穿越沙漠策略研究

摘要

本文研究了完成穿越沙漠游戏，在遵守游戏规则的情况下寻求最优策略即保留尽可能多的资金。对于已知天气与未知天气的单个玩家、已知天气与未知天气的多个玩家四种情况，分别制定一般情况下玩家的最优策略，并具体讨论对应的六个关卡的具体实施策略以及获得的最优解。

模型一基于已知所有天气的单个玩家制定最优策略，模型主要采用线性规划的方法，通过去点策略、去留策略、挖矿策略、购买策略等多个策略，形成对应的线性规划模型，确定行进路线、在起点、村庄购买水和食物的数量以及在矿山的挖矿天数等具体解，获得较优解；再使用启发式策略，不断更新路线与购买策略，使得较优解不断逼近最优解。以模型一为基础，我们具体分析第一问中“第一关”和“第二关”一般情况下玩家的最优策略，并获得最优收益。

模型二基于贝叶斯概率统计模型，通过利用条件概率，先验概率与后验概率的相互转化，分析已知在前几天天气的情况下对后期天气的影响问题，分别讨论在不考虑沙暴天气下的挖矿策略、考虑沙暴天气的挖矿策略以及资源消耗问题，得出不同情况下的行动策略。按照模型二为基础，结合第二问中的第三关和第四关的具体数据进行具体讨论，得出相应的策略，以获得最优收益。

模型三基于博弈论之纳什均衡，对于天气已知的情况每名玩家都选择平衡策略，对于天气未知的情况我们采用第二问中的贝叶斯概率统计模型，预测第二天的天气综合竞争者的决策做出尽可能稳健的策略决策。以模型三为基础，结合第三问中的第五关和第六关的具体数据寻求最优策略与收益。

关键词：线性规划、启发式策略、贝叶斯概率统计模型、博弈论、纳什均衡

一、问题重述

穿越沙漠游戏规则：

玩家利用已确定的初始资金购买一定数量的水和食物，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，穿越沙漠，必须在截止日期或之前到达终点，目标是在规定时间内到达终点，并保留尽可能多的资金。沙漠中有各种区域，包括起点、终点、矿山、村庄这些特殊区域以及其他普通的区域。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（拥有公共边界的两个区域称为相邻，仅有公共顶点而没有公共边界的两个区域不视作相邻），也可在原地停留。沙暴日必须在原地停留。

矿山：玩家在矿山停留时，可以选择通过挖矿获得资金，挖矿一天获得的资金量为基础收益，到达矿山当天不能挖矿，任何天气（包括沙暴）都可挖矿。

村庄：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。

天气因素：途中会遇到不同的天气，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，不同的天气规定的基础消耗量各不相同。

资源因素：穿越沙漠需要水和食物两种资源，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规定负重上限。若未到达终点而水或食物已耗尽，视为游戏失败。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，水或食物的基准价格是购买每一箱水或食物花费的资金。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。

基础消耗量的定义是玩家在原地停留一天消耗的资源数量，行走一天消耗的资源数量为基础消耗量的 2 倍，挖矿一天消耗的资源数量为基础消耗量的 3 倍。

试建立数学模型解决以下问题：

（1）已知整个游戏时段内每天天气状况，制定一般情况下仅一名玩家的最优策略，使得在规定时间内到达终点并且保留最多的资金，利用一般策略求解特定情况即“第一关”与“第二关”。

（2）仅知当天的天气状况，制定一般情况下仅一名玩家的最优策略，利用一般策略求解特定情况即“第三关”和“第四关”

（3）现在同时进行游戏的有 n 名玩家，在原有规则的基础上增加规则：$k(2 \leq k \leq n)$人在一天内从区域 A 走至区域 B 则消耗变为基础消耗的$2k$倍，k 人在同一天同一个矿山挖矿收益变为基础收益的$\frac{1}{k}$，多于 2 人在同一村庄购买水和食物消耗变为基础价格的 4 倍。

问题 1：给出在已知天气情况下的一般策略，并对第五关进行具体分析

问题 2：给出在未知天气情况下的一般策略，并对第六关进行具体分析

二、符号表

序号	符号	含义
1	d_{ij}	从 i 号到 j 号区域所需要的天数（距离）
2	W_k	第 k 天的天气
3	$C_i^f (i\epsilon 1,2,3)$	在第 i 种天气下食物每天的消耗
4	$C_i^w (i \in 1,2,3)$	在第 i 种天气下水每天的消耗
5	$Cost_{ij}^w$	从 i 号到 j 号区域水的消耗量
6	$Cost_{ij}^f$	从 i 号到 j 号区域食物的消耗量
7	B_f	食物的基准价格
8	B_w	水的基准价格
9	w_f	食物的每箱重量
10	w_w	水的每箱重量
11	e	挖矿每天的收入
12	R_i^f	第 i 次购买的食物数量
13	R_i^w	第 i 次购买的水的数量
14	S_i^f	第 i 天结束还剩的食物数量
15	S_i^w	第 i 天结束还剩的水的数量
16	M_i	第 i 天结束后资金
17	M_0	初始资金
18	D	截至时间
19	w_{max}	负重上限
20	K	单位价值比

三、已知天气的单个玩家策略

3.1 去点策略

将地图中各区域化成各节点，邻接区域之间连接无向边，从而使地图画为网络图

节点类型可分为起点、终点、村庄、矿山这些特殊点以及其他普通点，我们的策略应该围绕特殊点而制定，即我们的运动总是从一个特殊点去往另一个特殊点，并且路径是两点间的最短路径，因此我们可以确定我们可能会经过的点，即特殊点以及特殊点两两之间的最短路径上的点，我们通过 Dijkstra 算法确定特殊点两两之间的最短路径并标记经过的节点。根据这种规则，我们可以精简地图，去除我们完全没有必要经过的点（即普通点中不在最短路径上的点），其中精简后的网络图中可能存在地位相同的点，即这种点存在与否不会改变最短路径的长度，所以我们对地位相同的多个点保留一个出度更少的点即可（为达到点图最简化的目的）。

邻接矩阵 Adj = $\begin{pmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ a_{i1} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nj} & \cdots & a_{nn} \end{pmatrix}$

对于第一关，我们采取去点策略后可获得如下邻接矩阵：（其中 1 是起点，12 是矿山，15 是村庄，27 是终点）

	1	9	12	14	15	21	23	25	26	27	
1	1	0	0	0	0	0	0	1	0	0	1
9	0	1	0	0	1	1	0	0	0	0	2
12	0	0	1	1	0	0	0	0	0	0	3
14	0	0	1	1	1	0	0	0	0	0	4
15	0	1	0	1	1	0	0	0	0	0	5
21	0	1	0	0	0	1	1	0	0	1	6
23	0	0	0	0	0	1	1	0	1	0	7
25	1	0	0	0	0	0	0	1	1	0	8
26	0	0	0	0	0	0	1	1	1	1	9
27	0	0	0	0	0	1	0	0	1	1	10
	1	2	3	4	5	6	7	8	9	10	

邻接矩阵对应精简网络图：

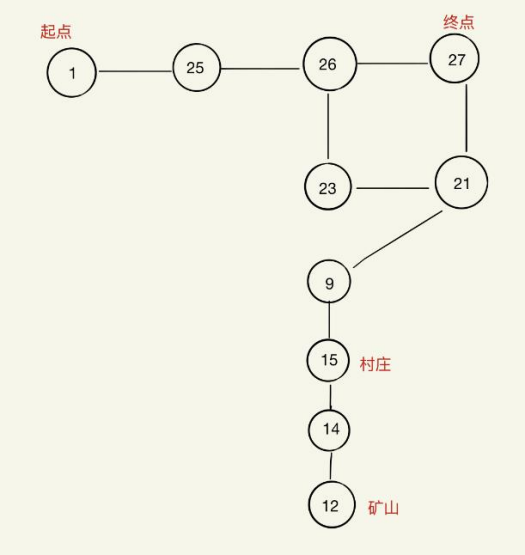

图 1 邻接矩阵对应精简网络图

3.2 去留策略

去留策略是根据当天天气是否选择移动或者选择等待消耗更小的其他天气：

若当前为 A 天气，选择直接移动的消耗为$2\begin{bmatrix} c_A^W \\ c_A^f \end{bmatrix}$，选择等待 k 天到第一次遇见 B 天气的消耗为$k\begin{bmatrix} c_A^W \\ c_A^f \end{bmatrix}+2\begin{bmatrix} c_B^W \\ c_B^f \end{bmatrix}$.

由此可以定义判别式$f(k)$：

$$f(k)=2\begin{bmatrix} c_A^W \\ c_A^f \end{bmatrix}-(k\begin{bmatrix} c_A^W \\ c_A^f \end{bmatrix}+2\begin{bmatrix} c_B^W \\ c_B^f \end{bmatrix}) \tag{1}$$

其中c_A^W、c_A^f是天气 A 的水和食物的基础消耗量，c_B^W、c_B^f是天气 B 的水和食物的基础消耗量。

当$f(k)\le 0$时，表明选择当天为 A 天气时直接移动的消耗更小，因此选择直接移动；

否则选择等待 k 天到 B 天气再移动。

食物与水的消耗系数为$2t_1C_1+2t_2C_2+t_3C_3$. 而为了躲避恶劣天气，选择t_2^1天高温天气前进，t_2^2天停在原地，导致了总共有 k 天停止前进的消耗为

$$2t_1'(C_1^f + C_1^w) + 2t_2^1\left(c_2^f + c_2^w\right) + t_2^2\left(c_2^f + c_2^w\right) + t_3'(C_3^f + C_3^w)$$

$$t_1' + t_2^1 = d_{ft}$$

$$t_2^2 + t_3' = k$$

$$t_3' \geq t_3, t_1' \geq t_1$$

食物与水的消耗系数为$2t_1'C_1 + \left[2\left(d_{ft} - t_1'\right) + k - t_3'\right]C_2 + t_3'C_3$

即当$(2\Delta t_1 + t_3' - k)C_2 \geq \Delta t_3 C_3 + 2\Delta t_1 C_1$时，选择在高温天气直接行进更加节省资源。上式中$\Delta t_i$是因为等待而出现的第 i 种天气的天数。

3.3 挖矿策略

考虑一般情况下从 f 点到 t 点的移动，不做耽搁直接去的消耗为

$$Cost_{ft} = \sum_{i=1}^{2} 2t_i \begin{bmatrix} C_i^w \\ C_i^f \end{bmatrix} + t_3 \begin{bmatrix} C_3^w \\ C_3^f \end{bmatrix} \tag{2}$$

$$\sum_{i=1}^{2} t_i = d_{ft}$$

其中d_{ft}是 f 点到 t 点的距离（即不考虑沙暴天气，从 f 点移动到 t 点的天数），

t_1表示晴朗，t_2表示高温，t_3表示沙暴。

则选择直接由起点前往终点的消耗为：

$$Cost_{se} = \sum_{i=1}^{2} 2t_i \begin{bmatrix} C_i^w \\ C_i^f \end{bmatrix} + t_3 \begin{bmatrix} C_3^w \\ C_3^f \end{bmatrix}$$

$$\sum_{i=1}^{2} t_i = d_{se}$$

其中d_{se}是起点到终点的距离，t_i 表示第 i 种天气的天数。

当在到达矿山时，根据当前的天气，挖矿的消耗为基础消耗量的 3 倍，即

$$3\sum_{i=1}^{3} t_{di} \begin{bmatrix} C_i^w \\ C_i^f \end{bmatrix}, i \in \{1,2,3\}$$

而挖矿的收益为et_d，可以定义函数$f(e,t_d)$是挖矿收益减去挖矿的耗费与路上耗费：

$$f(e,t_d)=et_d-[3\sum_{i=1}^{3}t_{di}\begin{bmatrix}C_i^w\\C_i^f\end{bmatrix}+\sum_{i=1}^{2}2t_i\begin{bmatrix}C_i^w\\C_i^f\end{bmatrix}+t_3\begin{bmatrix}C_3^w\\C_3^f\end{bmatrix}] \tag{3}$$

$$t_1+t_2=d_{sm}+d_{me}$$

$$\sum_{i=1}^{3}(t_i+t_{di})\le D$$

定义判别式G：

$$G=f(e,t_d)+\sum_{i=1}^{2}2t_i'\begin{bmatrix}C_i^w\\C_i^f\end{bmatrix}+t_3'\begin{bmatrix}C_3^w\\C_3^f\end{bmatrix} \tag{4}$$

$$\sum_{i=1}^{2}t_i'=d_{se}$$

当且仅当$G\ge 0$时，去矿山挖矿比由起点直接去终点收益更高。

由于起点直接去终点的策略很简单，因此下面的策略主要针对需要去挖矿的情况。

3.4 购买策略

1. 起点购买策略

初始购买的约束条件为：

$$\begin{cases}R_0^f w_f+R_0^w w_w\le w_{max}\\R_0^f B_f+R_0^w B_w\le M_0\\R_0^f\ge Cost_{sv}^f\\R_0^w\ge Cost_{sv}^w\end{cases} \tag{5}$$

其中R_0^w、R_0^f表示起点购买的水和食物的箱数，w_w、w_f表示水和食物每箱的重量

B_w、B_f表示购买一箱水和食物的资金

w_{max}表示最大负重，M_0表示初始资金

$Cost_{sv}^{w}$、$Cost_{sv}^{f}$表示起点到村庄路程上水和食物的消耗量

在满足以上约束的情况下，进行下列判别：

定义物品的单位价值是每箱价格：每箱质量，即$\frac{B}{w}$;

从而可以定义单位价值比K：

$$K = \frac{\frac{B_w}{w_w}}{\frac{B_f}{w_f}} \tag{6}$$

若$K > 1$ 则买水，若$K < 1$，则买食物，若$K = 1$，则 1:1 购买。

在满足起点到村庄需消耗资源量的情况下购买判别出的资源 T 直至达到最大负重或初始资金。

设x、y分别是在起点除去起点至村庄的路程消耗量额外购买的水和食物，x_0、y_0分别是起点至村庄水和食物的路程消耗量，R_1^w、R_1^f分别是第一次经过村庄购买的水和食物。

根据约束条件，可列出以下线性规划[1](7)：

$$\begin{cases} w_w x + w_f y + w_w Cost_{sv}^{w} + w_f Cost_{sv}^{f} \le w_{max} \\ w_w x + w_f y + w_w x_1 + w_f y_1 \le w_{max} \\ x + R_1^w \ge Cost_{vm}^{w} + 3\sum_{i=1}^{3} t_{di}\, C_i^w + Cost_{me}^{w} \\ y + R_1^f \ge Cost_{vm}^{f} + 3\sum_{i=1}^{3} t_{di}\, C_i^f + Cost_{me}^{f} \\ R_0^w = x + x_0 \\ R_0^f = y + y0 \\ R_0^w B_w + R_0^f B_f \le M_0 \\ R_1^w B_w + R_1^f B_f \le M_0 - (R_0^w B_w + R_0^f B_f) \\ \max\ T \end{cases} \tag{7}$$

2. 村庄购买策略

经过村庄有两种情况：1.从起点至矿山的路途中经过村庄；2.挖矿期间资源不足回到村庄补给资源

（1） 从起点至矿山的路途中经过村庄

由于我们在起点购买资源时已经尽可能地达到最大负重，因此在村庄最多补给

$$R_1 = \sum_{i=1}^{2} 2t_i \begin{bmatrix} C_i^w \\ C_i^f \end{bmatrix} + t_3 \begin{bmatrix} C_3^w \\ C_3^f \end{bmatrix}$$

$$\sum_{i=1}^{2} t_i = d_{sv}$$

$$s.t.\ \ R_1^w . B_w + R_1^f B_f \le M_0 - (R_0^f B_f + R_0^w B_w)$$

根据上述条件，可更新线性规划(8)：

$$\begin{cases} w_w x + w_f y + w_w Cost_{sv}^w + w_f Cost_{sv}^f \le w_{max} \\ w_w x + w_f y + w_w x_1 + w_f y_1 \le w_{max} \\ x + R_1^w \ge Cost_{vm}^w + 3\sum_{i=1}^{3} t_{di}\, C_i^w + Cost_{me}^w \\ y + R_1^f \ge Cost_{vm}^f + 3\sum_{i=1}^{3} t_{di}\, C_i^f + Cost_{me}^f \\ R_0^w = x + x_0 \\ R_0^f = y + y0 \\ R_1^w = \sum_{i=1}^{2} 2t_i C_i^w + t_3 C_3^w \\ R_1^f = \sum_{i=1}^{2} 2t_i C_i^f + t_3 C_3^f \\ \sum_{i=1}^{2} t_i = d_{sv} \\ R_0^w B_w + R_0^f B_f \le M_0 \\ R_1^w . B_w + R_1^f B_f \le M_0 - (R_0^f B_f + R_0^w B_w) \\ \max\ T \end{cases} \tag{8}$$

依据以上线性规划可以解得R_0^w、R_0^f、R_1^w、R_1^f

即确定起点购买的水和食物以及第一次经过村庄购买的水和食物。

（2） 挖矿期间资源不足回到村庄补给资源

第一次经过村庄后前往矿山挖矿直至资源不足，则可能需要第二次去村庄补给资源。

回到村庄补给资源后有两种选择：①直接回到终点；②回到矿山继续挖矿

决定如何选择取决于回到矿山继续挖矿的收益是否能大于消耗，我们可以发现这里引用 2.3 挖矿策略类似地定义$f_2(e,t_d)$ 是挖矿收益减去挖矿的耗费与路上耗费：

$$f_2(e,t_d') = et_d' - [3\sum_{i=1}^{3} t_{di}' \begin{bmatrix} C_i^w \\ C_i^f \end{bmatrix} + \sum_{i=1}^{2} 2t_i \begin{bmatrix} C_i^w \\ C_i^f \end{bmatrix} + t_3 \begin{bmatrix} C_3^w \\ C_3^f \end{bmatrix}] \tag{9}$$

$$t_1 + t_2 = d_{vm} + d_{me}$$

$$\sum_{i=1}^{3} (t_i + t_{di}') \le D - D_0$$

定义判别式G：

$$G = f(e,t_d') + \sum_{i=1}^{2} 2t_i' \begin{bmatrix} C_i^w \\ C_i^f \end{bmatrix} + t_3' \begin{bmatrix} C_3^w \\ C_3^f \end{bmatrix} \tag{10}$$

$$\sum_{i=1}^{2} t_i' = d_{ve}$$

其中e表示挖矿的基础收益，t_{di}'表示第二次挖矿在第 i 种天气下挖矿的天数

D表示总天数，D_0表示第二次到达村庄时花费的天数

d_{vm}表示村庄到矿山的距离，d_{me}表示矿山到终点的距离，d_{ve}表示村庄到终点的距离，

当且仅当$G > 0$时选择②，否则选择①

① 直接回到终点

这种选择只需要补给资源至满足从村庄到终点的资源并尽可能保证到达终点剩余的资源$S_e \to 0$：

$$R_2 = \sum_{i=1}^{2} 2t_i \begin{bmatrix} C_i^w \\ C_i^f \end{bmatrix} + t_3 \begin{bmatrix} C_3^w \\ C_3^f \end{bmatrix} - S_{v2} \tag{11}$$

$$\sum_{i=1}^{2} t_i = d_{ve}$$

$$s.t.\ \ R_2^w.B_w + R_2^f B_f \le M_0 - (R_0^f B_f + R_0^w B_w) - (R_1^f B_f + R_1^w B_w) + et_d$$

其中R_2表示第二次经过村庄时需购买的水和食物，S_{v2}表示第二次经过村庄时剩余的水和食物

d_{ve}表示村庄到终点的距离

② 回到矿山第二次挖矿

这种选择需要补给资源至满足村庄→矿山→终点的路程消耗量以及挖矿的消耗量并尽可能保证到达终点剩余的资源$S_e \to 0$：

$$R_2 = \sum_{i=1}^{2} 2t_i \begin{bmatrix} C_i^w \\ C_i^f \end{bmatrix} + t_3 \begin{bmatrix} C_3^w \\ C_3^f \end{bmatrix} + 3\sum_{i=1}^{3} t'_{di} \begin{bmatrix} C_i^w \\ C_i^f \end{bmatrix} + \sum_{i=1}^{2} 2t'_i \begin{bmatrix} C_i^w \\ C_i^f \end{bmatrix} + t'_3 \begin{bmatrix} C_3^w \\ C_3^f \end{bmatrix} - S_{v2} \tag{12}$$

$$\sum_{i=1}^{2} t_i = d_{vm}$$

$$\sum_{i=1}^{2} t'_i = d_{me}$$

$$R_2 = \begin{bmatrix} R_2^w \\ R_2^f \end{bmatrix}$$

$$s.t.\ \ R_2^w.B_w + R_2^f B_f \le M_0 - (R_0^f B_f + R_0^w B_w) - (R_1^f B_f + R_1^w B_w) + et_d$$

其中R_2表示第二次经过村庄时需购买的水和食物，S_{v2}表示第二次经过村庄时剩余的水和食物

t'_{di}表示第二次挖矿在第 i 种天气下挖矿的天数

d_{vm}表示村庄到矿山的距离，d_{me}表示矿山到终点的距离

3.5 启发式策略

启发式策略是指用启发式的方法通过试错不断更新优化购买以及路线策略。

通过上述策略获得的解属于较优解，还无法达到最优解，因此我们需要使用启发式搜索使得我们的较优解逼近最优解。

s:起点 v:村庄 m:矿山 e:终点

启发式搜索方向：

（1） 最终剩余资源S_e^w、S_e^f皆等于 0

经过线性规划确定的R_0、R_1、R_2，可能无法使得S_e等于 0，因此我们以R_0、R_1、R_2的购买策略根据我们当前的路线（s→v→m→v→e）求解出S_e

得到更新策略：

$$R_2 = R_2 - S_e$$

（2） 路线更新

我们可选的路线有

①s→e，②s→m→e，③s→m→v→e，④s→v→m→e，⑤s→v→m→v→e，

⑥s→v→m→v→m→v→e

根据我们的策略确定路线，可形成如下决策树：

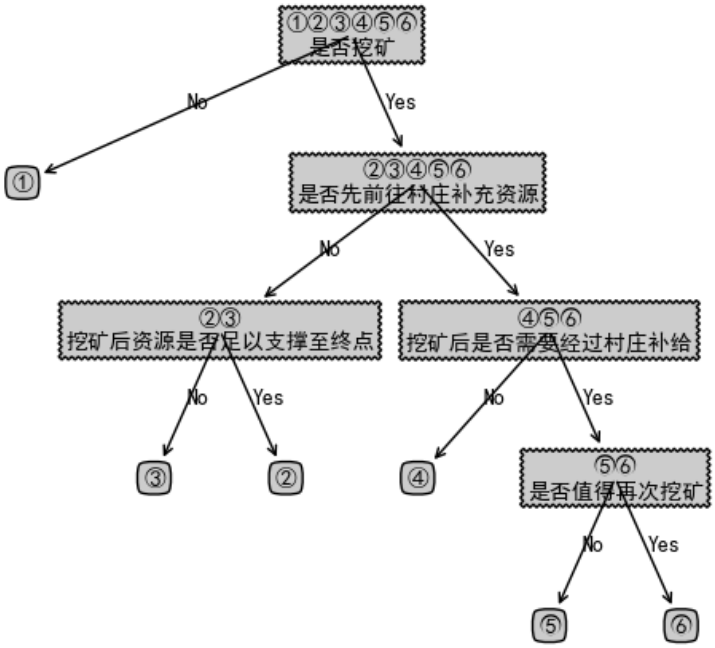

图 2 策略决策树

其中“是否值得再次挖矿”可循环判断直至天数或资金不足以再次挖矿停止。

3.6 第一关具体策略

如果不去挖矿，直接前往终点路程消耗为$2 \times (8 + 8 + 5) = 42$箱水和$2 \times (6 + 6 + 7) = 38$箱食物共计$42 \times 5 + 38 \times 10 = 590$元，故剩余$10000 - 590 = 9410$元，

根据我们的一般策略的，

1、去点：已述
2、挖矿：基础收益远大于路上的消耗，所以我们选择用尽可能多的时间去挖矿，在村庄进行临时补给

高温挖矿： $1000 - 3 \times (8 \times 5 + 6 \times 10) = 700$

晴朗挖矿： $1000 - 3 \times (5 \times 5 + 7 \times 10) = 715$

高温等待： $-(8 \times 5 + 6 \times 10) = -100$

晴朗等待： $-(7 \times 5 + 7 \times 10) = -95$

高温移动： $-2 \times (8 \times 5 + 6 \times 10) = -200$

晴朗移动： $-2 \times (5 \times 5 + 7 \times 10) = -190$

3、购买策略:

$$K=\frac{\frac{c_w}{w_w}}{\frac{c_f}{w_f}}=\frac{1}{3}<1$$

因为初始购买较村庄购买便宜，所以要在满足水的最低要求时尽可能的多买一些食物，还要尽可能保证最终到达终点时剩余的水和食物接近于 0

设d_{sv}表示起点去往村庄的距离，则起点至村庄的路程消耗量

$$Cost_{sv}=\sum_{i=1}^{2}2t_i\begin{bmatrix}C_i^f\\C_i^w\end{bmatrix}+t_3\begin{bmatrix}C_3^f\\C_3^w\end{bmatrix}$$

$$\sum_{i=1}^{2}t_i=d_{sv}$$

代入计算得$Cost_{sv}=\binom{98}{98}$.

设xy分别是在起点除去起点至村庄的路程消耗量额外购买的水和食物，x_1y_1表示第一次经过村庄购买的水和食物，另外水和食物要足够撑到终点，因为 10 号到达矿山，我们尽量让发生沙暴天气时，人在矿山且多挖矿赚钱，所以要人在 19 号离开矿山，而这八天如果一直挖矿需要$32+32+201=265$箱水，$24+24+183=231$箱食物，$265*3+231*2>1200$，故不可以一直挖矿，所以我们选择消耗较大的一天进行休息，也就是一个沙暴天气进行休息，这样我们就只需要$3*（32+32+181）=245$kg水，$2*（24+24+163）=211$kg食物

代入至线性规划组：$\begin{cases}3x+2y+3\times98+2\times98\le1200\\3x+2y+3x_1+2y_1\le1200\\y+y_1\ge211\\x+x_1\ge245\\\max\ y\end{cases}$

解得$\begin{cases}x=82\\y=232\\x_1=163\\y_1=0\end{cases}$

得初始购买$R_0^w=x+x_0=82+98=180$，$R_0^f=y+y_0=232+98=330$

4、第二次回到村庄是 20 号，从矿场去往终点需要 5 天，在 30 号到达终点，还可以挖三天矿，经过计算，挖矿产生3000元收益，折返矿山又绕远路返回需要消耗 $2\times(5+5)+3\times(5+8+10)+2\times(8+5+5+8+8)=157$箱水，共 $157\times10=1570$元，$2\times(7+7)+3\times(7+6+10)+2\times(6+7+7+6+6)=161$箱食物，共$161\times20=3220$元，付出大于收入，所以此时应当直接前往终点。

5、最终我们的资金是

$$M_E=M_0-(R_0^fB_f+R_0^wB_w)-(R_1^w.B_w+R_1^fB_f)-(R_2^w.B_w+R_2^fB_f)+et_d$$
$$=10000-4200-1630-740+7000=10430>9410$$

因此我们选择该方案获得最大资金 10430 元

3.7 第二关具体策略

第二关我们采用这样的路线：路过矿山先挖矿，去往村庄补给后前往距离终点更近的那个矿山继续挖矿，然后离开。

从起点直接去往矿山需要九天，消耗 114 箱水与 110 箱食物，设xy分别是在起点除去起点至矿山的路程消耗量额外购买的水和食物，此时的xy应满足:

$$3x+2y+3\times114+2\times110<1200$$

所以在矿山最多可以挖到 14 号即挖 5 天，需要消耗 133 箱水 117 箱食物，又有

$$x\geq133$$

$$y\geq117$$

xy只能是整数，固有唯一解

$$\begin{cases}x=134\\y=118\end{cases}$$

所以初始购买$R_0^w=x+x_0=134+114=248, R_0^f=y+y_0=118+110=228$

设x_1y_1表示第一次经过村庄购买的水和食物，第一次到达村庄时水和食物都只剩余 1 箱，而我们需要购买水和食物足够用于在第二个矿山挖矿并且撑到终点，从村庄前往第二个矿山需要 2 天，其中 17、18 号为沙暴，所以应需要 4 天，消耗$16+10+10+16=52$箱水，$12+10+10+12=44$箱食物，矿山到终点的路程为两天，为尽可能多挖矿我们选择最后两天去往终点，需消耗水 32 箱，食物 24 箱。

因为我们在 19 号到达第二个矿山，如果在第二个矿山九天都用来挖矿需要 $3\times(3\times8+5\times5+10)=177$箱，$3\times(3\times6+5\times7+10)=189$箱食物，而 $3\times(52+177+32)+2\times(44+189+24)>1200\ \mathrm{kg}$，故不可以一直挖矿，所以我们选择消耗较大的一天进行休息，也就是一个沙暴天气进行休息，这样我们就只需要

$3 \times （52 + 177 + 32 - 20） = 723\text{kg}$水，$2 \times （44 + 189 + 24 - 20） = 474\text{kg}$食物，$720 + 474 = 1194 < 1200\text{kg}$符合最大负重。
所以

$$\begin{cases} x_1 + 1 = \dfrac{723}{3} = 241 \\ y_1 + 1 = \dfrac{474}{2} = 237 \end{cases}$$

解得在村庄购买 240 箱水和 236 箱食物

所以我们最终的资金是

$$\begin{aligned} M_E &= M_0 - (R_0^f B_f + R_0^w B_w) - (R_1^w . B_w + R_1^f B_f) + et_d t_d' + et_d' \\ &= 10000 - 3520 - 7120 + 5000 + 8000 = 12360 > 10000 \end{aligned}$$

而直接从起点去往终点最终资金必然小于 10000 元，因此我们采用此方案获得最大资金 12360 元。

第一问的一二关具体解详见支撑材料的 Result.xlsx

四、未知天气的单个玩家策略

4.1 未知天气的普适性策略

当在仅知道当日天气为W_k的情况下，所作决策$P(action|W_k)$. 由贝叶斯公式可得

$$posterior = \frac{prior \times likelihood}{evidence}$$

$$P(action_i|W_k) = \frac{P(W_k|action_i)P(action_i)}{P(W_k)} \tag{13}$$

气温服从正态分布，即
对于地图中的每一个节点v,其度deg_v代表了**在当前节点v的决策数。**

在已知第i天的天气为W_k的情况下，预测第 $\text{i} + 1$天的天气W_{k+1}.

由贝叶斯公式可得，

$$P(W_{k+1}|W_k) = \frac{P(W_k|W_{k+1})P(W_{k+1})}{P(W_k)} \tag{14}$$

由于要保证在截止时间或之前到达终点，故当临近截至日期时，沙暴日出现的天数将产生重要影响，其出现天数的影响应该为指数级别. 当玩家所携带的资源较少需要补充时，无法根据确切的天气判定行动的时间以及补给之后的行动。玩家由 i 到 j 在

天数最小的情况下所需要的资源数量属于区间$[2d_{ij}C_1, t_3C_3 + 2d_{ij}\max\{C_1, C_2\}]$，即需要保证在到达终点之前所携带的资源量$\geq t_3C_3 + 2d_{ij}\max\{C_1, C_2\}$.

设现有的食物数量S_i^f：

(1)当没有沙暴天气时，仅需考虑高温与晴天的资源消耗问题。

在满足前文挖矿收益更高的前提下，在第 i 种天气挖矿的收益为$e - 3(C_i^f B_f + C_i^w B_w)$。在已知天气出现概率的情况下，每天挖矿收益的数学期望为

$$E_1 = \sum_{i=1}^{2} P_i[e - 3(C_i^f B_f + C_i^w B_w)] \tag{15}$$

由矿山到终点的时间为d_{me}，在路上的消耗的数学期望为

$$E_2 = 2P_1 d_{me}(C_1^f B_f + C_1^w B_w) + 2P_2 d_{me}(C_2^f B_f + C_2^w B_w) \tag{16}$$

故挖矿k天的最优化目标为$\max(kE_1 - E_2)$
经整理得

$$H = \underset{P_1}{argmax}([ke - 3k(C_1^f B_f + C_1^w B_w) - 2d_{me}(C_1^f B_f + C_1^w B_w)]P_1 + [ke - 3k(C_2^f B_f + C_2^w B_w) - 2P_2 d_{me}(C_2^f B_f + C_2^w B_w)](1 - P_1) \tag{17}$$

对上式求导可得

$$\frac{\partial H}{\partial P_1} = 3k[B_f(C_2^f - C_1^f) + B_f(C_2^w - C_1^w)] + 2d_{me}[B_f(C_2^f - C_1^f) + B_w(C_2^w - C_1^w)] \tag{18}$$

令上式为 0，即

$$B_f(C_2^f - C_1^f) + B_w(C_2^w - C_1^w) \geq 0 \tag{19}$$

时，

$$kE_1 - E_2 \propto P_1$$

即当满足(19)式时，晴天天气更为有利

(2)当存在沙暴天气时，定义在路上的损耗函数为

$$E_2' = 2P_1 d_{me}(C_1^f B_f + C_1^w B_w) + 2P_2 d_{me}(C_2^f B_f + C_2^w B_w) + e^{P_3(d_{me}+k)}(C_2^f B_f + C_2^w B_w) \tag{20}$$

此时在挖矿 k 天的最优化目标为$\max(k\sum_{i=1}^{3} P_i[e - 3(C_i^f B_f + C_i^w B_w)] - E_2')$

上式对 k 求导可得$\sum_{i=1}^{3} P_i[e - 3(C_i^f B_f + C_i^w B_w)] - P_3 e^{P_3(d_{me}+k)}(C_2^f B_f + C_2^w B_w)$
令上式等于 0，即得挖矿的最佳天数为

$$k = \max\{0, \frac{1}{P_3}\ln\left(\frac{1}{P_3}\sum_{i=1}^{3} P_i\left[e - 3\left(C_i^f B_f + C_i^w B_w\right)\right] - \left(C_2^f B_f + C_2^w B_w\right)\right)\} \quad (21)$$

4.2 第三关具体分析

第三关中，所给条件显然满足上述等式，从起点到矿山总共需要 3 天，从矿山到终点总共需要 2 天，而从起点直接到终点需要 4 天。结合题中所给条件，在高温天气挖矿消耗大于收益，在晴天挖矿赚 35 元。在第一天选择到达 4。此时已知前两天的天气，设为W_1, W_2.其第三、四天天气的概率为$P(W_3W_4|W_1W_2)$。带入上文所述损耗函数，我们认为在如此条件下，仅在晴天天气较多的情况下应取矿山，其余天气应该去终点。

4.3 第四关具体分析

表 1　1 人时各天气收益

天气	收益
高温(挖)	$200 - 3 \times (9 \times 5 + 9 \times 10) = -205$
晴朗(挖)	$200 - 3 \times (3 \times 5 + 4 \times 10) = 35$
高温(停)	$-(9 \times 5 + 9 \times 10) = -135$
晴朗(停)	$-(3 \times 5 + 4 \times 10) = -55$
高温(走)	$-2 \times (9 \times 5 + 9 \times 10) = -270$
晴朗(走)	$-2 \times (3 \times 5 + 4 \times 10) = -110$

从起点到矿山和到村庄距离一样远，存在出现沙暴的可能，大概率为 5 天，所以我们第一步要从起点往 13 号地点出发，通过概率模型预测未来的天气情况，如果晴天的天数较多且连续，则前往村庄挖矿，否则去村庄进行一次补给就前往终点

五、多个玩家策略

5.1 已知天气的一般性策略

首先去点策略和第一问相同，得到更为简练的邻接矩阵，但挖矿策略消耗函数变为：

$$f(e, t_d) = \frac{et_d}{k} - [3\sum_{i=1}^{3} t_{di}\begin{bmatrix} C_i^w \\ C_i^f \end{bmatrix} + \sum_{i=1}^{2} 2t_i\begin{bmatrix} C_i^w \\ C_i^f \end{bmatrix} + t_3\begin{bmatrix} C_3^w \\ C_3^f \end{bmatrix}] \quad (22)$$

$$\sum_{i=1}^{2} t_i' = d_{se}$$

这里的村庄多人购买花费会再次加倍，所以购买策略应与第一问相同，尽可能多的在起点购买。
每人在第一天都前进至在去往终点的最短路径上的下一个点，之后如果共行的人超过两人就采用随机挑选次近路线的方法规避最近路线上人数过多造成的消耗。

5.2 第五关具体分析

对于第五关，挖矿收益

表 2 k 人时各天气收益

天气	收益
高温(挖)	$\frac{200}{k} - 3\times(9\times5+9\times10) < -205$
晴朗(挖)	$\frac{200}{k} - 3\times(3\times5+4\times10) < 35$
高温(停)	$-(9\times5+9\times10) = -135$
晴朗(停)	$-(3\times5+4\times10) = -55$
高温(走)	$-2k(9\times5+9\times10) < -270$
晴朗(走)	$-2k(3\times5+4\times10) < -110$

由上表可知，挖矿唯一的挣钱可能是只有一人在矿山，且当天必须是晴天。

对于 k=2 时，只有 2 人来说，策略分别为

$$S_1 = \{\alpha_1, \alpha_2\}$$

$$S_2 = \{\beta_1, \beta_2\}$$

其中α_1 α_2分别是挖矿，不挖矿，β同理。

根据博弈论之纳什均衡可得矩阵对策$\Gamma = \{S_1, S_2; A\}$[2]

其中$A = \begin{bmatrix} 100-3cost & 200-3cost \\ 200-3cost & -2cost \end{bmatrix}$，$cost$为当天的基础消耗，根据上表可以看出，当 k=2 时，主动挖矿的收益为负，所以要尽可能快的离开。

当 k>2 时同样如此，主动挖矿入不敷出，按照天气表，最早可以在 3 号到达矿山，最多可以挖 4 天的矿，最多可挣$35\times4=140$元，而去往矿山需要绕远一步的距离，挖完矿后就是高温天气，最少需要花费$2\times(9\times5+9\times10)=270$元，采用挖矿的策略不仅不会赚钱，还会因为绕远路而必须消耗更多的水和食物.这里只能采用不主动挖矿的策略，每人在第一天都前往 4 号地点，每个人随机选择 3、7、6 三个地点，选到 3 的人下一步去往矿山，来到矿山后会有三种选择，挖矿，去往 10 或 11 地点，最终到达终点；选到 7 号地点的人也有 3 种选择：11 号，12 号和 6 号，选到 11 号的人没有必要去矿山继续挖矿，按照挖矿判别式折返消耗大于挖矿收益，所以下一步都应该直接前往终点；选到 6 号的人有两种选择，可以直接到达终点也可以通过 12 号到达终点。

5.3 未知天气情况的一般性策略

第一步按照去往终点的最短路路径随机前往下一个地点，之后我们借鉴集成学习中的Stacking 方法的思想[3]，将第一天的所有人的决策放入学习器产生第一步结果，按照规避他人又尽量走近路的准则放入学习器中再产生下一步的决策与结果。其中还要考虑天气的情况，当预测之后较长一段时间不会出现晴朗，就选择不挖矿。

5.4 第六关具体分析

所有人从起点出发，走第一步时不知道其他人的策略，因为第六关这个地图较为对称，所以人在第一天随机前往 2 号或者 6 号地点，2 号和 6 号又可以随机前往 3、7 和 11 号这三个地点，如果别人的资源剩余量小于自己，并且未来会有晴天出现，那么就去挖矿。

六、模型的评价与改进

6.1 模型优点

1、本文主要建立线性规划模型求解，具有清晰直观的特点。在已知天气的情况下分各种具体情况讨论，得出了一般情况下从一个区域移动到另一个区域的一般性策略，根据一般性策略获得较优解，再利用启发式搜索的思想使得较优解不断逼近最优解。

2、原创度较高，文中的模型的建立均为原创公式推导，在文中给出推到步骤，环环相扣。

3、本文对游戏中复杂的情况进行了合理的假设与简化。

6.2 模型缺点

1、本文在天气未知的情况下所给出的模型对于已知条件的利用有所不足。

2、在多名玩家参与的情况下的博弈分析与概率预测分析的还不够透彻，模型单薄。

3、在天气未知的情况下，各种影响因素的相关性密切度不足。

4、在具体数据的分析中，针对具体策略的分析有一定程度上依赖于穷举搜索。

5、最优解的搜索策略还存在不足，难以证明解的全局最优性，有可能陷入局部最优。

参考文献

[1]司守奎,数学建模算法与应用[M],北京市:国防工业出版社,2017。
[2]刁在筠,运筹学[M],北京市:高等教育出版社,2016。
[3]周志华,机器学习理论导引[M],北京市:机械工业出版社,2020。

附录

文件列表：

1、迪杰斯特拉程序　　dij.cpp
2、决策树生成程序　　drawtree.py
3、第一关线性规划　　第一关线性规划.m
4、第一问 Result 文件　Result.xlsx
5、决策树图片　　DecisionTree.png

源代码：

迪杰斯特拉程序(dij.cpp):

```
#include<stdio.h>
#include<cstring>
#include<iostream>
#define INF 0x3f3f3f3f
using namespace std;
const int mSize=100;
int a[2][2] = {1, 0, 2, 3};
int maze[mSize][mSize];
int d[mSize];
bool used[mSize];
int v;//顶点数
int s;//开始点
void dijkstra(int s)
{
	fill(d, d + v,INF);
	fill(used, used + v, false);
	d[s] = 0;
	while (true)
	{
		int ve = -1;//从尚未使用的顶点中选择距离最小的顶点

		for (int u = 0; u < v;u++)
```

```
            {
                if(!used[u]&&(ve==-1||d[u]<d[ve]))
                    ve = u;
            }
            if(ve==-1)
                break;
            used[ve] = 1;
            for (int u = 0; u < v;u++)
            {
                d[u] = min(d[u], d[ve] + maze[ve][u]);
            }
        }
}
int main()
{
    freopen("input.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    cin >> v>>s;//输入点的个数
    for (int i = 0; i < v;i++)
        for (int j = 0; j < v;j++)
            cin >> maze[i][j];
    dijkstra(s);
    for (int i = 0; i < v;i++)
        cout << d[i] << "\t";
    cout << endl;

    fclose(stdin);
    fclose(stdout);
    return 0;
}
```

决策树生成程序(drawtree.py):

```
import matplotlib.pyplot as plt


decision_node = dict(boxstyle="sawtooth", fc="0.8")
leaf_node = dict(boxstyle="round4", fc="0.8")
arrow_args = dict(arrowstyle="<-")

def plot_node(node_text, center, parent, node_type):
    # create_plot.ax1
    create_plot.ax1.annotate(node_text,xy=parent,xycoords='axes
fraction',xytext=center,textcoords='axes fraction',
```

```
va='center',ha='center',bbox=node_type,arrowprops=arrow_args)

def create_plot(tree):
    # 创建新图形并清空绘图区
    figure = plt.figure(1,facecolor='lightcyan')
    figure.clf()


    axprops = dict(xticks=[],yticks=[])
    create_plot.ax1 = plt.subplot(111,frameon = False,**axprops)
    plot_tree.total_w = float(get_num_leafs(tree))
    plot_tree.total_d = float(get_tree_depth(tree))
    plot_tree.x_off = -0.5/plot_tree.total_w
    plot_tree.y_off = 1.
    plot_tree(tree,(0.5,1.),'')
    plt.savefig('C:/Users/70922/Desktop/DecisionTree.png')
    plt.show()

def plot_mid_text(center,parent,txt_string):
    # 中间文本的坐标，上减下加上下
    x_mid = (parent[0]-center[0])/2. + center[0]
    y_mid = (parent[1]-center[1])/2. + center[1]
    create_plot.ax1.text(x_mid,y_mid,txt_string)

def plot_tree(tree,parent,node_text):
    # 计算叶子数量
    num_leafs = get_num_leafs(tree)
    depth = get_tree_depth(tree)
    first_str = list(tree.keys())[0]
    # 定位
    center = (plot_tree.x_off + (1.0+float(num_leafs))/2./plot_tree.total_w,plot_tree.y_off)
    # 中间的文本
    plot_mid_text(center,parent,node_text)
    # 节点
    plot_node(first_str,center,parent,decision_node)
    second_dict = tree[first_str]
    plot_tree.y_off -= 1./plot_tree.total_d
    # 开始画了，也是递归
    for key in second_dict.keys():
        if type(second_dict[key]).__name__ == 'dict':
            plot_tree(second_dict[key],center,str(key))
        else:
```

```
                plot_tree.x_off += 1./plot_tree.total_w
                plot_node(second_dict[key],(plot_tree.x_off,plot_tree.y_off),center,leaf_node)
                plot_mid_text((plot_tree.x_off,plot_tree.y_off),center,str(key))
        plot_tree.y_off += 1./plot_tree.total_d


def get_num_leafs(tree):
        """递归求叶子"""
        num_leafs = 0
        first_str = list(tree.keys())[0]
        second_dict = tree[first_str]
        for key in second_dict.keys():
                # 如果节点还是一个字典，就说明还可以继续
                if type(second_dict[key]).__name__ == 'dict':
                        num_leafs += get_num_leafs(second_dict[key])
                else:
                        # 每次发现一个节点就加一，最终的那个子叶也是加个 1 就跑了
                        num_leafs +=1

        return num_leafs

def get_tree_depth(tree):
        max_depth = 0
        first_str = list(tree.keys())[0]
        second_dict = tree[first_str]
        for key in second_dict.keys():
                if type(second_dict[key]).__name__ == 'dict':
                        this_depth = 1 + get_tree_depth(second_dict[key])
                else:
                        this_depth = 1
                if this_depth > max_depth:
                        max_depth = this_depth
        return max_depth

def retrieve_tree(i):

        list_of_trees = [{'①②③④⑤⑥\n 是否挖矿':
                                        {'No': '①','Yes':
                                                {'②③④⑤⑥\n 是否先前往村庄补充资源':
                                                        {'No': {'②③\n 挖矿后资源是否足以支撑至终点
':{'No':'③','Yes':'②'}},
                                                         'Yes': {'④⑤⑥\n 挖矿后是否需要经过村庄补
给':
```

```
                                                    {'No': '④','Yes':{'⑤⑥\n 是否值得再次挖矿':{'No':'⑤','Yes':'⑥'}}
                                                     }}}}}}]
    return list_of_trees[i]
plt.rcParams['font.sans-serif']=['SimHei']#黑体
mytree=retrieve_tree(0)
create_plot(mytree)
```

第一关线性规划(第一关线性规划.m):

```
f=[-1,-1,0,0];
intcon=2;
A=[
0,-1, 0,-1;
3, 2, 0, 0;
-1,0,-1, 0;
0, 0, 3, 2;
-1,0, 0, 0;
0,-1, 0, 0;
0,0, -1, 0;
0,0, 0, -1;
];%负号指把不等式两边同时乘以-1 把大于等于号转化为小于等于号
b=[-211,710,-245,490,0,0,0,0];
Aeq = [0 0 0 0];
beq = 0;
lb=[0,0,0,0];
ub=[];

[x,fval] = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub);
```