + ak) * m1 + l * m2;

        if (tmp1 > M) break; //超出负重上限
```

```
        for (int al = 0;; al++) //买al箱食物
        {
            if (ak == 0 && al == 0) continue;
            int tmp2 = al * m2;
            if (tmp1 + tmp2 > M || f[d][u][k][l] - 2*ak*c1
- 2*al*c2 < 0) //超出负重或钱不够用
                break;
            mx(d, u, k + ak, l + al, d, u, k, l, - 2*ak*c1
- 2*al*c2); //当前层的更新
        }
    }
}

const int N = 1e5 + 50;
short ans[N], tot = 0;
Data ansp[N];

void get_weather()
{
    wt[0] = rand() % 2;
    for (int i = 1; i < day; i++)
    {
        double sun = 0.5, high = 0.5, sand = 0; //没有沙暴,
各五十
        for (int j = 1; j <= 2; j++)
        {
            if (i - j <= 0) break;
            if (wt[i - j] == 0)
            {
                high += sun * 0.15;
                sun *= 0.85;
            }
            else
            {
                sun += high * 0.15;
                high *= 0.85;
            }
        }
        wt[i] = rand()%1000 + 1 <= sun * 1000 ? 0 : 1;
```

```
    }
}
int pcnt[2];
int fans[N], ftot, ts[N];
void get_ans()
{
    short mxans = -1;
    Data st;
    for (int i = 1; i <= day; i++)
        for (int k = 0; k <= v1; k++)
        {
            int lim = (M - k * m1) / m2;
            for (int l = 0; l <= lim; l++)
            {
                if (f[i][ed][k][l] > mxans)
                {
                    mxans = f[i][ed][k][l] + k*c1/2 +
k*c2/2;
                    st = Data(i, ed, k, l);
                }
            }
        }
    fans[++ftot] = mxans;
    ts[st.i]++;
    while (st.i)
    {
        ansp[++tot] = st;
        ans[tot] = f[st.i][st.j][st.k][st.l];
        if (st.j == 4) {++pcnt[1]; return ;}
        st = p[st.i][st.j][st.k][st.l];
    }
    ++pcnt[0];
}
void dp()
{


    memset(f, -0x3f, sizeof(f)); //不可达状态设为 -inf

    for (int k = 0; k <= v1; k++) //带多少箱水

    {

        int lim = (M - k * m1) / m2;  //最多还能带多少箱食物

        for (int l = 0; l <= lim; l++)
```

```
            f[0][1][k][l] = c0 - k*c1 - l*c2; //初值
        }

        for (int w, d = 0; d < day; d++) //天
        {
            w = wt[d]; //当前天气

            for (int u = 1; u <= n; u++) //当前点
            {
                if (pv[u]) //当前位于村庄
                    for (int k = 0; k <= v1; k++)
                    {
                        int lim = (M - k * m1) / m2;
                        for (int l = 0; l <= lim; l++)
                        {
                            if (f[d][u][k][l] < 0) continue;
                            buy(d, u, k, l);
                        }
                    }

                for (int k = 0; k <= v1; k++)
                {
                    int lim = (M - k * m1) / m2;
                    for (int l = 0; l <= lim; l++)
                    {
                        if (f[d][u][k][l] < 0) continue; //当前状
态不合法，跳过

                        mx(d + 1, u, k - r1[w], l - r2[w], d, u,
k, l); //1. 停

                        if (w != 2) //不是沙暴
                            for (int v, i = head[u]; i; i =
e[i].nxt) //2. 移动
                            {
                                v = e[i].v; //下一个点
```

```
                        mx(d + 1, v, k - (r1[w]<<1), l -
(r2[w]<<1), d, u, k, l); //两倍消耗
                    }
                if (pm[u]) //当前位于矿场
                    mx(d + 1, u, k - 3 * r1[w], l - 3 *
r2[w], d, u, k, l, mny); //3.挖矿
            }
        }
    }
}
int main()
{
    freopen("gk3.out", "w", stdout);
    srand(time(0));
    build_G(); //建图
    int T = 100;
    while (T--)
    {
        get_weather();
        dp();
        get_ans();
    }
    cout<<pcnt[0]<<" "<<pcnt[1]<<endl;
    for (int i = 1; i <= ftot; i++)
        cout<<fans[i]<<" ";
    puts("");
    for (int i = 1; i <= 10; i++)
        cout<<ts[i]<<" ";
    return 0;
}
```

第二问第四关动态规划

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <cstdio>
#include <cmath>
#include <algorithm>
#include <string>
```

```cpp
#include <cstring>
using namespace std;

const int n = 10; //简化后节点数

const int M = 1200;  //负重上限

const int day = 30; //30天

const int c0 = 10000;  //初始资金

const int c1 = 5, c2 = 10;  //水和食物的基准价格

const int m1 = 3, m2 = 2;  //水和食物的每箱质量


int etop, head[n + 1];
struct Edge
{
    int v, nxt;
}e[n * n];
void add(int u, int v)
{
    e[++etop].v = v;
    e[etop].nxt = head[u];
    head[u] = etop;
}
void adds(int u, int v) {add(u, v); add(v, u);}

int pv[n+1], pm[n+1]; //是否为村庄、矿山

int ed = 10; //10为终点

void build_G()
{
    for (int i = 1; i <= 9; i++)
        if (i != 6) adds(i, i + 1);
    adds(5, 7);
    adds(6, 8);
    pm[7] = 1;
    pv[6] = 1;
}

//晴朗0 高温1 沙暴2

int wt[day];
```

```
int r1[] = {3, 9, 10}, r2[] = {4, 9, 10}; // 基础消耗量

int mny = 1000; //挖矿收益


struct Data
{
    char i, j;
    short k, l;
    Data () {}
    Data (char _i, char _j, short _k, short _l)
    {
        i = _i, j = _j, k = _k, l = _l;
    }
}p[day + 1][n + 1][M/m1 + 1][M/m2 + 1]; //path

short f[day + 1][n + 1][M/m1 + 1][M/m2 + 1];

void mx(int ni, int nj, int nk, int nl, int i, int j, int
k, int l, int val=0)
{
    if (nk < 0 || nl < 0) return ;
    if (f[ni][nj][nk][nl] < f[i][j][k][l] + val)
    {
        f[ni][nj][nk][nl] = f[i][j][k][l] + val;
        p[ni][nj][nk][nl] = Data(i, j, k, l);
    }
}
int v1 = M / m1;
void buy(int d, int u, int k, int l)
{

    for (int ak = 0;; ak++) //买ak箱水

    {
        int tmp1 = (k + ak) * m1 + l * m2;

        if (tmp1 > M) break; //超出负重上限


        for (int al = 0;; al++) //买al箱食物

        {
            if (ak == 0 && al == 0) continue;
            int tmp2 = al * m2;
            if (tmp1 + tmp2 > M || f[d][u][k][l] - 2*ak*c1
- 2*al*c2 < 0) //超出负重或钱不够用
```

```
            break;
        mx(d, u, k + ak, l + al, d, u, k, l, - 2*ak*c1
- 2*al*c2); //当前层的更新
        }
    }
}

const int N = 1e5 + 50;
short ans[N], tot = 0;
Data ansp[N];

void get_weather()
{
    wt[0] = rand() % 2;
    for (int i = 1; i < day; i++)
    {
        double sun = 0.3, high = 0.5, sand = 0.2;
        for (int j = 1; j <= 2; j++)
        {
            if (i - j <= 0) break;
            if (wt[i - j] == 0)
            {
                high += sun * 0.15;
                sun *= 0.85;
            }
            else if (wt[i - j] == 1)
            {
                sand += high * 0.15;
                high *= 0.85;
            }
            else
            {
                sun = 0.3, high = 0.5, sand = 0.2;
            }
        }
        int tmp = rand()%1000 + 1;
        wt[i] = tmp <= sun * 1000 ? 0 : (tmp >
(sun+high)*1000 ? 2 : 1);
    }
}
void get_ans()
{
    short mxans = -1;
```

```
    Data st;
    for (int i = 1; i <= day; i++)
        for (int k = 0; k <= v1; k++)
        {
            int lim = (M - k * m1) / m2;
            for (int l = 0; l <= lim; l++)
            {
                if (f[i][ed][k][l] > mxans)
                {
                    mxans = f[i][ed][k][l] + k*c1/2 +
k*c2/2;
                    st = Data(i, ed, k, l);
                }
            }
        }
    tot = 0;
    while (st.i)
    {
        ansp[++tot] = st;
        ans[tot] = f[st.i][st.j][st.k][st.l];
        st = p[st.i][st.j][st.k][st.l];
    }
    reverse(ans + 1, ans + tot + 1);
    reverse(ansp + 1, ansp + tot + 1);
    for (int i = 1; i <= tot; i++)
    {
        Data u = ansp[i];
        printf("%d %d %d %d %d\n", u.i, u.j, ans[i], u.k,
u.l);
    }
}
void dp()
{
    memset(f, -0x3f, sizeof(f)); //不可达状态设为 -inf

    for (int k = 0; k <= v1; k++) //带多少箱水

    {

        int lim = (M - k * m1) / m2;  //最多还能带多少箱食物

        for (int l = 0; l <= lim; l++)

            f[0][1][k][l] = c0 - k*c1 - l*c2; //初值

    }
```

```
    for (int w, d = 0; d < day; d++) //天
    {
        w = wt[d]; //当前天气
        for (int u = 1; u <= n; u++) //当前点
        {
            if (pv[u]) //当前位于村庄
                for (int k = 0; k <= v1; k++)
                {
                    int lim = (M - k * m1) / m2;
                    for (int l = 0; l <= lim; l++)
                    {
                        if (f[d][u][k][l] < 0) continue;
                        buy(d, u, k, l);
                    }
                }
            for (int k = 0; k <= v1; k++)
            {
                int lim = (M - k * m1) / m2;
                for (int l = 0; l <= lim; l++)
                {
                    if (f[d][u][k][l] < 0) continue; //当前状
态不合法，跳过
                    mx(d + 1, u, k - r1[w], l - r2[w], d, u,
k, l); //1. 停
                    if (w != 2) //不是沙暴
                        for (int v, i = head[u]; i; i =
e[i].nxt) //2. 移动
                        {
                            v = e[i].v; //下一个点
                            mx(d + 1, v, k - (r1[w]<<1), l -
(r2[w]<<1), d, u, k, l); //两倍消耗
```

```
                    }
                if (pm[u]) //当前位于矿场
                    mx(d + 1, u, k - 3 * r1[w], l - 3 *
r2[w], d, u, k, l, mny); //3.挖矿
            }
        }
    }
}
int main()
{
    freopen("gk4.out", "w", stdout);
    srand(time(0));
    build_G(); //建图
    int T = 5;
    while (T--)
    {
        get_weather();
        dp();
        get_ans();
    }
    return 0;
}
```