赛区评阅编号（由赛区组委会填写）：

2020 高教社杯全国大学生数学建模竞赛

承　诺　书

我们仔细阅读了《全国大学生数学建模竞赛章程》和《全国大学生数学建模竞赛参赛规则》（2019 年修订稿，以下简称为“竞赛章程和参赛规则”，可从全国大学生数学建模竞赛网站下载）。

我们完全清楚，在竞赛开始后参赛队员不能以任何方式，包括电话、电子邮件、“贴吧”、QQ 群、微信群等，与队外的任何人（包括指导教师）交流、讨论与赛题有关的问题；无论主动参与讨论还是被动接收讨论信息都是严重违反竞赛纪律的行为。

我们完全清楚，抄袭别人的成果是违反竞赛章程和参赛规则的行为；如果引用别人的成果或资料（包括网上资料），必须按照规定的参考文献的表述方式列出，并在正文引用处予以标注。

我们以中国大学生名誉和诚信郑重承诺，严格遵守竞赛章程和参赛规则，以保证竞赛的公正、公平性。如有违反竞赛章程和参赛规则的行为，我们将受到严肃处理。

我们授权全国大学生数学建模竞赛组委会，可将我们的论文以任何形式进行公开展示（包括进行网上公示，在书籍、期刊和其他媒体进行正式或非正式发表等）。

我们参赛选择的题号（从 A/B/C/D/E 中选择一项填写）：B

我们的报名参赛队号（12 位数字全国统一编号）：

参赛学校（完整的学校全称，不含院系名）：南京大学

参赛队员 (打印并签名) ：1. 曾嘉馨 171840063 数学系

2. 徐佳美 181860117 计算机科学与技术系

3. 朱倩 181860154 计算机科学与技术系

指导教师或指导教师组负责人 (打印并签名)：

（指导教师签名意味着对参赛队的行为和论文的真实性负责）

日期: 2020 年 09 月 13 日

（请勿改动此页内容和格式。此承诺书打印签名后作为纸质论文的封面，注意电子版论文中不得出现此页。以上内容请仔细核对，如填写错误，论文可能被取消评奖资格。）

赛区评阅编号（由赛区组委会填写）：

2020 高教社杯全国大学生数学建模竞赛

编 号 专 用 页

赛区评阅记录（可供赛区评阅时使用）：

评阅人						
备注						

送全国评阅统一编号（由赛区组委会填写）：

全国评阅随机编号（由全国组委会填写）：

（请勿改动此页内容和格式。此编号专用页仅供赛区和全国评阅使用，参赛队打印后装订到纸质论文的第二页上。注意电子版论文中不得出现此页。）

穿越沙漠游戏策略设计

摘要

在穿越沙漠游戏中，玩家要在规定时间内到达终点，并保留尽可能多的资金。本文在单个、多个玩家，已知所有天气、仅知当天天气几个条件两两组合的四种情境下分别探讨最佳游戏策略。

对于问题一：采用**动态规划**，初始状态为第 0 天在起点。通过状态转移方程，利用第 $i-1$ 天在 j 地及其邻居的最大剩余资金量求出第 i 天在 j 地的最大剩余资金量，进而求出截止日期当天在终点的最大剩余资金量，从而得到最优策略。将关卡一、二中的数据代入，求得关卡一最大剩余资金量为 12820 元，关卡二最大剩余资金量为 16410 元。

对于问题二：由于只知道当天天气不知道后续情况，因此玩家自己需要预判接下来的可能情况，但是预判的天数过少会导致只顾眼前利益，天数过多情况复杂变动也越大，综合考虑之后使用**决策树**来预判三天内的情况，同时考虑到玩家在过程中会积累一定经验值，需要在探索和经验中进行平衡，因此引进 **Q-learning** 模型，通过学习速率和折扣因子来进行平衡。

对于问题三：利用**博弈论**的思想，对参与游戏的玩家作出一些假设，并对特殊地点能够采取的各种行动计算收益，然后依据收益函数找到纳什均衡点。在此假设所有玩家都趋向于达成纳什均衡，以此作为玩家选择策略的依据。

问题三（1）：首先求出最优路径，然后按特殊地点的分布将路线分割为若干段，每一段都使用类似的方式求得局部最优解，从而得到总的最优解。对第五关，计算出两个玩家的最优选择都是通过最短路径直达终点，他们游戏结束时拥有的资金都为 9020 元。

问题三（2）：对每个特殊地点可能采取的行动都进行了评估，得出了每种策略的收益期望函数。与第一小问类似，通过收益函数求得纳什均衡点，从而确定行动方案。对第六关，通过评估和比较得出这样的策略：玩家将用最短路径赶往特殊地点，在途中会尽量和其他玩家错开；抵达村庄时，三个人都选择买完就走，不作停留；抵达矿山时，三个人都选择挖矿，直到物资告急不得不离开前往下一个地点为止。

关键字： 动态规划　决策树　Q-learning　博弈论

1 问题重述

1.1 问题背景

在穿越沙漠游戏中：玩家凭借一张地图，从起点出发，在沙漠中行走，途中会遇到不同的天气。可在起点用初始资金购买一定数量的资源（水和食物），也可在村庄购买，并可在矿山补充资金。目标是在规定时间内到达终点，并保留尽可能多的资金。

游戏的基本规则如下：

（1）游戏目标：在截止日期或之前到达终点，并保留尽可能多的资金。未到达终点前穿越沙漠所需的资源（水或食物）不能耗尽，每天玩家拥有的水和食物质量之和不能超过负重上限。

（2）移动策略：每天玩家可移动到与目前区域相邻的另一区域（有公共边界），也可在原地停留。玩家在原地停留一天消耗的资源数量称为基础消耗量，行走一天消耗的资源数量为基础消耗量的 2 倍。

（3）天气影响：每天的天气为“晴朗”、“高温”或“沙暴”，沙漠中所有区域的天气相同。不同天气玩家的资源基础消耗量不同，且沙暴日玩家必须在原地停留。

（4）资源获取：玩家可在起点处用初始资金以基准价格购买水和食物（只有第 0 天可以购买），也可在村庄用剩余的资金购买水和食物，每箱价格为基准价格的 2 倍。

（5）资金获取：玩家在矿山停留时，可通过挖矿获得资金，挖矿一天获得的资金量称为基础收益, 消耗的资源数量为基础消耗量的 3 倍；不挖矿消耗的资源数量为基础消耗量。到达矿山当天不能挖矿，沙暴日也可挖矿。玩家到达终点后可退回剩余的水和食物，每箱退回价格为基准价格的一半。

（6）多人游戏：当有多名玩家时，若某天其中的任意 k 名玩家路线相同，则任一玩家消耗的资源数量均为基础消耗量的 $2k$ 倍；若某天其中的任意 k 名玩家在同一矿山挖矿，则任一玩家消耗的资源数量均为基础消耗量的 3 倍，且一天通过挖矿获得的资金是基础收益的 $\frac{1}{k}$；若某天其中的任意 k 名玩家在同一村庄购买资源，每箱价格均为基准价格的 4 倍。其他情况下消耗资源数量与资源价格与单人游戏相同。

1.2 求解问题

根据游戏的不同设定，建立数学模型，解决以下问题:

1. 只有一名玩家，玩家已知每天天气状况，给出一般情况下玩家的最优策略。求解附件中的“第一关”和“第二关”。

2. 只有一名玩家，玩家仅知道当天的天气状况，据此决定当天的行动方案，给出一般情况下玩家的最佳策略，并对附件中的“第三关”和“第四关”进行具体讨论。

3. 有 n 名玩家，他们有相同的初始资金，且同时从起点出发。

（1）假设在整个游戏时段内每天天气状况事先全部已知，每名玩家的行动方案需在第 0 天确定且此后不能更改。给出一般情况下玩家应采取的策略，并对附件中的“第五关”进行具体讨论。

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

2 问题分析

2.1 问题一分析

由于每天天气状况已知，游戏中不存在任何未知因素，因此一定存在最优解。游戏目标是在规定时间到达并保留尽可能多的资金，该目标可转化为求出截止日期当天在终点的剩余最大资金量 m，而 m 的求解实际上可转化为前一天在终点或终点邻居的剩余最大资金量的求解。这样一直迭代到第一天，可以得到动态规划的雏形。考虑用二维数组 $c[i][j]$ 记录在第 i 天到达 j 地最大剩余资金数对应的资源消耗和资金赚取，通过前一天的邻居节点 k（考虑到原地停留，还包括自身）对应的 $c[i][k]$ 和第 i 天到达 j 地的资源消耗、资金赚取情况获得状态转移方程。

2.2 问题二分析

在这一问题中，玩家仅知道当前的天气，我们考虑强化学习进行训练，以训练的经验来判断这一情况下，玩家应该采取的行动。为了解决这一问题，我们建立了一个基于决策树的 Q-learning 模型，在探索和经验中进行平衡。具体的操作我们将在之后详细介绍。

2.3 问题三（1）分析

这一问题固定了天气状况和地图，因此在单人行动时有唯一解。其他玩家的加入影响了玩家行动时的消耗和收益，因此，玩家需要通过其他玩家的行动调整自己的策略，然而，参与游戏的多个玩家彼此之间信息不共通，无法依靠对手的策略进行决策，只能推测对手的行动来选择自己的路线。问题的关键在于预测其他玩家的行动，为了解决这个问题，我们使用了博弈论的相关理论，建立了一个均衡模型。

2.4 问题三（2）分析

与前一小问相比，这一题中玩家拥有更改策略的自由，可以及时调整，但是天气不固定又带来了变数，这意味着玩家还要考虑不同天气发生的概率带来的影响。在天气不固定的情况下，得出一个固定的策略是不足以解决所有问题的，于是我们针对特定的地点，比如矿山和村庄，分别分析了玩家采取何种行动能够保障自己的收益。

3 模型假设

假设 1 第二题中，根据题目对天气的限制条件猜测的各种天气出现概率在游戏过程中不变。

4 符号说明

下表列出了我们在建模过程中使用的符号及其含义，部分符号我们会在描述模型时进一步给出说明。

表 **1**　符号说明

符号	符号说明	单位
$source$	起点编号	-
$dest$	终点编号	-
$weightLimit$	负重上限	千克
$originMoney$	初始资金	元
$deadline$	截止日期	天
$basicGain$	基础收益	元
$money$	在矿山赚取的资金	元
$foodNum$	消耗的食物	箱
$waterNum$	消耗的水	箱
$foodSize$	食物每箱质量	千克
$waterSize$	水每箱质量	千克
$foodPrice$	食物基准价格	元/箱
$waterPrice$	水基准价格	元/箱

5 模型的建立与求解

5.1 问题一：动态规划

游戏目标是规定时间达到终点并保留最大剩余资金，剩余资金量可用初始资金 + 挖矿所得 + 终点变卖资源 - 购买资源花销表示，由于初始资金一定，购买资源花销可以通过食物和水的消耗数量计算出，而最优策略中一定会避免到终点还有剩余资源的情况（资源变卖会折损资金），因此仅记录消耗的水、食物和挖矿所得资金即可求得剩余最大资金量。

令 $c[i][j]$ 表示第 i 天在 j 地的累计资源（水、食物）消耗情况和资金赚取情况, 即 $c[i][j]$ 实际包括消耗的水 $c[i][j].waterNum$, 消耗的食物 $c[i][j].foodNum$, 赚取的资金 $c[i][j].money$ 三个数据。为使剩余资金最大,在对可能方案的衡量中应尽量使 $c[i][j].waterNum * waterPrice + c[i][j].foodNum * foodPrice - c[i][j].money$ 取最小。

考虑到天气对消耗资源量的影响,用 $resource[i]$ 表示第 i 天的天气,$resource[weather[i]]$ 表示第 i 天对应天气的资源消耗情况。

现在考虑第 i 天处于 j 地所有可能的决策：①原地停留不挖矿（基础消耗），②移动（2 倍的基础消耗），③挖矿（3 倍的基础消耗，收获基础收益）。

下面计算所有决策产生的结果（以下的加法运算表示各部分的 $waterNum$,$foodNum$,$money$ 各自相加）：

决策一：原地停留

$$res1 = cost[i-1][j] + resource[weather[i]]$$

决策二：非沙暴天，可以移动

$$res2 = cost[i-1][k] + 2 * recource[weather]$$

决策三：挖矿

$$res3 = cost[i-1][j] + resource[weather[i]] * 3 + basicGain$$

此时还需考虑两个问题：

- **背包上限**

  在计算每个决策的结果之前还需判断此时累计所需的资源重量是否超过背包上限，若超过背包上限则需将该结果舍弃。对于背包上限问题，在未经过村庄时为初始的负重上限，而在经过村庄补充资源后的累计消耗资源总重很有可能会超过初始的背包上限（因为一部分资源在达到村庄前消耗，所以实际每天的负载量并未超过上限），为此额外设置一个变量 $supply$ 记录因经过村庄而增加的累计资源上限，其值为到达村庄后消耗的累计资源总重，这样实际的背包上限可以表示为 $weightLimit+supply$。在不超出背包上限的前提下，分别计算 $res1$,$res2$,$res3$ 对应的资金消耗与资金赚取的差值，取最小者，用其对应的 res 值更新 $c[i][j]$。在更新 $c[i][j]$ 的同时，同时记录下使其更新的顶点的编号，用于最优策略的路径计算。

- **计算顺序**

  动态规划中很重要的一点就是结果更新的顺序，这将决定结果更新的准确性。在该题中，每个结点的结果由其可达结点的结果更新，任意两两结点之间相互影响，并没有明显的先后顺序。为此在第 0 天到截止日期下各顶点的顺序遍历这样的两层循环外增设布尔变量 $change$，记录本轮的两层循环遍历之后是否至少有一个顶点 j 在第 i 天的结果 $c[i][j]$ 发生变化, 若有变化则继续遍历，若无则返回结果。

最后 $c[deadline][dest]$ 即为最优策略对应的资源消耗和资金赚取，结合路径和每日玩家行为（行进，停留或挖矿）即可获得耗费资金最少的起点购买物资情况和村庄补充物资情况，进而获取每日剩余资源和剩余资金的求解。

完成一般情况下的编程后，将关卡一和关卡二的参数设定、天气、地图分别代入，完成表格 Result.xlsx 的填写。

最终得到关卡一的最大剩余资金为 12820 元，行进路线和各天所处位置表示为下图（各区域内的绿色数字表示第几天玩家位于该地）：

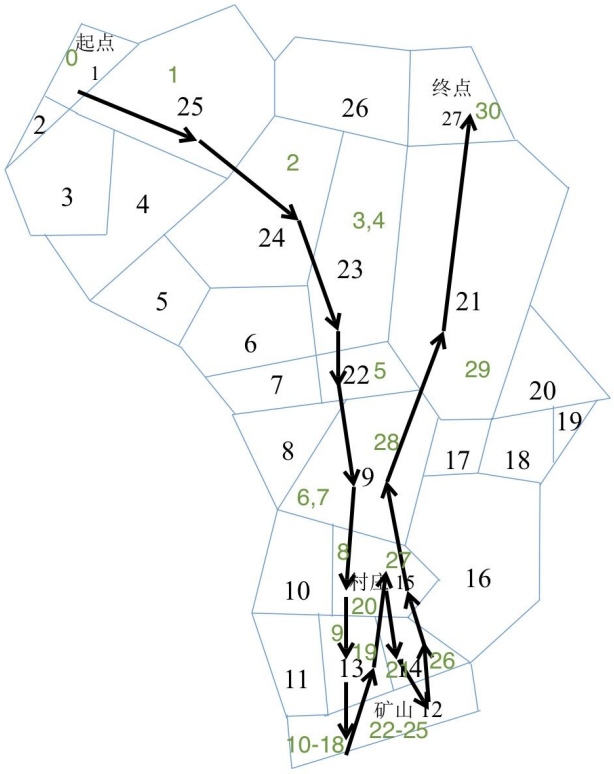

图 **1**　关卡一

关卡二的最大剩余资金为 16410 元，行进路线和各天所处位置表示为下图：

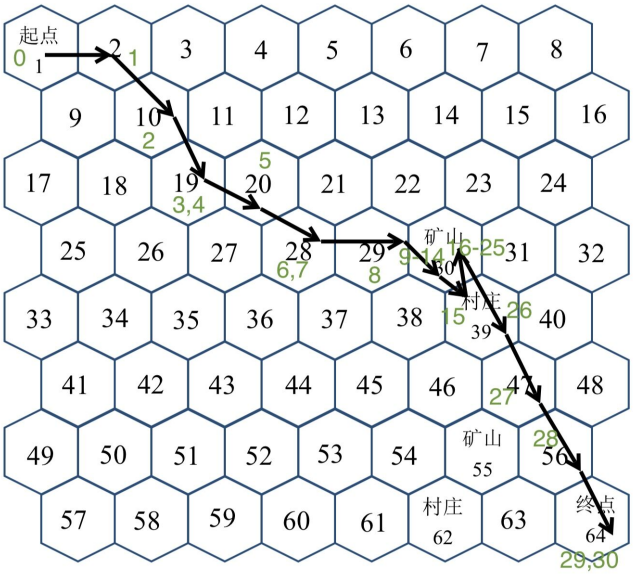

图 **2**　关卡二

5.2 问题二:Q-learning

Q-learing 是一种优良的强化学习算法，其中 Q 表示动作效用函数（action-utility function），用于评价在特定状态下采取某个动作的优劣。它组成了智能体（agent）的记忆。

Q-learing 模型包括状态，动作，奖赏三个要素，智能体———玩家根据当前状态来选择下一步要采取的动作，并记录被反馈的奖赏，以便下次采取更优的策略。在模型中，不同类型参数的组合是有限的，若天气 m 种，地点类型 n 种，背包状态 w 种，把 Q 当做一个表格，则一层共有 $m*n*w$ 行，每个动作都有一个对应的效用值。我们接下来将确定这一问题中三个要素是如何体现的。

状态的选择

我们用一个向量来表示当前的状态，它包括当前的天气 α、当前地点的类型 β 及当前已使用的物资 γ，γ 用背包剩余容量/最快到达终点所需物资来计算，数值越大，表示物资越充足。于是，状态的表达可记为 (α, β, γ)。

动作的选择

在这个游戏中，玩家有停留、挖矿、行走、购买物资这四种动作。在确定状态后，我们选择该状态下最有益的动作。由于非沙尘暴天非矿山地停留无意义，在村庄可以虚拟装满背包，矿山处挖矿总能获得收益，因此状态确定后最有益的动作也随之确定。

奖赏的选择

对于决策树中的一层，若物资不能保证到达终点，给予-2000 的奖赏。若物资保证还能继续到达终点，则根据状态给予相应的奖励。通过决策树来探索局部的收益，搜索

深度定为 3；根据天气，地点类型，背包充裕程度，一层共有 27 条分支。层分支的编号及具体情况可见支撑材料 Q 表。理想情况下最多 $27^3 = 19683$ 条。在当前节点选择收益最大的一条分支，定为局部利益。

接下来我们用伪代码的方式来描述训练过程：

```
初始化 Q ={};
while 训练轮数>0:
   Weather[]←rand();//随机生成天气用于学习
   while Q未收敛:
      初始化玩家位置为起点，开始新一轮游戏
      while S!=物资不足:
         使用策略W，获得动作 x=W(S)
         使用动作x进行游戏，获得玩家的新位置状态S'
         获得S'处的基础奖励值score[S,X];
         计算深度为3时之后两步的最大收益;
         Corr = max( p[0]*max(neibors[0]) +p[1] *max(neibors[1])+ p[2] * max(neibors[2]) );
         R(S, X) = score[S,X] + corr;//这一步的局部最大收益
         //更新Q S ← S'
         Q[S, X] ←(1 - p)*Q[S, X] + p * ( R(S, X) + q * max Q[S',X] )
```

其中，R 为深度为 3 时的利益，max 为以前学到的利益，p 为学习速率，q 为折扣因子，$Score[S, X]$ 是基础步的奖励分值；$Corr$ 是该路径抉择下后两步的抉择所得的最大分值。

根据公式可以看出，p 越大，保留之前训练的效果越少，越重视局部利益；q 越大，max 所起到的作用就越大，越重视以往经验。

对没有其他限制的一般关卡，我们设置默认参数如下：

表 **2**　默认参数

α	0（晴朗），1（高温），2（沙尘暴）
β	0（普通地点），2（矿山），3（村庄）
γ	0，1，2
p	0.6
q	0.4
$p[0]$	$\frac{1}{3}$
$p[1]$	$\frac{1}{3}$
$p[2]$	$\frac{1}{3}$

针对第三关: 没有沙暴天气，可以确定一条最优路径为：

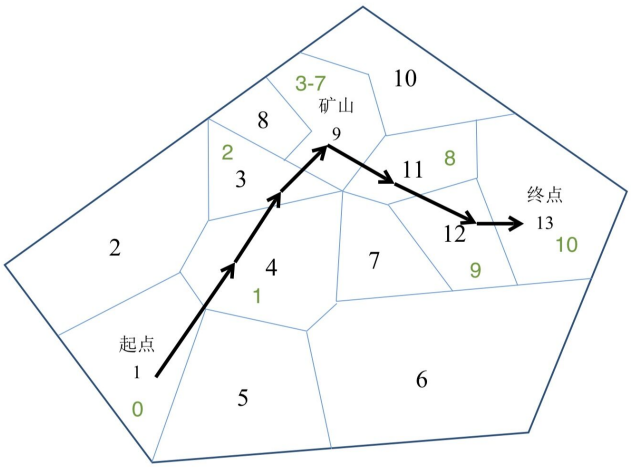

图 **3**　关卡三

不妨假设晴朗和高温天气概率均等，即 p[0]=0.5 ; p[1]=0.5 ; P[2]=0;

针对第四关：沙暴的概率很小，不妨令 p[0]=0.45;p[1]=0.45; p[2]=0.1; 大致有两种路径选择：路径选择一为经过村庄到矿山然后到终点，可以只经过一次，也可以在时间盈余而物资不够，需要到矿山后再到村庄再到矿山挖矿。路径选择二为先去矿山挖矿，资源不足时去村庄补给并回到矿山继续挖矿，并确保在规定时间和现有物资未耗尽时回到终点。

TODO：贴路径图

5.3 问题三（1）

在建模之前，我们补充几个假设，这些假设能够简化模型，但现实中未必能够实现。

- 所有玩家智力水平一致，且绝对理性。他们趋向于达成纳什均衡，而不是破坏规则，使局面变得更混乱不可控;
- 所有玩家都是风险厌恶型，他们趋向于携带充足的物资，离开起点时，他们会尽量将物资补充到最满，离开村庄前，他们会补充自己此前的消耗，直到资金耗尽;
- 如果玩家判断自己物资耗尽前无法抵达下一个补充物资的地点，他们会停留在原地，不再对其他玩家产生影响。

除了这些假设以外，我们还要补充一些定义，这些定义能让模型的叙述更精简。

- **特殊地点**

  我们定义村庄、矿山、终点这三类为特殊地点，只有这几个点会对玩家的选择产生影响。我们认为，玩家从一个特殊地点赶到另一个特殊地点只会按照最优的方式行动，即通过最短路径抵达特殊地点，途中除非遇见沙暴，否则不停留。

- **紧急状态**

  玩家处于紧急状态是指，如果玩家继续停留在当前地点，将因为物资或时间不足而导致无法走到终点，游戏失败。

在确定了天气和地图的情况下，我们可以通过类似第一题的动态规划法求得最佳路径，确定前往特殊地点的顺序。根据博弈论的知识，我们知道，每个人都会选择对自己更有利的方案，体现在游戏中就是更能避免自己失败的方案。在这一局多人游戏中，他们不会选择会导致其他玩家所得利益超过自己利益的方案，所以他们都会按照最佳路径行走。

最佳路径分为两种情况，遇到这两种情况时，最优的处理方案分别是：

1. **最佳路径为直达终点的最短路**

   此时，玩家需要计算路上的最大消耗，并按消耗购买物资。最短路的路程是一定的，但可能有多条路线，玩家事前不知道其他玩家的策略，因此他们会以相等的概率随机选择一条最短路。

2. **最佳路径经过村庄或矿山**

   如果路径经过村庄或矿山，那么玩家将会同时抵达第一个村庄或矿山。他们可以选择不同的路径抵达，但所耗时间和物资都是一样的。由于玩家事前不知道其他玩家的策略，因此他们会以相等的概率随机选择一条最短路径。

   在村庄，玩家可能会选择休息一天，让一部分玩家先离开，从而用较低价格购入村庄物品。但是，玩家的智力水平是一致的，所以留下的玩家大概率会超过 2 个，那么这一天的停留就没有意义，因此，所有玩家的选择都是购入物资后离开。

   在矿山，玩家可能会选择挖矿、休息或者直接离开。我们对某一玩家 i 进行分析。玩家 i 首先应当预估自己抵达矿山时的状态，如果处于紧急状态，则应继续赶路，如果不处于紧急状态，即不会因为停留在矿山而导致无法走到终点，就可以停留在矿山。玩家 i 可以估计停留在矿山时的玩家数量，列出在矿山的 k 个玩家的所有可能策略，然后求出纳什均衡，判断这三类玩家分别占总数的多少，并以此为概率随机选择一种方案。

   由此，我们得出这种情况下，一个玩家为了不让自己获得的利益少于一起进行游戏的其他人，应当采取这样的策略：

1. 玩家根据动态规划的办法计算出最佳路径，如果有多条，则以相等的概率随机选择一条最佳路径；
2. 在确定路径后，玩家按特殊地点分为若干段，每一段的起点和终点都是特殊地点，而且途中不经过特殊地点。
3. 对每一段路程，预估将要抵达的特殊地点可能遇到的人数，列出可能策略并求出纳什均衡，按概率随机选择一种策略，然后继续对下一段路程实行同样的操作。
4. 最后敲定行走路径和特殊地点的停留时间，购买物资，尽可能满足最大消耗。

接下来，我们对具体的案例进行分析。

对于第五关，最佳路径只可能在两种路径中产生，一种是走最短路径抵达矿山以后，按照每天的收益决定挖矿、休息或是离开，另一种路径是走最短路径直达终点。

经过矿山

我们分别计算晴朗天气和高温天气挖矿一天消耗的资源，并将它们等价折算成资金，得到

表 **3**　挖矿消耗

消耗 / 天气	水	食物	资金
晴朗	9	12	165
高温	12	27	405

挖矿一天获得的收益是 200 元，因此晴朗天气挖矿可以得到 35 元的净收益，但高温天气挖矿则会损失收益，因此高温天气不挖矿是更好的选择。根据给定的天气情况，我们可以看到，最后几天都是高温，因此在第 7 天离开矿山前往终点是最优选择。

于是，我们得到了经过矿山的最佳路径：

1. 从 1 出发，通过 1-2-3-9 或 1-4-3-9 路线抵达矿山，这一段路程的消耗为:

表 **4**　第一段路的消耗

消耗 / 天数	水	食物	资金
1	6	8	110
2	18	18	270
3	6	8	110
总消耗	30	34	490

2. 第 4 天至第 6 天在矿山挖矿，这一段时间的消耗和收益为:

表 **5**　第二段路的消耗

消耗 / 天数	水	食物	资金
4	9	12	-35
5	9	12	-35
6	9	12	-35
总消耗	27	36	-105

其中，资金的负号表示此时有净收益，即矿山挖矿一天赚得的钱不仅补足了消耗的资金，还有剩余。

3. 第 7 天离开矿山，通过 9-10-13 或 9-11-13 抵达终点，这一段路程的消耗为:

表 **6**　第三段路的消耗

消耗 / 天数	水	食物	资金
7	18	18	270
8	18	18	270
总消耗	36	36	540

综合上面的结果，我们得到这一路径的总消耗：

表 **7**　过矿山的最优路径总消耗

	水	食物	负重	资金
总消耗	93	106	491	925

经过矿山

同样地，我们可以计算出直达终点的总消耗：

表 **8**　直达终点的最优路径总消耗

	水	食物	负重	资金
总消耗	30	34	158	490

因此，最佳路径为直达终点。为了不让对方获得比自己更高的收益，两位玩家都会选择走直达终点的最佳路径，而直达终点的最佳路径有且仅有一条，因此他们的总消耗为

表 **9**　两位玩家参与时的最优路径总消耗

	水	食物	负重	资金
总消耗	60	68	316	980

最后，他们各自剩下 9020 元。

5.4 问题三（2）

在这一题的情景中，玩家选择策略时无法预知第二天的天气，出于简便考虑，我们假设玩家选择离开而因为沙暴无法实现时，玩家将会停留在原地休息，而不进行其他活动。

为了叙述的方便，我们定义以下几种状态：

物资的三种状态:

- **物资耗尽**

  剩下的水和食物不够，哪怕天天晴朗都无法抵达下一个物资补充地点
- **物资告急**

  剩下的水和食物数量有限，假设接下来的天气随机变化，抵达下一个物资补充地点路上损耗的物资期望超过了现有物资，如果再不离开有可能因为极端天气而被淘汰
- **物资充足**

  正常的状态

时间的两种状态:

- **时间告急**

  剩下的时间有限，假设接下来的天气随机变化，抵达终点消耗的时间期望超过了剩余时间，如果再不离开有可能因为极端天气而被淘汰

- **时间充足**

正常的状态

我们沿用 3.（1）解答中的假设和定义，假定从一个特殊地点前往下一个特殊地点的路上，除非出现沙暴天气，否则所有人都不停顿。每天行动结束后，玩家会根据地图判断接下来的路是否有多种选择，如果有多种选择，他们将以同等的概率随机选择其中一条路线。

抵达村庄后，玩家有两种选择：在村庄停留或直接离开。我们首先判断这里的所有人是否已经时间告急，如果是，所有人都将继续前进，如果不是，我们可以分析当前在村庄且不处于时间告急状态的 k 个玩家所采取的策略。这 k 个玩家中，有 s 个玩家处于物资告急状态，他们下一步一定是离开村庄。剩下的 k-s 个玩家可能采取不同的策略，一种策略是停留一天，等人少一点再走，以降低消耗，但是玩家智力水平相同，意味着所有人都会这么想，最终所有人都会留在村庄，反而增加了无益的消耗，因此我们假定玩家路过村庄时不会停留。

抵达矿山后，玩家有三种选择：休息一天，挖矿或直接离开。同样地，我们首先判断这里的所有人是否已经时间告急，如果是，所有人都将继续前进，如果不是，还可以分析当前在矿山且不处于时间告急状态的 k 个玩家所采取的策略。这 k 个玩家中，有 s 个玩家物资告急，他们的下一步一定是离开，剩下的 k-s 个玩家中，一部分玩家选择休息一天，一部分玩家选择挖矿，剩下的玩家直接离开。我们对他们的策略求得纳什均衡，从而计算出玩家采取各种策略的概率。

这三种策略的收益函数分别是：

$$U_r = -p_1(w1 + f1) - p_2(w2 + f2) - p_3(w3 + f3) \tag{1}$$

$$U_m = -3p_1(w1 + f1) - 3p_2(w2 + f2) - 3p_3(w3 + f3) + \frac{1}{m}E \tag{2}$$

$$U_l = -2lp_1(w1 + f1) - 2lp_2(w2 + f2) - p_3(w3 + f3) \tag{3}$$

其中，U_r、U_m、U_l 表示选择休息、选择挖矿和选择离开的玩家各自的收益期望，p_i、w_i、f_i(i=1,2,3) 分别表示晴朗、高温、沙暴天气发生的概率以及对应天气条件下水和食物的基础消耗量，m、l 分别表示选择挖矿和选择离开的人数，E 表示挖矿一天的基础收益。

根据收益函数，我们可以确定纳什均衡点，从而确定玩家采取各种策略的概率。玩家将依靠这一预测选择自己的策略，以期获得较高的收益。

接下来我们对具体情形进行分析。

由于天气情况无法确定，我们假设最佳路径过矿山，并分析玩家在两类特殊地点会采取怎样的措施。与 3.（1）的假设一致，玩家会通过最短路径前往特殊地点。在出发前，玩家会对可能的对局情况作出预测，并依据预测购买合适的物资。

在村庄里，他们趋向于一同购买物资并离开，因为留下来的人必然会比先走的人损失更多，而他们都不愿意损害自己的利益。

到矿山处，他们趋向于一起挖矿直至物资告急。在消耗最大的沙暴天气，如果三个人一起挖矿，他们将平分 1000 元的基础收益，而各自损失价值 450 元的资源，但如果一个人不挖而剩下两人挖，则两人会平分 1000 元的基础收益，比不挖的人赢得更多收益。为了不让竞争对手获得更多收益，三个人都会选择挖矿。其他天气亦然，只有选择挖矿才能保证自己获得的收益不比其他二人少。

由此，我们确定了玩家在这一地图中特殊地点应当选择的策略。

6 模型评价

6.1 优点

- 在动态规划中进行了适当剪枝，减少了部分不必要的计算。
- 通过决策树 +Q-learning 对探索和学习进行了平衡，而不是只顾眼前利益。
- 合理引入了一些假设和约定，使模型完善化。
- 适当简化了地图，将地图抽象为几个点，运用图论的知识简化了问题。
- 分情况讨论了各种策略对自己和他人的影响。

6.2 缺点

- 博弈论相关的部分过于理想化，不够符合实际。
- 问题二过程中用不变的天气类型概率来预测全局天气，而没有考虑实际情况下沙漠天气的变动情况。

6.3 改进思路

我们的模型还可以进一步优化，但由于种种限制，这些想法没能实现。

1. 在 Q-learning 部分，如果能获得足够的训练样本，把学习结果和正确结果比对并修正一些系数和选择，学习结果会更好。
2. 观测大量真实沙漠天气数据，大致得出各种天气的概率，天气类型的连续性，来对天气概率进行修正和动态调整。

参考文献

[1] 徐涛, 赵慧伟, 吕宗磊. 多人不完备信息博弈的一种解法及改进 [J]. 武汉大学学报 (工学版),2011,44(06):792-796+805.

[2] 郑应平. 多人决策与博弈论 (六)[J]. 信息与控制,1987(06):41-45.

[3] 阮晓钢, 刘鹏飞, 朱晓庆. 基于气味奖励引导的 QGlearning 环境认知方法 [J/OL]. 清华大学学报 (自然科学版):1-7[2020-09-13].

附录 A

1.1 头文件

```
//Problem.h
#pragma once
#include <iostream>
#include <vector>
#include <fstream>
#include <sstream>
using namespace std;
/*
   +------------基本参数设置--------------+

*/

const int MAX_STEP = 30;//限制日期数
const int MAX_SPOT = 65; //地图上最多的地点数
const int MAXCOST = 2000;
const int waterpri = 5;
const int foodpri = 10;
const int watersz = 3;
const int foodsz = 2;
const int MAXPAC = 1200;
const int MYMONEY = 10000;

enum {
   //地区类型
   ORD = 0,//普通地点
   SRC = 1,//起点
   MINE = 2,//矿山
   VIL=3,//村庄
   DST = 4,//终点
   //天气类型
   SUNNY=0,HYPE=1,SAND=2,
};


struct graph {
   int state = 0;//标记本身
   int dis = 0;//该点距离终点的最短路径长度->方便后续剪枝
   vector<int> neibors;//邻居地点

};

struct Resource;
```

```
struct Resource {
  int money = 0;
  int water = 0;
  int food = 0;
  Resource() {}
  Resource(int x, int y, int z) : money(x), water(y), food(z) {}
  Resource(const Resource &tmp) : money(tmp.money), water(tmp.water), food(tmp.food) {}
  //重载操作符
  Resource operator + (Resource tmp) {
    Resource res;
    res.money = money + tmp.money;
    res.water=water + tmp.water;
    res.food = food+ tmp.food;
    return res;
  }
  Resource operator - (Resource tmp) {
    Resource res;
    res.money = money - tmp.money;
    res.water = water - tmp.water;
    res.food = food - tmp.food;
    return res;
  }
  Resource operator * (int num) {
    Resource res;
    res.money = money * num ;
    res.water = water * num ;
    res.food = food * num ;
    return res;
  }


  void operator = (Resource tmp) {
    money = tmp.money;
    water = tmp.water;
    food = tmp.food;
  }
  bool operator == (Resource tmp) {
    if (money == tmp.money && water == tmp.water && food == tmp.food)
      return true;
    return false;
  }
  bool operator < (Resource tmp) {
    if ( water < tmp.water && food < tmp.food)
      return true;
    return false;
  }
```

```
};

class PROBLEM
{
protected:

   vector<graph> map;//游戏地图
   Resource dpmap[MAX_STEP + 2][MAX_SPOT + 2];//动态规划的记录
   Resource package;//背包
   string filename;//输入文件名来建立相应游戏地图

public:
   PROBLEM() {}
   PROBLEM(string setname): filename(setname) {}
   ~PROBLEM(){
      vector<graph>().swap(map);//释放vector内存空间
   }
   friend class TEST;//方便测试类调用
   friend class Sol2;
    vector <Resource> resource;
   void construct_table();
   void bfs_getDis();//计算其他点到终点的最短距离
   int set1_dp(int weather[],int dest);//动态规划寻找路径
   void init();//一些初始化操作
   int getres_set1();//动态规划之后计算结果
   void check_path(int day,int dest);//打印路径
   int check_vil(int reachday, int dest);//判断路径上是否经过村庄

};

class Sol2 {

private:
   //天气概率参数，可以更改以获得其他情况
   int psunny = 0.45;//晴天概率
   int phype = 0.45;//高温概率
   int psand = 0.1;//沙尘暴概率

protected:

public:
   void init();
   void get_R();//计算基础奖励
   void get_Q(int weather[],PROBLEM set4);//学习：综合局部收益和经验值
   int judge_state(int weather,PROBLEM set4,int i,int j);//用于决策树搜索，计算后续决策的收益
```

```
};
```

1.2 问题一源程序

```
//Problem.cpp
#include "Problem.h"
#include <queue>

Resource sunny(0, 5, 7);
Resource hype(0, 8, 6);
Resource sandstorm(0, 10, 10);
Resource digmine(1000, 0, 0);

int supply[4];//水量，食物量，增加的容量，花费的钱币

int Path[MAX_STEP + 2][MAX_SPOT + 2];
/*
   从文件中读入并建图
   attention; 不含dis
*/
void PROBLEM::construct_table() {
   resource.push_back(sunny);
   resource.push_back(hype);
   resource.push_back(sandstorm);

   graph blank;
   map.push_back(blank);//地点从1开始标记,所以先push一个空的0

   ifstream infile1(filename);
   string line;//数据格式:  点编号 点状态 邻居节点
   //按行读取
   while (getline(infile1, line)) {
      istringstream in(line);
      graph temp;

      int from = 0, state = 0, to = 0;
      in >> from >> state;
      temp.state = state;

      while (in >> to) {
         temp.neibors.push_back(to);
      }
      map.push_back(temp);

   }
```

```
    infile1.close();
    bfs_getDis();
}

/*
    计算每个点到终点的最短距离
*/
void PROBLEM::bfs_getDis() {
    int size = map.size();
    bool *visited = new bool[size];
    for (int i = 0; i < size; i++)
        visited[i] = false;
    visited[0] = true;
    queue<int> que;
    int destnum = size - 1;
    que.push(destnum);//check:size-1是终点编号吗
    map[destnum].dis = 0;
    while (!que.empty())
    {
        int from = que.front();
        visited[from] = true;
        que.pop();
        for (int i = 0; i < map[from].neibors.size(); i++) {
            int neibor = map[from].neibors[i];
            if (visited[neibor] == false) {
                map[neibor].dis = map[from].dis + 1;
                que.push(neibor);
                visited[neibor] = true;
            }

        }
    }
    delete[]visited;

}

/*
    考虑最小消耗量,初始化为极大
*/
void PROBLEM::init() {
    Resource initres(0, MAXCOST, MAXCOST);
    for (int i = 0; i <= MAX_STEP; i++) {
        for (int j = 0; j <= MAX_SPOT; j++) {
            dpmap[i][j] = initres;
            Path[i][j] = 0;
            if (i == 0 && j == 1)
                dpmap[0][1].food = 0; dpmap[0][1].water = 0; dpmap[0][1].money = 0;
```

```
    }
    }

  }


/*
  返回花销少的，已考虑赚的钱
*/
bool mincmp(Resource a, Resource b) {
  int money1 = a.water * waterpri + a.food *foodpri - a.money;
  int money2 = b.water * waterpri + b.food *foodpri - b.money;
  if (money1 <= money2) {
    return true;
  }
  return false;
}

/*
  第一题，动态规划(优化：带剪枝)
*/

int PROBLEM::set1_dp(int weather[], int dest) {

  init();

  memset(supply, 0, sizeof(supply));
  bool haschange = true;
  while (haschange) {
    haschange = false;
    for (int i = 1; i <= MAX_STEP; i++) {//第i步
      for (int j = 1; j < dest; j++) {//某个点
        if (dpmap[i - 1][j].food < MAXCOST) {//剪枝：判断是否到了j点，带MAXCOST的可以不予计算
          for (auto neibor : map[j].neibors) {//走到下一步邻居节点
            //TODO:计算i-30天有几天沙尘暴
            if (i + map[neibor].dis >
                MAX_STEP)//剪枝，在当前节点必须预留足够的时间到终点，时间不够的话可以不予考虑
              break;
            if (j != dest && weather[i] != SAND) {//到终点游戏结束
              Resource newres1 = dpmap[i - 1][j] + resource[weather[i]] * 2;
              if (((newres1.water*watersz + newres1.food*foodsz) < (MAXPAC + check_vil(i,
                  neibor) * supply[2]))
                && ((newres1.water*waterpri + newres1.food*foodpri) < (MYMONEY +
                    check_vil(i, neibor) * supply[3]))) {//保证背包装的下,钱也不能超过
                if (mincmp(dpmap[i][neibor], newres1) == false) {
                  dpmap[i][neibor] = newres1;
                  haschange = true;
```

```
                Path[i][neibor] = j;
                /*if (neibor == 27)
                  cout << "第"<< i<<" 天"<< "i reach dest from" << j << endl;*/
                  //如果走过来是一个村庄，虚拟装满背包
                if (map[neibor].state == VIL) {
                  if (supply[2]==0||supply[0] * watersz + supply[1] * foodsz <
                      supply[2]) {//补给少的那份
                    int newpri = 2 * (waterpri * dpmap[i][neibor].water + foodpri *
                        dpmap[i][neibor].food);
                    supply[0] = dpmap[i][neibor].water; supply[1] =
                        dpmap[i][neibor].food;
                    supply[2] = supply[0] * watersz + supply[1] * foodsz;
                    supply[3] = dpmap[i][neibor].money;
                  }
                }
              }
            }


          }
        }

        //自身状态转换
        //原地停留无其他行为
        if (j != dest && map[j].state != MINE )
            {//除去矿山外没有必要主动停留,所以只要考虑沙尘暴无法走动
          Resource newres2 = dpmap[i - 1][j] + resource[weather[i]];
          int newsize = newres2.water*watersz + newres2.food*foodsz;
          int newpri = newres2.water*waterpri + newres2.food*foodpri;
          int pass_vli = check_vil(i, j);
            if (   ( newsize < (MAXPAC + pass_vli* supply[2]) )
              && ( newpri < (MYMONEY + pass_vli * supply[3]) ) )
                {//保证背包装的下,钱也不能超过
              if (mincmp(dpmap[i][j], newres2) == false) {
                dpmap[i][j] = newres2;
                haschange = true;
                Path[i][j] = j;
                if (map[j].state == VIL) {
                  if (supply[2] == 0 || supply[0] * watersz + supply[1] * foodsz <
                      supply[2]) {//补给少的那份
                    supply[0] = dpmap[i][j].water; supply[1] = dpmap[i][j].food;
                    supply[2] = supply[0] * watersz + supply[1] * foodsz;
                    supply[3] = dpmap[i][j].money;
                  }
                }
              }
            }
```

```
            }

            //如果是在矿山判断是否挖矿
            if (map[j].state == MINE && dpmap[i - 1][j].food < MAXCOST) {
               Resource newres1 = dpmap[i - 1][j] + resource[weather[i]] * 3 + digmine;
               if (((newres1.water*watersz + newres1.food*foodsz) < (MAXPAC + supply[2] *
                    check_vil(i, j)))
                  && ((newres1.water*waterpri + newres1.food*foodpri)) < (MYMONEY +
                       check_vil(i, j) * supply[3])) {//保证背包装的下,钱也不能超过
                  if (mincmp(dpmap[i][j], newres1) == false) {
                     dpmap[i][j] = newres1;
                     cout << "矿山编号" << j << "上一步花费的食物" << dpmap[i - 1][j].food << "
                         ";
                     cout << i << "I have dig mine" << endl;
                     haschange = true;
                     Path[i][j] = j;
                  }
               }
            }


            //cout <<"i="<< i << endl;
         }




      }
   }
}



///////
cout << "supply" << supply[0] << " " << supply[1] << " " << supply[2] << " " << supply[3] <<
    endl;
//运行完后计算

//for (int i = 1; i <= MAX_STEP; i++) {//第i步
// for (int j = 1; j < map.size(); j++) {//某个点
//
//   cout << i << " " << dpmap[i][j].food << " " << dpmap[i][j].water << " " <<
    dpmap[i][j].money << endl;
// }
//}
return getres_set1();
```

```
  ////////
}


int PROBLEM::getres_set1() {
  int dest = map.size() - 1;
  Resource mincost(0, MAXCOST, MAXCOST);
  int reachday = 0;
  for (int i = 1; i <= MAX_STEP; i++) {
    if (mincmp(mincost, dpmap[i][dest]) == false) {
      reachday = i;
      mincost = dpmap[i][dest];
    }
    cout << i<< " " << dpmap[i][dest].food <<" "<< dpmap[i][dest].water <<" "<<
        dpmap[i][dest].money << endl;

  }
  int pass_vil = check_vil(reachday, dest);
  if (mincost.water*watersz + mincost.food*foodsz < MAXPAC + pass_vil * supply[2])
     {//保证背包装的下
    check_path(reachday, map.size() - 1);
    cout << "花费水，食物，赚到的钱" << mincost.water << " " << mincost.food << " " <<
        mincost.money << endl;
    if (mincost.water*watersz + mincost.food*foodsz < MAXPAC) {//不用村庄补给就能装下
      int cost = mincost.water*waterpri + mincost.food*foodpri - mincost.money;
      cout << "cost" << cost << endl;
      return MYMONEY - cost;
    }
    else {
      //优先在起点购买食物
      int foodnum = (MAXPAC > foodsz * mincost.food) ? mincost.food : (MAXPAC /
          2);//在起点能购买的食物的数目
      cout << "foodnum=" << foodnum << endl;
      int waternum = (MAXPAC > (foodnum * 2) ? (MAXPAC - foodnum * 2) : 0) /
          3;//在起点能购买的水的数目
      cout << "waternum=" << waternum << endl;
      int buycost = foodnum * foodpri + waternum * waterpri +
        ((mincost.food > foodnum) ? ((mincost.food - foodnum) * 2 * foodpri) : 0)
        + ((mincost.water > waternum) ? ((mincost.water - waternum) * 2 * waterpri) : 0);
            //多出的部分在村庄按两倍价格购买;
      cout << "buycost" << buycost << endl;
      return MYMONEY - buycost + mincost.money;

    }

  }
  return -1;//表示不能到达
```

```
}

void PROBLEM::check_path(int reachday, int dest) {
   cout << "路径（倒序): " << dest << " "<<endl;
   while (reachday > 0) {
      cout << Path[reachday][dest] << " :";
      cout << dpmap[reachday][dest].food << " " << endl;
      dest = Path[reachday][dest];
      reachday--;
   }
   cout << endl;
}

int PROBLEM::check_vil(int reachday, int dest) {//dest是点编号

   while (reachday > 0) {
      if (map[dest].state == VIL)
         return 1;
      dest = Path[reachday][dest];
      reachday--;
   }
   return 0;
}
```

1.3 问题二源程序

```
//Sol.cpp
#include "Problem.h"
#include <stdlib.h>
#include <time.h>



Resource sunny(0, 3, 4);
Resource hype(0, 9, 9);
Resource sandstorm(0, 10, 10);
Resource digmine(1000, 0, 0);
Resource walk_val;
Resource stay_val;
Resource stay_mine;
const double PGREEDY = 0.6;
const double QLEARN = 0.4;

int basic_score[] = { 0, -2000, 2 ,3,-2000,5, 6,-2000,4, 4, //0 is an impl
                        -2000,1,2,2000,4,5,2000 ,3,3,
```

```
                        -2000,1 ,2,2000,4,5 ,2000,3,3 };
int R[30][30][30];//对于set4,实际是27^3=19683种状态
double Q[MAX_STEP+2][MAX_SPOT+2];//记录第i天j地的分值

int dpscore[MAX_STEP + 2][MAX_SPOT + 2];
int Path[MAX_STEP + 2][MAX_SPOT + 2];
const int minscore = -10000;
const int ori3_psum = 1;


void Sol2::init() {
   Resource walk_val = sunny * 2 * psunny + hype * 2 * phype;
   Resource stay_val = sandstorm * psand;
   Resource stay_mine = sunny * 3 * psunny + hype * 3 * phype + sandstorm * 3 * psand + digmine;

}

void Sol2::get_R() {
   // 单层有27种状态，深度为3总的为27^3;
   int p1 = 0, p2 = 0, p3 = 0;
   for (int i = 1; i <= 27; i++) {
      if (i <= 9)
         p1 = psunny;
      else if (i <= 18)
         p1 = phype;
      else
         p1 = psand;
      //在随机生成weather数组后，可以确定天气，p1=1;
      p1 = 1;
      for (int j = 0; j <= 27; j++) {
         if (j <= 9)
            p2 = psunny;
         else if (j <= 18)
            p2 = phype;
         else
            p2 = psand;
         for (int k = 0; k <= 27; k++) {
            if (k <= 9)
               p3 = psunny;
            else if (k <= 18)
               p3 = phype;
            else
               p3 = psand;
            R[i][j][k] = p1 * basic_score[i] + p1 * p2 * basic_score[j] + p1 * p2 * p3
                *basic_score[k];
         }
      }
```

```
  }



  //图中节点1,2,3,5,6,7,11,21属于同一类节点，三个深度均为普通节点
  /*int temp[] = { 1,2,3,5,6,7,11,21 };
  Resource ord3_val = walk_val;*/



}

int Sol2::judge_state(int weather,PROBLEM set4,int i,int j) {
  if (i > MAX_STEP)//没有这步
    return 0;
  if (weather == SUNNY) {
    //2: 判断地点类型
    if (set4.map[j].state == ORD) {
      //3:判断背包充裕程度
      double rate = (MAXPAC - set4.dpmap[i][j].food * foodsz - set4.dpmap[i][j].water
          *waterpri)
        / (set4.map[j].dis *(sunny.food *foodsz + sunny.water*watersz));
      if (rate < 1) { //line 1
        return 1;
      }
      else if (rate < 2) {//line 2
        return 2;
      }
      else {//line 3
        return 3;
      }

    }
    else if (set4.map[j].state == VIL) {
      //3:判断背包充裕程度
      double rate = (MAXPAC - set4.dpmap[i][j].food * foodsz - set4.dpmap[i][j].water
          *waterpri)
        / (set4.map[j].dis *(sunny.food *foodsz + sunny.water*watersz));
      if (rate < 1) { //line
        return 4;
      }
      else if (rate < 2) {//line
        return 5;
      }
      else {//line
        return 6;
      }
```

```
    }
    else if (set4.map[j].state == MINE) {
      double rate = (MAXPAC - set4.dpmap[i][j].food * foodsz - set4.dpmap[i][j].water
          *waterpri)
        / (set4.map[j].dis *(sunny.food *foodsz + sunny.water*watersz));
      if (rate < 1) { //line
        return 7;
      }
      else if (rate < 2) {//line
        return 8;
      }
      else {//line
        return 9;
      }
    }

}

else if (weather == HYPE) {
    //2: 判断地点类型
    if (set4.map[j].state == ORD) {
      //3:判断背包充裕程度
      double rate = (MAXPAC - set4.dpmap[i][j].food * foodsz - set4.dpmap[i][j].water
          *waterpri)
        / (set4.map[j].dis *(sunny.food *foodsz + sunny.water*watersz));
      if (rate < 1) { //line 1
        return 10;
      }
      else if (rate < 2) {//line 2
        return 11;
      }
      else {//line 3
        return 12;
      }

    }
    else if (set4.map[j].state == VIL) {
      //3:判断背包充裕程度
      double rate = (MAXPAC - set4.dpmap[i][j].food * foodsz - set4.dpmap[i][j].water
          *waterpri)
        / (set4.map[j].dis *(sunny.food *foodsz + sunny.water*watersz));
      if (rate < 1) { //line
        return 13;
      }
      else if (rate < 2) {//line
        return 14;
```

```
      }
      else {//line
        return 15;
      }

    }
    else if (set4.map[j].state == MINE) {
      double rate = (MAXPAC - set4.dpmap[i][j].food * foodsz - set4.dpmap[i][j].water
          *waterpri)
        / (set4.map[j].dis *(sunny.food *foodsz + sunny.water*watersz));
      if (rate < 1) { //line
        return 16;
      }
      else if (rate < 2) {//line
        return 17;
      }
      else {//line
        return 18;
      }
    }
  }
  else if (weather == SAND) {
    //2: 判断地点类型
    if (set4.map[j].state == ORD) {
      //3:判断背包充裕程度
      double rate = (MAXPAC - set4.dpmap[i][j].food * foodsz - set4.dpmap[i][j].water
          *waterpri)
        / (set4.map[j].dis *(sunny.food *foodsz + sunny.water*watersz));
      if (rate < 1) { //line 1
        return 19;
      }
      else if (rate < 2) {//line 2
        return 20;
      }
      else {//line 3
        return 21;
      }

    }
    else if (set4.map[j].state == VIL) {
      //3:判断背包充裕程度
      double rate = (MAXPAC - set4.dpmap[i][j].food * foodsz - set4.dpmap[i][j].water
          *waterpri)
        / (set4.map[j].dis *(sunny.food *foodsz + sunny.water*watersz));
      if (rate < 1) { //line
        return 22;
      }
```

```
        else if (rate < 2) {//line
          return 23;
        }
        else {//line
          return 24;
        }

      }
      else if (set4.map[j].state == MINE) {
        double rate = (MAXPAC - set4.dpmap[i][j].food * foodsz - set4.dpmap[i][j].water
            *waterpri)
          / (set4.map[j].dis *(sunny.food *foodsz + sunny.water*watersz));
        if (rate < 1) { //line
          return 25;
        }
        else if (rate < 2) {//line
          return 26;
        }
        else {//line
          return 27;
        }
      }
    }
}

void Sol2::get_Q(int weather[],PROBLEM set4) {
    srand((unsigned)time(NULL));
    // TODO 随机生成weather
    //attention:注意修改PROBLEM.cpp中地点，资源消耗等相关参数;
    //tip:在头文件定义中修改psunny等参数可以调整天气的概率
    int studycircle = 1;//调整学习的次数
    while (studycircle > 0) {
      for (int i = 0; i < 35; i++) {
        double randnum = rand() / double(RAND_MAX);
        int random = 0;
        if (randnum < psunny)
          random = 0;
        else if (randnum < psunny + phype)
          random = 1;
        else
          random = 2;
        //int random = rand() % 3;
        weather[i] = random;
      }

      set4.construct_table();
      bool has_change = true;
```

```
    while (has_change)//未收敛
    {
       for (int i = 1; i <= MAX_STEP; i++) {//第i天
          for (int j = 1; j <= MAX_SPOT; j++) {
             int state1 = judge_state(weather[i], set4, i, j);
             for (auto neibor1 : set4.map[j].neibors) {
                int state2 = 0;
                if (i + 1 <= MAX_STEP) {
                   int state2 = judge_state(weather[i + 1], set4, i + 1, neibor1);
                   for (auto neibor2 : set4.map[neibor1].neibors) {
                      int state3 = 0;
                      if (i + 2 <= MAX_STEP)
                         int state3 = judge_state(weather[i + 2], set4, i + 2, neibor2);
                      Q[i][j] = (1 - PGREEDY)* Q[i][j] + PGREEDY * (R[1][1][1] + QLEARN *
                          Q[i][j]);

                   }
                }
             }
          }
       }

       studycircle--;
    }

}
```

1.4 调用模块

```
//main.cpp
#include "Problem.h"

class TEST
{
public:
   TEST();
   ~TEST();
   void test_consgraph_getdis(PROBLEM set) {
      int size = set.map.size();
      for (int i = 1; i < size; i++) {
         /*cout << endl << "点编号: " << i << " 点类型: " << set.map[i].state << " 邻居节点: ";
         for (auto j : set.map[i].neibors)
            cout << j << " ";*/
         cout << i << "距离终点:" << set.map[i].dis << endl;
```

```
    }
  }
  /*void test_dpmap(PROBLEM set) {
    for (int i = 1; i < MAX_STEP; i++) {
      for (int j = 1; j < MAX_SPOT; j++) {
        cout << dpmap[i][j] << " ";
      }
      cout << endl;
    }
  }*/
private:

};

TEST::TEST()
{
}

TEST::~TEST()
{
}

int main() {

   //第一题的main////////////////////////////////////////////
  PROBLEM set1("SET2.txt");
  set1.construct_table();
  set1.bfs_getDis();
  /*TEST smalltest;
  smalltest.test_consgraph_getdis(set1);*/
  int set1_weather[35] = { 0, HYPE,HYPE,SUNNY, SAND,SUNNY,HYPE, //0 is a blank impl
                   SAND,SUNNY,HYPE, HYPE,SAND,HYPE,
                   SUNNY,HYPE,HYPE,HYPE,SAND,SAND,
                   HYPE,HYPE,SUNNY,SUNNY,HYPE,SUNNY,
                   SAND,HYPE,SUNNY,SUNNY,HYPE,HYPE
  };


  int set_ans = set1.set1_dp(set1_weather, 64);
  /*cout << "i begin to test path 29 " << endl;
  set1.check_path(29, 27);*/
  cout << set_ans << endl;
   ////////////////////////////////////////////////////////

  //第二题的main////////////////////////////////////////////
  PROBLEM set4("SET4_6.txt");
  Sol2 sol4;
```

```
  int set4_weather[35];
  sol4.get_R();
  sol4.get_Q(set4_weather, set4);

  // */ ////////////////////////////
  system("pause");
  return 0;




}
```