

基于动态规划、统计分析、静态博弈的穿越沙漠游戏策略设计

摘要

本文研究一个穿越沙漠小游戏的最佳策略问题。我们先在分析游戏机制的基础上编程实现了游戏逻辑模拟，再使用了图动态规划模型得到前两个具体关卡的最优解。再基于动态规划求解得到的数据，使用统计分析方法得出更复杂情况的最优策略规则。将这些规则应用于第三、四关后使用随机模拟的方法对具体方案进行调整和检验。最后使用静态博弈模型考虑 2 人、3 人共同参与游戏时的最优策略机制，在第五、六关上进行具体实现和讨论。

针对问题一：我们先将游戏机制转化为 python 程序，模拟后发现了对游戏状态及其转移规律的数学表示后论证了**动态规划**模型的合理性。在对这一模型算法的具体实现、复杂度进行充分分析后进行 C++ 实现，计算得到第一关最优策略得分 10470，用时 24 天，第二关最优策略得分 12730，用时 30 天。

从建模过程和最优方案中提炼出“走最短路”这一确定性策略和“能走则走”在目前游戏设定下成立的规律。

我们对动态规划模型进行了比较彻底的时间、空间复杂度优化(第一问例子中秒级求解)使其与游戏机制模拟一起成为后续问题使用大量数据分析和随机模拟验证的基础。

针对问题二：为了解决问题二中的随机性，我们先对最佳策略使用**统计分析方法**来设计决策方案，再使用**随机模拟进行验证**和评价。对于小规模第三关，我们用之前的动态规划模型对 1024 种天气情况进行逐一求解，对其中体现出的路线、行动、购买策略进行统计，得出结论应当直接在前三天走到终点，在起点买保证能够到达终点的最少物资量。我们使用随机模拟算法比较该方案与其他合理方案的期望收益、失败概率，发现其具有明显优势。严格计算和随机模拟均显示其过关得分期望为 9350。

对于规模较大不适合穷举最优解的第四关，我们采取抽样统计，通过随机生成不同的天气情况进行动态规划求解，分析最优解，结合第三关经验得出“买足物资，及时补给”的降低风险策略。得到的具体策略分析得到其失败概率控制在 2.5% 以内并且在一般天气情况下拥有超过初始资金的预期结果。

针对问题三：对于多人游戏，我们在分析机制后建立了静态博弈模型。我们先对第五关的几种较优单人策略进行分析，建立起两两间的博弈收益函数表，根据收益表分别进行纯策略和混合策略分析，最后得出均衡的解为双方采用 3 天到达终点的策略并会选择在起点购买足够生存的物资。第六关我们采取类似的方法对于两人局部的竞争进行分析得出了一系列局部策略，由此指导玩家进行决策优化。

最后我们分析了模型的优缺点和灵敏度，结果显示模型对于这一类问题具有比较好的适应性，提炼出的规则可以有效指导玩家决策。

关键字： 动态规划 统计分析 随机模拟 静态博弈

一、问题重述

1.1 游戏设定

本题考虑一个策略选择问题。玩家在一张沙漠地图内，初始拥有一定的资金并购买一定数量的水和食物，从起点出发，在保证生存的情况下，在规定的时间内到达终点并保留尽可能多的资金。途中玩家的资源消耗受到天气和行动的影响，但玩家可以在矿山补充资金，在村庄补充资源。

主要的游戏要点如下：

- 1 游戏以天为单位，玩家选择停留或行走。若在矿区停留可以选择挖矿，经过村庄可以选择购买资源。
- 2 资源分为水和食物，玩家在沙漠中的水和食物有负重上限，若未到达终点而资源耗尽则游戏失败。
- 3 玩家在原地停留的资源消耗量称为基础消耗量，行走一天消耗量为基础消耗的两倍，挖矿的消耗量为基础消耗的三倍。
- 4 天气分为“晴朗”、“高温”和“沙暴”，不同天气玩家的基础消耗量不同。“沙暴”天气时玩家必须在原地停留。
- 5 玩家在第 0 天在起点以基本价格购买水和食物，之后若要购买资源需要在村庄购买，但价格为基准价格的两倍。在到达终点后资源以基准价格的一半退回。
- 6 玩家在矿山挖矿可以获得基础收益，到达矿山当天不能挖矿。

1.2 问题提出

根据不同的游戏参数设定 (地图、负重上限、初始资金、截止日期、基础收益、资源价格、资源重量、基础消耗量)，在有以下三个问题：

问题一：只有一名玩家，整个游戏时间内所有的天气已知，给出该玩家的最优策略，求解“第一关”和“第二关”。

问题二：只有一名玩家，玩家仅知道当天的天气情况，据此决定当天的行动方案，给出最佳策略并对“第三关”和“第四关”进行讨论。

问题三：有 n 名玩家同时从起点出发，若某天 k 名玩家的路线相同，则他们每人的资源消耗量为基础消耗的 $2k$ 倍；若某天 k 人在同一矿山挖矿，收益是基础收益的 $\frac{1}{k}$ 但消耗量不变；若某天 k 人在同一村庄购买资源，每箱价格为基准价格的 4 倍。其余情况与单人游戏相同。

(1) 若整个游戏世界内天气全部已知，每名玩家的行动方案在第 0 天确定且不更改，给出玩家的策略并对“第五关”讨论。

(2) 若所有玩家仅知道当天天气情况，每个玩家在当天行动结束后知道其余玩家当天的行动方案和剩余的资源数量，随后确定第二天的行动方案，给出玩家应该采取的策略并对“第六关”进行讨论。

二、模型的假设

2.1 对于天气不确定情况，各种情况的出现概率固定

在处理天气不确定的策略时，我们认为应当有对于天气概率的一些预期信息。为了简化我们处理为各种情况按固定概率出现。

2.2 确定天气下的最优方案包含了随机天气下优秀方案的共性

除了确定的天气情况下我们可以得出确定的最优解外，含有博弈和随机天气的情况均不能得到严格最优解。我们认为单人确定性天气下得到的最优方案包含了方案优化中的特征，据此我们可以通过数据分析提取这些优化策略。

2.3 多人游戏时，每个玩家都是理性人

多人游戏中，玩家总希望自己能存活并获得尽可能多的收益。由于玩家在沙漠中行走没有合作机制，因此合作博弈的基本条件不成立 [5]，因此所有玩家都进行不合作博弈，即完全为自己的利益考虑。

2.4 这是一个纯粹的游戏，不涉及生命伦理问题，即我们可以对游戏者在沙漠中“死亡”的可能性进行分数量化

本模型允许给出对游戏者在沙漠中因缺乏食物而失败的方案，并对此进行量化评分。这对于实际人穿越沙漠问题这样的解应当排除，因为生命无价。

三、符号说明

符号	意义
$InitialMoney$	初始资金
$BasicIncome$	挖矿的基础收益
$WeightLimit$	负重上限
p_w	水的基础价格
p_f	食物的基础价格
c_{wi}	第 i 天水的基础消耗量
c_{fi}	第 i 天食物的基础消耗量
w_w	一箱水质量
w_f	一箱食物质量
d	截止日期
p	地图上地区的总个数

四、图动态规划模型

4.1 图模型

为了后续算法表述和程序计算，首先我们需要将附件中给的图形化地图转化为图 $G = (V, E)$ ：每个地区块为一个顶点， V 表示顶点集合， E 表示边集，若 i 地区和 j 地区相邻，则 $(i, j) \in E, (j, i) \in E$ 。

观察发现并不是所有的顶点和边的存在都会对游戏目标产生影响。我们每天如果选择行走的话，总会选择最短的路径走向终点、村庄、矿山三者之一。由此可以对图作较大简化。我们将这一猜想具体表述如下：

定义 1 功能点：终点、村庄、矿山三类点的总称。

定义 2 玩家在第 i 天开始时到 j 点的距离：玩家从第 i 天所在点走到 j 点所需要经过最少天数，记作 $d_i(j)$ 。

定义 3 j 点到 j' 的最短路径：构成从 j 点走到 j' 点所需最少天数的方案中的点集和边集，记作 $P_{j,j'} = (V_{j,j'}, E_{j,j'})$

猜想 1 单人玩家在游戏过程中如果选择行走, 则至少存在一个功能点 j , $d_{i+1}(j) < d_i(j)$ 。

证明: 设玩家选择行走的某天 i 开始时处在节点 j , 设 j' 为玩家之后最先到达的功能节点。(由于成功过关时要到达终点, 因此 j' 必存在)。

记玩家让 $d_{i+1}(j') < d_i(j')$ 的方案 (朝着目标 j' 走), 并且之后每天行走都遵循这个规则为策略 1, 到 j' 用时 t_1 , 其中走路天数 t_{1w} 天, 让 $d_{i+1}(j') \geq d_i(j')$ 的方案 (不朝着目标 j' 走) 为策略 2, 到 j' 用时 t_{2w} , 其中走路天数 t_{2w} 。则有 $t_{2w} > d_i(j') = t_{1w}$

玩家根据天气制定的移动策略 (停留或行走) 对两种路线均适用。则使用策略 1 的玩家到达 j' 后可以等待使用策略 2 的玩家直到其也到达 j' , 此时两种情况具有相同的金钱、日期、位置, 哪一种留下的食物和水更多则为更优方案。

由于策略 2 走路天数多, 两种情况在到达目的 j' 前每天都是同样等待或是选择行走 (路线可以不一样, 不影响消耗), 前 t_1 时间两种情况消耗相同水和食物, 而策略 1 先到目的 j' , 在之后 $t_2 - t_1$ 时间内策略 1 在等待而策略 2 有等待有行走, 消耗更多食物和水。因此策略 1 更优。由此证明了我们猜想每一步行走都应当靠近某个功能点 (也就是即将到达的那个)。

推论 1 由起点和所有功能点, 以及这些点走向各功能点的所有最短路径之并集构成的新图 G' 对于游戏目标来说与原图 G 等效

证明: 由于玩家每次行走总会使其与某个功能点之间距离减少, 可以等效为其在当前点到该功能点之间最短路径上的行走。图的其余部分与玩家所有可以选择的决策 (行走、停留、购买、挖矿、出售) 无关, 因此对于游戏目标来说, 原地图 G 与新地图 G' 是等效的。

根据推论 1 我们完成了对完全图 $G = (V, E)$ 的简化, 得到了简化图 $G' = (V', E')$, 其中, V' 为所有走向功能点的最短路上的所有点构成的集合, E' 为 V' 中所有相邻的点连成的的边。将第一关的地图简化后的示意图如下 1 所示。

基于对地图的图论表示和分析, 我们将整个游戏的逻辑进行程序实现, 可以通过输入策略来实现游戏过程的自动模拟。源码见附录中主程序 `desert_game.py` 和策略输入程序 `strategy.py`。据此我们大致得出了前两关最高分的大致范围。

我们会在后续第一问的动态规划求解时对完整的图和简化图分别进行计算, 并通过比较结果进一步验证我们的猜想。

4.2 动态规划思想来源

对于单人玩家在确定天气下策略问题, 我们可以用一组参数——(天数, 位置, 水量, 食物量, 金钱数) 描述当前游戏的状态。[4]

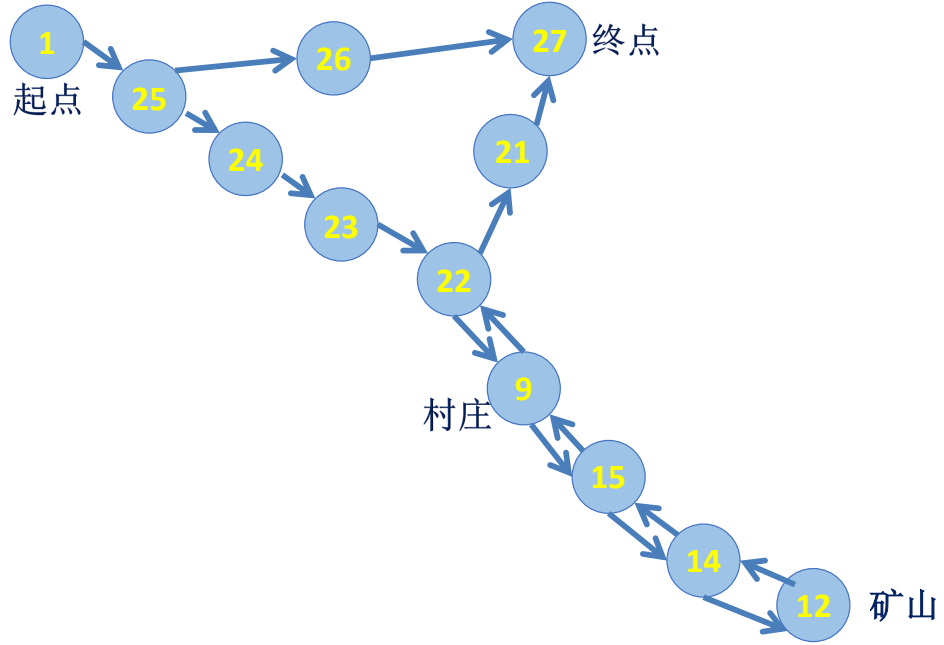


图 1 简化图（第一关）

记录下这一组参数，就好像给游戏存档，下次再读档时后续的游戏决策仅与此存档参数有关，而与如何到达这一参数所描述的状态无关。即决策具有**无后效性**。

因此，如果我们使用搜索的策略模拟玩家的所有决策，到达相同状态的分支就会进行重复搜索。即这个问题含有许多**重叠子问题**。

再进一步分析，对于金钱最大这一目标，该问题存在**最优子结构**。即对于相同的转移策略来说，金钱较多的状态转移到达的状态也是金钱更多的。

完成这些分析，一个在图上的动态规划模型就呼之欲出了。

4.3 动态规划的要素确定

状态变量 $s = (i, j, k, l)$ ，表示时间处于第 i 天结束，玩家的位置在 j 点，拥有 k 箱水和 l 箱食物。

指标函数记为 $dp(s)$ ，表示第 i 天结束时，玩家的位置在 j 点，拥有 k 箱水和 l 箱食物时**可能拥有的最多金钱数**。

决策变量 $x(s)$ ，表示在状态 s 时应当采取的状态转移策略。

则本游戏的最佳结果可表示为

$$\max_{i,k,l} dp(i, FinalPoint, k, l)$$

其中 $FinalPoint$ 表示地图上的终点编号。

4.4 状态空间分析

状态变量 s 含有 4 个整数分量。 i 表示天数, 设游戏天数 d , 则有 $0 \leq i \leq d$ 。

j 表示在地图上的位置, 设地图含有点的数量为 p , 则 $1 \leq j \leq p$ 。

k, l 表示水、食物的数量, 则有 $k \geq 0, l \geq 0$ 且 $kw_w + lw_f \leq WeightLimit$ 。

其中 w_w, w_f 分别表示单箱水和食物的质量, $WeightLimit$ 表示负重上限。

综上, 整个状态空间 $S = \{(i, j, k, l) \in \mathbb{N}^4 | i \leq d, j \leq p, kw_w + lw_f \leq WeightLimit\}$, \mathbb{N} 表示非负整数集合。

状态空间大小 $|S| \approx d \times p \times \frac{1}{2} \frac{WeightLimit}{w_w} \times \frac{WeightLimit}{w_f}$ 。代入一组有代表性的游戏数据可以算得 $|S| \approx 30 \times 64 \times \frac{1}{2} \frac{1200}{3} \frac{1200}{2} \approx 2.3 \times 10^8$ 。其所需的空间复杂度是一台计算机可以接受的。

4.5 状态转移方程

根据状态定义, 我们可以通过游戏规则列写状态转移方程。状态转移有购买和行动两类, 共 5 种情况。购买可以分为在起点购买和在村庄购买, 不改变天数; 行动可以分为停留、行走和挖矿, 会增加天数。下面将分别讨论 5 种情况下 $dp(i, j, k, l)$ 的转移方程。

4.5.1 起点买资源

初始时有状态

$$dp(0, 1, 0, 0) = InitialMoney$$

$InitialMoney$ 为初始资金。时间参数 $i = 0$ 表示第 0 天结束, 即第 1 天开始。位置参数 $j = 1$ 为起点。

在起点可以以基础价格购买食物和水。因此 $i = 0, j = 1$ 的所有状态 $s = (i, j, k, l)$ 都可以由初始状态到达 (如果钱够的话), $dp(s)$ 就是初始时的钱减去购买水和食物花费的钱。

$$dp(0, 1, k, l) = InitialMoney - kp_w - lp_f, \quad \forall (0, 1, k, l) \in S \quad (1)$$

其中 p_w, p_f 分别表示水和食物的基础价格。

4.5.2 村庄买资源

如果 j 位置是一个村庄, 那么从 (i, j, k, l) 状态出发购买水和食物 (钱足够的情况下) 就可以到达状态集合 S 中的另一状态 $(i, j, k + k', l + l')$ 。

相应可以对 $dp(s)$ 进行更新。

$$dp(i, j, k + k', l + l') = \max\{dp(i, j, k + k', l + l'), dp(i, j, k, l) - 2k'p_w - 2l'p_f\} \quad (2)$$

4.5.3 停留

停留使状态中天数 i 增加 1，位置 j 不变，食物和水按照游戏规则进行消耗。

$$dp(i+1, j, k - c_{wi}, l - c_{fi}) = \max\{dp(i+1, j, k - c_{wi}, l - c_{fi}), dp(i, j, k, l)\}, \quad (3)$$

要求 $k \geq c_{wi}, l \geq c_{fi}$ 。其中 c_{wi}, c_{fi} 分别为水和食物在第 i 天情况下基础消耗量。

4.5.4 挖矿

挖矿时水和食物的消耗为基础消耗量的 3 倍，增加收入，天数增加，位置不变。

$$dp(i+1, j, k - 3c_{wi}, l - 3c_{fi}) = \max\{dp(i+1, j, k - 3c_{wi}, l - 3c_{fi}), dp(i, j, k, l) + BaseIncome\} \quad (4)$$

要求 $k \geq 3c_{wi}, l \geq 3c_{fi}$ ，其中 $BaseIncome$ 为每关游戏参数中的基础收益。

4.5.5 移动

移动的时候，水和食物的消耗量都是基础消耗量的 2 倍，天数增加，位置变化。

$$dp(i+1, j', k - 2c_{wi}, l - 2c_{fi}) = \max\{dp(i+1, j', k - 2c_{wi}, l - 2c_{fi}), dp(i, j, k, l)\} \quad (5)$$

要求 $k \geq 2c_{wi}, l \geq 2c_{fi}, (j, j') \in E$ 。 E 为图的边集。

4.6 动态规划算法实现

Algorithm 1 图动态规划

Input: S : 合法状态 (i, j, k, l) 的集合

E : 图的边集

BasicIncome: 基础收益

InitialMoney: 初始资金

d : 截止天数

p : 图中地区点总数

Output: $\max_{i,k,l} dp(i, FinalPoint, k, l)$: 这一关的最高过关分数

Initialize: $dp(i, j, k, l) = -INF$ for $\forall (i, j, k, l) \in S$

- 1: **for** $k, l: (0, 1, k, l) \in S$ **do**/*1 起点购买物资*/
- 2: $dp(0, 1, k, l) = InitialMoney - kp_w - lp_f$
- 3: **end for**
- 4: **for** $i = 0 \rightarrow d$ **do**
- 5: **for** $j = 1 \rightarrow p$ **do**
- 6: **if** j 是村庄 **then**/*2 每天行动前，在村庄买物资更新*/


```

7:         for  $k, l : (i, j, k, l) \in S$  do
8:             for  $k', l' : k' \geq 0, l' \geq 0, (i, j, k + k', l + l') \in S$  do
                 $dp(i, j, k + k', l + l') = \max\{dp(i, j, k + k', l + l'), dp(i, j, k, l) - k'p_w - l'p_f\}$ 
9:             end for
10:        end for
11:    end if
12:    for  $k, l : (i, j, k, l) \in S$  do /*3 停留情况的更新 */
13:         $dp(i + 1, j, k - c_{wi}, l - c_{fi}) = \max\{dp(i + 1, j, k - c_{wi}, l - c_{fi}), dp(i, j, k, l)\}$ 
14:    end for
15:    if  $j$  是矿山 then /*4 挖矿情况的更新 */
16:        for  $k, l : (i, j, k, l) \in S$  do
17:             $dp(i + 1, j, k - 3c_{wi}, l - 3c_{fi}) = \max\{dp(i + 1, j, k - 3c_{wi}, l - 3c_{fi}),$ 
                 $dp(i, j, k, l) + BasicIncome\}$ 
18:        end for
19:    end if
20:    for  $j' : (j, j') \in E$  do /*5 移动情况的更新 */
21:         $dp(i + 1, j', k - 2c_{wi}, l - 2c_{fi}) = \max\{dp(i + 1, j', k - 2c_{wi}, l - 2c_{fi}),$ 
                 $dp(i, j, k, l)\}$ 
22:    end for
23: end for
24: end for

```

注意

1. 伪代码 Input 中的集合 S 代表了在实际程序中输入的一系列游戏参数, $(i, j, k, l) \in S$ 的判断在实际程序中就是一些简单的逻辑判断。终点 *FinalPoint* 这个参数也包括在状态集合 S 中了。为了让算法表述更简洁, 伪代码中省去了处理这些逻辑的细节。
2. 伪代码 Input 中的边集 E 实际用邻接表实现。
3. $-INF$ 表示负无穷, 实际实现时可以使用一个很小的数。
4. 读者可能有疑问, 为何算法循环中每天需要同时更新停留、挖矿、移动。这可以从搜索的角度理解, 动态规划的更新过程并不是模拟一个玩家在进行游戏, 而可以理解为模拟玩家所有可能到达状态及其之间转移方式可能性的过程。
5. 伪代码中在村庄买物资更新一部分循环达到 6 层。在第 i 天第 j 个节点的情况下要对 k, l, k', l' 四个变量循环遍历。这里复杂度似乎为 $O(n^4)$ 。这里这样写是为了算法表达的简单明了, 具体在程序实现时会做一定的优化, 使复杂度变为 $O(n^2)$ 。优化思想简要介绍如下:

如果从 (i, j, k, l) 状态通过购买食物到达 $(i, j, k + k', l + l')$ 时剩下的钱少于

$dp(i, j, k', l')$ 。那么就可以确定由 $(i, j, k + k', l + l')$ 转移到 $(i, j, k + k' + k'', l + l' + l'')$ 一定比从 (i, j, k, l) 转移到 $(i, j, k + k' + k'', l + l' + l'')$ 更好，即买东西过程也存在**最优子结构**，基于此，只要对 (k, l) 所在的二维状态空间做一次遍历即可，复杂度 $O(n^2)$ 。

4.7 复杂度分析

设共 d 天， p 个点，由重量限制得可以携带的食物或水 n 箱。(两者同一量级的)

算法最外层对 i 的循环 $O(d)$ ，对 j 的循环 $O(p)$ 。对于每个 (i, j) 。若在村庄，由之前分析购买物资更新复杂度 $O(n^2)$ 。计算停留、挖矿均为对 (k, l) 简单遍历，复杂度 $O(n^2)$ 。由于关卡地图均为稀疏图，各点度数较小，因此移动的复杂度也为 $O(n^2)$

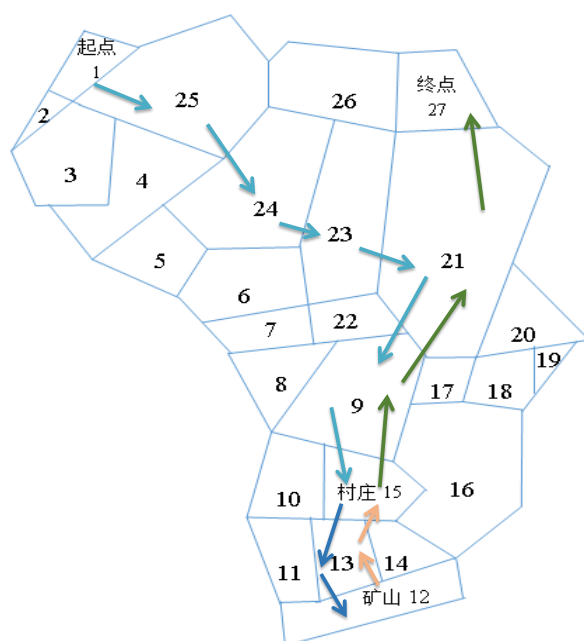
因此根据复杂度分析的运算规则，整个算法复杂度 $O(d \times p \times (n^2 + n^2 + n^2)) = O(pd n^2)$ 。与开始分析存储动态规划状态表的空间复杂度在同一量级。实际使用 C++ 编程实现对问题一求解程序 (源码见附录) 运行时间在秒级。

五、问题一解答

5.1 第一关

5.1.1 最佳方案

经过动态规划算法求解，我们得到了第一关最佳结果行动方案。



日期	所在区域	剩余资金数	剩余水量	剩余食物量
0	1	5780	178	333
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	5780	80	235
9	13	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	13	11150	10	38
21	15	11150	10	38
22	9	10470	26	26
23	21	10470	10	14
24	27	10470	0	0

图 2 最优路线图（第一关）

第一关玩家在截止日前到达终点时的资金最高为 10470 元，用时 24 天。走法如图所示：初始买水 178，食物 333，1－8 天去村庄，买水 163 到负重上限，9－10 天去矿区，11－17 天挖矿，18 天停留，19 天挖矿，20－21 去村庄，买 36 水，16 食物，后 22－24 天去终点，食物和水恰好清零。

具体的每天行动与资源情况如 Result.xlsx 表格中所示。

5.1.2 策略变化分析

除了得到最终 30 天内到达终点的最高分外，我们还考虑了截止天数为 $1 \leq i \leq 30$ 天玩家最高得分以及相应策略的变化情况，最高得分随截止日期变化如下：

截止日期	1	3	22	23	24
最高得分	0	9410	9500	10430	10470

表 1 最高得分随截止日期增加的上升情况

从表中可以看出，最快可以三天到达终点，最后最多剩余 9410 元，可以从地图中看出这是直接前往终点的策略。这在 22 天之前到达的方案之中都是最优的，即前去挖矿且 22 天内到达终点的方案不如直接前往终点。到第 24 天上升为 10470 元，且之后到 30 天不再更新，即我们得到的结果，这是最优的行走策略。

为了查看挖矿情况下截止日期与最高得分的关系，我们将 3 天前往终点的路线删去重新计算，得到的最高变化趋势如下：

截止天数	1	9	20	21	22	23	24
最高资金	0	8330	8395	8950	9500	10430	10470

表 2 最高得分随截止日期增加的上升情况（除去 3 天到达的情况策略）

可以看出，删去 3 天速达路线后，仍然是 9 天直接到达终点不挖矿的结果维持到了截止日期少于 20 天。之后的变化与有完整路线的地图结果一致。

具体查看程序输出的路径信息，我们可以发现在截止天数较少时最优策略倾向于节约资源开支，直接到达终点。20 天即之后到达的情况才会考虑挖矿并在村庄购买资源补充。具体是补充一次还是两次需要结合天气情况确定，没有明显的规律。

5.2 第二关

经过动态规划算法求解，我们得到了第二关最佳结果行动方案。

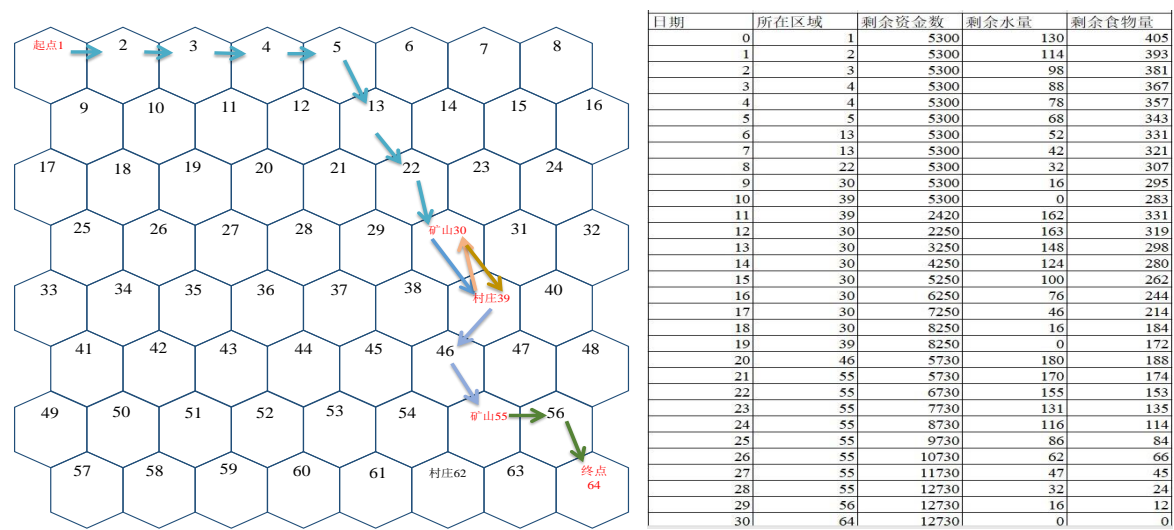


图 3 最优路线图（第二关）

第二关玩家在截止日前到达终点时的资金最高为 12730 元，用时 30 天，走法如图所示：

初始买水 130 箱，食物 405，1-10 天经过 30 号矿山前往 39 号村庄且不在矿山处停留挖矿，到达村庄时水刚好耗尽，第 11 天遇上沙暴在村庄停留，买水 172，食物 58。第 12 天买水 17，不买食物，离开前往 30 号矿山。第 13 天到第 18 天在矿山挖矿。第 19 天前往 39 号村庄。第 20 天购买 196 箱水，28 箱食物，前往 55 号矿山，并于第 21 天抵达。第 22 天到第 28 天挖矿。第 29-30 天从矿山出发到达终点，食物和水恰好清零。

5.2.1 策略变化分析

同样地，我们可以作截止时间和最高得分如下：

截止天数	14	15	16	19	20	21	22	23	24	29	30
最高得分	7390	8790	9485	9555	10285	10760	11180	11590	12020	12355	12730

表 3 截止天数的最高资金

由于本关卡有两个村庄和矿山，所以决策会有更多的情况，所以最高得分表随截止时间增加的更新也比较频繁，可以看出第 30 天到达终点是最优解。

5.3 一般情况下的最优策略

通过对结果分析，我们得到以下基本的策略设计原则。

(1) **到达终点处时玩家的资源恰好耗尽。**为了使到达终点时资金尽可能多，玩家没有理由买多于生存需求的水和食物。玩家在起点和村庄购买资源的价格都高于最后在终点返还的资金，因此在天气情况已知的情况下，玩家知道维持自己生存所需要的最少资源，所以有能力在到达终点时控制资源恰好耗尽。此策略**仅适用于全局天气已知情况**。

(2) **起点处在保证生存的情况下多买食物。**根据条件，村庄处的资源价格是起点处的资源价格的两倍，所以需要尽量多在起点买资源，而在村庄仅保证生存需要即可。根据计算，挖矿的收益大于沙暴天气挖矿的消耗，因此收益较大，玩家选择前去挖矿。为了保证挖矿的时间，需要装上尽量多的食物和水。根据食物和水的价格表，定义衡量资源携带效率为基准价格与每箱质量之比。

$$\eta = \frac{prize}{mass} \quad (6)$$

由于背包有负重上限，因此玩家倾向于在起点购买价格较贵且质量较小的资源，即资源携带效率 η 较大的资源。水和食物的比较见下表。

资源	每箱质量 (kg)	(元/箱)	携带效率 (元/千克)
水	3	5	1.67
食物	2	10	5.00

表 4 资源携带效率比较

由结果知，食物的携带效率远大于水，因此在确保生存的情况下尽量多买食物是更好的策略。

(3) **玩家每次移动总会与目的地的最短距离缩小。**根据分析可知，玩家在沙漠中的功能地只有三个：村庄、终点和矿山。当玩家选定目的地后，最佳移动策略为以最短路向目的地前进。动态规划的计算中我们分别以原始图和简化后的图作为输入运行算法，得到了完全相同的输出，再次**验证**了在开始做简化图模型时猜想的正确性。

对于更大规模的地图，或者没有条件使用计算机运行动态规划算法时，可以利用上述策略设计单人确定天气情况下穿越沙漠游戏策略。

通过比对简化地图与动态规划得到的最优解，我们发现最佳策略就是按照最短路前进的，这也进一步验证了该策略的正确性。

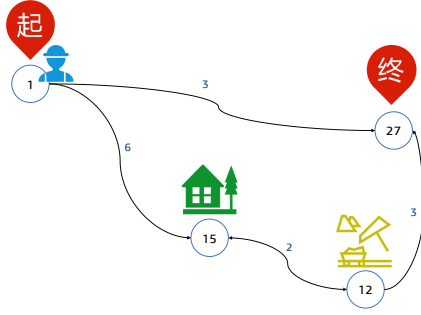


图 4 简化后的地图（第一关）

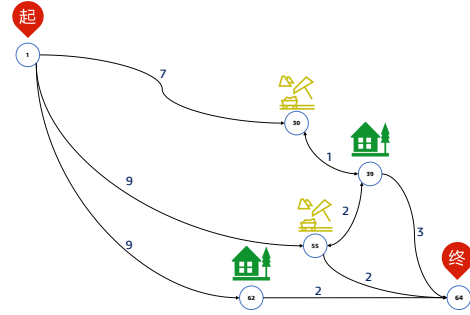


图 5 简化后的地图（第二关）

六、统计评价模型与第二问方案

第三、四关由于存在随机性，不能够给出一个确定性的最优策略，因此我们在设计方案之后需要建立方法对方案的评价进行评价。

在设计中，我们将方案分解为购买策略、路线策略、行走-停留决策三部分。在设计完成后利用随机模拟得到的期望收益和失败概率来对方案进行评价。

6.1 基于完全统计的路线、行动选择、购买策略设计

第三关天气是随机的，但因为图比较小并且只有 10 天，所有天气一共 $2^{10} = 1024$ 种情况，我们可以利用动态规划算法给出每一种情况的最优解，用统计方法观察规律。（输出 1024 种最优方案的程序见附录 `desert_game_dp`(第三关-简化图-路径回溯-天气枚举).cpp。

6.1.1 初步结论：确定路线

通过观察最优解策略，我们发现即使在全部晴天状态下仍然没有去挖矿。可以猜想这一关不能够挖矿。

事实上，由于第三关没有村庄，又因为高温天气消耗很大，且挖矿收益很小，即使在晴朗的天气挖矿，收益仅为 $200 - 165 = 35$ ，挖矿 5 天的收益 175 还不足以弥补由于绕路导致至少 220 的资源消耗。若有高温天挖矿损失只会更大。

因此玩家的决定只有一个——直接走向终点。从图中很容易得到从起点到终点的最短路，需要玩家行走三天。在路径完全确定的情况下，玩家需要决策的就是在起点购买资源的策略和移动策略。

6.1.2 统计结论 1：确定行走-停留方式

有了确定的路线，我们再对 1024 种天气情况下的到达终点天数进行统计
为了理解这一数据的意义，我们对移动策略进行简单的分类讨论：

到达天数	3	4	5	6
最优解数量	576	320	112	16
占比 (%)	56.25	31.25	10.94	1.56

表 5 1024 种最优走法的到达天数统计

- 1 晴朗天气移动：晴朗天的资源基础消耗最少，所以若遇到晴朗天气，玩家一定选择移动。
- 2 高温天气停留：高温天气移动会消耗 18 箱水和 18 箱食物，折合金钱为 270 元，比两次晴朗天气的消耗量还多，因此高温天若不连续出现，在高温天停留可能比较明智。
- 3 高温天气移动：高温天移动的消耗很大，但若遇到连续的高温天而不移动，很容易会将携带的资源耗尽，又由于本关的路径很短，若在高温天移动而尽早到达终点，可以减少进一步由于高温消耗资源，即减少失败的概率。

可见最优解中过半选择 3 天不停留直接走向终点，而仅有约 10% 会等待两天或三天。根据上述分析可以得知，选择高温天行走是相对比较保守的方案，只要买足物资就不存在失败风险，而选择等待则以风险换取省物资的收益。

考虑到动态规划搜索到最优解的特点，搜索到的最优解一定是一个非常冒险的方案，这里面存在很严重的**幸存者偏差**。即使如此仍然有大概率选择 3 天到达，那么我们可以认为在决策过程中选择高温等待的概率应当很小。

6.1.3 统计结论 2：确定初始购买方案

设有 n 个最优解，第 i 个最优解的购买策略为买 w_i 箱水和 f_i 箱食物，则样本均值和标准差分别为

$$\bar{w} = \sum_{i=1}^n w_i, \bar{f} = \sum_{i=1}^n f_i$$

$$\sigma_w = \frac{1}{n-1} \sum_{i=1}^n (\bar{w} - w_i)^2, \sigma_f = \frac{1}{n-1} \sum_{i=1}^n (\bar{f} - f_i)^2$$

我们将 $[\bar{w} - \sigma_w, \bar{w} + \sigma_w]$ 作为水的优秀购买区间，将 $[\bar{f} - \sigma_f, \bar{f} + \sigma_f]$ 作为食物的优秀购买区间。

经过动态规划我们可以计算得到上述统计量如下表

优秀解的分布图如下

(1) 购买方式分析：

两种资源的均值大致相等，这说明大部分的优秀策略都选择了购买尽量等量的水和食物。由第一问我们得到在起点处尽量多买食物，但在本关没有村庄，如何生存和省钱

资源	样本均值	样本标准差	最高两峰代表点位
水	38.4	3.1	[42, 53]
食物	37.2	1.1	[41, 53]

表 6 资源购买的样本统计量计算

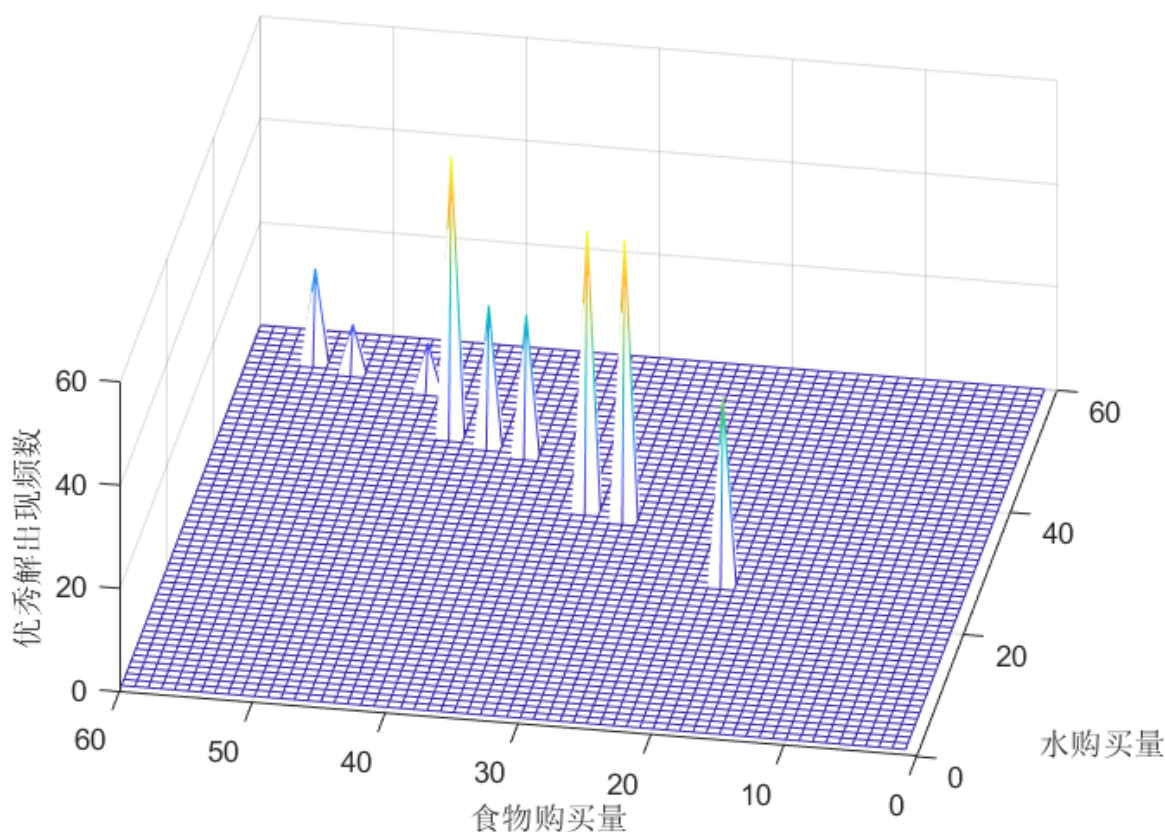


图 6 1024 个最优解起点购买方案统计

是主要目标，而晴朗和高温天气下水和食物的消耗箱数大致相等，因此为了更加省钱，玩家不能对于某一种资源有购买偏好而导致最终一种资源余量很大但另一种资源被较快耗尽的情况。

(2) 购买量分析：考虑幸存者偏差，我们不应当取平均值或众数作为购买方案，而应该选择统计图中水、食物量最大的两个峰区作为第一天为晴天、高温两种情况下的购买策略。

因此我们得到的优秀购买方案参考值为晴天 42 水、44 食物，高温 54 水，54 食物。（对之前的两峰参考值进行调整使得食物和水配比平衡，恰好可以满足之后两天都是高温的行走需求）。

6.1.4 方案搜索空间的确定

综合以上分析，我们的方案设计框架为：

- 1 路线：按最短路线 3 天到终点。
- 2 行走：三种情况也可以统一理解为晴天行走，高温以 p 的概率移动 ($0 \leq p \leq 1$)。倾向于 $p \approx 1$ 的情况。
- 3 购买：晴天 42 水、44 食物，高温 54 水，54 食物。在这一基础上进行调整。

有了以上方案设计指导，搜索空间已经很小，可以写程序进行搜索，接下来就是设计评价指标来筛选最优方案了。

6.1.5 方案的评价模型

为了综合评价方案的好坏，我们认为要从最终收益的期望和存活概率两方面综合考量。

对于确定的策略参数，进行 n 轮随机模拟，其中成功通关 n_1 次，第 i 次最终剩余钱 m_i 。则有

$$LoseRate = \frac{n - n_1}{n} \quad (7)$$

$$AvgIncome = \frac{\sum_{i=1}^{n_1} m_i}{n_1} \quad (8)$$

$LoseRate$ 为失败率， $AvgIncome$ 表示在成功过关情况下平均得分。

再定义完全平均意义下的收入 $Income$ ，认为失败的收益为 0，没有额外惩罚。

$$Income = \frac{\sum_{i=1}^{n_1} m_i}{n} \quad (9)$$

6.2 第三关解答：随机模拟结果呈现与讨论

我们利用模拟程序（见附录和支撑材料 `simulation.py`），按照晴天-高温各 50% 的概率设定对方案中不同的行走概率 p 进行随机模拟，结果如下：

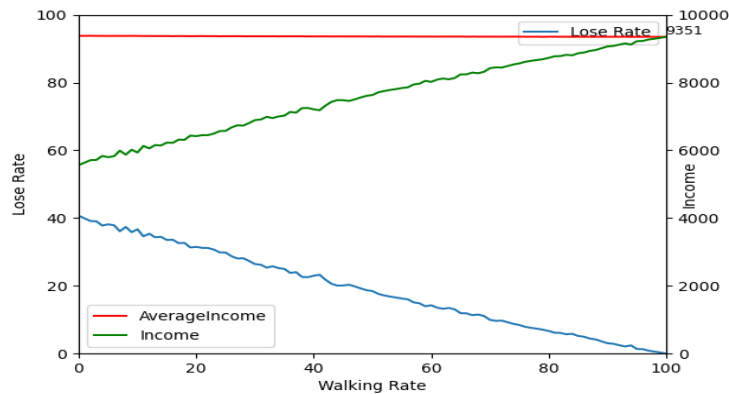


图 7 失败率、成功时平均收入、总平均收入随高温天移动概率的关系图

可以看到当高温天确定移动时模拟得到的期望收益为 9351（可以严格计算得期望是 9350），并且没有失败风险。可见该随机模拟计算期望的仿真度很高。同时可以明显发现随着高温天移动概率的下降，失败概率（蓝线）急剧上升而成功时平均收益（红线）变化不明显。

将两者平均得到总的预期收益（绿线）与高温天行动概率的关系。可以明显看出概率应当为 1 最好。

接下来简单讨论初始时带上当天和 2 个高温天行走所需水和食物的购买方案的优越性。在这一购买方案基础上进行微调，可以发现最终受益基本是一个单峰函数，这里仅给出多买 5 箱 (9314) 和少买 5 箱的结果 (≈ 7000)

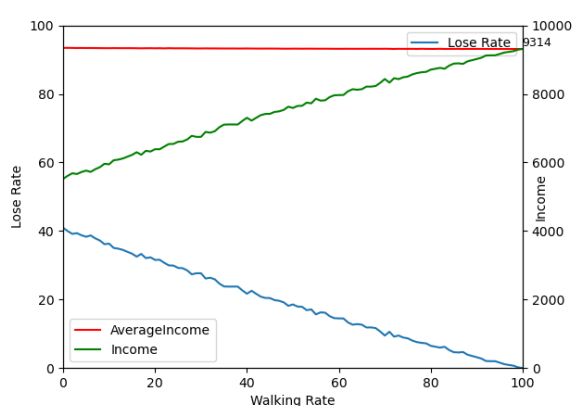


图 8 多买 5 箱食物和水

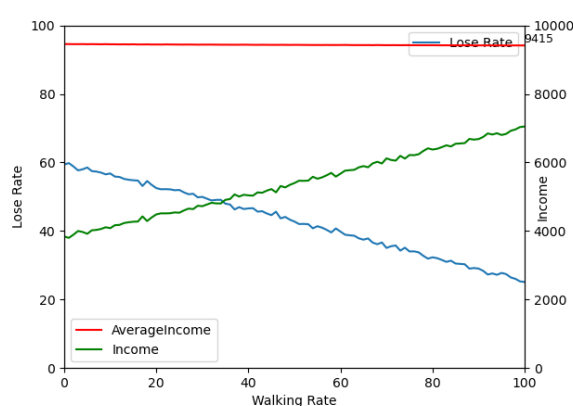


图 9 少买 5 箱食物和水

根据得分来看，不管天气如何，三天直接走向终点得分最高，是最合理的策略。相应购买保证成功的最少食物。一点也不能少（上面少买 5 箱的结果可以看到失败率急剧上升，最终期望结果下降 2000 左右）。

下面我们将在这些分析得到的结果基础上处理第四关的情况。

6.3 第四关图的简化和分析

通过第三关的分析，在这一组食物、水消耗的数据下，当玩家不知道天气并且没有沙暴天，距离目的地较近时，购买充足的水和食物一直不停留地前进是最好的策略。

从本关的地图来看，除了功能点外，还有一个非常重要的点 13，我们把它称为决策点。首先，决策点距离起点只有四天的路程，而且是玩家在沙漠中的必经点，这和第三关的情况非常相似。

同时由于决策点距离村庄和矿山都只有一天行程，在决策点的状态直接影响玩家的决策，因此是玩家做决策的关键点。

此外，最后走向终点的路同样和第三关类似，玩家最终要走向终点，而这一段路程的起点（最后一次决定目的地）的点必须为村庄或矿山，而这两个点距离终点的距离都

为三天路程，这和第三关的情况完全相同。

而一个比较大的问题在于沙暴天气的处理。

综上所述，我们可以对问题进行简化：

- 离开起点的路——从起点到决策点 13 的路采用第三关的结论。
- 走向终点的路——从村庄或矿山到终点的路采用第三关的结论。

6.4 天气等效模型

虽然我们的移动策略得到简化，但我们难以在购买和路线方面给出较有说服力的方案。我们知道三种天气都有可能发生，天气的未知性是使得玩家失败率升高的根本原因，若我们能够将天气进行等效，以一定比例地把晴朗和沙暴天全部转化为高温天，那么购买策略就能大大简化，玩家按照等效高温天进行购买资源，可以有效地提高自己的存活率。因此我们充分考虑在三种天气下玩家的所有行为，并对他们的收益和消耗进行计算，从资源消耗和挖矿收益两方面来进行天气等效。

对于给定的天气 $i, i = 1, 2, 3$ （分别为晴朗、高温和沙暴），设对应的水的基础消耗箱数为 c_{wi} ，基础价格为 p_w ，食物的基础消耗箱数为 c_{fi} ，基础价格为 p_f 。根据第一关的数据，我们发现玩家在起点购买食物数与在村庄购买量之比近似为 $8:2$ ，购买水量之比约为 $7:3$ 。但由于天气未知，根据之前分析，玩家更关注存活率，这和第一关购买水量的初衷一致，为方便计算，我们认为在本关玩家在起点购买的食物箱数和水箱数与在村庄购买数之比均为 $7:3$ 。设 $B_{ij}, j = 1, 2, 3$ 分别为在 i 天气下停留、行走和挖矿的净收益，且综合净收益为 B_i ，则得到如下方程组

$$\begin{cases} B_{i1} = -(0.7 + 0.3 \times 2)(c_{fi}p_i + c_{wi}p_i) = -1.3(c_{fi}p_i + c_{wi}p_i) \\ B_{i2} = 2B_{i1} \\ B_{i3} = 1000 + 3B_{i1} \\ B_i = \sum_{j=1}^3 B_{ij} \end{cases}$$

设晴朗和沙暴天以 $x:y$ 的比例等效为高温天，则可以列得净收益等效方程

$$xB_1 + yB_3 = (x + y)B_2$$

经过计算可得， $\frac{x}{y} = 0.214 \approx \frac{5}{1}$ 。因此，从净收益的角度，晴朗与沙暴天以 $5:1$ 等效为高温天，即每 5 天晴朗和 1 天沙暴与 6 天高温等效。

6.5 第四关解答：基于动态规划策略的分析与调整

游戏设定中已知沙暴天气较少，我们取 10%，我们将一定比例的晴天和沙暴天进行等效，增加了高温天且减少了一定晴天。

确定了天气我们就可以使用第一问中的动态规划模型进行求解。在天气等效后我们主要关注其购买策略，可以分析得出以下策略：

- 1 在起点的购买策略和第一问的结果一致，是优先考虑购买食物，食物的量在 300 箱左右，水的量在 150 箱左右。
- 2 第一次到决策点时，若水的量小于 100 箱就要去村庄补给，否则去挖矿。
- 3 第一次到村庄补给时只尽量多地购买水而不购买食物，一般补充水至 235 箱左右，此时食物剩余约 235 箱左右。

我们发现这些策略和第一问的结论极其相似，这也符合我们的预期，因为我们将晴朗天和沙暴天等效为高温天，则天气的确定性降低，购买策略就相对固定了。这些策略和第三关有差别，可以认为是由于村庄可以补给以及可以挖矿赚钱这两个主要因素造成的。

等效天气的方法仍旧无法确定路线策略，但经过分析购买策略，可以推理出一些路径策略。由于在村庄补给食物与水至 235 箱左右，大约可以支持 9 天的挖矿并留足直接返回终点不停留的资源；或是挖矿 12 天左右后返回村庄补给并直接回到终点。

我们认为这两种策略都有可能是比较优秀的路线，因此我们随机生成了 100 种 30 天的天气进行动态规划求解，筛选出前 20 的优秀解，统计它们在第一次补给后的策略，发现有 93.41% 的解选择了路径一，即第一次补给后挖矿，保留够直接返回终点的资源直接返回终点。

路径一的压倒性的优势是我们没有预料到的，但我们仔细分析，可以得到原因。首先，挖矿并留足能直接返回终点的策略和第三关的最优策略一致，玩家可以避免由于更多挖矿而资源耗尽，或是减少由于天气恶劣造成的损失。其次，由于从矿山到村庄需要走两天，而村庄和矿山距离终点都是四天行程，若多挖矿会多走至少两天的路程，更有可能遇到高温或沙暴，使资源大大减少，甚至增加失败的风险。因此，路径一是更合理的选择。

考虑沙暴概率 10% 的情况，对于几个关键点的简单的计算表明该策略失败概率控制在 2.5% 以内。

对于该策略进行与第三关类似的随机模拟 (问题一中的 python 模拟程序) 验证了以上结论，并计算得期望收益达到 10500。

6.6 总结：一般情况下玩家的最优策略

综合以上分析，我们可以归纳出当天气未知时玩家的宏观优秀策略。

6.6.1 地图中没有村庄和矿山的情况

当玩家距离下一个目的地较近时，玩家不应该等待，应该尽快到达目的地。因为更长时间的停留会大大增加资源的消耗，甚至会有失败的风险。即使等待最终能到达终点，最后的资金与快速通过方案也相差无几，通过避开高温天行走而节省的金钱并不多，

在购买资源时按照最坏的情况购买，保证能够到达目的地。我们进行游戏的首要目的是通过，若不通过净收益为 -10000 元，而相对保守地购买可以在保证 100% 通过的前提下，最终的收益也较高，是在天气不确定时最好的策略。

6.6.2 地图中有村庄和矿山的情况

当地图中有村庄和矿山时，村庄的合理购买可以保证存活，矿山挖矿可以增加最后的收益，因此最佳策略和上一种情况有所不同。

玩家的购买策略是在起点优先考虑携带效率高的资源，即价格贵且需求大的资源，另一种资源可以以最坏情况买到下一次补给之前。在村庄大量补充携带效率低的资源，尽量不补充携带效率高的资源。如此购买的原因在第一关和第四关都有详细的阐述。

玩家的行动策略为首先在接近村庄和矿山的点进行决策，若剩余资源较少则先去补充资源，若剩余资源较多就直接去挖矿。当挖矿到资源剩余按最坏打算仅够前往下一个功能点时，前往下一个功能点。一般地，玩家应该在向下一个功能点的同时，与终点的距离逐渐缩小，否则将显著提高失败的风险。

七、静态博弈模型：问题三策略分析与设计

由于在这一问中两关都具有不止一个玩家，并且玩家在游戏状态更新会受到对方情况的影响，因此每个玩家为了实现自己的游戏目标，必须考虑对方的行动策略。因此用博弈的模型来考虑。

第五关双方同时进行一次决策，为单阶段静态博弈。第六关双方进行多次决策，每次决策都是同时进行的，为多阶段静态博弈。

7.1 两人单阶段博弈：第五关

7.1.1 博弈设定与求解目标

有两位玩家 A、B。我们假设两个玩家都是具有充分思维能力的理性人，能够充分考虑自己的策略和对方的选择。设计的目标是使 A 能够在 B 按照符合 B 利益前提下行动时让自己获得最大的期望收益。

因为 A、B 地位是平等的，即两个玩家的策略集是完全对称的，因此我们为 A 设定的策略对 B 也是同样适用的。

7.1.2 思路分析

可行的方案大致有两类：第一类为纯策略 [1]，两个玩家使用同一种固定策略，走同一条路线。第二类为混合策略，玩家以从一个策略组 $S = \{S_1, S_2 \cdots S_k\}$ 中以 $(P_1, P_2 \cdots P_k)$ 的概率选择一种策略，给出的混合策略应当满足纳什均衡 [3]，即玩家选择任何一种策略最后的平均收益都是一样的。如果混合策略不满足纳什均衡，就存在好的策略和坏的策略，玩家出于利己的角度，会倾向性地采用混合策略中的好策略，那么之前找到的混合策略就不能稳定存在了。

7.1.3 博弈收益表的计算

为了建立博弈收益表以对双方的策略进行分析，在接下来的部分中，我们把玩家从起点到终点消耗的水和食物的价格的相反数做为收益，并把失败失败的收益记为-10000 (初始资金的相反数)。

下面计算一下会被纳入策略组中的几种策略的消耗量：

S_1 ：第一天从 1 到 5，第二天停留，第三到第四天从 5 走到终点。消耗的食物和水价值 465。

S_2 ：第一天到第三天沿 $1 \rightarrow 5 \rightarrow 6 \rightarrow 13$ 一直行走，第三天可到达终点，消耗的食物和水的价格是 490。

S_3 ：沿着 $1 \rightarrow 4 \rightarrow 7 \rightarrow 12 \rightarrow 13$ 的路线行走，并且只在晴天行走，消耗的食物和水的价格是 575。

S_4 ：按照 S_2 的行走策略行走，但是假设对方采用 S_2 的行走策略，更改食物和水的购买方案。消耗的食物和水的价格是 980。

S_5 ：按照 S_2 的行走策略行走，但是假设对方采用 S_1 的行走策略，更改食物和水的购买方案。消耗的食物和水的价格是 600。

S_6 ：按照 S_1 的行走策略行走，但是假设对方也采用该行走策略，更改食物和水的购买方案。消耗的食物和水的价格是 795。

S_7 ：按照 S_3 的行走策略行走，但是假设对方也这么走，更改食物和水的购买方案，消耗的价格是 1015。

7.1.4 最优纯策略

首先寻找纯策略。最优的纯策略已经很明显了：由于失败的代价非常大，保证存活方案中收益最大的方案就是最佳纯策略，即两玩家都采用 S_6 。这样玩家到达终点时平均收益为-795，且保证存活。

7.1.5 最优混合策略

接下来寻找混合策略。首先讨论只采用 S_4 和 S_6 的情况，两玩家采取策略的情况共有 4 种，对应的收益如表7所示：

表 7 混合策略分析

收益 玩家 A 策略	玩家 B 策略	
	S_4	S_6
S_4	(-980,-980)	(-800,-685)
S_6	(-685,-800)	(-795,-795)

设两位玩家采用 S_4, S_6 的概率分别为 P_1, P_2 ，首先寻找纳什平衡点，让玩家采用 S_4 和 S_6 的收益相等：

$$-980P_1 - 800P_2 = -685P_1 - 795P_2$$

$$P_1 + P_2 = 1$$

求解，可得 $P_1 = -0.02$, $P_2 = 1.02$ 。这说明这种策略组合不存在纳什均衡点，两位玩家会严重倾向于使用 S_6 策略。

同理，对组合 S_1 和 S_3 的概率进行求解，然后算得玩家到达终点时平均收益为-5260.2。由于存在不小的失败率，即使是采用在单人模式下非常好的策略，在多人游戏中的平均收益也非常低。

我们又对 S_6, S_7 等两个策略的组合，以及部分 3 个策略的组合进行了计算，发现在失败率等于 0 的情况下，不存在纳什均衡点。计算的结果告诉我们：玩家如果想保证存活，就应当采用采用纯策略而非混合策略。

7.1.6 求解结果

通过比较，采用纯策略 S_6 是两类方案中的最优方案。

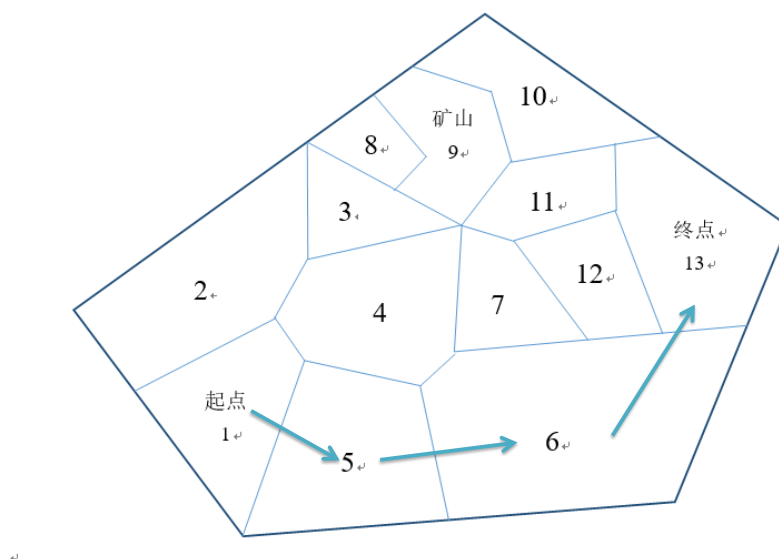


图 10 最优纯策略路线图

7.1.7 允许交流情况下的最优策略

如果两玩家之间可以交流的话，那么就一人采用 S_1 ，一人采用 S_3 ，具体谁采用最优路线可以通过掷骰子决定，确保对两位玩家都是公平的。这样平均消耗的钱为： $\frac{465}{2} + \frac{575}{2} = 520$ 。比不交流的情况更优。

7.2 三人多阶段博弈：第六关

第五关是单阶段博弈，采用混合博弈的模型解决。第六关是多阶段博弈，玩家可以牺牲短期利益，换取长期利益。即使不能直接交流，也可以通过观察分析对方之前的行为，达成某种程度上的合作，实现共赢。

多阶段博弈的关键是寻找到纳什均衡点，因为纳什均衡点才能稳定存在，同时意味着每个玩家都有比较好的收益。

根据 [2] 中的命题 10，我们可以取消各个阶段的联系，寻找每个阶段的纳什均衡点 $\sigma^{1*}, \sigma^{2*} \dots \sigma^{n*}$ ，那么存在一个多阶段博弈的子博弈完全均衡，该完全均衡的路径与 $\sigma^{1*}, \sigma^{2*} \dots \sigma^{n*}$ 相同。这个命题告诉我们，在这种游戏场景之下，如果对每一天进行孤立，在不考虑过去与未来的情况下寻找单日的纳什平衡点，那么把找到的 30 天的方案连起来看的话，就能构成 30 天方案中的一种平衡方案。利用该命题，可以把每一天看做一个阶段，分情况讨论，从而简化运算。

7.2.1 两玩家高温天抵达同一地点博弈

假设两个玩家在同一高温天抵达了同一个地点，下面为他们寻找纳什均衡点。基础消耗的价格是 135，如果两玩家都行走，每位玩家到达下一个点的消耗是 540；如果一人停留，一个人行走，那么停留的人到达下一个点的消耗是 405，行走的人是 270；如果两人都停留，那么消耗的期望必定大于 405，因为最优情况下也只能做到做到一个人是 405。表8展示出了以上分析结果，两人都停留的期望收益按照以下方式近似计算：停留之后，每人选择走或继续停留的概率是 1/2，这样计算出来的两人停留的消耗均为 585。

表 8 行走策略分析

收益 \ 玩家 B 策略		停留	行走
玩家 A 策略	停留	(-585,-585)	(-405,-270)
	行走	(-270,-405)	(-540,-540)

可以发现，当玩家 A 行走的时候，玩家 B 的最佳策略是停留；当玩家 B 行走的时候，玩家 A 的最佳策略是行走，因此 (玩家 A 行走，玩家 B 停留) 是一个纳什均衡点，由对称性可知，(玩家 A 停留，玩家 B 行走) 也是一个纳什均衡点。由于玩家知道对方的食物和水的剩余情况，因此从某种程度上可以进行合作，对双方都有利。在两者物资不等的情况下，物资多的玩家选择停留，物资少的玩家选择行走。当两者物资相等的时候，采用第五关中的混合博弈模型求解纳什均衡点，假设两位玩家选择停留和行走的概率分别为 P_1, P_2 ，有下列方程：

$$-585P_1 - 405P_2 = -270P_1 - 540P_2$$

$$P_1 + P_2 = 1$$

解得 $P_1 = 0.3, P_2 = 0.7$ 。于是得到在双方物资相同时走到同一地点的策略：每位玩家以 0.7 的概率选择行走，以 0.3 的概率选择停留。

7.2.2 两玩家高温天抵达同一座矿山博弈

如果两个玩家同时在矿山区，并且食物和水充足，那么容易得到表 9 所示的结果，分析可知，无论玩家 B 采用什么策略，玩家 A 的最优策略都是挖矿，对玩家 B 也是如此。于是 (玩家 A 挖矿，玩家 B 挖矿) 构成纳什均衡点，这意味着在该情况下两玩家都应该挖矿。

表 9 挖矿策略分析

收益 \ 玩家 B 策略		
	挖矿	停留
玩家 A 策略	挖矿	(95,95)
	停留	(-135,595)

7.2.3 两玩家高温天抵达村庄博弈

从前几关的分析可以发现，在未知天气的情况下，玩家会购买更多的水和食物提升存活率，因此在抵达村庄的时候往往还留有一些水和食物，因此这里对两个玩家同时抵达村庄，且资源还足以支撑他们存活几天的情况进行分析。假设两人都准备购买基础价格为 650 的物资，然后前往矿山挖矿。选取从两人抵达村庄到两人均抵达矿山这段时间进行分析。容易得到，两人均购买然后行走的收益为-3140；一人购买行走，一人停留的话，先离开村庄的人的收益为 $-650 \times 2 - 270 + 595 = -975$ ，后离开的人的收益为 $-135 - 650 \times 2 - 270 = -1705$ 。难点在于两人都停留的收益，这里近似认为如果都停留，那么之后两人都以 50% 的概率选择购买离开或者继续停留。设每人的收益为 $-135 - x$ ，那么有：

$$\frac{1}{4}[(-135 - x) + (-975) + (-1705) + (-3140)] = -x$$

求得 $x = 1985$ ，做出表10。

表 10 村庄购买策略分析

收益 \ 玩家 B 策略		
	停留	购买
玩家 A 策略	停留	(-2120,-2120)
	购买	(-975,-1705)

不存在纯策略纳什均衡，使用混合策略模型，求得停留与购买的概率分别为 55.6% 和 44.4%。

7.2.4 第六关小结

本文把 30 天的游戏当做多阶段博弈，运用多阶段博弈的性质对多阶段博弈进行拆分，并采用纯策略博弈纳什均衡、多阶段博弈纳什均衡等博弈论的方法给出了许多局部情况下玩家的行动策略，比如当两玩家在高温天同时到达一个地点（非村庄，非矿山）时，物资少的玩家继续行走，物资多的玩家停留，如果两玩家物资相当，那么都以 70% 的概率行走，以 30% 的概率停留。当两玩家在高温天同时抵达一座矿山的时候，两位玩家都会选择在第二天挖矿。当两玩家在高温天同时抵达村庄时，以 55.6% 的概率选择第二天在村庄停留，以 44.4% 的概率选择在第二天开始的时候买东西离开村庄。

7.3 第三问一般策略

7.3.1 天气已知且存在多个玩家的单阶段博弈

由于中途失败造成的损失巨大，玩家的首要目的是生存，因此要在起点处购买足够的食物和水，然后猜测其他玩家可能会采取的行走路线，这些路线是天气已知的单玩家模式下的较优行走策略，运用博弈论的方法寻找纳什均衡点，均衡点给出的策略就是玩家的行动策略。

7.3.2 天气未知且存在多个玩家的多阶段博弈

考虑到中途失败的损失巨大，玩家的首要目的依然是生存，因此要在起点处购买足够的食物和水，按照天气未知单玩家场景下较优的行走路线行进，如果遇到与其他玩家在同一天到达了同一个地点的情况，就运用博弈论的方法寻找均衡点，以此与其他玩家进行某种程度上的合作实现共赢，比如物资少的玩家继续行走，物资多的玩家停留；一起到达村庄时，以较大的概率选择第二天在村庄停留，以较小的概率选择选择在第二天开始的时候补充物资离开村庄。

八、模型总结

8.1 灵敏度分析

对于第一问我们采用的是确定性算法，不存在灵敏度问题。因为我们的动态规划模型只要在算力充足的情况下得到任何地图的全局最优解，并能够回溯出每一步购买和移动策略。

第二问我们通过统计分析和随机模拟，得到了特定地图特定条件的优秀解。在数据选择上有一定人工参与主观性，这些策略和第一问有相同之处，但又由于天气未知有很强的随机性。灵敏度相对另外两问模型较高。

同时有一些宏观的策略是适用于大多数情况的，但对于极端的条件我们的策略可能就不是最优的策略，只能在统计意义上保持优秀。

第三问的两人全局方案博弈和局部策略博弈都基于严格的博弈论基础，其结果对于各种情况也是普适的，不会因为输入的微小扰动而失效。

8.2 优点分析

8.2.1 整体模型充分地结合数学推导、算法实现、仿真模拟，对于问题给出了全面而深入的分析

对于一个游戏策略问题，我们综合使用了图论、运筹学、统计学、博弈论等知识和工具进行分析，并熟练地编程实现。给出了对游戏“最优策略”的各种不同表述，如带有递推搜索思想的算法、统计的最优、决策的一般规则、博弈的平衡等，多角度地帮助玩家理解这个游戏并确定最优策略。

8.2.2 动态规划的算法实现进行充分的时间、空间复杂度优化

第一问的动态规划不仅给出全局最优解，我们还在分析算法时间、空间复杂度的基础上进行了彻底优化，达到单问题的秒级求解，使得后续的大量样本统计成为可能。

8.2.3 统计加随机模拟对于第三关随机天气下的策略设计给出了详细的论证

第三关的最终结果呈现后可能很容易猜到。但给出充分的思想来源和令人信服的论证并不容易。我们从统计结果抽取策略，并利用随机模拟较为完整地论证了该方案确实由于其他合理方案。

8.2.4 第三问的二人博弈模型给出一系列有效的局部决策

由于多人游戏的复杂性使得一些经验性结论（即局部优化方案）比确定性的计算机算法更有意义。因此这一部分我们用数学推导给出的可靠局部决策结论能够更有效地帮助实际游戏。

8.3 缺点分析

8.3.1 利用确定天气情况下结果求解后两问时没有定量分析幸存者偏差

尽管我们可以通过动态规划回溯出优秀解，但在天气未知的情况下这些解有非常大的运气成分。最高收益和存活率二者是相互制衡的，而我们在分析一些优秀解的时候虽然也重点考虑了存活率，但无法显式地给出描述幸存者偏差的量并加以讨论。

8.3.2 给出的策略需要一定的算力支撑

我们给出的有些策略难以通过直觉或人工计算快速得到验证，都需要一定的程序和算力进行实现，这些结果可能不易于被人从直观上理解。

8.3.3 对于多人玩家的情况没有给出完全最优解

虽然我们给出了局部最优策略，但对于三人的多阶段静态博弈没有给出完全最优解。由于博弈的过程难以由程序体现，最后的博弈过程没有进行模拟和全局计算。

8.4 总结与感想

本文我们对穿越沙漠游戏的策略进行了由浅入深的分析，对于越来越复杂的问题也有确定性策略求解转化为带有随机性，局部性优化，并利用各种评价方法进行讨论分析。

其实该问题本身的递进过程就是对一定的现实背景进行建模，我们在这二次建模的过程中也充分感受到数学各个分支的基本思想之间存在着广泛的联系，可能就会在某个有趣的问题上汇合。只有创造性的思维加上扎实的理论、计算基本功才能较好地处理一个实际问题。

参考文献

- [1] Strategy (game theory). [https://en.wikipedia.org/wiki/Strategy_\(game_theory\)](https://en.wikipedia.org/wiki/Strategy_(game_theory)).
- [2] 多阶段纳什均衡. http://faculty.haas.berkeley.edu/stadelis/Game%20Theory/econ160_week5.pdf.
- [3] John Forbes Nash Jr. Equilibrium points in n-person games. *National Academy of Sciences*, 1950.
- [4] 《运筹学》教材编写组. 运筹学 (修订版). 清华大学出版社, 1990.
- [5] 胡奇英. 随机运筹学: *Stochastic operations research*. 清华大学出版社, 2012.

附录

A 支撑材料列表

Result.xlsx : 前两关的结果呈现

代码.rar : 所有 C++、python 的程序源代码，具体有三类：

1. desert_game.py 和 strategy.py 为游戏逻辑的直接模拟程序和相应玩家策略输入程序。
名字略有不同的是不同的参数
2. simulation.py 为第三关对游戏策略进行随机模拟并作图的程序。
3. desert_game_dp(第 X 关-XX 图-XXXX) 为动态规划算法的 C++ 源码，具体含义见文件名，分散地用于模型各处。

图.rar：含有文中统计图、示意图的文件夹

B 动态规划 C++ 代码

desert_game_dp(第一关-完整图-路径回溯).cpp

```
#include <iostream>
#include <cstdio>
#include <vector>
#include <stack>
#include <map>

using namespace std;

const int INF=0x3f3f3f3f;

const int num_of_points=27;  ///!!修改1,: 点数
const int deadline=30;      //期限
const int initial_money=10000; //初始金钱
const int base_income=1000; //基本收益
const int weight_limit=1200; //负重上限
const int water_weight=3,water_price=5,food_weight=2,food_price=10; //价格与重量
bool mine_list[num_of_points+1]={},village_list[num_of_points+1]={};

int water_consumption_list[3]={5,8,10},food_consumption_list[3]={7,6,10}; //消耗列表
int weather_list[deadline]={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};
//0晴朗, 1高温, 2沙暴

struct state
{
    char i,j;
    short k,l;
    state(short ii=0,short jj=0,short kk=0,short ll=0):i(ii),j(jj),k(kk),l(ll){}
    bool operator!=(const state &st2){
        return i!=st2.i || j!=st2.j || k!=st2.k || l!=st2.l;
    }
};

vector<short> world_map[num_of_points+1];
```

```

short wmap[num_of_points+1][num_of_points+1]
    {{},{2,25},{1,3},{2,25,4},{3,25,24,5},{4,24,6},{5,24,23,7},{6,8,22},{7,9,22},{8,22,21,17,16,15,10},{9,15,11}};
//!!修改2: 地图
map<short,short> mapping;

short *dp[deadline+1][num_of_points+1][weight_limit/water_weight+1];
    //dp[i][j][k][l]表示第i天结束时, j点有k水、l食物情况下的最多金钱
state *dp_state[deadline+1][num_of_points+1][weight_limit/water_weight+1];

// 更新规则:采用“我为人人”型dp
/* 天数i从0循环到29, 编号j从1-10, k,l完全循环
    更新 停留、挖矿、走动
    若 (k,l减去行动代价后均为正, 则与原有值取较大值)

    注意: 每天开始时对村庄点状态进行更新
        (需要四重循环, 对于每个k,l, 向上k',l'进行计算更新, 计算量600多亿, ×30天),
    但特别注意, 向上更新时, 一旦更新不动就可以break. 比如 对于某个k,l,
        循环k',l'向上更新, 一旦更新不动, 直接对k'l'双循环break
    共5种更新: 起点买, 村庄买, 停留, 挖矿, 走动
*/

void inline init_space()
{
    for(int i=0;i<=deadline;++i)
        for(int j=0;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
            {
                dp[i][j][k]=new short [(weight_limit-k*3)/2 +1]; //加了之后堆不会损坏
                dp_state[i][j][k]=new state[(weight_limit-k*3)/2 +1];
            }
}

void inline init()
{
    for(int i=1;i<=num_of_points;++i){
        mapping[i]=i;
        for(int j=0;j<=num_of_points&&wmap[i][j];++j)
            world_map[i].push_back(wmap[i][j]);
    }
    //!!修改3: 映射,村庄、矿表
    mine_list[12]=1;village_list[15]=1;

    for(int i=0;i<=deadline;++i)
        for(int j=0;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
                for (int l =0;k*3+l*2<=weight_limit;++l){
                    dp[i][j][k][l]=-INF;

```

```

    }
    dp[0][1][0][0]=initial_money;

    for(int k=0;k<=weight_limit/water_weight;++k){
        int money_left=dp[0][1][0][0]-k*water_price,weight_left= weight_limit-k*water_weight;
        if (money_left<0)
            break;
        for (int l =0;l<=weight_limit/food_weight;++l){
            int money_left2=money_left-l*food_price;
            if (money_left2<0 || l*food_weight>weight_left)
                break;
            if(money_left2>dp[0][1][k][l])
            {
                dp[0][1][k][l]=money_left2;
                dp_state[0][1][k][l]=state(0,1,0,0);
            }
            //cout<<dp[0][1][k][l]<<'\\n';
        }
    }
}

inline void buy(int i,int j,int k,int l)
{
    int max_food=weight_limit/food_weight;
    for(int m=k;m<=weight_limit/water_weight;++m){
        for(int n=1;m*3+n*2<=weight_limit && n<max_food;++n)
        {
            if(m==k &&n==1)
                continue;
            int money_left= dp[i][j][k][l] - (m-k)*water_price*2 - (n-1)*food_price*2;
            if(money_left<=dp[i][j][m][n] || money_left<0 ) // 1\
                三角形区域的右上部分三角无需再搜索了，其他还是要更新的
            {
                max_food=n;
                break;
            }
            dp[i][j][m][n]=money_left;
            dp_state[i][j][m][n]=state(i,j,k,l);
        }
    }
}

int main()
{
    freopen("1_complete_result.txt","w",stdout); //!!修改4: 输出文件名

```



```

init_space();
init();
int best_ever=-INF;

for(int i=0;i<deadline;++i){
    int ii=i+1;
    //cerr<<"i="<<i<<'\\n';
    for(int j=1;j<=num_of_points;++j){
        if(village_list[j]){ //每天开始更新前一天前，村庄先购物更新
            for(int k=0;k<=weight_limit/water_weight;++k)
                for (int l =0; k*3+l*2<=weight_limit;++l)
                    if(dp[i][j][k][l]>=0){
                        buy(i,j,k,l);
                        //cout<<"更新村庄! "<<endl;
                    }
        }

        //更新停留
        int
            water_consumption=water_consumption_list[weather_list[i]],food_consumption=food_consumption_list[weather_list[i]];
        for(int k=water_consumption;k<=weight_limit/water_weight;++k){
            for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                int kk=k-water_consumption,ll=l-food_consumption;

                if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][j][kk][ll])
                {
                    dp[ii][j][kk][ll]=dp[i][j][k][l];
                    dp_state[ii][j][kk][ll]=state(i,j,k,l);

                    //cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\\n';
                }
            }
        }

        //更新挖矿
        if (mine_list[j]){
            water_consumption=water_consumption_list[weather_list[i]]*3,food_consumption=food_consumption_list[weather_list[i]];
            for(int k=water_consumption;k<=weight_limit/water_weight;++k){
                for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                    int kk=k-water_consumption,ll=l-food_consumption;

                    if(dp[i][j][k][l]>=0 && dp[i][j][k][l]+base_income>dp[ii][j][kk][ll])
                    {
                        dp[ii][j][kk][ll]=dp[i][j][k][l]+base_income;
                        dp_state[ii][j][kk][ll]=state(i,j,k,l);
                    }
                }
            }
        }
    }
}

```

```

    }
}

/*for(int k=0;k<=400;++k)
    for (int l =0;k*3+l*2<=1200;++l)
        if(dp[i][j][k][l]>9900)
            cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\n';//*/

//非沙暴，更新走动
if (weather_list[i]<2){
    water_consumption=water_consumption_list[weather_list[i]]*2,food_consumption=food_consumption_list[weather_list[i]]*2;
    for(vector<short>::iterator
        iter=world_map[j].begin();iter!=world_map[j].end();++iter){
        for(int k=water_consumption;k<=weight_limit/water_weight;++k){
            for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                int kk=k-water_consumption,ll=l-food_consumption;

                if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][*iter][kk][ll])
                {
                    dp[ii][*iter][kk][ll]=dp[i][j][k][l];
                    dp_state[ii][*iter][kk][ll]=state(i,j,k,l);
                }
            }
        }
    }
}

}

}

int target=num_of_points;
int res=-INF;
state last_state;
for(int i=0;i<=deadline;++i){
    for(int k=0;k<=weight_limit/water_weight;++k){
        for (int l =0;k*3+l*2<=weight_limit;++l){
            if(dp[i][target][k][l]>res)
            {
                res=dp[i][target][k][l];
                last_state=state(i,target,k,l);
            }
        }
    }
}

if(res>best_ever){
    cerr<<res<<" "<<best_ever<<endl;
    best_ever=res;
}

```

```

    stack<state> sta;
    while(last_state!=state(0,0,0,0)){
        sta.push(last_state);
        last_state=dp_state[last_state.i][last_state.j][last_state.k][last_state.l];
    }

    cout <<i+1<< "days,score:" <<res<< "\nsolution:\n"; //此i是外面的大i

    while(!sta.empty())
    {
        last_state=sta.top();sta.pop();
        cout<<int(last_state.i)<<" "<<mapping[last_state.j]<<" "<<int(last_state.k)<<"
            "<<int(last_state.l)<<"
            "<<dp[last_state.i][last_state.j][last_state.k][last_state.l]<<endl;
    }
    cout<<"\n\n";
}

return 0;
}

```

desert_game_dp(第一关-简化图-路径回溯).cpp

```

#include <iostream>
#include <cstdio>
#include <vector>
#include <stack>
#include <map>

using namespace std;

const int INF=0x3f3f3f3f;

const int num_of_points=11;
const int deadline=30;           //期限
const int initial_money=10000; //初始金钱
const int base_income=1000; //基本收益
const int weight_limit=1200; //负重上限
const int water_weight=3,water_price=5,food_weight=2,food_price=10; //价格与重量
bool mine_list[num_of_points+1]={},village_list[num_of_points+1]={};

int water_consumption_list[3]={5,8,10},food_consumption_list[3]={7,6,10}; //消耗列表
int weather_list[deadline]={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};
    //0晴朗, 1高温, 2沙暴

struct state

```

```

{
    char i,j;
    short k,l;
    state(short ii=0,short jj=0,short kk=0,short ll=0):i(ii),j(jj),k(kk),l(ll){}
    bool operator!=(const state &st2){
        return i!=st2.i || j!=st2.j || k!=st2.k || l!=st2.l;
    }
};

vector<short> world_map[num_of_points+1];
short wmap[num_of_points+1][num_of_points+1] {{},{2},{3},{4},{5},{6},{7,10},{6,8},{7,9},{8},{11}};

map<int,int> mapping;

int dp[deadline+1][num_of_points+1][weight_limit/water_weight+1][weight_limit/food_weight+1];
    //dp[i][j][k][l]表示第i天结束时，j点有k水、l食物情况下的最多金钱
state
    dp_state[deadline+1][num_of_points+1][weight_limit/water_weight+1][weight_limit/food_weight+1];

// 更新规则:采用“我为人人”型dp
/* 天数i从0循环到29， 编号j从1-10，k,l完全循环
    更新 停留、挖矿、走动
    若(k,l减去行动代价后均为正，则与原有值取较大值)

    注意：每天开始时对村庄点状态进行更新
        (需要四重循环，对于每个k,l,向上k',l'进行计算更新，计算量600多亿，×30天)，
    但特别注意，向上更新时，一旦更新不动就可以break。比如 对于某个k,l，
        循环k',l'向上更新，一旦更新不动，直接对k'l'双循环break
    共5种更新： 起点买，村庄买，停留，挖矿，走动
*/

void inline init()
{
    for(int i=1;i<=num_of_points;++i)
        for(int j=0;j<=num_of_points&&wmap[i][j];++j)
            world_map[i].push_back(wmap[i][j]);
    mapping[1]=1;mapping[2]=25; mapping[3]=24; mapping[4]=23; mapping[5]=22; mapping[6]=9;
        mapping[7]=15; \
    mapping[8]=14; mapping[9]=12;mapping[10]=25; mapping[11]=27;

    mine_list[9]=1; village_list[7]=1;

    for(int i=0;i<=deadline;++i)
        for(int j=0;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
                for (int l =0;k*3+l*2<=weight_limit;++l){

```

```

        dp[i][j][k][l] = -INF;
    }
    dp[0][1][0][0] = initial_money;

    for(int k=0; k<=weight_limit/water_weight; ++k){
        int money_left = dp[0][1][0][0] - k*water_price;
        weight_left = weight_limit - k*water_weight;
        if (money_left < 0)
            break;
        for (int l = 0; l <= weight_limit/food_weight; ++l){
            int money_left2 = money_left - l*food_price;
            if (money_left2 < 0 || l*food_weight > weight_left)
                break;
            if (money_left2 > dp[0][1][k][l])
            {
                dp[0][1][k][l] = money_left2;
                dp_state[0][1][k][l] = state(0, 1, 0, 0);
            }
            //cout << dp[0][1][k][l] << '\n';
        }
    }
}

inline void buy(int i, int j, int k, int l)
{
    int max_food = weight_limit/food_weight;
    for(int m=k; m<=weight_limit/water_weight; ++m){
        for(int n=l; m*3+n*2<=weight_limit && n<max_food; ++n)
        {
            if(m==k && n==l)
                continue;
            int money_left = dp[i][j][k][l] - (m-k)*water_price*2 - (n-l)*food_price*2;
            if(money_left <= dp[i][j][m][n] || money_left < 0) // 三角形区域的右上部分三角无需再搜索了，其他还是要更新的
            {
                max_food = n;
                break;
            }
            dp[i][j][m][n] = money_left;
            dp_state[i][j][m][n] = state(i, j, k, l);
        }
    }
}

int main()
{

```

```

freopen("1_simple_result.txt", "w", stdout);
init();

//for(int i=0;i<=num_of_points;++i) cout<<village_list[i]<<mine_list[i]<<" ";
int best_ever=-INF;

for(int i=0;i<deadline;++i){
    int ii=i+1;
    //cerr<<"i="<<i<<'\n';
    for(int j=1;j<=num_of_points;++j){
        if(village_list[j]){ //每天开始更新下一天前，村庄先购物更新
            for(int k=0;k<=weight_limit/water_weight;++k)
                for (int l =0; k*3+l*2<=weight_limit;++l)
                    if(dp[i][j][k][l]>=0){
                        buy(i,j,k,l);
                        //cout<<"更新村庄! "<<endl;
                    }
        }

        //更新停留
        int
            water_consumption=water_consumption_list[weather_list[i]], food_consumption=food_consumption_list[weather_list[i]];
        for(int k=water_consumption;k<=weight_limit/water_weight;++k){
            for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                int kk=k-water_consumption, ll=l-food_consumption;

                if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][j][kk][ll])
                {
                    dp[ii][j][kk][ll]=dp[i][j][k][l];
                    dp_state[ii][j][kk][ll]=state(i,j,k,l);

                    //cout<<i<< ' '<<j<< " "<<k<< " "<<l<< " "<<dp[i][j][k][l]<<'\n';
                }
            }
        }

        //更新挖矿
        if (mine_list[j]){
            water_consumption=water_consumption_list[weather_list[i]]*3, food_consumption=food_consumption_list[weather_list[i]];
            for(int k=water_consumption;k<=weight_limit/water_weight;++k){
                for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                    int kk=k-water_consumption, ll=l-food_consumption;

                    if(dp[i][j][k][l]>=0 && dp[i][j][k][l]+base_income>dp[ii][j][kk][ll])
                    {
                        dp[ii][j][kk][ll]=dp[i][j][k][l]+base_income;
                        dp_state[ii][j][kk][ll]=state(i,j,k,l);
                    }
                }
            }
        }
    }
}

```

```

    }
    }
}

/*for(int k=0;k<=400;++k)
    for (int l =0;k*3+l*2<=1200;++l)
        if(dp[i][j][k][l]>9900)
            cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\\n';//*/

//非沙暴，更新走动
if (weather_list[i]<2){
    water_consumption=water_consumption_list[weather_list[i]]*2,food_consumption=food_consumption_list[weather_list[i]]*2;
    for(vector<short>::iterator
        iter=world_map[j].begin();iter!=world_map[j].end();++iter){
        for(int k=water_consumption;k<=weight_limit/water_weight;++k){
            for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                int kk=k-water_consumption,ll=l-food_consumption;

                if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][*iter][kk][ll])
                {
                    dp[ii][*iter][kk][ll]=dp[i][j][k][l];
                    dp_state[ii][*iter][kk][ll]=state(i,j,k,l);
                }
            }
        }
    }
}

}

}

}

int target=num_of_points;
int res=-INF;
state last_state;
for(int i=0;i<=deadline;++i){
    for(int k=0;k<=weight_limit/water_weight;++k){
        for (int l =0;k*3+l*2<=weight_limit;++l){
            if(dp[i][target][k][l]>res)
            {
                res=dp[i][target][k][l];
                last_state=state(i,target,k,l);
            }
        }
    }
}

}

if(res>best_ever){

```

```

        cerr<<res<<" "<<best_ever<<endl;
        best_ever=res;
        stack<state> sta;
        while(last_state!=state(0,0,0,0)){
            sta.push(last_state);
            last_state=dp_state[last_state.i][last_state.j][last_state.k][last_state.l];
        }
        cout <<i+1<< "days,score:" <<res<< "\nsolution:\n";

        while(!sta.empty())
        {
            last_state=sta.top();sta.pop();
            cout<<int(last_state.i)<<" "<<mapping[last_state.j]<<" "<<int(last_state.k)<<"
                "<<int(last_state.l)<<"
                "<<dp[last_state.i][last_state.j][last_state.k][last_state.l]<<endl;
        }
        cout<<"\n\n\n";
    }
}
return 0;
}

```

desert_game_dp(第二关-完整图-路径回溯).cpp

```

#include <iostream>
#include <cstdio>
#include <vector>
#include <stack>
#include <map>

using namespace std;

const int INF=0x3f3f3f3f;

const int num_of_points=64;  ///!!修改1,: 点数
const int deadline=30;      //期限
const int initial_money=10000; //初始金钱
const int base_income=1000; //基本收益
const int weight_limit=1200; //负重上限
const int water_weight=3,water_price=5,food_weight=2,food_price=10; //价格与重量
bool mine_list[num_of_points+1]={},village_list[num_of_points+1]={};

int water_consumption_list[3]={5,8,10},food_consumption_list[3]={7,6,10}; //消耗列表
int weather_list[deadline]={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};
    //0晴朗, 1高温, 2沙暴

struct state

```



```

{
    char i,j;
    short k,l;
    state(short ii=0,short jj=0,short kk=0,short ll=0):i(ii),j(jj),k(kk),l(ll){}
    bool operator!=(const state &st2){
        return i!=st2.i || j!=st2.j || k!=st2.k || l!=st2.l;
    }
};

vector<short> world_map[num_of_points+1];
short
    wmap[num_of_points+1][num_of_points+1]={ {},{1,2,9},{10,9,1,2,3},{3,10,2,11,4},{4,11,3,12,5},{5,13,4,12,6},{
//!!修改2: 地图
map<short,short> mapping;

short *dp[deadline+1][num_of_points+1][weight_limit/water_weight+1];
    //dp[i][j][k][l]表示第i天结束时, j点有k水、l食物情况下的最多金钱
state *dp_state[deadline+1][num_of_points+1][weight_limit/water_weight+1];

// 更新规则:采用“我为人人”型dp
/* 天数i从0循环到29, 编号j从1-10, k,l完全循环
    更新 停留、挖矿、走动
        若(k,l减去行动代价后均为正, 则与原有值取较大值)

    注意: 每天开始时对村庄点状态进行更新
        (需要四重循环, 对于每个k,l, 向上k',l'进行计算更新, 计算量600多亿, ×30天),
    但特别注意, 向上更新时, 一旦更新不动就可以break. 比如 对于某个k,l,
        循环k',l'向上更新, 一旦更新不动, 直接对k'l'双循环break
    共5种更新: 起点买, 村庄买, 停留, 挖矿, 走动
*/

void inline init_space()
{
    for(int i=0;i<=deadline;++i)
        for(int j=0;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
            {
                dp[i][j][k]=new short [(weight_limit-k*3)/2 +1]; //加了之后堆不会损坏
                dp_state[i][j][k]=new state[(weight_limit-k*3)/2 +1];
            }
}

void inline init()
{
    for(int i=1;i<=num_of_points;++i){
        mapping[i]=i;
        for(int j=0;j<=num_of_points&&wmap[i][j];++j)

```

```

        world_map[i].push_back(wmap[i][j]);
    }
    ///修改3: 映射,村庄、矿表
    mine_list[30]=1;mine_list[55]=1; village_list[39]=1;village_list[62]=1;

    for(int i=0;i<=deadline;++i)
        for(int j=0;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
                for (int l =0;k*3+l*2<=weight_limit;++l){
                    dp[i][j][k][l]=-INF;
                }
    dp[0][1][0][0]=initial_money;

    for(int k=0;k<=weight_limit/water_weight;++k){
        int money_left=dp[0][1][0][0]-k*water_price,weight_left= weight_limit-k*water_weight;
        if (money_left<0)
            break;
        for (int l =0;l<=weight_limit/food_weight;++l){
            int money_left2=money_left-l*food_price;
            if (money_left2<0 || l*food_weight>weight_left)
                break;
            if(money_left2>dp[0][1][k][l])
            {
                dp[0][1][k][l]=money_left2;
                dp_state[0][1][k][l]=state(0,1,0,0);
            }
            //cout<<dp[0][1][k][l]<<'\n';
        }
    }
}

inline void buy(int i,int j,int k,int l)
{
    int max_food=weight_limit/food_weight;
    for(int m=k;m<=weight_limit/water_weight;++m){
        for(int n=1;m*3+n*2<=weight_limit && n<max_food;++n)
        {
            if(m==k &&n==1)
                continue;
            int money_left= dp[i][j][k][l] - (m-k)*water_price*2 - (n-1)*food_price*2;
            if(money_left<=dp[i][j][m][n] || money_left<0 ) // 1\
                三角形区域的右上部分三角无需再搜索了, 其他还是要更新的
            {
                max_food=n;
                break;
            }
        }
    }
}

```

```

    }
    dp[i][j][m][n]=money_left;
    dp_state[i][j][m][n]=state(i,j,k,l);
}
}

int main()
{
    freopen("2_complete_result.txt","w",stdout); ///修改4: 输出文件名
    init_space();
    init();
    int best_ever=-INF;

    for(int i=0;i<30;++i){
        int ii=i+1;
        //cerr<<"i="<<i<<'\\n';
        for(int j=1;j<=num_of_points;++j){
            if(village_list[j]){ //每天开始更新前一天前, 村庄先购物更新
                for(int k=0;k<=weight_limit/water_weight;++k)
                    for (int l =0; k*3+l*2<=weight_limit;++l)
                        if(dp[i][j][k][l]>=0){
                            buy(i,j,k,l);
                            //cout<<"更新村庄! "<<endl;
                        }
            }

            //更新停留
            int
                water_consumption=water_consumption_list[weather_list[i]],food_consumption=food_consumption_list[weather_list[i]];
            for(int k=water_consumption;k<=weight_limit/water_weight;++k){
                for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                    int kk=k-water_consumption,ll=l-food_consumption;

                    if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][j][kk][ll])
                    {
                        dp[ii][j][kk][ll]=dp[i][j][k][l];
                        dp_state[ii][j][kk][ll]=state(i,j,k,l);

                        //cout<<i<<' ' <<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\\n';
                    }
                }
            }

            //更新挖矿
            if (mine_list[j]){
                water_consumption=water_consumption_list[weather_list[i]]*3,food_consumption=food_consumption_list[weather_list[i]];
            }
        }
    }
}

```

```

        for(int k=water_consumption;k<=weight_limit/water_weight;++k){
            for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                int kk=k-water_consumption,ll=l-food_consumption;

                if(dp[i][j][k][l]>=0 && dp[i][j][k][l]+base_income>dp[ii][j][kk][ll])
                {
                    dp[ii][j][kk][ll]=dp[i][j][k][l]+base_income;
                    dp_state[ii][j][kk][ll]=state(i,j,k,l);
                }
            }
        }
    }

    /*for(int k=0;k<=400;++k)
        for (int l =0;k*3+l*2<=1200;++l)
            if(dp[i][j][k][l]>9900)
                cout<<i<< ' '<<j<< " "<<k<< " "<<l<< " "<<dp[i][j][k][l]<<'\n';//*/

    //非沙暴，更新走动
    if (weather_list[i]<2){
        water_consumption=water_consumption_list[weather_list[i]]*2,food_consumption=food_consumption_list[weather_list[i]]*2;
        for(vector<short>::iterator
            iter=world_map[j].begin();iter!=world_map[j].end();++iter){
            for(int k=water_consumption;k<=weight_limit/water_weight;++k){
                for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                    int kk=k-water_consumption,ll=l-food_consumption;

                    if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][*iter][kk][ll])
                    {
                        dp[ii][*iter][kk][ll]=dp[i][j][k][l];
                        dp_state[ii][*iter][kk][ll]=state(i,j,k,l);
                    }
                }
            }
        }
    }
}

int target=num_of_points;
int res=-INF;
state last_state;
for(int i=0;i<=deadline;++i){
    for(int k=0;k<=weight_limit/water_weight;++k){
        for (int l =0;k*3+l*2<=weight_limit;++l){
            if(dp[i][target][k][l]>res)

```

```

        {
            res=dp[i][target][k][l];
            last_state=state(i,target,k,l);
        }
    }
}

if(res>best_ever){
    cerr<<res<<" "<<best_ever<<endl;
    best_ever=res;
    stack<state> sta;
    while(last_state!=state(0,0,0,0)){
        sta.push(last_state);
        last_state=dp_state[last_state.i][last_state.j][last_state.k][last_state.l];
    }

    cout <<i+1<< "days,score:" <<res<< "\nsolution:\n";

    while(!sta.empty())
    {
        last_state=sta.top();sta.pop();
        cout<<int(last_state.i)<<" "<<mapping[last_state.j]<<" "<<int(last_state.k)<<"
            "<<int(last_state.l)<<"
            "<<dp[last_state.i][last_state.j][last_state.k][last_state.l]<<endl;
    }
    cout<<"\n\n\n";
}

return 0;
}

```

desert_game_dp(第二关-简化图-路径回溯).cpp

```

#include <iostream>
#include <cstdio>
#include <vector>
#include <stack>
#include <map>

using namespace std;

const int INF=0x3f3f3f3f;

const int num_of_points=18;  ///!!修改1,: 点数
const int deadline=30;      //期限
const int initial_money=10000; //初始金钱

```

```

const int base_income=1000; //基本收益
const int weight_limit=1200; //负重上限
const int water_weight=3,water_price=5,food_weight=2,food_price=10; //价格与重量
bool mine_list[num_of_points+1]={},village_list[num_of_points+1]={};

int water_consumption_list[3]={5,8,10},food_consumption_list[3]={7,6,10}; //消耗列表
int weather_list[deadline]={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};
    //0晴朗, 1高温, 2沙暴

struct state
{
    char i,j;
    short k,l;
    state(short ii=0,short jj=0,short kk=0,short ll=0):i(ii),j(jj),k(kk),l(ll){}
    bool operator!=(const state &st2){
        return i!=st2.i || j!=st2.j || k!=st2.k || l!=st2.l;
    }
};

vector<short> world_map[num_of_points+1];
short wmap[num_of_points+1][num_of_points+1] {{},{2}, {3}, {4},{5}, {6},{7,13}, {6,8}, {7,9},
    {8,10},
    {9,11,12},{10,12,15,16,17},{10,11,17,18},{6,14},{13,15},{11,14,16},{11,15,17},{11,12,16,18},{}};
    };
    ///!!修改2: 地图
map<int,int> mapping;

int dp[deadline+1][num_of_points+1][weight_limit/water_weight+1][weight_limit/food_weight+1];
    //dp[i][j][k][l]表示第i天结束时, j点有k水、l食物情况下的最多金钱
state
    dp_state[deadline+1][num_of_points+1][weight_limit/water_weight+1][weight_limit/food_weight+1];

// 更新规则:采用“我为人人”型dp
/* 天数i从0循环到29, 编号j从1-10, k,l完全循环
    更新 停留、挖矿、走动
    若 (k,l减去行动代价后均为正, 则与原有值取较大值)

    注意: 每天开始时对村庄点状态进行更新
        (需要四重循环, 对于每个k,l, 向上k',l'进行计算更新, 计算量600多亿, ×30天 ),
    但特别注意, 向上更新时, 一旦更新不动就可以break. 比如 对于某个k,l,
        循环k',l'向上更新, 一旦更新不动, 直接对k'l'双循环break
    共5种更新: 起点买, 村庄买, 停留, 挖矿, 走动
*/

void inline init()
{
    for(int i=1;i<=num_of_points;++i){

```

```

        mapping[i]=i;
        for(int j=0;j<=num_of_points&&wmap[i][j];++j)
            world_map[i].push_back(wmap[i][j]);
    }
    ///修改3: 映射,村庄、矿表
    mapping[4]=11;mapping[5]=20; mapping[6]=28; mapping[7]=29;
    mapping[8]=30; mapping[9]=39;mapping[10]=47; mapping[11]=55;
    mapping[12]=56; mapping[13]=37;mapping[14]=45; mapping[15]=54;
    mapping[16]=62; mapping[17]=63;mapping[18]=64;

    mine_list[8]=1;mine_list[11]=1; village_list[9]=1;village_list[9]=1;village_list[16]=1;

    for(int i=0;i<=deadline;++i)
        for(int j=0;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
                for (int l =0;k*3+l*2<=weight_limit;++l){
                    dp[i][j][k][l]=-INF;
                }
    dp[0][1][0][0]=initial_money;

    for(int k=0;k<=weight_limit/water_weight;++k){
        int money_left=dp[0][1][0][0]-k*water_price,weight_left= weight_limit-k*water_weight;
        if (money_left<0)
            break;
        for (int l =0;l<=weight_limit/food_weight;++l){
            int money_left2=money_left-l*food_price;
            if (money_left2<0 || l*food_weight>weight_left)
                break;
            if(money_left2>dp[0][1][k][l])
            {
                dp[0][1][k][l]=money_left2;
                dp_state[0][1][k][l]=state(0,1,0,0);
            }
            //cout<<dp[0][1][k][l]<<'\n';
        }
    }
}

inline void buy(int i,int j,int k,int l)
{
    int max_food=weight_limit/food_weight;
    for(int m=k;m<=weight_limit/water_weight;++m){
        for(int n=1;m*3+n*2<=weight_limit && n<max_food;++n)
        {
            if(m==k &&n==1)

```

```

        continue;
    int money_left= dp[i][j][k][l] - (m-k)*water_price*2 - (n-l)*food_price*2;
    if(money_left<=dp[i][j][m][n] || money_left<0 ) // 1\
        三角形区域的右上部分三角无需再搜索了，其他还是要更新的
    {
        max_food=n;
        break;
    }
    dp[i][j][m][n]=money_left;
    dp_state[i][j][m][n]=state(i,j,k,l);
}
}
}

int main()
{
    freopen("2_simple_result.txt","w",stdout);
    init();

    //for(int i=0;i<=num_of_points;++i) cout<<village_list[i]<<mine_list[i]<<" ";
    int best_ever=-INF;

    for(int i=0;i<30;++i){
        int ii=i+1;
        //cerr<<"i="<<i<<'\n';
        for(int j=1;j<=num_of_points;++j){
            if(village_list[j]){ //每天开始更新前一天前，村庄先购物更新
                for(int k=0;k<=weight_limit/water_weight;++k)
                    for (int l =0; k*3+l*2<=weight_limit;++l)
                        if(dp[i][j][k][l]>=0){
                            buy(i,j,k,l);
                            //cout<<"更新村庄! "<<endl;
                        }
            }
        }

        //更新停留
        int
            water_consumption=water_consumption_list[weather_list[i]],food_consumption=food_consumption_list[weather_list[i]];
        for(int k=water_consumption;k<=weight_limit/water_weight;++k){
            for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                int kk=k-water_consumption,ll=l-food_consumption;

                if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][j][kk][ll])
                {
                    dp[ii][j][kk][ll]=dp[i][j][k][l];
                    dp_state[ii][j][kk][ll]=state(i,j,k,l);
                }
            }
        }
    }
}

```



```

        //cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\n';
    }
}
}

//更新挖矿
if (mine_list[j]){
    water_consumption=water_consumption_list[weather_list[i]]*3,food_consumption=food_consumption_list[weather_list[i]]*2;
    for(int k=water_consumption;k<=weight_limit/water_weight;++k){
        for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
            int kk=k-water_consumption,ll=l-food_consumption;

            if(dp[i][j][k][l]>=0 && dp[i][j][k][l]+base_income>dp[ii][j][kk][ll])
            {
                dp[ii][j][kk][ll]=dp[i][j][k][l]+base_income;
                dp_state[ii][j][kk][ll]=state(i,j,k,l);
            }
        }
    }
}

/*for(int k=0;k<=400;++k)
    for (int l =0;k*3+l*2<=1200;++l)
        if(dp[i][j][k][l]>9900)
            cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\n';//*/

//非沙暴，更新走动
if (weather_list[i]<2){
    water_consumption=water_consumption_list[weather_list[i]]*2,food_consumption=food_consumption_list[weather_list[i]]*3;
    for(vector<short>::iterator iter=world_map[j].begin();iter!=world_map[j].end();++iter){
        for(int k=water_consumption;k<=weight_limit/water_weight;++k){
            for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                int kk=k-water_consumption,ll=l-food_consumption;

                if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][*iter][kk][ll])
                {
                    dp[ii][*iter][kk][ll]=dp[i][j][k][l];
                    dp_state[ii][*iter][kk][ll]=state(i,j,k,l);
                }
            }
        }
    }
}
}
}

```

```

int target=num_of_points;
int res=-INF;
state last_state;
for(int i=0;i<=deadline;++i){
    for(int k=0;k<=weight_limit/water_weight;++k){
        for (int l =0;k*3+l*2<=weight_limit;++l){
            if(dp[i][target][k][l]>res)
            {
                res=dp[i][target][k][l];
                last_state=state(i,target,k,l);
            }
        }
    }
}
if(res>best_ever){
    cerr<<res<<" "<<best_ever<<endl;
    best_ever=res;
    stack<state> sta;
    while(last_state!=state(0,0,0,0)){
        sta.push(last_state);
        last_state=dp_state[last_state.i][last_state.j][last_state.k][last_state.l];
    }

    cout <<i<< "days,score:" <<res<< "\nsolution:\n";

    while(!sta.empty())
    {
        last_state=sta.top();sta.pop();
        cout<<int(last_state.i)<<" "<<mapping[last_state.j]<<" "<<int(last_state.k)<<"
            "<<int(last_state.l)<<"
            "<<dp[last_state.i][last_state.j][last_state.k][last_state.l]<<endl;
    }
    cout<<"\n\n\n";
}

return 0;
}

```

desert_game_dp(第三关-完整图-路径回溯-天气枚举).cpp

```

#include <iostream>
#include <cstdio>
#include <vector>
#include <stack>
#include <map>

```



```

void inline init_space()
{
    for(int i=0;i<=deadline;++i)
        for(int j=0;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
            {
                dp[i][j][k]=new short [(weight_limit-k*3)/2 +1]; //加1了之后堆不会损坏
                dp_state[i][j][k]=new state[(weight_limit-k*3)/2 +1];
            }
}

void inline init(int weather)
{
    for(int i=0;i<deadline;++i,weather>>=1)
        weather_list[i]=weather&1;

    for(int i=1;i<=num_of_points;++i){
        mapping[i]=i;
        for(int j=0;j<=num_of_points&&wmap[i][j];++j)
            world_map[i].push_back(wmap[i][j]);
    }
    //!修改3: 映射,村庄、矿表
    mine_list[9]=1;

    for(int i=0;i<=deadline;++i)
        for(int j=0;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
                for (int l =0;k*3+l*2<=weight_limit;++l){
                    dp[i][j][k][l]=-INF;
                }
    dp[0][1][0][0]=initial_money;

    for(int k=0;k<=weight_limit/water_weight;++k){
        int money_left=dp[0][1][0][0]-k*water_price,weight_left= weight_limit-k*water_weight;
        if (money_left<0)
            break;
        for (int l =0;l<=weight_limit/food_weight;++l){
            int money_left2=money_left-l*food_price;
            if (money_left2<0 || l*food_weight>weight_left)
                break;
            if(money_left2>dp[0][1][k][l])
            {
                dp[0][1][k][l]=money_left2;
                dp_state[0][1][k][l]=state(0,1,0,0);
            }
        }
    }
}

```

```

        //cout<<dp[0][1][k][1]<<'\\n';
    }
}

inline void buy(int i,int j,int k,int l)
{
    int max_food=weight_limit/food_weight;
    for(int m=k;m<=weight_limit/water_weight;++m){
        for(int n=1;m*3+n*2<=weight_limit && n<max_food;++n)
        {
            if(m==k &&n==1)
                continue;
            int money_left= dp[i][j][k][1] - (m-k)*water_price*2 - (n-1)*food_price*2;
            if(money_left<=dp[i][j][m][n] || money_left<0 ) // 1\
                三角形区域的右上部分三角无需再搜索了，其他还是要更新的
            {
                max_food=n;
                break;
            }
            dp[i][j][m][n]=money_left;
            dp_state[i][j][m][n]=state(i,j,k,l);
        }
    }
}

int main()
{
    freopen("3_complete_result_weatherEnumerate.txt","w",stdout); ///!修改4: 输出文件名
    init_space();
    for(int w=0;w< (1<<deadline);++w)
    {
        init(w);
        for(int i=0;i<deadline;++i){
            int ii=i+1;
            //cerr<<"i="<<i<<'\\n';
            for(int j=1;j<=num_of_points;++j){
                if(village_list[j]){ //每天开始更新前一天前，村庄先购物更新
                    for(int k=0;k<=weight_limit/water_weight;++k)
                        for (int l =0; k*3+l*2<=weight_limit;++l)
                            if(dp[i][j][k][l]>=0){
                                buy(i,j,k,l);
                                //cout<<"更新村庄! "<<endl;
                            }
                }
            }
        }
    }
}

```

```

//更新停留
int
    water_consumption=water_consumption_list[weather_list[i]],food_consumption=food_consumption_list[weather_list[i]]
for(int k=water_consumption;k<=weight_limit/water_weight;++k){
    for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
        int kk=k-water_consumption,ll=l-food_consumption;

        if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][j][kk][ll])
        {
            dp[ii][j][kk][ll]=dp[i][j][k][l];
            dp_state[ii][j][kk][ll]=state(i,j,k,l);

            //cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\n';
        }
    }
}

//更新挖矿
if (mine_list[j]){
    water_consumption=water_consumption_list[weather_list[i]]*3,food_consumption=food_consumption_list[weather_list[i]]*3
    for(int k=water_consumption;k<=weight_limit/water_weight;++k){
        for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
            int kk=k-water_consumption,ll=l-food_consumption;

            if(dp[i][j][k][l]>=0 && dp[i][j][k][l]+base_income>dp[ii][j][kk][ll])
            {
                dp[ii][j][kk][ll]=dp[i][j][k][l]+base_income;
                dp_state[ii][j][kk][ll]=state(i,j,k,l);
            }
        }
    }
}

/*for(int k=0;k<=400;++k)
    for (int l =0;k*3+l*2<=1200;++l)
        if(dp[i][j][k][l]>9900)
            cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\n';//*/

//非沙暴，更新走动
if (weather_list[i]<2){
    water_consumption=water_consumption_list[weather_list[i]]*2,food_consumption=food_consumption_list[weather_list[i]]*2
    for(vector<short>::iterator
        iter=world_map[j].begin();iter!=world_map[j].end();++iter){
        for(int k=water_consumption;k<=weight_limit/water_weight;++k){
            for (int l =food_consumption;k*3+l*2<=weight_limit;++l){

```



```

        last_state=sta.top();sta.pop();
        cout<<int(last_state.i)<<" "<<mapping[last_state.j]<<" "<<int(last_state.k)<<"
            "<<int(last_state.l)<<"
            "<<dp[last_state.i][last_state.j][last_state.k][last_state.l]<<endl;
    }
    cout<<"\n\n\n";
}
else
    cout<<"no solution \n\n\n";
}
return 0;
}

```

desert_game_dp(第三关-简化图-路径回溯-天气枚举-购买统计).cpp

```

#include <iostream>
#include <cstdio>
#include <vector>
#include <stack>
#include <map>
#include <cstring>
#include <fstream>
using namespace std;

const int INF=0x3f3f3f3f;

const int num_of_points=4; ///!修改1: 地区数量
const int deadline=10;      ///!修改8期限
const int initial_money=10000; ///初始金钱
const int base_income=200;  ///!修改6: 基本收益
const int weight_limit=1200; ///负重上限
const int water_weight=3,water_price=5,food_weight=2,food_price=10; ///价格与重量
bool mine_list[num_of_points+1]={},village_list[num_of_points+1]={};

int water_consumption_list[3]={3,9,10},food_consumption_list[3]={4,9,10};///!修改7: 消耗列表
int weather_list[deadline]; ///!修改5: 天气生成。 0晴朗, 1高温, 2沙暴

struct state
{
    char i,j;
    short k,l;
    state(short ii=0,short jj=0,short kk=0,short ll=0):i(ii),j(jj),k(kk),l(ll){}
    bool operator!=(const state &st2){
        return i!=st2.i || j!=st2.j || k!=st2.k || l!=st2.l;
    }
};

```



```

vector<short> world_map[num_of_points+1];
short wmap[num_of_points+1][num_of_points+1]={},{2},{3},{4}};
///  
修改2: 地图

map<short,short> mapping;
short *dp[deadline+1][num_of_points+1][weight_limit/water_weight+1];
    //dp[i][j][k][l]表示第i天结束时, j点有k水、l食物情况下的最多金钱
state *dp_state[deadline+1][num_of_points+1][weight_limit/water_weight+1];

int num_of_cases[weight_limit/water_weight+1][weight_limit/food_weight+1]={};

// 更新规则:采用“我为人人”型dp
/* 天数i从0循环到29, 编号j从1-10, k,l完全循环
    更新 停留、挖矿、走动
    若(k,l减去行动代价后均为正, 则与原有值取较大值)

    注意: 每天开始时对村庄点状态进行更新
        (需要四重循环, 对于每个k,l,向上k',l'进行计算更新, 计算量600多亿, *30天),
    但特别注意, 向上更新时, 一旦更新不动就可以break. 比如 对于某个k,l,
        循环k',l'向上更新, 一旦更新不动, 直接对k'l'双循环break
    共5种更新: 起点买, 村庄买, 停留, 挖矿, 走动
*/

void inline init()
{
    memset(num_of_cases,0,sizeof(num_of_points));

    for(int i=0;i<=deadline;++i)
        for(int j=0;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
            {
                dp[i][j][k]=new short [(weight_limit-k*3)/2 +1]; //加1了之后堆不会损坏
                dp_state[i][j][k]=new state[(weight_limit-k*3)/2 +1];
            }

    for(int i=1;i<=num_of_points;++i){
        mapping[i]=i;
        for(int j=0;j<=num_of_points&&wmap[i][j];++j)
            world_map[i].push_back(wmap[i][j]);
    }
    mapping[2]=4;mapping[3]=6,mapping[4]=13;
    ///  
修改3: 映射,村庄、矿表

```

```

//mine_list[9]=1;
for(int i=0;i<=deadline;++i)
    for(int j=0;j<=num_of_points;++j)
        for(int k=0;k<=weight_limit/water_weight;++k)
            for (int l =0;k*3+l*2<=weight_limit;++l){
                dp[i][j][k][l]=-INF;
            }
dp[0][1][0][0]=initial_money;

for(int k=0;k<=weight_limit/water_weight;++k){
    int money_left=dp[0][1][0][0]-k*water_price,weight_left= weight_limit-k*water_weight;
    if (money_left<0)
        break;
    for (int l =0;l<=weight_limit/food_weight;++l){
        int money_left2=money_left-l*food_price;
        if (money_left2<0 || l*food_weight>weight_left)
            break;
        if(money_left2>dp[0][1][k][l])
        {
            dp[0][1][k][l]=money_left2;
            dp_state[0][1][k][l]=state(0,1,0,0);
        }
        ////cout<<dp[0][1][k][l]<<'\n';
    }
}

}

void inline init_for(int weather)
{
    for(int i=0;i<deadline;++i,weather>=>1)
        weather_list[i]=weather&1;

    for(int i=1;i<=deadline;++i)
        for(int j=1;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
                for (int l =0;k*3+l*2<=weight_limit;++l){
                    dp[i][j][k][l]=-INF;
                }
}

inline void buy(int i,int j,int k,int l)
{

```

```

int max_food=weight_limit/food_weight;
for(int m=k;m<=weight_limit/water_weight;++m){
    for(int n=1;m*3+n*2<=weight_limit && n<max_food;++n)
    {
        if(m==k &&n==1)
            continue;
        int money_left= dp[i][j][k][l] - (m-k)*water_price*2 - (n-l)*food_price*2;
        if(money_left<=dp[i][j][m][n] || money_left<0 ) // 1\
            三角形区域的右上部分三角无需再搜索了，其他还是要更新的
        {
            max_food=n;
            break;
        }
        dp[i][j][m][n]=money_left;
        dp_state[i][j][m][n]=state(i,j,k,l);
    }
}
}

int main()
{
    freopen("3_complete_result_weatherEnumerate_buyingStatistic.txt","w",stdout);
    ///!!修改4: 输出文件名
    init();

    for(int w=0;w< (1<<deadline);++w)
    {
        init_for(w);
        for(int i=0;i<deadline;++i){
            int ii=i+1;
            //cerr<<"i="<<i<<'\\n';
            for(int j=1;j<=num_of_points;++j){
                if(village_list[j]){ //每天开始更新前一天前，村庄先购物更新
                    for(int k=0;k<=weight_limit/water_weight;++k)
                        for (int l =0; k*3+l*2<=weight_limit;++l)
                            if(dp[i][j][k][l]>=0){
                                buy(i,j,k,l);
                                ///cout<<"更新村庄! "<<endl;
                            }
                }

                //更新停留
                int
                water_consumption=water_consumption_list[weather_list[i]],food_consumption=food_consumption_list
                for(int k=water_consumption;k<=weight_limit/water_weight;++k){
                    for (int l =food_consumption;k*3+l*2<=weight_limit;++l){

```

```

        int kk=k-water_consumption,ll=l-food_consumption;

        if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][j][kk][ll])
        {
            dp[ii][j][kk][ll]=dp[i][j][k][l];
            dp_state[ii][j][kk][ll]=state(i,j,k,l);

            ////cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\\n';
        }
    }
}

//更新挖矿
if (mine_list[j]){
    water_consumption=water_consumption_list[weather_list[i]]*3,food_consumption=food_consumption_list[weather_list[i]]*3;
    for(int k=water_consumption;k<=weight_limit/water_weight;++k){
        for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
            int kk=k-water_consumption,ll=l-food_consumption;

            if(dp[i][j][k][l]>=0 && dp[i][j][k][l]+base_income>dp[ii][j][kk][ll])
            {
                dp[ii][j][kk][ll]=dp[i][j][k][l]+base_income;
                dp_state[ii][j][kk][ll]=state(i,j,k,l);
            }
        }
    }
}

/*for(int k=0;k<=400;++k)
    for (int l =0;k*3+l*2<=1200;++l)
        if(dp[i][j][k][l]>9900)
            //cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\\n';//*/

//非沙暴，更新走动
if (weather_list[i]<2){
    water_consumption=water_consumption_list[weather_list[i]]*2,food_consumption=food_consumption_list[weather_list[i]]*2;
    for(vector<short>::iterator
        iter=world_map[j].begin();iter!=world_map[j].end();++iter){
        for(int k=water_consumption;k<=weight_limit/water_weight;++k){
            for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                int kk=k-water_consumption,ll=l-food_consumption;

                if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][*iter][kk][ll])
                {
                    dp[ii][*iter][kk][ll]=dp[i][j][k][l];
                    dp_state[ii][*iter][kk][ll]=state(i,j,k,l);
                }
            }
        }
    }
}

```

```

    }
    }
    }
}

int target=num_of_points;
int res=-INF, arrive_day;
state last_state;
for(int i=0; i<=deadline; ++i){
    for(int k=0; k<=weight_limit/water_weight; ++k){
        for (int l =0; k*3+l*2<=weight_limit; ++l){
            if(dp[i][target][k][l]>res)
            {
                res=dp[i][target][k][l];
                arrive_day=i;
                last_state=state(i, target, k, l);
            }
        }
    }
}

if(res>0){
    while(last_state.i!=0){
        last_state=dp_state[last_state.i][last_state.j][last_state.k][last_state.l];
    }

    cerr<<w<<" "<<res<<" "<<last_state.k<<" "<<last_state.l<<endl;
    num_of_cases[last_state.k][last_state.l]+=1;
}

}

ofstream outFile;
outFile.open("3_complete_result_weatherEnumerate_buyingStatistic.csv", ios::ate);

for(int i=0; i<=weight_limit/water_weight; ++i) {
    outFile <<num_of_cases[i][0];
    for(int j=1; j<=weight_limit/food_weight; ++j)
        outFile <<" "<<num_of_cases[i][j];
    outFile<<'\n';
}

outFile.close();

```

```

int ss=0;
for(int i=0; i<=400; ++i)
    for(int j=1;j<=600; ++j)
        ss+=num_of_cases[i][j];
cerr<<"ss="<<ss<<endl;

return 0;
}

```

desert_game_dp(第三关-简化图-路径回溯-天气枚举，天数统计).cpp

```

#include <iostream>
#include <cstdio>
#include <vector>
#include <stack>
#include <map>

using namespace std;

const int INF=0x3f3f3f3f;

const int num_of_points=4;  ///!!修改1: 地区数量
const int deadline=10;      ///!!修改8期限
const int initial_money=10000; ///初始金钱
const int base_income=200;  ///!!修改6: 基本收益
const int weight_limit=1200; ///负重上限
const int water_weight=3,water_price=5,food_weight=2,food_price=10; ///价格与重量
bool mine_list[num_of_points+1]={},village_list[num_of_points+1]={};

int daysum[11]={};
int water_consumption_list[3]={3,9,10},food_consumption_list[3]={4,9,10};///!!修改7: 消耗列表
int weather_list[deadline]; ///!!修改5: 天气生成。 0晴朗, 1高温, 2沙暴

struct state
{
    char i,j;
    short k,l;
    state(short ii=0,short jj=0,short kk=0,short ll=0):i(ii),j(jj),k(kk),l(ll){}
    bool operator!=(const state &st2){
        return i!=st2.i || j!=st2.j || k!=st2.k || l!=st2.l;
    }
};

vector<short> world_map[num_of_points+1];
short wmap[num_of_points+1][num_of_points+1]={ {},{2},{3},{4}};
///!!修改2: 地图

```

```

map<short,short> mapping;
short *dp[deadline+1][num_of_points+1][weight_limit/water_weight+1];
    //dp[i][j][k][l]表示第i天结束时，j点有k水、l食物情况下的最多金钱
state *dp_state[deadline+1][num_of_points+1][weight_limit/water_weight+1];

// 更新规则：采用“我为人人”型dp
/* 天数i从0循环到29， 编号j从1-10，k,l完全循环
    更新 停留、挖矿、走动
    若（k,l减去行动代价后均为正，则与原有值取较大值）

    注意：每天开始时对村庄点状态进行更新
        （需要四重循环，对于每个k,l,向上k',l'进行计算更新，计算量600多亿，×30天），
    但特别注意，向上更新时，一旦更新不动就可以break。比如 对于某个k,l，
        循环k',l'向上更新，一旦更新不动，直接对k'l'双循环break
    共5种更新： 起点买，村庄买，停留，挖矿，走动
*/

void inline init()
{
    for(int i=0;i<=deadline;++i)
        for(int j=0;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
            {
                dp[i][j][k]=new short [(weight_limit-k*3)/2 +1]; //加1了之后堆不会损坏
                dp_state[i][j][k]=new state[(weight_limit-k*3)/2 +1];
            }

    for(int i=1;i<=num_of_points;++i){
        mapping[i]=i;
        for(int j=0;j<=num_of_points&&wmap[i][j];++j)
            world_map[i].push_back(wmap[i][j]);
    }
    mapping[2]=4;mapping[3]=6,mapping[4]=13;
    ///修改3：映射,村庄、矿表

    //mine_list[9]=1;
    for(int i=0;i<=deadline;++i)
        for(int j=0;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
                for (int l =0;k*3+l*2<=weight_limit;++l){
                    dp[i][j][k][l]=-INF;
                }
    dp[0][1][0][0]=initial_money;

```

```

for(int k=0;k<=weight_limit/water_weight;++k){
    int money_left=dp[0][1][0][0]-k*water_price,weight_left= weight_limit-k*water_weight;
    if (money_left<0)
        break;
    for (int l =0;l<=weight_limit/food_weight;++l){
        int money_left2=money_left-l*food_price;
        if (money_left2<0 || l*food_weight>weight_left)
            break;
        if(money_left2>dp[0][1][k][l])
        {
            dp[0][1][k][l]=money_left2;
            dp_state[0][1][k][l]=state(0,1,0,0);
        }
        ///cout<<dp[0][1][k][l]<<'\\n';
    }
}

void inline init_for(int weather)
{
    for(int i=0;i<deadline;++i,weather>=>1)
        weather_list[i]=weather&1;

    for(int i=1;i<=deadline;++i)
        for(int j=1;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
                for (int l =0;k*3+l*2<=weight_limit;++l){
                    dp[i][j][k][l]=-INF;
                }
}

inline void buy(int i,int j,int k,int l)
{
    int max_food=weight_limit/food_weight;
    for(int m=k;m<=weight_limit/water_weight;++m){
        for(int n=l;m*3+n*2<=weight_limit && n<max_food;++n)
        {
            if(m==k &&n==l)
                continue;
            int money_left= dp[i][j][k][l] - (m-k)*water_price*2 - (n-l)*food_price*2;
            if(money_left<=dp[i][j][m][n] || money_left<0 ) // \\
                三角形区域的右上部分三角无需再搜索了，其他还是要更新的
            {
                max_food=n;
            }
        }
    }
}

```



```

        break;
    }
    dp[i][j][m][n]=money_left;
    dp_state[i][j][m][n]=state(i,j,k,l);
}
}

int main()
{
    freopen("3_complete_result_weatherEnumerate.txt","w",stdout); ///!!修改4: 输出文件名
    init();

    for(int w=0;w< (1<<deadline);++w)
    {

        init_for(w);
        for(int i=0;i<deadline;++i){
            int ii=i+1;
            //cerr<<"i="<<i<<'\\n';
            for(int j=1;j<=num_of_points;++j){
                if(village_list[j]){ //每天开始更新下一天前, 村庄先购物更新
                    for(int k=0;k<=weight_limit/water_weight;++k)
                        for (int l =0; k*3+l*2<=weight_limit;++l)
                            if(dp[i][j][k][l]>=0){
                                buy(i,j,k,l);
                                ////cout<<"更新村庄! "<<endl;
                            }
                }

                //更新停留
                int
                water_consumption=water_consumption_list[weather_list[i]],food_consumption=food_consumption_list[weather_list[i]];
                for(int k=water_consumption;k<=weight_limit/water_weight;++k){
                    for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                        int kk=k-water_consumption,ll=l-food_consumption;

                        if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][j][kk][ll])
                        {
                            dp[ii][j][kk][ll]=dp[i][j][k][l];
                            dp_state[ii][j][kk][ll]=state(i,j,k,l);

                            ////cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\\n';
                        }
                    }
                }
            }
        }
    }
}

```

```

//更新挖矿
if (mine_list[j]){
    water_consumption=water_consumption_list[weather_list[i]]*3,food_consumption=food_consumption_list[weather_list[i]]*3;
    for(int k=water_consumption;k<=weight_limit/water_weight;++k){
        for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
            int kk=k-water_consumption,ll=l-food_consumption;

            if(dp[i][j][k][l]>=0 && dp[i][j][k][l]+base_income>dp[ii][j][kk][ll])
            {
                dp[ii][j][kk][ll]=dp[i][j][k][l]+base_income;
                dp_state[ii][j][kk][ll]=state(i,j,k,l);
            }
        }
    }
}

/*for(int k=0;k<=400;++k)
    for (int l =0;k*3+l*2<=1200;++l)
        if(dp[i][j][k][l]>9900)
            //cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\n';//*/

//非沙暴，更新走动
if (weather_list[i]<2){
    water_consumption=water_consumption_list[weather_list[i]]*2,food_consumption=food_consumption_list[weather_list[i]]*2;
    for(vector<short>::iterator
        iter=world_map[j].begin();iter!=world_map[j].end();++iter){
        for(int k=water_consumption;k<=weight_limit/water_weight;++k){
            for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                int kk=k-water_consumption,ll=l-food_consumption;

                if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][*iter][kk][ll])
                {
                    dp[ii][*iter][kk][ll]=dp[i][j][k][l];
                    dp_state[ii][*iter][kk][ll]=state(i,j,k,l);
                }
            }
        }
    }
}

}

}

int target=num_of_points;
int res=-INF,arrive_day;
state last_state;

```

```

        for(int i=0;i<=deadline;++i){
            for(int k=0;k<=weight_limit/water_weight;++k){
                for (int l =0;k*3+l*2<=weight_limit;++l){
                    if(dp[i][target][k][l]>res)
                    {
                        res=dp[i][target][k][l];
                        arrive_day=i;
                        last_state=state(i,target,k,l);
                    }
                }
            }
        }

        cerr<<w<<". "<<arrive_day<<endl;
        daysum[arrive_day]+=1;

        for(int i=1;i<=10;++i)
            cout<<daysum[i]<<" ";
    }
    return 0;
}

```

desert_game_dp(第三关-简化图-路径回溯-天气枚举).cpp

```

#include <iostream>
#include <cstdio>
#include <vector>
#include <stack>
#include <map>

using namespace std;

const int INF=0x3f3f3f3f;

const int num_of_points=4;  ///!!修改1: 地区数量
const int deadline=10;      ///!!修改8期限
const int initial_money=10000; ///初始金钱
const int base_income=200;  ///!!修改6: 基本收益
const int weight_limit=1200; ///负重上限
const int water_weight=3,water_price=5,food_weight=2,food_price=10; ///价格与重量
bool mine_list[num_of_points+1]={},village_list[num_of_points+1]={};

int water_consumption_list[3]={3,9,10},food_consumption_list[3]={4,9,10};///!!修改7: 消耗列表
int weather_list[deadline]; ///!!修改5: 天气生成。 0晴朗, 1高温, 2沙暴

struct state
{

```

```

char i,j;
short k,l;
state(short ii=0,short jj=0,short kk=0,short ll=0):i(ii),j(jj),k(kk),l(ll){}
bool operator!=(const state &st2){
    return i!=st2.i || j!=st2.j || k!=st2.k || l!=st2.l;
}
};

vector<short> world_map[num_of_points+1];
short wmap[num_of_points+1][num_of_points+1]={},{2},{3},{4}};
//!!修改2: 地图

map<short,short> mapping;
short *dp[deadline+1][num_of_points+1][weight_limit/water_weight+1];
    //dp[i][j][k][l]表示第i天结束时, j点有k水、l食物情况下的最多金钱
state *dp_state[deadline+1][num_of_points+1][weight_limit/water_weight+1];

// 更新规则:采用“我为人人”型dp
/* 天数i从0循环到29, 编号j从1-10, k,l完全循环
    更新 停留、挖矿、走动
    若 (k,l减去行动代价后均为正, 则与原有值取较大值)

    注意: 每天开始时对村庄点状态进行更新
        (需要四重循环, 对于每个k,l, 向上k',l'进行计算更新, 计算量600多亿, *30天),
    但特别注意, 向上更新时, 一旦更新不动就可以break. 比如 对于某个k,l,
        循环k',l'向上更新, 一旦更新不动, 直接对k'l'双循环break
    共5种更新: 起点买, 村庄买, 停留, 挖矿, 走动
*/

void inline init()
{
    for(int i=0;i<=deadline;++i)
        for(int j=0;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
            {
                dp[i][j][k]=new short [(weight_limit-k*3)/2 +1]; //加1了之后堆不会损坏
                dp_state[i][j][k]=new state[(weight_limit-k*3)/2 +1];
            }

    for(int i=1;i<=num_of_points;++i){
        mapping[i]=i;
        for(int j=0;j<=num_of_points&&wmap[i][j];++j)
            world_map[i].push_back(wmap[i][j]);
    }
    mapping[2]=4;mapping[3]=6,mapping[4]=13;
    //!!修改3: 映射,村庄、矿表

```

```

//mine_list[9]=1;
for(int i=0;i<=deadline;++i)
    for(int j=0;j<=num_of_points;++j)
        for(int k=0;k<=weight_limit/water_weight;++k)
            for (int l =0;k*3+l*2<=weight_limit;++l){
                dp[i][j][k][l]=-INF;
            }
dp[0][1][0][0]=initial_money;

for(int k=0;k<=weight_limit/water_weight;++k){
    int money_left=dp[0][1][0][0]-k*water_price,weight_left= weight_limit-k*water_weight;
    if (money_left<0)
        break;
    for (int l =0;l<=weight_limit/food_weight;++l){
        int money_left2=money_left-l*food_price;
        if (money_left2<0 || l*food_weight>weight_left)
            break;
        if(money_left2>dp[0][1][k][l])
        {
            dp[0][1][k][l]=money_left2;
            dp_state[0][1][k][l]=state(0,1,0,0);
        }
        //cout<<dp[0][1][k][l]<<'\n';
    }
}

}

void inline init_for(int weather)
{
    for(int i=0;i<deadline;++i,weather>>=1)
        weather_list[i]=weather&1;

    for(int i=1;i<=deadline;++i)
        for(int j=1;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
                for (int l =0;k*3+l*2<=weight_limit;++l){
                    dp[i][j][k][l]=-INF;
                }
}

inline void buy(int i,int j,int k,int l)
{

```

```

int max_food=weight_limit/food_weight;
for(int m=k;m<=weight_limit/water_weight;++m){
    for(int n=1;m*3+n*2<=weight_limit && n<max_food;++n)
    {
        if(m==k &&n==1)
            continue;
        int money_left= dp[i][j][k][l] - (m-k)*water_price*2 - (n-l)*food_price*2;
        if(money_left<=dp[i][j][m][n] || money_left<0 ) // 1\
            三角形区域的右上部分三角无需再搜索了，其他还是要更新的
        {
            max_food=n;
            break;
        }
        dp[i][j][m][n]=money_left;
        dp_state[i][j][m][n]=state(i,j,k,l);
    }
}
}

int main()
{
    freopen("3_complete_result_weatherEnumerate.txt","w",stdout); ///!修改4: 输出文件名
    init();

    for(int w=0;w< (1<<deadline);++w)
    {
        init_for(w);
        for(int i=0;i<deadline;++i){
            int ii=i+1;
            //cerr<<"i="<<i<<'\\n';
            for(int j=1;j<=num_of_points;++j){
                if(village_list[j]){ //每天开始更新前一天前，村庄先购物更新
                    for(int k=0;k<=weight_limit/water_weight;++k)
                        for (int l =0; k*3+l*2<=weight_limit;++l)
                            if(dp[i][j][k][l]>=0){
                                buy(i,j,k,l);
                                //cout<<"更新村庄! "<<endl;
                            }
                }
            }

            //更新停留
            int
                water_consumption=water_consumption_list[weather_list[i]],food_consumption=food_consumption_list[
            for(int k=water_consumption;k<=weight_limit/water_weight;++k){
                for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                    int kk=k-water_consumption,ll=l-food_consumption;

```

```

        if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][j][kk][ll])
        {
            dp[ii][j][kk][ll]=dp[i][j][k][l];
            dp_state[ii][j][kk][ll]=state(i,j,k,l);

            //cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\n';
        }
    }
}

//更新挖矿
if (mine_list[j]){
    water_consumption=water_consumption_list[weather_list[i]]*3,food_consumption=food_consumption_list[weather_list[i]]*3;
    for(int k=water_consumption;k<=weight_limit/water_weight;++k){
        for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
            int kk=k-water_consumption,ll=l-food_consumption;

            if(dp[i][j][k][l]>=0 && dp[i][j][k][l]+base_income>dp[ii][j][kk][ll])
            {
                dp[ii][j][kk][ll]=dp[i][j][k][l]+base_income;
                dp_state[ii][j][kk][ll]=state(i,j,k,l);
            }
        }
    }
}

/*for(int k=0;k<=400;++k)
    for (int l =0;k*3+l*2<=1200;++l)
        if(dp[i][j][k][l]>9900)
            cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\n';//*/

//非沙暴，更新走动
if (weather_list[i]<2){
    water_consumption=water_consumption_list[weather_list[i]]*2,food_consumption=food_consumption_list[weather_list[i]]*2;
    for(vector<short>::iterator
        iter=world_map[j].begin();iter!=world_map[j].end();++iter){
        for(int k=water_consumption;k<=weight_limit/water_weight;++k){
            for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                int kk=k-water_consumption,ll=l-food_consumption;

                if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][*iter][kk][ll])
                {
                    dp[ii][*iter][kk][ll]=dp[i][j][k][l];
                    dp_state[ii][*iter][kk][ll]=state(i,j,k,l);
                }
            }
        }
    }
}

```

```

    }
    }
}

int target=num_of_points;
int res=-INF,arrive_day;
state last_state;
for(int i=0;i<=deadline;++i){
    for(int k=0;k<=weight_limit/water_weight;++k){
        for (int l =0;k*3+l*2<=weight_limit;++l){
            if(dp[i][target][k][l]>res)
            {
                res=dp[i][target][k][l];
                arrive_day=i;
                last_state=state(i,target,k,l);
            }
        }
    }
}

if(res>0){
    cerr<<w<<" "<<res<<endl;

    stack<state> sta;
    while(last_state!=state(0,0,0,0)){
        sta.push(last_state);
        last_state=dp_state[last_state.i][last_state.j][last_state.k][last_state.l];
    }
    cout<<"weather:";
    for(int q=0;q<deadline;++q)
        cout<<weather_list[q];
    cout<<"\n";

    cout <<arrive_day<< "days,score:" <<res<< "\nsolution:\n";

    while(!sta.empty())
    {
        last_state=sta.top();sta.pop();
        cout<<int(last_state.i)<<" "<<mapping[last_state.j]<<" "<<int(last_state.k)<<"
            "<<int(last_state.l)<<"
            "<<dp[last_state.i][last_state.j][last_state.k][last_state.l]<<endl;
    }
    cout<<"\n\n\n";
}
}

```


}

```

//dp[i][j][k][l]表示第i天结束时，j点有k水、l食物情况下的最多金钱
state
    dp_state[deadline+1][num_of_points+1][weight_limit/water_weight+1][weight_limit/food_weight+1];

// 更新规则:采用“我为人人”型dp
/* 天数i从0循环到29， 编号j从1-10，k,l完全循环
    更新 停留、挖矿、走动
        若 (k,l减去行动代价后均为正，则与原有值取较大值)

    注意：每天开始时对村庄点状态进行更新
        (需要四重循环，对于每个k,l,向上k',l'进行计算更新，计算量600多亿，×30天)，
    但特别注意，向上更新时，一旦更新不动就可以break。比如 对于某个k,l,
        循环k',l'向上更新，一旦更新不动，直接对k'l'双循环break
    共5种更新： 起点买，村庄买，停留，挖矿，走动
*/

void inline init()
{
    for(int i=1;i<=num_of_points;++i){
        mapping[i]=i;
        for(int j=0;j<=num_of_points&&wmap[i][j];++j)
            world_map[i].push_back(wmap[i][j]);
    }
    ///修改3: 映射,村庄、矿表
    mapping[4]=8;mapping[5]=13;mapping[6]=14;mapping[7]=18;mapping[8]=19;mapping[9]=20;mapping[10]=25;
    mine_list[7]=1;village_list[6]=1;

    for(int i=0;i<=deadline;++i)
        for(int j=0;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
                for (int l =0;k*3+l*2<=weight_limit;++l){
                    dp[i][j][k][l]=-INF;
                }
    dp[0][1][0][0]=initial_money;

    for(int k=0;k<=weight_limit/water_weight;++k){
        int money_left=dp[0][1][0][0]-k*water_price,weight_left= weight_limit-k*water_weight;
        if (money_left<0)
            break;
        for (int l =0;l<=weight_limit/food_weight;++l){
            int money_left2=money_left-l*food_price;
            if (money_left2<0 || l*food_weight>weight_left)
                break;
            if(money_left2>dp[0][1][k][l])
                {

```

```

        dp[0][1][k][1]=money_left2;
        dp_state[0][1][k][1]=state(0,1,0,0);
    }
    //cout<<dp[0][1][k][1]<<'\n';
}
}
}

inline void buy(int i,int j,int k,int l)
{
    int max_food=weight_limit/food_weight;
    for(int m=k;m<=weight_limit/water_weight;++m){
        for(int n=1;m*3+n*2<=weight_limit && n<max_food;++n)
        {
            if(m==k &&n==1)
                continue;
            int money_left= dp[i][j][k][1] - (m-k)*water_price*2 - (n-1)*food_price*2;
            if(money_left<=dp[i][j][m][n] || money_left<0 ) // |\
                三角形区域的右上部分三角无需再搜索了，其他还是要更新的
            {
                max_food=n;
                break;
            }
            dp[i][j][m][n]=money_left;
            dp_state[i][j][m][n]=state(i,j,k,l);
        }
    }
}

int main()
{
    freopen("4_simple_result_highTemperature.txt","w",stdout);
    ///!!修改4: 输出文件名

    init();

    //for(int i=0;i<=num_of_points;++i) cout<<village_list[i]<<mine_list[i]<<" ";
    int best_ever=-INF;

    for(int i=0;i<30;++i){
        int ii=i+1;
        //cerr<<"i="<<i<<'\n';
        for(int j=1;j<=num_of_points;++j){
            if(village_list[j]){ //每天开始更新前一天前，村庄先购物更新
                for(int k=0;k<=weight_limit/water_weight;++k)

```

```

        for (int l =0; k*3+l*2<=weight_limit;++l)
            if(dp[i][j][k][l]>=0){
                buy(i,j,k,l);
                //cout<<"更新村庄! "<<endl;
            }
    }

    //更新停留
    int
        water_consumption=water_consumption_list[weather_list[i]],food_consumption=food_consumption_list[weather_list[i]];
    for(int k=water_consumption;k<=weight_limit/water_weight;++k){
        for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
            int kk=k-water_consumption,ll=l-food_consumption;

            if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][j][kk][ll])
            {
                dp[ii][j][kk][ll]=dp[i][j][k][l];
                dp_state[ii][j][kk][ll]=state(i,j,k,l);

                //cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\n';
            }
        }
    }

    //更新挖矿
    if (mine_list[j]){
        water_consumption=water_consumption_list[weather_list[i]]*3,food_consumption=food_consumption_list[weather_list[i]];
        for(int k=water_consumption;k<=weight_limit/water_weight;++k){
            for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                int kk=k-water_consumption,ll=l-food_consumption;

                if(dp[i][j][k][l]>=0 && dp[i][j][k][l]+base_income>dp[ii][j][kk][ll])
                {
                    dp[ii][j][kk][ll]=dp[i][j][k][l]+base_income;
                    dp_state[ii][j][kk][ll]=state(i,j,k,l);
                }
            }
        }
    }

    /*for(int k=0;k<=400;++k)
        for (int l =0;k*3+l*2<=1200;++l)
            if(dp[i][j][k][l]>9900)
                cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\n';//*/

    //非沙暴，更新走动

```

```

        if (weather_list[i]<2){
            water_consumption=water_consumption_list[weather_list[i]]*2,food_consumption=food_consumption_list[weather_list[i]]*2;
            for(vector<short>::iterator
                iter=world_map[j].begin();iter!=world_map[j].end();++iter){
                for(int k=water_consumption;k<=weight_limit/water_weight;++k){
                    for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                        int kk=k-water_consumption,ll=l-food_consumption;

                        if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][*iter][kk][ll])
                        {
                            dp[ii][*iter][kk][ll]=dp[i][j][k][l];
                            dp_state[ii][*iter][kk][ll]=state(i,j,k,l);
                        }
                    }
                }
            }
        }
    }

    int target=num_of_points;
    int res=-INF;
    state last_state;
    for(int i=0;i<=deadline;++i){
        for(int k=0;k<=weight_limit/water_weight;++k){
            for (int l =0;k*3+l*2<=weight_limit;++l){
                if(dp[i][target][k][l]>res)
                {
                    res=dp[i][target][k][l];
                    last_state=state(i,target,k,l);
                }
            }
        }
    }
    if(res>best_ever){
        cerr<<res<<" "<<best_ever<<endl;
        best_ever=res;
        stack<state> sta;
        while(last_state!=state(0,0,0,0)){
            sta.push(last_state);
            last_state=dp_state[last_state.i][last_state.j][last_state.k][last_state.l];
        }

        cout <<i<< "days,score:" <<res<< " \nsolution:\n";

        while(!sta.empty())
        {

```

```

        last_state=sta.top();sta.pop();
        cout<<int(last_state.i)<<" "<<mapping[last_state.j]<<" "<<int(last_state.k)<<"
            "<<int(last_state.l)<<"
            "<<dp[last_state.i][last_state.j][last_state.k][last_state.l]<<endl;
    }
    cout<<"\n\n\n";
}
}

return 0;
}

```

desert_game_dp(第四关-简化图-路径回溯-天气随机).cpp

```

#include <iostream>
#include <cstdio>
#include <vector>
#include <stack>
#include <map>
#include <windows.h>
#include <cstdlib>
#include <ctime>
using namespace std;

const int INF=0x3f3f3f3f;

const int num_of_points=10;  ///!!修改1: 地区数量
const int deadline=30;      ///!!修改8期限
const int initial_money=10000; //初始金钱
const int base_income=1000;  ///!!修改6: 基本收益
const int weight_limit=1200; //负重上限
const int water_weight=3,water_price=5,food_weight=2,food_price=10; //价格与重量
bool mine_list[num_of_points+1]={},village_list[num_of_points+1]={};

int water_consumption_list[3]={3,9,10},food_consumption_list[3]={4,9,10};///!!修改7: 消耗列表
short weather_list[deadline]={0, 1, 0, 1, 1, 1, 2, 0, 1, 1, 2, 2, 1, 2, 1, 0, 1, 1, 0, 1, 0,
    0, 0, 1, 2, 2, 1, 1, 1, 0};
///!!修改5: 天气生成。 0晴朗, 1高温, 2沙暴

struct state
{
    char i,j;
    short k,l;
    state(short ii=0,short jj=0,short kk=0,short ll=0):i(ii),j(jj),k(kk),l(ll){}
    bool operator!=(const state &st2){
        return i!=st2.i || j!=st2.j ||k!=st2.k ||l!=st2.l;
    }
}

```

```

};

vector<short> world_map[num_of_points+1];
short
    wmap[num_of_points+1][num_of_points+1]={ {},{2},{3},{4},{5},{6,7},{8},{8},{6,7,9},{10},{}};
///  
修改2: 地图

map<short,short> mapping;
short *dp[deadline+1][num_of_points+1][weight_limit/water_weight+1];
    ///  
//dp[i][j][k][l]表示第i天结束时, j点有k水、l食物情况下的最多金钱
state *dp_state[deadline+1][num_of_points+1][weight_limit/water_weight+1];

void inline init()
{
    for(int i=0;i<=deadline;++i)
        for(int j=0;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
            {
                dp[i][j][k]=new short [(weight_limit-k*3)/2 +1]; ///  
加1了之后堆不会损坏
                dp_state[i][j][k]=new state[(weight_limit-k*3)/2 +1];
            }

    for(int i=1;i<=num_of_points;++i){
        mapping[i]=i;
        for(int j=0;j<=num_of_points&&wmap[i][j];++j)
            world_map[i].push_back(wmap[i][j]);
    }
    mapping[4]=8;mapping[5]=13;mapping[6]=14;mapping[7]=18;mapping[8]=19;mapping[9]=20;mapping[10]=25;
    mine_list[7]=1;village_list[6]=1;
    ///  
修改3: 映射,村庄、矿表

    ///  
//mine_list[9]=1;
    for(int i=0;i<=deadline;++i)
        for(int j=0;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
                for (int l =0;k*3+l*2<=weight_limit;++l){
                    dp[i][j][k][l]=-INF;
                }
    dp[0][1][0][0]=initial_money;

    for(int k=0;k<=weight_limit/water_weight;++k){
        int money_left=dp[0][1][0][0]-k*water_price,weight_left= weight_limit-k*water_weight;

```

```

        if (money_left<0)
            break;
        for (int l =0;l<=weight_limit/food_weight;++l){
            int money_left2=money_left-l*food_price;
            if (money_left2<0 || l*food_weight>weight_left)
                break;
            if(money_left2>dp[0][1][k][1])
            {
                dp[0][1][k][1]=money_left2;
                dp_state[0][1][k][1]=state(0,1,0,0);
            }
            //cout<<dp[0][1][k][1]<<'\n';
        }
    }
}

bool flag=1;
void inline shuffle_weather()
{
    if(flag){
        flag=0;return ;
    }
    srand(time(0));
    int changetime=rand()%30;
    while(changetime--)
    {
        int i= rand()%deadline,j=rand()%deadline;
        int tmp=weather_list[i];
        weather_list[i]=weather_list[j];
        weather_list[j]=tmp;
    }
}

void inline init_for(int weather)
{
    shuffle_weather();
    //for(int i=0;i<deadline;++i,weather>>=1) weather_list[i]=weather_list_list[weather][i];

    for(int i=1;i<=deadline;++i)
        for(int j=1;j<=num_of_points;++j)
            for(int k=0;k<=weight_limit/water_weight;++k)
                for (int l =0;k*3+l*2<=weight_limit;++l){
                    dp[i][j][k][1]=-INF;
                }
}

```



```

inline void buy(int i,int j,int k,int l)
{
    int max_food=weight_limit/food_weight;
    for(int m=k;m<=weight_limit/water_weight;++m){
        for(int n=l;m*3+n*2<=weight_limit && n<max_food;++n)
        {
            if(m==k &&n==l)
                continue;
            int money_left= dp[i][j][k][l] - (m-k)*water_price*2 - (n-l)*food_price*2;
            if(money_left<=dp[i][j][m][n] || money_left<0 ) // |\
                三角形区域的右上部分三角无需再搜索了，其他还是要更新的
            {
                max_food=n;
                break;
            }
            dp[i][j][m][n]=money_left;
            dp_state[i][j][m][n]=state(i,j,k,l);
        }
    }
}

int main()
{
    freopen("4_simple_result_weatherRandom.txt","w",stdout);
    ///!!修改4: 输出文件名

    init();

    for(int w=0;w< 100;++w)
    {
        init_for(w);
        for(int i=0;i<deadline;++i){
            int ii=i+1;
            //cerr<<"i="<<i<<'\n';
            for(int j=1;j<=num_of_points;++j){
                if(village_list[j]){ //每天开始更新前一天前，村庄先购物更新
                    for(int k=0;k<=weight_limit/water_weight;++k)
                        for (int l =0; k*3+l*2<=weight_limit;++l)
                            if(dp[i][j][k][l]>=0){
                                buy(i,j,k,l);
                                //cout<<"更新村庄! "<<endl;
                            }
                }
            }

            //更新停留
            int

```

```

        water_consumption=water_consumption_list[weather_list[i]],food_consumption=food_consumption_list[i]
    for(int k=water_consumption;k<=weight_limit/water_weight;++k){
        for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
            int kk=k-water_consumption,ll=l-food_consumption;

            if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][j][kk][ll])
            {
                dp[ii][j][kk][ll]=dp[i][j][k][l];
                dp_state[ii][j][kk][ll]=state(i,j,k,l);

                //cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\n';
            }
        }
    }

    //更新挖矿
    if (mine_list[j]){
        water_consumption=water_consumption_list[weather_list[i]]*3,food_consumption=food_consumption_list[i]
        for(int k=water_consumption;k<=weight_limit/water_weight;++k){
            for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                int kk=k-water_consumption,ll=l-food_consumption;

                if(dp[i][j][k][l]>=0 && dp[i][j][k][l]+base_income>dp[ii][j][kk][ll])
                {
                    dp[ii][j][kk][ll]=dp[i][j][k][l]+base_income;
                    dp_state[ii][j][kk][ll]=state(i,j,k,l);
                }
            }
        }
    }

    /*for(int k=0;k<=400;++k)
        for (int l =0;k*3+l*2<=1200;++l)
            if(dp[i][j][k][l]>9900)
                cout<<i<<' '<<j<<" "<<k<<" "<<l<<" "<<dp[i][j][k][l]<<'\n';//*/

    //非沙暴，更新走动
    if (weather_list[i]<2){
        water_consumption=water_consumption_list[weather_list[i]]*2,food_consumption=food_consumption_list[i]
        for(vector<short>::iterator
            iter=world_map[j].begin();iter!=world_map[j].end();++iter){
            for(int k=water_consumption;k<=weight_limit/water_weight;++k){
                for (int l =food_consumption;k*3+l*2<=weight_limit;++l){
                    int kk=k-water_consumption,ll=l-food_consumption;

                    if(dp[i][j][k][l]>=0 && dp[i][j][k][l]>dp[ii][*iter][kk][ll])

```

```

        {
            dp[ii][*iter][kk][ll]=dp[i][j][k][l];
            dp_state[ii][*iter][kk][ll]=state(i,j,k,l);
        }
    }
}
}
}
}

}

int target=num_of_points;
int res=-INF,arrive_day;
state last_state;
for(int i=0;i<=deadline;++i){
    for(int k=0;k<=weight_limit/water_weight;++k){
        for (int l =0;k*3+l*2<=weight_limit;++l){
            if(dp[i][target][k][l]>res)
            {
                res=dp[i][target][k][l];
                arrive_day=i;
                last_state=state(i,target,k,l);
            }
        }
    }
}
if(res>0){
    cerr<<w<<" "<<res<<endl;

    stack<state> sta;
    while(last_state!=state(0,0,0,0)){
        sta.push(last_state);
        last_state=dp_state[last_state.i][last_state.j][last_state.k][last_state.l];
    }
    cout<<"weather:";
    for(int q=0;q<deadline;++q)
        cout<<weather_list[q];
    cout<<"\n";

    cout <<arrive_day<< "days,score:" <<res<< "\nsolution:\n";

    while(!sta.empty())
    {
        last_state=sta.top();sta.pop();
        cout<<int(last_state.i)<<" "<<mapping[last_state.j]<<" "<<int(last_state.k)<<"
            "<<int(last_state.l)<<"

```

```

        "<<dp[last_state.i][last_state.j][last_state.k][last_state.l]<<endl;
    }
    cout<<"\n\n\n";
}
else cout<<"no solution \n\n\n";
}
return 0;
}

```

C 第三关随机与统计作图模拟 python 代码

simulation.py

```

from random import *
from matplotlib import pyplot as plt
import numpy as np
water_consumption_list=[3,9,10]; #消耗列表
food_consumption_list=[4,9,10];
water_weight=3; #价格与重量
water_price=5;
food_weight=2;
food_price=10;

current_food=0;
current_water=0;
money=10000-current_food*food_price-current_water*water_price;

bias=-5;
def buyfood(weather):

    if weather:
        water=54;
        food=54;
    else:
        water=42;
        food=44;
    water-=bias;food-=bias;
    return water,food;

def judge(weather,current_water,current_food,position):
    choice= randint(1,100)<=possibility;
    if weather:
        return choice;
    return 1;

```

```

income_list=[];

loserate_list=[];
possibility=80;

for possibility in range(101):
    print(possibility);
    losetimes=0;
    wintimes=0;
    totalincome=0;
    for i in range(10000):
        weather1=randint(0,1);
        current_water,current_food= buyfood(weather1);
        money=10000-current_food*food_price-current_water*water_price;

        position=0;
        for day in range(1,11):
            weather=randint(0,1)
            if day==1:
                weather=weather1;

            go = judge(weather,current_water,current_food,position)
            position+=go;
            current_water-= water_consumption_list[weather]*(go+1);
            current_food-= food_consumption_list[weather]*(go+1);

            #print (position)

            if current_water<0 or current_food<0:
                losetimes+=1;
                break;
            if position==3:
                wintimes+=1;
                totalincome+= money +(current_food*food_price+current_water*water_price)/2;
                break;
            if day==10:
                losetimes+=1;
                break;
    loserate_list.append(losetimes/100);
    income_list.append(totalincome/wintimes);

exp_list=[]
for i in range(101):
    exp_list.append( income_list[i]* (1-loserate_list[i]/100));

```

```

fig = plt.figure()
x=range(101);

ax1 = fig.add_subplot(111)
ax1.plot(x, loserate_list )

ax1.set_ylim([0, 100]);
ax1.set_ylabel('Lose Rate');

ax1.set_xlabel('Walking Rate');

plt.legend(["Lose Rate"])

#ax1.set_title("Income");

ax2 = ax1.twinx() # this is the important function
ax2.plot(x, income_list, 'r')
ax2.plot(x, exp_list, 'g')

ax2.set_xlim([0, 100])
ax2.set_ylim([0, 10000])

ax2.set_ylabel('Average Income',labelpad=-30);
ax2.set_ylabel('Income')
ax2.set_xlabel('Walking Rate');

plt.text(103, income_list[100], '%.0f' %(income_list[100]) , ha='center', va=
        'bottom',fontSize=9)

plt.legend(["AverageIncome","Income"])

plt.savefig("3bias%.0f.png" %bias);

plt.show()

```

D 游戏逻辑的 python 模拟代码

desert_game.py

```

from strategy import *;
from sys import *

```

```

world_map={1: [1,2,25],2: [2,1,3],3: [3,2,25,4],4: [4,3,25,24,5],5: [5,4,24,6],6: [6,5,24,23,7],7: [7,6,8,22],8: [8,7,9
degree={1:2,2:2,3:3,4:4,5:3,6:4,7:3,8:3}; #数一下每个点度数, 校验地图正确性

start_point=1; end_point=27; #起点终点
village_list=[15];          # 村庄表
mine_list=[12];             # 矿点表

weather_list=[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]; #0晴朗, 1高温, 2沙暴
deadline=30;                 #期限
initial_money=10000; #初始金钱
base_income=1000;   #基本收益
weight_limit=1200;   #负重上限
water_consumption_list=[5,8,10]; #消耗列表
food_consumption_list=[7,6,10];
water_weight=3;       #价格与重量
water_price=5;
food_weight=2;
food_price=10;

money_earn_sum=0;
water_consumption_sum=0;
food_consumption_sum=0;

#print(world_map);
def check_map(world_map):
    for i in world_map.keys():
        for j in world_map[i]:
            if not i in world_map[j]:
                print(i,j,' 地图错误,双向不匹配');
            if i in degree.keys() and len(world_map[i])!=degree[i]+1:
                print(i,j,' 地图错误, 度数不匹配');
    if not (start_point in world_map.keys() and end_point in world_map.keys()):
        print('起点/终点不在地图内');
    for i in village_list:
        if not i in world_map.keys():
            print('村庄不在地图内');
    for i in mine_list:
        if not i in world_map.keys():
            print('矿点不在地图内');
    if not len(weather_list)==deadline:
        print('天数与天气列表不匹配');
    print('地图校验通过');
def display_state():
    print('第',current_date+1,"天, 在",current_position,"点")
    print('水: ',current_water, "食物: ",current_food,"钱: ",current_money);

```

```

def display_setting():
    print('水价格:',water_price,"食物价格: ",food_price);
    print('水消耗',water_consumption_list, ' 食物消耗: ',food_consumption_list);
def game_over():
    print('第',current_date+1,'天,你失败了');
    print('共计挖矿',money_earn_sum,'共计消耗水:',water_consumption_sum,'共计消耗食物:',food_consumption_sum)
    exit(0);
def get_weather(date):
    return weather_list[date];

def buy_water(water_num,food_num,multiple=1):
    global
        start_point,end_point,deadline,initial_money,base_income,weight_limit,food_weight,food_price;
    global water_weight,water_price,current_money,current_water,current_food,current_date;
    spend_money= multiple*(water_num*water_price+food_num*food_price);
    print('购买水: ',water_num,'购买食物: ',food_num,'花钱',spend_money);
    print('重量: ', (water_num+current_water)*water_weight+(food_num+current_food)*food_weight)
    if spend_money > current_money:
        print('金额不足');game_over();
    if (water_num+current_water)*water_weight+(food_num+current_food)*food_weight>weight_limit:
        print('负重过大');game_over();
    current_money-=spend_money;
    current_water+=water_num;
    current_food+=food_num;

def consume(multiple=1):
    global
        start_point,end_point,deadline,initial_money,base_income,weight_limit,food_weight,food_price;
    global
        water_weight,water_price,current_money,current_water,current_food,current_date,current_position;
    global money_earn_sum,water_consumption_sum,food_consumption_sum

    consume_water = water_consumption_list[get_weather(current_date)]*multiple;
    consume_food= food_consumption_list[get_weather(current_date)]*multiple;
    if consume_water>current_water or consume_food >current_food:
        print('食物/水消耗殆尽'); game_over();
    current_water-= consume_water;current_food-=consume_food;
    water_consumption_sum+=consume_water;
    food_consumption_sum +=consume_food;

def compute_consumption(begin_date, daynum, multiple_list):
    if daynum!=len(multiple_list):
        print('时间长度与倍数列表不匹配');
    water_sum=0;
    food_sum=0;
    for i in range(daynum):

```



```

        water_sum+= multiple_list[i]* water_consumption_list[get_weather(begin_date+i)];
        food_sum += multiple_list[i]* food_consumption_list[get_weather(begin_date+i)];
    return water_sum,food_sum;

def main_game():
    global
        start_point,end_point,deadline,initial_money,base_income,weight_limit,food_weight,food_price;
    global
        water_weight,water_price,current_money,current_water,current_food,current_date,current_position;
    global money_earn_sum,water_consumption_sum,food_consumption_sum
    print('游戏开始! 游戏设定为: ');display_setting();
    print('目前状态为: '); display_state();

    strategy= get_strategy
        (current_money,current_water,current_food,current_position,current_date+1,world_map,start_point,
        end_point,village_list,mine_list,weather_list,deadline,initial_money,base_income,weight_limit,food_consumption_sum)
    #约定: 只能走之前买, 即不允许当天食物变成负数
    print(strategy)

    buy_water(strategy[1],strategy[2]);
    while current_date<deadline and current_position!=end_point:
        display_state();
        if(get_weather(current_date)==2 and strategy[0]!=current_position):
            print('沙暴天气不能移动! ');game_over();
        if not strategy[0] in world_map[current_position]:
            print('两个地区不相邻! ');game_over();

        if current_position in village_list:
            buy_water(strategy[1],strategy[2],2);
            consume( (strategy[0]!=current_position) +1);
        elif current_position in mine_list and strategy[0]==current_position and strategy[1]:
            consume(3);
            current_money +=base_income;
            money_earn_sum+=base_income;
        else:
            consume( (strategy[0]!=current_position) +1);
        current_position=strategy[0];
        current_date+=1;
        strategy=get_strategy(current_money,current_water,current_food,current_position,current_date+1,world_map,
            end_point,village_list,mine_list,weather_list,deadline,initial_money,base_income,weight_limit,food_consumption_sum)

    if current_position==end_point:
        print('成功过关');
        display_state();
        print('共计挖矿',money_earn_sum,'共计消耗水:',water_consumption_sum,'共计消耗食物:',food_consumption_sum)

```

```

world_map,start_point,
    end_point,village_list,mine_list,weather_list,deadline,initial_money,base_income,weight_limit,food_consumpt
check_map(world_map);
current_money= initial_money;current_water=0;current_food=0;current_position=
    start_point;current_date=0;
main_game();

```

strategy.py

```

def change_settings():
    world_map={1:[1,2,25],2:[2,1,3],3:[3,2,25,4],4:[4,3,25,24,5],5:[5,4,24,6],6:[6,5,24,23,7],7:[7,6,8,22],8:[8,
    degree={1:2,2:2,3:3,4:4,5:3,6:4,7:3,8:3}; #数一下每个点度数，校验地图正确性

    start_point=1; end_point=27; #起点终点
    village_list=[15]; # 村庄表
    mine_list=[12]; # 矿点表

    weather_list=[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];
    #0晴朗，1高温，2沙暴
    deadline=30; #期限
    initial_money=10000; #初始金钱
    base_income=1000; #基本收益
    weight_limit=1200 #负重上限
    water_consumption_list=[5,8,10]; #消耗列表
    food_consumption_list=[7,6,10];
    water_weight=3; #价格与重量
    water_price=5;
    food_weight=2;
    food_price=10;
    return [world_map,start_point,
        end_point,village_list,mine_list,weather_list,deadline,initial_money,base_income,weight_limit,food_consumpt

def get_strategy
    (current_money,current_water,current_food,current_position,current_date,world_map,start_point,
    end_point,village_list,mine_list,weather_list,deadline,initial_money,base_income,weight_limit,food_consumpt
    #print('这是策略')
    '''1-10天去矿区，中途补163水，11-17挖矿7天,18休息，之后5天去终点，中途路过村庄补充一下 '''
    # 村庄或者起点[目的地，水（几箱），食物
    if current_date==1:
        return [25,178,333];
    if current_date==2:
        return [24];
    if current_date==3:
        return [23];
    if current_date==4:
        return [23];

```

```
if current_date==5:
    return [22]

if current_date==6:
    return [9];
if current_date==7:
    return [9];
if current_date==8:
    return [15];
if current_date==9:
    return [13,163,0];
if current_date==10:
    return [12];
if current_date==11:
    return [12,1];
if current_date==12:
    return [12,1];
if current_date==13:
    return [12,1];
if current_date==14:
    return [12,1];
if current_date==15:
    return [12,1];
if current_date==16:
    return [12,1];
if current_date==17:
    return [12,0];
if current_date==18:
    return [12,0];

if current_date==19:
    return [12,1];
if current_date==20:
    return [14];
if current_date==21:
    return [15];
if current_date==22:
    return [9,36,16];
if current_date==23:
    return [21];
if current_date==24:
    return [27];

return [27]
```

desert_game2.py

```
from strategy2 import *;
from sys import *
import numpy as np

world_map={1:[15,27],15:[12,27],12:[15] }
distance_matrix= np.zeros((30,30), dtype=np.int);
distance_matrix[1][15]=6;distance_matrix[1][27]=3;distance_matrix[15][12]=2;distance_matrix[15][27]=3;distance_

start_point=1; end_point=27; #起点终点
village_list=[15];          # 村庄表
mine_list=[12];             # 矿点表

weather_list=[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]; #0晴朗, 1高温, 2沙暴
deadline=30;                 #期限
initial_money=10000; #初始金钱
base_income=1000;   #基本收益
weight_limit=1200;   #负重上限
water_consumption_list=[5,8,10]; #消耗列表
food_consumption_list=[7,6,10];
water_weight=3;        #价格与重量
water_price=5;
food_weight=2;
food_price=10;

def display_state(state):
    print('第',state['current_date'], "天结束, 在",state['current_position'], "点")
    print('水: ',state['current_water'],
          "食物: ",state['current_food'], "钱: ",state['current_money']);
def display_setting():
    print('水价格:',water_price,"食物价格: ",food_price);
    print('水消耗',water_consumption_list, ' 食物消耗: ',food_consumption_list);

def game_over(state):
    print('第',state['current_date']+1, '天,你失败了');
    print('共计挖矿',state['money_earn_sum'],'共计消耗水:',state['water_consumption_sum'],'共计消耗食物:',state['
    exit(0);

def get_weather(date):
    return weather_list[date];

def buy(state, water_num,food_num,multiple=1):
    global
        start_point,end_point,deadline,initial_money,base_income,weight_limit,food_weight,food_price,water_weight,
```

```

spend_money= multiple*(water_num*water_price+food_num*food_price);
print('购买水: ',water_num,'购买食物: ',food_num,'花钱',spend_money);
print('重量: ',
      (water_num+state['current_water'])*water_weight+(food_num+state['current_food'])*food_weight)
if spend_money > state['current_money']:
    print('金额不足');game_over(state);
if
    (water_num+state['current_water'])*water_weight+(food_num+state['current_food'])*food_weight>weight_lim
    print('负重过大');game_over(state);
state['current_money']-=spend_money;
state['current_water']+=water_num;
state['current_food'] +=food_num;
return state;

def consume(state,multiple=1): #state进来应当是引用?
    global
        start_point,end_point,deadline,initial_money,base_income,weight_limit,food_weight,food_price,water_weight

    consume_water = water_consumption_list[get_weather(state['current_date'])]*multiple;
    consume_food= food_consumption_list[get_weather(state['current_date'])]*multiple;
    if consume_water>state['current_water'] or consume_food >state['current_food']:
        print('食物/水消耗殆尽'); game_over(state);
    state['current_water']-= consume_water; state['current_food']-=consume_food;
    state['water_consumption_sum']+=consume_water;
    state['food_consumption_sum'] +=consume_food;

def update (state,next_position, mining=0 ):
    global
        start_point,end_point,deadline,initial_money,base_income,weight_limit,food_weight,food_price,water_weight
    if state['current_position']==next_position: #停留或挖矿: 消耗食物、增加金钱、增加天数
        if next_position in mine_list and mining:
            consume(state,3);
            state['current_money']+=base_income;
            state['money_earn_sum']+=base_income;
        else:
            consume(state,1);
            state['current_date']+=1;
    else: # 行进: 消耗食物, 改变位置, 增加天数
        distance=distance_matrix[state['current_position']][next_position];
        while distance:
            if get_weather(state['current_date'])<2: #非沙暴就往前走
                consume(state,2);
                distance-=1;

```

```

        else:
            consume(state,1);
            state['current_date']+=1;
            display_state(state);

            state['current_position']=next_position;
        return state;

def main_game(state):
# BFS外壳，以字典state表示当前状态作为队列中元素，每轮操作具体如下

# 1. 获取用户策略strategy
# 2. 判断命令合法性

# 3. 计算购买行为

# 4 -1 计算挖矿行为
# -2 计算停留行为
# -3 计算运动行为 。

    global
        start_point,end_point,deadline,initial_money,base_income,weight_limit,food_weight,food_price,water_weight

    print('游戏开始! 游戏设定为: ');display_setting();
    print('目前状态为: '); display_state(state);

    #约定：只能走之前买，即不允许当天食物变成负数
    q=[state];

    while len(q):
        state=q.pop(0);
        if state['current_date']>=deadline or state['current_position']==end_point:
            q=[state];
            break; #到达终点
        display_state(state);
        strategy= get_strategy
            (state['current_money'],state['current_water'],state['current_food'],state['current_position'],state
            end_point,village_list,mine_list,weather_list,deadline,initial_money,base_income,weight_limit,food_c
        if(get_weather(state['current_date'])==2 and strategy[0]!=state['current_position']):
            print('沙暴天气不能移动! ');game_over(state);

        print(state['current_position']);
        if not strategy[0] in world_map[state['current_position']] and not
            strategy[0]==state['current_position']:
            print('两个地区不相邻! ');game_over(state);

        if state['current_position'] == start_point: #图设计不回起点，因此不用担心二次购买
            state=buy(state.copy(),strategy[1],strategy[2],1);

```

```

        elif state['current_position'] in village_list:
            state=buy(state.copy(),strategy[1],strategy[2],2);

            state=update(state.copy(),strategy[0],len(strategy)==2 and strategy[1]); #
                更新时间的3种行为：停留、挖矿、行走的数据更新在此处计算
            q.append(state);
            state=q[0];
            if state['current_position']==end_point:
                print('成功过关');
                display_state(q[0]);
                print('共计挖矿',state['money_earn_sum'],'共计消耗水:',state['water_consumption_sum'],'共计消耗食物:',state['food_consumption_sum']);

world_map,start_point,
    end_point,village_list,mine_list,weather_list,deadline,initial_money,base_income,weight_limit,food_consumption_list)=change_settings()

state={'current_money':initial_money,'current_water':0,'current_food':0,'current_position':start_point,'current_date':0,
'water_consumption_sum':0,'food_consumption_sum':0}
# 8个状态参数： 钱、水、食物、位置、天数、 总赚钱、总耗水、总耗食
main_game(state);

```

strategy2.py

```

def change_settings():

    world_map={1:[15,27],15:[12,27],12:[15] }
    start_point=1; end_point=27; #起点终点
    village_list=[15];          # 村庄表
    mine_list=[12];             # 矿点表

    weather_list=[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];
        #0晴朗, 1高温, 2沙暴
    deadline=30;                 #期限
    initial_money=10000; #初始金钱
    base_income=1000;   #基本收益
    weight_limit=1200    #负重上限
    water_consumption_list=[5,8,10]; #消耗列表
    food_consumption_list=[7,6,10];
    water_weight=3;        #价格与重量
    water_price=5;
    food_weight=2;
    food_price=10;
    return [world_map,start_point,
            end_point,village_list,mine_list,weather_list,deadline,initial_money,base_income,weight_limit,food_consumption_list]

def get_strategy
    (current_money,current_water,current_food,current_position,current_date,world_map,start_point,

```

```

end_point,village_list,mine_list,weather_list,deadline,initial_money,base_income,weight_limit,food_consumpt
#print('这是策略')
'''1-10天去矿区，中途补163水，11-17挖矿7天,18休息，之后5天去终点，中途路过村庄补充一下'''
# 村庄或者起点[目的地，水（几箱），食物]
if current_date==1:
    return [15,98,452];
if current_date==9:
    return [12,163,0];

if current_date==11:
    return [12,1];
if current_date==12:
    return [12,1];
if current_date==13:
    return [12,1];
if current_date==14:
    return [12,1];
if current_date==15:
    return [15];
if current_date==16:
    return [12,1];
if current_date==17:
    return [15,213,0];
if current_date==18:
    return [15,0,0];
if current_date==19:
    return [12,0,0];
if current_date==21:
    return [12,1];
if current_date==22:
    return [12,1];
if current_date==23:
    return [12,1];
if current_date==24:
    return [12,1];
if current_date==25:
    return [12,1];
if current_date==26:
    return [15];
if current_date==28:
    return [27,0,0];

```

desert_game3.py

```

from strategy3 import *;
from sys import *

```



```

import numpy as np

world_map = {1: [1, 2, 25], 2: [2, 1, 3], 3: [3, 2, 25, 4], 4: [4, 3, 25, 24, 5], 5: [5, 4,
    24, 6],
    6: [6, 5, 24, 23, 7], 7: [7, 6, 8, 22], 8: [8, 7, 9, 22], 9: [9, 8, 22, 21, 17, 16,
    15, 10],
    10: [10, 9, 15, 11, 13], 11: [11, 10, 12, 13], 12: [12, 11, 13, 14], 13: [13, 10,
    11, 12, 14, 15],
    14: [14, 12, 13, 15, 16], 15: [15, 9, 10, 13, 14, 16], 16: [16, 14, 15, 9, 17, 18],
    17: [17, 9, 21, 18, 16], 18: [18, 17, 16, 19, 20, 21], 19: [19, 18, 20], 20: [20,
    18, 19, 21],
    21: [21, 17, 18, 20, 9, 22, 23, 27], 22: [22, 7, 8, 9, 21, 23], 23: [23, 6, 22, 21,
    24, 26],
    24: [24, 4, 5, 6, 23, 25, 26], 25: [25, 1, 3, 4, 24, 26], 26: [26, 25, 24, 23, 27],
    27: [27, 21, 26]};

degree = {1: 2, 2: 2, 3: 3, 4: 4, 5: 3, 6: 4, 7: 3, 8: 3}; # 数一下每个点度数, 校验地图正确性

start_point = 1;
end_point = 27; # 起点终点
village_list = [15]; # 村庄表
mine_list = [12]; # 矿点表

deadline = 30; # 期限
initial_money = 10000; # 初始金钱
base_income = 1000; # 基本收益
weight_limit = 1200; # 负重上限
water_consumption_list = [5, 8, 10]; # 消耗列表
food_consumption_list = [7, 6, 10];
water_weight = 3; # 价格与重量
water_price = 5;
food_weight = 2;
food_price = 10;

money_earn_sum = 0;
water_consumption_sum = 0;
food_consumption_sum = 0;

# print(world_map);
def check_map(world_map):
    for i in world_map.keys():
        for j in world_map[i]:
            if not i in world_map[j]:
                pass
            if i in degree.keys() and len(world_map[i]) != degree[i] + 1:
                pass

```

```

if not (start_point in world_map.keys() and end_point in world_map.keys()):
    print('起点/终点不在地图内');
for i in village_list:
    if not i in world_map.keys():
        print('村庄不在地图内');
for i in mine_list:
    if not i in world_map.keys():
        print('矿点不在地图内');

def display_state():
    global weather_list
    weather='高温' if weather_list[current_date]==1 else '晴朗'
    print('第', current_date + 1, f"天, {weather}.在", current_position, "点")
    print('水: ', current_water, "食物: ", current_food, "钱: ", current_money);

def display_setting():
    print('水价格:', water_price, "食物价格: ", food_price);
    print('水消耗', water_consumption_list, ' 食物消耗: ', food_consumption_list);

def game_over():
    pass
    #print('第', current_date + 1, '天,你失败了');

    #print('共计挖矿', money_earn_sum, '共计消耗水:', water_consumption_sum, '共计消耗食物:',
        food_consumption_sum)

def buy_water(water_num, food_num, multiple=1):
    global start_point, end_point, deadline, initial_money, base_income, weight_limit,
        food_weight, food_price;
    global water_weight, water_price, current_money, current_water, current_food, current_date;
    spend_money = multiple * (water_num * water_price + food_num * food_price);
    #print('购买水: ', water_num, '购买食物: ', food_num, '花钱', spend_money);
    #print('重量: ', (water_num + current_water) * water_weight + (food_num + current_food) *
        food_weight)

    if spend_money > current_money:
        print('金额不足');
        game_over();

```

```

if (water_num + current_water) * water_weight + (food_num + current_food) * food_weight >
    weight_limit:
    print('负重过大');
    game_over();

current_money -= spend_money;
current_water += water_num;
current_food += food_num;

def consume(multiple=1):
    global start_point, end_point, deadline, initial_money, base_income, weight_limit,
        food_weight, food_price;
    global water_weight, water_price, current_money, current_water, current_food, current_date,
        current_position;
    global money_earn_sum, water_consumption_sum, food_consumption_sum
    global weather_list
    weather=weather_list[current_date]
    consume_water = water_consumption_list[weather] * multiple;
    consume_food = food_consumption_list[weather] * multiple;
    if consume_water > current_water or consume_food > current_food:
        #print('食物/水消耗殆尽');
        game_over();
        return 0 # 0 表示游戏结束
    current_water -= consume_water;
    current_food -= consume_food;

    water_consumption_sum += consume_water;
    food_consumption_sum += consume_food;
    return 1

def compute_consumption(begin_date, daynum, multiple_list):
    if daynum != len(multiple_list):
        print('时间长度与倍数列表不匹配');
    water_sum = 0;
    food_sum = 0;
    for i in range(daynum):
        water_sum += multiple_list[i] * water_consumption_list[get_weather(begin_date + i)];
        food_sum += multiple_list[i] * food_consumption_list[get_weather(begin_date + i)];
    return water_sum, food_sum;

def sell(food,water):
    global water_price,food_price
    return food*0.5*food_price+water*0.5*water_price

def main_game():

```

```

global start_point, end_point, deadline, initial_money, base_income, weight_limit,
    food_weight, food_price;
global water_weight, water_price, current_money, current_water, current_food, current_date,
    current_position;
global money_earn_sum, water_consumption_sum, food_consumption_sum
global weather_list
global water_to_buy, food_to_buy

strategy = get_strategy(current_money, current_water, current_food, current_position,
    current_date + 1, world_map,
        start_point, end_point, village_list, mine_list, weather_list,
        deadline, initial_money,
        base_income, weight_limit, food_consumption_list,
        water_consumption_list, food_weight,
        food_price, water_weight, water_price, water_to_buy, food_to_buy);
# 约定：只能走之前买，即不允许当天食物变成负数

buy_water(strategy[1], strategy[2]);
while current_date < deadline and current_position != end_point:
    #display_state();

    if (weather_list[current_date-1] == 2 and strategy[0] != current_position):
        print('沙暴天气不能移动! ');
        game_over();
        exit()

    if not strategy[0] in world_map[current_position]:
        print('两个地区不相邻! ');
        print(f'{current_position},{strategy[0]},{world_map[current_position]}')
        game_over();
        exit()

    if current_position in village_list:
        buy_water(strategy[1], strategy[2], 2);
        flag=consume((strategy[0] != current_position) + 1);
        if flag==0:
            return -1 # -1表示死亡
    elif current_position in mine_list and strategy[0] == current_position and strategy[1]:
        flag = consume(3);
        if flag==0:
            return -1
        current_money += base_income;
        money_earn_sum += base_income;

    else:
        flag=consume((strategy[0] != current_position) + 1);
        if flag==0:
            return -1
    current_position = strategy[0];

```

```

        current_date += 1;
        strategy = get_strategy(current_money, current_water, current_food, current_position,
                                current_date + 1,
                                world_map, start_point, end_point, village_list, mine_list,
                                weather_list, deadline,
                                initial_money, base_income, weight_limit, food_consumption_list,
                                water_consumption_list,
                                food_weight, food_price, water_weight, water_price);

    if current_position == end_point:

        sell_sum=sell(current_food,current_water)
        current_money+=sell_sum
        current_water,current_food=0,0

        return current_money
    return -2 # 表示没有到达终点 (但是没有死亡)

def generate_weather(hot_posibility=0.5):
    global weather_list
    weather_list=[]
    for i in range(10):
        r=1 if np.random.rand()<hot_posibility else 0
        weather_list.append(r)
    return weather_list

def adjust_water_and_food(water,food):
    global water_to_buy,food_to_buy
    water_to_buy=water
    food_to_buy=food

world_map, start_point, end_point, village_list, mine_list, deadline, initial_money,
    base_income, weight_limit, food_consumption_list, water_consumption_list, food_weight,
    food_price, water_weight, water_price = change_settings();
num_of_sim=100

weather_list=[]
water_to_buy=0
food_to_buy=0

num_of_hot=[3,4,5]
num_of_sunny=[3]
for n_hot in num_of_hot:
    for n_sun in num_of_sunny:
        death_cnt = 0

```

```

left_sum = 0
cnt = 0
for i in range(num_of_sim):
    water=n_hot*9+3*2*3+(n_sun-3)*3
    food=n_hot*9+4*2*3+(n_sun-3)*4
    adjust_water_and_food(water,food)

    current_money = initial_money;
    current_water = 0;
    current_food = 0;
    current_position = start_point;
    current_date = 0;

    li = generate_weather()
    np_li = np.array(li)

    # w,f=compute_consumption(9, 2+7+1+2,[2,2]+[3]*7+[1,2,2]);
    money_left = main_game();
    if money_left < 0:
        death_cnt += 1
    else:
        left_sum += money_left
        cnt += 1

print(f"death time:{death_cnt}, average money left:{left_sum / cnt} h_hot={n_hot}
      n_sun={n_sun}")

```

strategy3.py

```

import numpy as np

def get_weather(hot_possibility=0.5):
    r=np.random.rand()
    if r<hot_possibility:
        return 1 # 1代表高温
    else:
        return 0

def change_settings():
    world_map = {1: [1,2,4,5], 2: [2,1,3,4], 3: [3,2,4,8,9], 4: [4,1,2,3,5,6,7], 5: [5,1,4,7,6],
                  6: [6,5, 4, 7, 12, 13], 7: [7,4, 6, 12, 11], 8: [8,3,9], 9: [9,3,8,11,10],
                  10: [10,9,11,13], 11: [11,9,10,13,12,7], 12: [12,7, 11, 13, 6], 13:
                  [13,10,11,12,6],
                  };
    degree = {1: 2, 2: 2, 3: 3, 4: 4, 5: 3, 6: 4, 7: 3, 8: 3}; # 数一下每个点度数, 校验地图正确性

```

```

start_point = 1;
end_point = 13; # 起点终点
village_list = []; # 村庄表
mine_list = [9]; # 矿点表


deadline = 10; # 期限
initial_money = 10000; # 初始金钱
base_income = 200; # 基本收益
weight_limit = 1200; # 负重上限
water_consumption_list = [3, 9, 10]; # 消耗列表
food_consumption_list = [4, 9, 10];
water_weight = 3; # 价格与重量
water_price = 5;
food_weight = 2;
food_price = 10;


return [world_map, start_point, end_point, village_list, mine_list, deadline, initial_money,
        base_income, weight_limit, food_consumption_list, water_consumption_list,
        food_weight, food_price,
        water_weight, water_price];


def get_strategy(current_money, current_water, current_food, current_position, current_date,
                world_map, start_point,
                end_point, village_list, mine_list, weather_list, deadline, initial_money,
                base_income, weight_limit,
                food_consumption_list, water_consumption_list, food_weight, food_price,
                water_weight, water_price
                ,buy_water=54,buy_food=60):
    # print('这是策略')
    # 村庄或者起点[目的地, 水(几箱), 食物]
    # current_position, current_weather, current_date
    current_weather=weather_list[current_date-1]
    if current_weather==0:
        if current_position==1:
            return [5,buy_water,buy_food] # 4高温, 3晴天
        elif current_position==5:
            return [6]
        elif current_position==6:
            return [13]
    elif current_weather==1:
        if current_position==1:
            if current_date>1:

```

```
        return [1]
    else:
        return [1, buy_water, buy_food]
if current_position==5:
    if deadline-current_date+1==2:
        return [6]
    else:
        return [5]
if current_position==6:
    if deadline-current_date+1==1:
        return [13]
    else:
        return [6]
return []
```