

Problem 568: Maximum Vacation Days

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

LeetCode wants to give one of its best employees the option to travel among

n

cities to collect algorithm problems. But all work and no play makes Jack a dull boy, you could take vacations in some particular cities and weeks. Your job is to schedule the traveling to maximize the number of vacation days you could take, but there are certain rules and restrictions you need to follow.

Rules and restrictions:

You can only travel among

n

cities, represented by indexes from

0

to

$n - 1$

. Initially, you are in the city indexed

0

on

Monday

.

The cities are connected by flights. The flights are represented as an

$n \times n$

matrix (not necessarily symmetrical), called

flights

representing the airline status from the city

i

to the city

j

. If there is no flight from the city

i

to the city

j

,

`flights[i][j] == 0`

; Otherwise,

`flights[i][j] == 1`

. Also,

$\text{flights}[i][i] == 0$

for all

i

.

You totally have

k

weeks (each week has

seven days

) to travel. You can only take flights at most once per day and can only take flights on each week's Monday morning. Since flight time is so short, we do not consider the impact of flight time.

For each city, you can only have restricted vacation days in different weeks, given an

$n \times k$

matrix called

days

representing this relationship. For the value of

$\text{days}[i][j]$

, it represents the maximum days you could take a vacation in the city

i

in the week

j

You could stay in a city beyond the number of vacation days, but you should work on the extra days, which will not be counted as vacation days.

If you fly from city

A

to city

B

and take the vacation on that day, the deduction towards vacation days will count towards the vacation days of city

B

in that week.

We do not consider the impact of flight hours on the calculation of vacation days.

Given the two matrices

flights

and

days

, return

the maximum vacation days you could take during

k

weeks

Example 1:

Input:

flights = [[0,1,1],[1,0,1],[1,1,0]], days = [[1,3,1],[6,0,3],[3,3,3]]

Output:

12

Explanation:

One of the best strategies is: 1st week : fly from city 0 to city 1 on Monday, and play 6 days and work 1 day. (Although you start at city 0, we could also fly to and start at other cities since it is Monday.) 2nd week : fly from city 1 to city 2 on Monday, and play 3 days and work 4 days. 3rd week : stay at city 2, and play 3 days and work 4 days. Ans = $6 + 3 + 3 = 12$.

Example 2:

Input:

flights = [[0,0,0],[0,0,0],[0,0,0]], days = [[1,1,1],[7,7,7],[7,7,7]]

Output:

3

Explanation:

Since there are no flights that enable you to move to another city, you have to stay at city 0 for the whole 3 weeks. For each week, you only have one day to play and six days to work. So the maximum number of vacation days is 3. Ans = $1 + 1 + 1 = 3$.

Example 3:

Input:

```
flights = [[0,1,1],[1,0,1],[1,1,0]], days = [[7,0,0],[0,7,0],[0,0,7]]
```

Output:

21

Explanation:

One of the best strategies is: 1st week : stay at city 0, and play 7 days. 2nd week : fly from city 0 to city 1 on Monday, and play 7 days. 3rd week : fly from city 1 to city 2 on Monday, and play 7 days. Ans = 7 + 7 + 7 = 21

Constraints:

n == flights.length

n == flights[i].length

n == days.length

k == days[i].length

1 <= n, k <= 100

flights[i][j]

is either

0

or

1

.

0 <= days[i][j] <= 7

Code Snippets

C++:

```
class Solution {  
public:  
    int maxVacationDays(vector<vector<int>>& flights, vector<vector<int>>& days)  
    {  
  
    }  
};
```

Java:

```
class Solution {  
public int maxVacationDays(int[][][] flights, int[][][] days) {  
  
}  
}
```

Python3:

```
class Solution:  
    def maxVacationDays(self, flights: List[List[int]], days: List[List[int]]) ->  
        int:
```

Python:

```
class Solution(object):  
    def maxVacationDays(self, flights, days):  
        """  
        :type flights: List[List[int]]  
        :type days: List[List[int]]  
        :rtype: int  
        """
```

JavaScript:

```
/**  
 * @param {number[][][]} flights  
 * @param {number[][][]} days  
 * @return {number}  
 */
```

```
var maxVacationDays = function(flights, days) {  
};
```

TypeScript:

```
function maxVacationDays(flights: number[][][], days: number[][][]): number {  
};
```

C#:

```
public class Solution {  
    public int MaxVacationDays(int[][][] flights, int[][][] days) {  
          
    }  
}
```

C:

```
int maxVacationDays(int** flights, int flightsSize, int* flightsColSize,  
int** days, int daysSize, int* daysColSize) {  
  
}
```

Go:

```
func maxVacationDays(flights [][]int, days [][][]int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun maxVacationDays(flights: Array<IntArray>, days: Array<IntArray>): Int {  
          
    }  
}
```

Swift:

```
class Solution {  
func maxVacationDays(_ flights: [[Int]], _ days: [[Int]]) -> Int {  
}  
}  
}
```

Rust:

```
impl Solution {  
pub fn max_vacation_days(flights: Vec<Vec<i32>>, days: Vec<Vec<i32>>) -> i32 {  
}  
  
}  
}
```

Ruby:

```
# @param {Integer[][]} flights  
# @param {Integer[][]} days  
# @return {Integer}  
def max_vacation_days(flights, days)  
  
end
```

PHP:

```
class Solution {  
  
/**  
 * @param Integer[][] $flights  
 * @param Integer[][] $days  
 * @return Integer  
 */  
function maxVacationDays($flights, $days) {  
  
}  
}
```

Dart:

```
class Solution {  
int maxVacationDays(List<List<int>> flights, List<List<int>> days) {
```

```
}
```

```
}
```

Scala:

```
object Solution {  
    def maxVacationDays(flights: Array[Array[Int]], days: Array[Array[Int]]): Int  
    = {  
  
        }  
    }  
}
```

Elixir:

```
defmodule Solution do  
    @spec max_vacation_days(flights :: [[integer]], days :: [[integer]]) ::  
        integer  
    def max_vacation_days(flights, days) do  
  
    end  
    end
```

Erlang:

```
-spec max_vacation_days(Flights :: [[integer()]], Days :: [[integer()]]) ->  
    integer().  
max_vacation_days(Flights, Days) ->  
    .
```

Racket:

```
(define/contract (max-vacation-days flights days)  
  (-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))  
    exact-integer?)  
)
```

Solutions

C++ Solution:

```

/*
 * Problem: Maximum Vacation Days
 * Difficulty: Hard
 * Tags: array, dp
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

class Solution {
public:
    int maxVacationDays(vector<vector<int>>& flights, vector<vector<int>>& days)
    {

    }
};

```

Java Solution:

```

/**
 * Problem: Maximum Vacation Days
 * Difficulty: Hard
 * Tags: array, dp
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

class Solution {
public int maxVacationDays(int[][][] flights, int[][][] days) {
    }

}

```

Python3 Solution:

```

"""
Problem: Maximum Vacation Days
Difficulty: Hard
Tags: array, dp

```

```

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:

def maxVacationDays(self, flights: List[List[int]], days: List[List[int]]) ->
int:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
def maxVacationDays(self, flights, days):
"""
:type flights: List[List[int]]
:type days: List[List[int]]
:rtype: int
"""

```

JavaScript Solution:

```

/**
 * Problem: Maximum Vacation Days
 * Difficulty: Hard
 * Tags: array, dp
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

/**
 * @param {number[][]} flights
 * @param {number[][]} days
 * @return {number}
 */
var maxVacationDays = function(flights, days) {

```

```
};
```

TypeScript Solution:

```
/**  
 * Problem: Maximum Vacation Days  
 * Difficulty: Hard  
 * Tags: array, dp  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(n) or O(n * m) for DP table  
 */  
  
function maxVacationDays(flights: number[][], days: number[][]): number {  
  
};
```

C# Solution:

```
/*  
 * Problem: Maximum Vacation Days  
 * Difficulty: Hard  
 * Tags: array, dp  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(n) or O(n * m) for DP table  
 */  
  
public class Solution {  
    public int MaxVacationDays(int[][] flights, int[][] days) {  
  
    }  
}
```

C Solution:

```
/*  
 * Problem: Maximum Vacation Days  
 * Difficulty: Hard
```

```

* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
int maxVacationDays(int** flights, int flightsSize, int* flightsColSize,
int** days, int daysSize, int* daysColSize) {

}

```

Go Solution:

```

// Problem: Maximum Vacation Days
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxVacationDays(flights [][]int, days [][][]int) int {
}

```

Kotlin Solution:

```

class Solution {
    fun maxVacationDays(flights: Array<IntArray>, days: Array<IntArray>): Int {
    }
}

```

Swift Solution:

```

class Solution {
    func maxVacationDays(_ flights: [[Int]], _ days: [[[Int]]]) -> Int {
    }
}

```

Rust Solution:

```
// Problem: Maximum Vacation Days
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
    pub fn max_vacation_days(flights: Vec<Vec<i32>>, days: Vec<Vec<i32>>) -> i32
    {
        }

    }
}
```

Ruby Solution:

```
# @param {Integer[][]} flights
# @param {Integer[][]} days
# @return {Integer}
def max_vacation_days(flights, days)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[][] $flights
     * @param Integer[][] $days
     * @return Integer
     */
    function maxVacationDays($flights, $days) {
        }

    }
```

Dart Solution:

```
class Solution {  
    int maxVacationDays(List<List<int>> flights, List<List<int>> days) {  
        }  
    }
```

Scala Solution:

```
object Solution {  
    def maxVacationDays(flights: Array[Array[Int]], days: Array[Array[Int]]): Int  
    = {  
        }  
    }
```

Elixir Solution:

```
defmodule Solution do  
  @spec max_vacation_days(flights :: [[integer]], days :: [[integer]]) ::  
  integer  
  def max_vacation_days(flights, days) do  
  
  end  
end
```

Erlang Solution:

```
-spec max_vacation_days(Flights :: [[integer()]], Days :: [[integer()]]) ->  
integer().  
max_vacation_days(Flights, Days) ->  
.
```

Racket Solution:

```
(define/contract (max-vacation-days flights days)  
  (-> (listof (listof exact-integer?)) (listof (listof exact-integer?))  
       exact-integer?)  
  )
```