一、编程题
ACM:路口最短时间问题
题目描述：
假定街道是棋盘型的，每格距离相等，车辆通过每格街道需要时间均为 timePerRoad；街道的街口（交叉点）有交通灯，灯的周期 T(=lights[row][col])各不相同；车辆可直行、左转和右转，其中直行和左转需要等相应 T 时间的交通灯才可通行，右转无需等待。
现给出 n*m 个街口的交通灯周期，以及起止街口的坐标，计算车辆经过两个街口的最短时间。
其中：
1）起点和终点的交通灯不计入时间，且可以任意方向经过街口
2）不可超出 n*m 个街口，不可跳跃，但边线也是道路（即 lights[0][0] -> lights[0][1] 是有效路径）

入口函数定义:

```
/**
* lights : n*m 个街口每个交通灯的周期，值范围[0,120]，n 和 m 的范围为[1,9]
* timePerRoad : 相邻两个街口之间街道的通过时间,范围为[0,600]
* rowStart : 起点的行号
* colStart : 起点的列号
* rowEnd : 终点的行号
* colEnd : 终点的列号
* return : lights[rowStart][colStart] 与 lights[rowEnd][colEnd] 两个街口之间的最短通行时间
*/
int calcTime(int[][] lights,int timePerRoad,int rowStart,int colStart,int rowEnd,int colEnd)
```

补充说明：
示例 1
输入：
[[1,2,3],[4,5,6],[7,8,9]],60,0,0,2,2
输出：
245
说明：
行走路线为 (0,0) -> (0,1) -> (1,1) -> (1,2) -> (2,2) 走了 4 格路，2 个右转，1 个左转，共耗时 60+0+60+5+60+0+60=245

代码：

```
#
# 返回通过指定路口之间的最短时间
# @param lights int 整型二维数组 n*m 个街口每个交通灯的周期，值范围[0,120]，n 和 m 的范围为[1,9]
# @param timePerRoad int 整型 相邻两个街口之间街道的通过时间,范围为[0,600]
# @param rowStart int 整型 起点的行号
# @param colStart int 整型 起点的列号
```

```
# @param rowEnd int 整型 终点的行号
# @param colEnd int 整型 终点的列号
# @return int 整型
#
class Solution:
    def is_turn_right(self, cur_row, rowEnd, cur_col, colEnd, x, y):
        if (x==0 and y==0):
            return False
        if (cur_row == rowEnd+1 and cur_col == colEnd):
            #right turn
            if (x==1 and y==0):
                return True
            else: #straight or turn left
                return False
        elif (cur_row == rowEnd-1 and cur_col == colEnd):
            #right turn
            if (x==-1 and y==0):
                return True
            else: #straight or turn left
                return False
        elif (cur_row == rowEnd and cur_col == colEnd+1):
            #right turn
            if (x==0 and y==-1):
                return True
            else: #straight or turn left
                return False
        elif (cur_row == rowEnd and cur_col == colEnd-1):
            #right turn
            if (x==0 and y==1):
                return True
            else: #straight or turn left
                return False

    def dfs(self , lights, total_time,num_rows, num_cols,cur_time, timePerRoad ,
            cur_row , cur_col , rowEnd , colEnd, x, y):
        if (cur_row == rowEnd and cur_col == colEnd):
            self.total_time.append(total_time)
            return
        total_time += timePerRoad
        next_direction = []
        if (rowEnd > cur_row):
            next_direction.append([0,1])
        elif (rowEnd < cur_row):
            next_direction.append([0,-1])
```

```
            if (colEnd > cur_col):
                next_direction.append([1,0])
            elif (colEnd < cur_col):
                next_direction.append([-1,0])

            for [next_x, next_y] in next_direction:
                next_col = cur_col + next_x
                next_row = cur_row + next_y
                if (next_row >= num_rows or next_row < 0 or next_col >= num_cols or next_col <
0):
                    continue
                #print(total_time)
                flag = self.is_turn_right(cur_row,next_row,cur_col,next_col,x,y)
                if (not flag) and not (x==0 and y==0):
                    total_time += cur_time
                #print(flag, cur_row, next_row, cur_col, next_col, x, y)
                #print(total_time)
                self.dfs(lights, total_time, num_rows, num_cols, lights[next_row][next_col],
timePerRoad,
                                next_row, next_col, rowEnd, colEnd, next_x, next_y)


        def calcTime(self , lights , timePerRoad , rowStart , colStart , rowEnd , colEnd ):
            # write code here
            num_rows = len(lights)
            num_cols = len(lights[0])
            self.total_time = []
            #total_time = abs(rowEnd-rowStart)*abs(colEnd-colStart)*timePerRoad
            self.dfs(lights , 0, num_rows, num_cols, 0, timePerRoad ,
                        rowStart , colStart , rowEnd , colEnd, 0, 0)
            return min(self.total_time)
```