

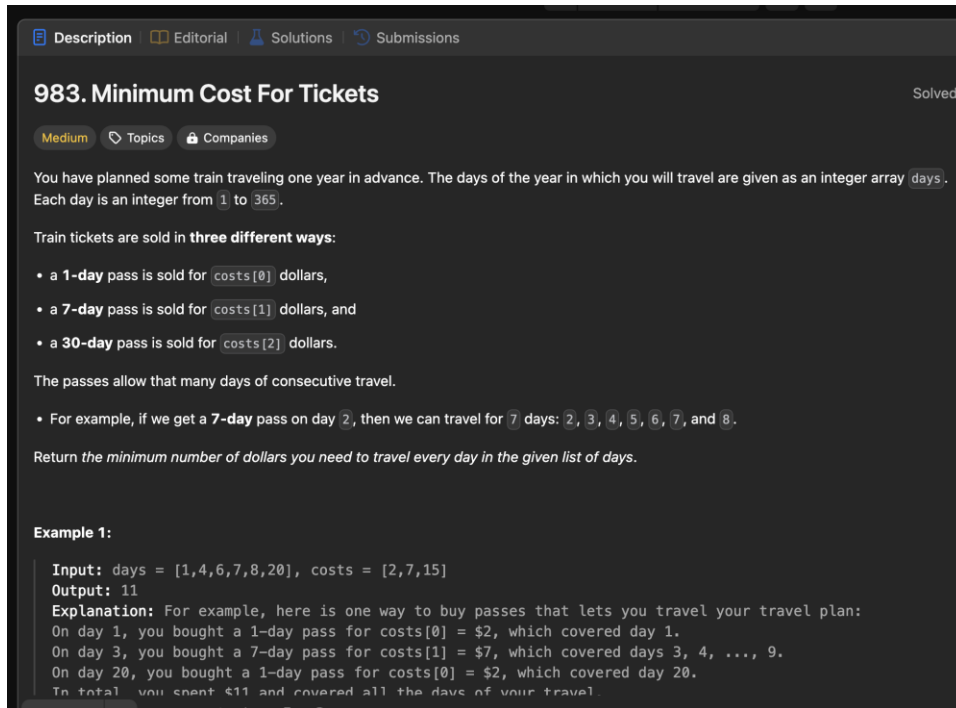
# MCS – 253P ADVANCED PROGRAMMING AND PROBLEM SOLVING

## LAB 9 Writeup(Minimum Cost For Tickets)

Aswin Sampath  
[saswin@uci.edu](mailto:saswin@uci.edu)

53844684

### Question



The screenshot shows the LeetCode interface for problem 983, 'Minimum Cost For Tickets'. The problem is categorized as 'Medium'. The description states: 'You have planned some train traveling one year in advance. The days of the year in which you will travel are given as an integer array `days`. Each day is an integer from 1 to 365. Train tickets are sold in three different ways: a 1-day pass is sold for `costs[0]` dollars, a 7-day pass is sold for `costs[1]` dollars, and a 30-day pass is sold for `costs[2]` dollars. The passes allow that many days of consecutive travel. For example, if we get a 7-day pass on day 2, then we can travel for 7 days: 2, 3, 4, 5, 6, 7, and 8. Return the minimum number of dollars you need to travel every day in the given list of days.' An example is provided: 'Input: days = [1,4,6,7,8,20], costs = [2,7,15], Output: 11'. The explanation details the optimal purchase strategy: a 1-day pass for day 1 (\$2), a 7-day pass for days 3-9 (\$7), and a 1-day pass for day 20 (\$2), totaling \$11.

### Understanding the Problem

The problem involves a frog trying to cross a river by jumping on stones placed at various positions in ascending order across the river. The frog starts at the first stone (position 0) and aims to reach the last stone. The frog's jumps must follow specific rules:

If the frog's last jump was  $k$  units, its next jump must be either  $k - 1$ ,  $k$ , or  $k + 1$  units.

The frog can only jump in the forward direction.

The task is to determine if the frog can reach the last stone following the given rules.

### Identifying Edge Cases

An empty input: When the input list of stones is empty, the frog cannot move, so it cannot reach the last stone.

A single stone: When there's only one stone, if it's not positioned at 0, the frog cannot reach it. If it's positioned at 0, the frog is already there.

## **Effective Test Cases**

A list of stones with a clear path to the last stone:

Input: stones = [0, 1, 3, 5, 6, 8, 12, 17]

Expected Output: true

A list of stones with no possible path to the last stone due to a large gap:

Input: stones = [0, 1, 2, 3, 4, 8, 9, 11]

Expected Output: false

An array with only two stones:

Input: stones = [0, 1]

Expected Output: true

## **Algorithmic Solution**

- It creates a map (mp) to store the positions of stones.
- Initializes a 2D DP array (dp) to store results of recursive calls to avoid redundant calculations.
- Implements a recursive function solve to check if the frog can reach the last stone from a specific stone position num by considering all possible valid jumps.
- The solve function performs backtracking, checking if jumps of k-1, k, or k+1 units from the current stone position can lead to the last stone.

## **Time and Space Complexity Analysis**

The time complexity for this algorithm is  $O(n)$  where n is the number of days in a year (365). It iterates through each day and computes the minimum cost for each day.

The space complexity is  $O(1)$  for the DP array, which stores costs for each day.