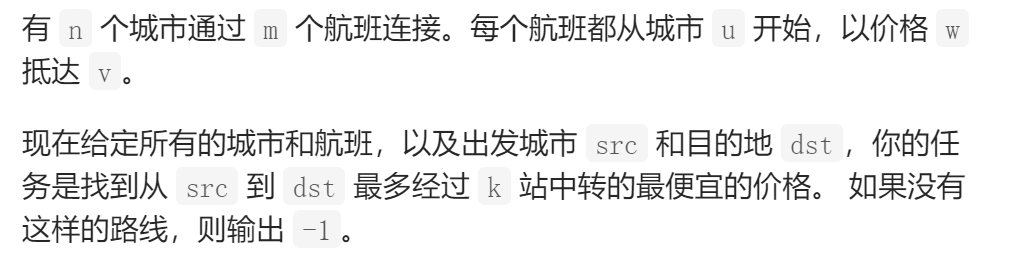
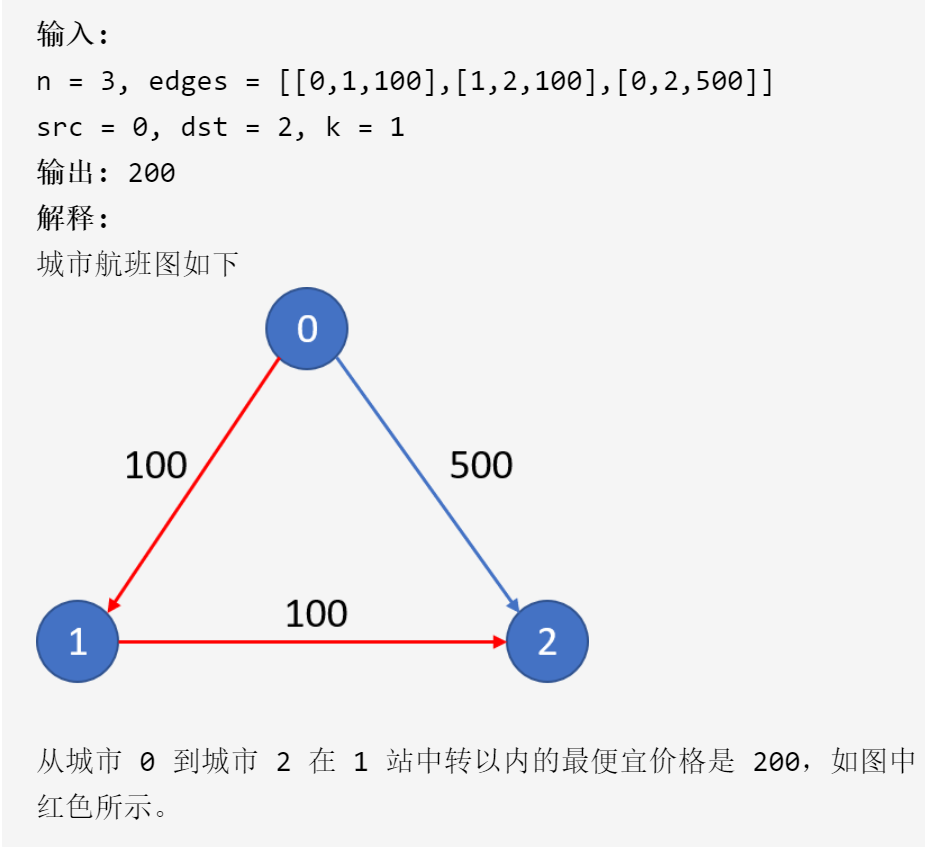
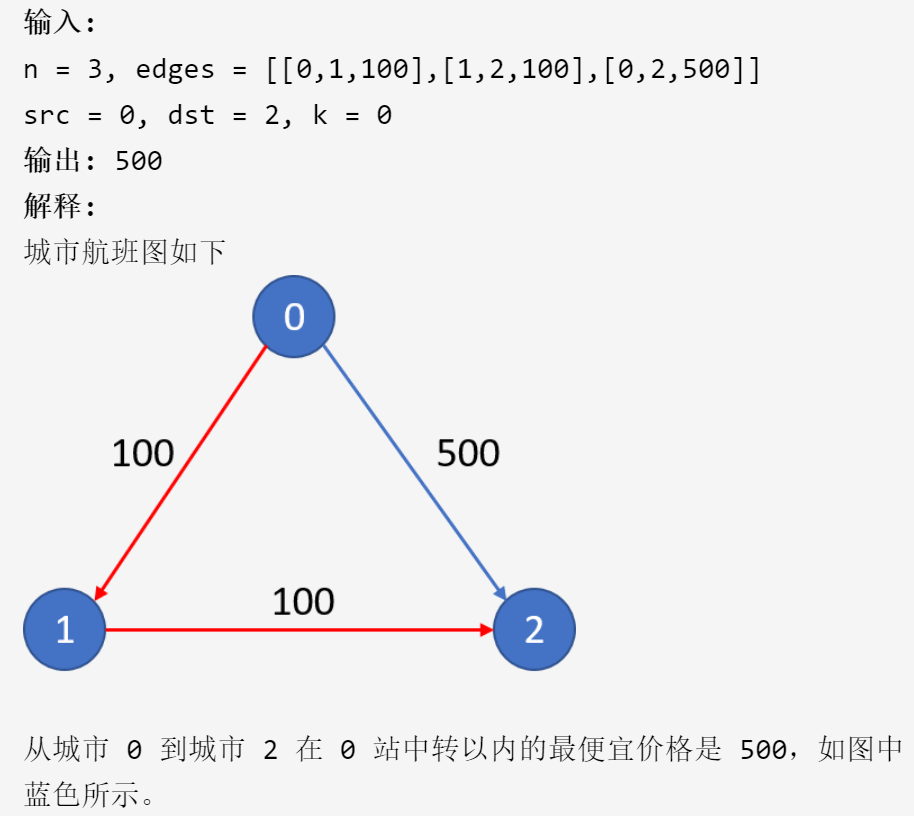
#### #787. K 站中转内最便宜的航班---【图论问题中的动态规划】

## 题目描述







## 动态规划分析

### 1.寻找最优子结构

1. 设a1,a2,a3....,am表示其中的m个城市；设起点为src,中点为dst,中转次数为k;
2. 寻找从src到dst中转k次的最便宜路线的问题--->可以转换为--->从m个城市中随意找出一个中转城市b(1)，且有一趟从ai到dst的航班。则需要寻找从src到b(1)经过k-1次中转的最便宜路线；
3. 依此类推：从m个城市中随意找出一个中转城市bi,且有一趟从b(i)到b(i-1)的航班，则需要寻找一个从src到b(i)经过k-i次中转的最便宜的路线。

**动态规划求解问题具有一定的暴力性质：在这个问题中，我门并不能确定的找出这k个中转城市，因此，我们不得不考虑所有的城市都有这样的一种可能；但动态规划的优点是不会重复的求解同一个问题。**

### 2.二维动态数组和状态转移方程的确定

1. 要确定动态数组的含义，首先要分析在分解子问题的时候变动的量和不变的量。在该问题中，改变的量是“终点站”和“中转站点的个数”；不变的量是“起始站点”；dp[i] [j]要表示的值肯定直接或间接的与所求有关，在本题中表示的就是最少的花销。
2. dp[dst] [j]表示从起始站点到达dst站点经过 j 个中转站最便宜的价钱。
3. 状态转移方程dp[nowEnd] [i] = min(dp[nowBegin] [i - 1] + price, dp[nowEnd] [i]); （price 表示的是从nowEnd到nowBegin的航班价格）**确定动态数组的含义的时候要结合不变量和变量来分析，不可想当然的套来套去；**

### 3.动态数组的初始化

* 遍历flights数组，dp[dst] [0]=price;

## 3代码实现

* class Solution {  
  public:  
   int findCheapestPrice(int n, vector<vector<int>>& flights, int src, int dst, int K) {  
   //动态规划---初始化为最大  
   vector<vector<int>>dp(n,vector<int>(K+1,INT\_MAX));  
   //数组初始化  
   /\*  
   for(int i=0;i<=k;i++){  
   //航班没有重复，且不存在自环  
   dp[src][i]=0;  
   }  
   \*/  
   //直达处理  
   for(int i=0;i<flights.size();i++){  
   if(flights[i][0]==src)  
   dp[flights[i][1]][0]=flights[i][2];  
   }  
    
   //遍历  
   for(int i=1;i<=K;i++){  
   for(int j=0;j<flights.size();j++){  
   int start=flights[j][0];  
   int end=flights[j][1];  
   int price=flights[j][2];  
   if(dp[start][i-1]!=INT\_MAX){  
   dp[end][i]=min(dp[start][i-1]+price,dp[end][i]);  
   }  
   }  
   }  
    
   //寻找结果  
   int res=INT\_MAX;  
   for(int i=0;i<=K;i++){  
   if(dp[dst][i]<res)  
   res=dp[dst][i];  
   }  
    
   return res!=INT\_MAX?res:-1;  
    
   }  
  };