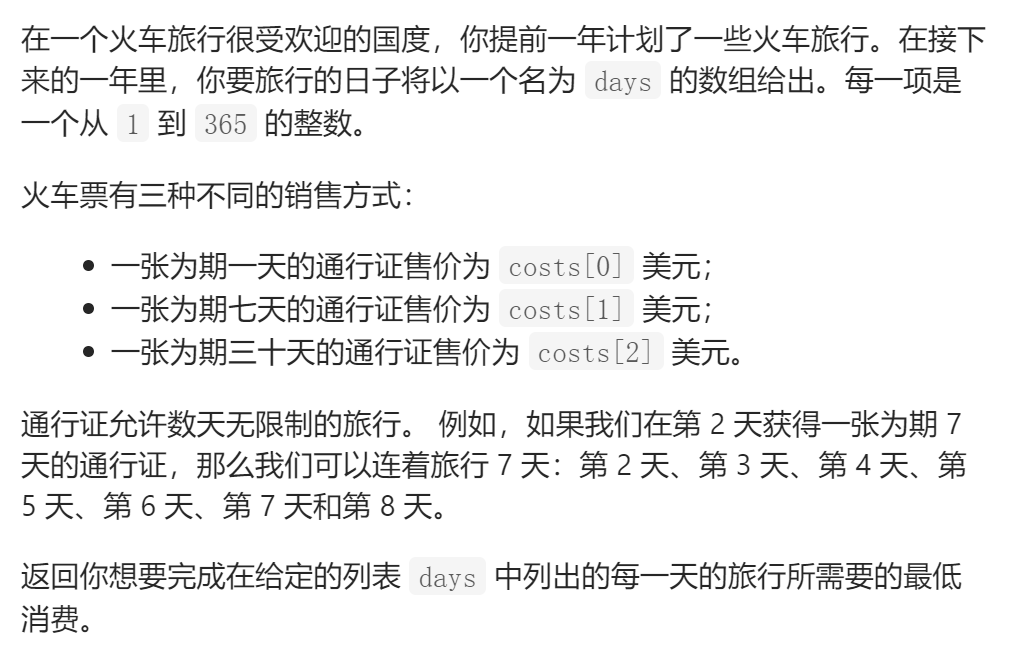
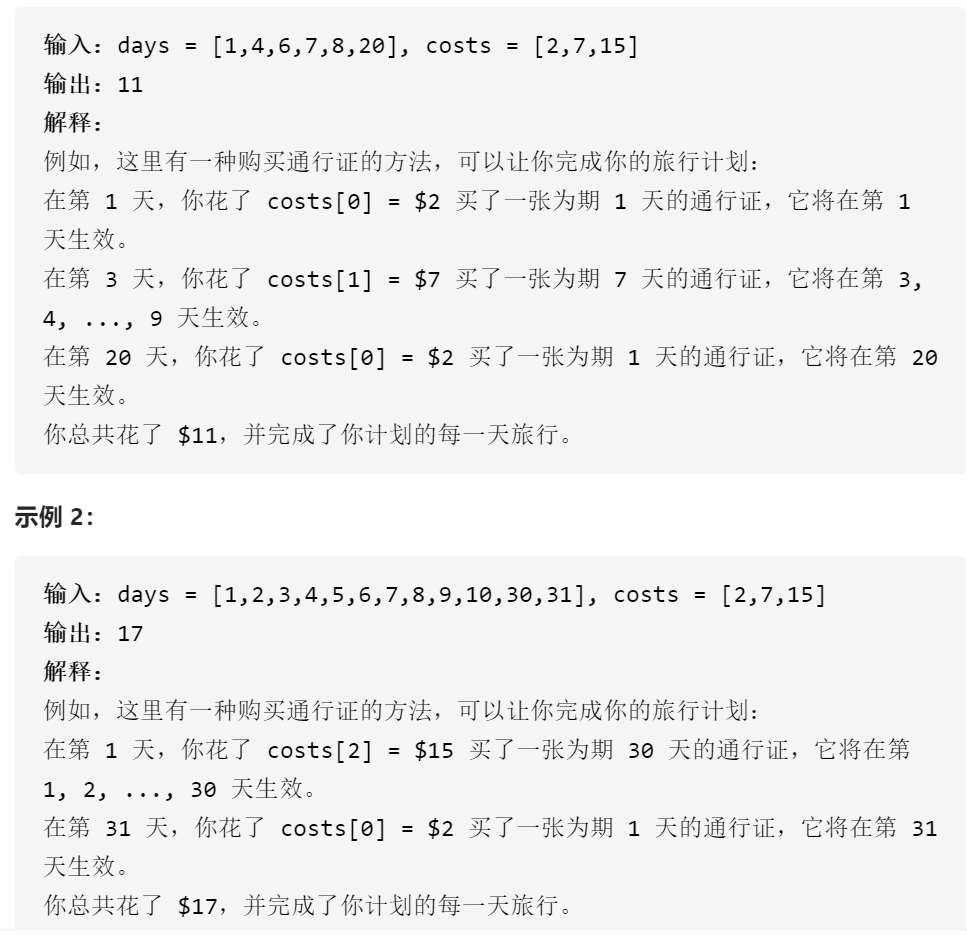
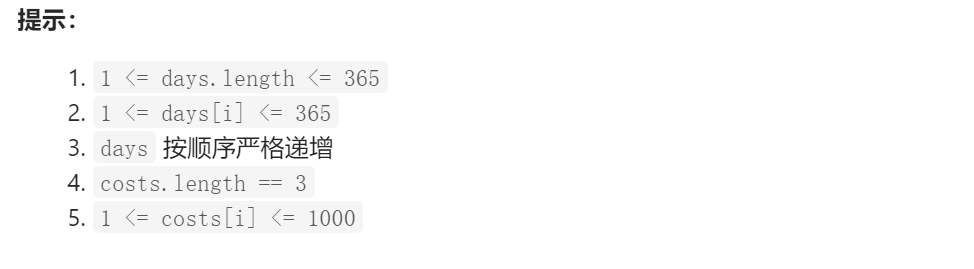
# [983. 最低票价]-【动态规划】【反序求解】-2021\3\30

## 题目描述：







## 题目分析：

### 推导动态数组：

1. 两个动态的变量：旅行计划的days、购买通行症的种类costs.
2. 所求结果：最少的花费
3. 推出动态数组：dp[days] [costs]=min\_price;

### 一个贪心的性质：

**购买通行证越晚越好，故只在需要旅行的那天才会做出是否购买通行证的选择**；

### 动态数组降维，是否一定得逆序求解？

1. 将dp[days] [costs]将维为dp[days]；
2. dp[days]：为从该天开始到结束需要的最少花费；
3. 动态转移方程：dp[days]=min(dp[days+1]+costs[0],dp[days+7]+costs[1],dp[days+30]+cost[2]);
4. 数组的初始化：只有当天数属于days时才进行动态转移方程的一个判断；若不属于直接dp[i]=dp[i+1];

### 逆序求解代码

class Solution {  
public:  
 int mincostTickets(vector<int>& days, vector<int>& costs) {  
 //一个贪心的性质：  
 //车票越晚持有越好，所以不旅行的日子不买通行症  
  
 //动态规划，逆序求解  
 //一个顺着想复杂的东西，为何反着想就容易了？  
  
 //设置dp数组  
 int MAX\_DAY=days[days.size()-1];  
 vector<int>dp(MAX\_DAY+1,0);  
 //设置一个天数的集合  
 unordered\_set<int>DAYS;  
 for(int i=0;i<days.size();i++){  
 DAYS.insert(days[i]);  
 }  
  
 for(int i=MAX\_DAY;i>=0;i--){  
 if(DAYS.count(i)){  
 //改天需要旅游  
 int temp1=0,temp2=0,temp3=0;  
 if(i+1<=MAX\_DAY)temp1=dp[i+1];  
 if(i+7<=MAX\_DAY)temp2=dp[i+7];  
 if(i+30<=MAX\_DAY)temp3=dp[i+30];  
 dp[i]=min(min(temp1+costs[0],temp2+costs[1]),temp3+costs[2]);  
 }else{  
 dp[i]=dp[i+1];  
 }  
 }  
  
 return dp[0];  
   
 }  
};

### 主要时贪心+动态规划+正序求解也是可行的

class Solution {  
public:  
 int mincostTickets(vector<int>& days, vector<int>& costs) {  
 //一个贪心的性质：  
 //车票越晚持有越好，所以不旅行的日子不买通行症  
  
 //动态规划，逆序求解  
 //一个顺着想复杂的东西，为何反着想就容易了？  
  
 //设置dp数组  
 int MAX\_DAY=days[days.size()-1];  
 vector<int>dp(MAX\_DAY+1,0);  
 //设置一个天数的集合  
 unordered\_set<int>DAYS;  
 for(int i=0;i<days.size();i++){  
 DAYS.insert(days[i]);  
 }  
 /\*  
 //逆序求解  
 for(int i=MAX\_DAY;i>=0;i--){  
 if(DAYS.count(i)){  
 //改天需要旅游  
 int temp1=0,temp2=0,temp3=0;  
 if(i+1<=MAX\_DAY)temp1=dp[i+1];  
 if(i+7<=MAX\_DAY)temp2=dp[i+7];  
 if(i+30<=MAX\_DAY)temp3=dp[i+30];  
 dp[i]=min(min(temp1+costs[0],temp2+costs[1]),temp3+costs[2]);  
 }else{  
 dp[i]=dp[i+1];  
 }  
 }  
 \*/  
  
 //正序求解  
 for(int i=days[0];i<=MAX\_DAY;i++){  
 if(DAYS.count(i)){  
 //该天需要旅游  
 int temp1=0,temp2=0,temp3=0;  
 if(i-1>=0)temp1=dp[i-1];  
 if(i-7>=0)temp2=dp[i-7];  
 if(i-30>=0)temp3=dp[i-30];  
 dp[i]=min(min(temp1+costs[0],temp2+costs[1]),temp3+costs[2]);  
 }else{  
 dp[i]=dp[i-1];  
 }  
 }  
 return dp[MAX\_DAY];  
   
 }  
};