LeetCode 568. Maximum Vacation Days

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:

1. 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.
2. The cities are connected by flights. The flights are represented as a N*N matrix (not necessary 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, flights[i][i] = 0 for all i.
3. You totally have K weeks (each week has 7 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 don't consider the impact of flight time.
4. For each city, you can only have restricted vacation days in different weeks, given an N*K matrix called days representing this relationship. For the value of days[i][j], it represents the maximum days you could take vacation in the city i in the week j.

You're given the flights matrix and days matrix, and you need to output 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:
Ans = 6 + 3 + 3 = 12.

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.
```

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:
Ans = 1 + 1 + 1 = 3.

Since there is no flights 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.
```

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:
Ans = 7 + 7 + 7 = 21

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.
```

Note:

1. N and K are positive integers, which are in the range of [1, 100].
2. In the matrix flights, all the values are integers in the range of [0, 1].
3. In the matrix days, all the values are integers in the range [0, 7].
4. You could stay at a city beyond the number of vacation days, but you should work on the extra days, which won't be counted as vacation days.
5. If you fly from the city A to the 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.
6. We don't consider the impact of flight hours towards the calculation of vacation days.

给定 N 个城市，K 周时间。

矩阵 flights 描述 N 个城市之间是否存在航班通路。

若 flights[i][j] = 1，表示 i 与 j 存在通路，否则表示不存在。特别的，flights[i][i]恒等于 0。

矩阵 days 表示可以在某城市逗留的最长天数。

例如 days[i][j] = k，表示第 i 个城市第 j 周最长可以逗留 k 天。

初始位于 0 号城市，每周可以选择一个能够到达的城市逗留（也可以留在当前城市）。

求最优策略下的最长逗留总天数。

解题思路：

动态规划（Dynamic Programming）

dp[w][c]表示第 w 周选择留在第 c 个城市可以获得的最大总收益

初始令 dp[w][0] = 0, dp[w][1 .. c - 1] = -1

当 dp[w][c] < 0 时，表示第 c 个城市在第 w 周时还不可达。

状态转移方程：

```
for w in (0 .. K)
    for sc in (0 .. N)
        if dp[w][sc] < 0:
            continue
        for tc in (0 .. N)
            if sc == tc or flights[sc][tc] == 1:
                dp[w + 1][tc] = max(dp[w + 1][tc], dp[w][sc] + days[tc][w])
```

```
#include<iostream>
#include<stdio.h>
#include<vector>
#include<unordered_set>
#include<unordered_map>
#include<limits.h>
#include<set>
#include<algorithm>
#include<sstream>
#include<stdlib.h>
#include<string.h>
using namespace std;

int        maxVacationDays(vector<vector<int>>        &flights,
vector<vector<int>> &days)
{
    int N = days.size(); // city number
    int K = days[0].size(); // week
    int dp[K+1][N];
    dp[0][0] = 0;
```

```
    for(int i=0;i<K+1;i++)
        for(int j=0;j<N;j++)
        dp[i][j]=-1;
    dp[0][0] = 0;
    for(int w=0;w<K;w++)
    {
        for(int sc=0;sc<N;sc++)
        {
            if (dp[w][sc]<0) continue;
            for(int tc=0;tc<N;tc++)
            {
                if(sc==tc || flights[sc][tc]==1)
                {
                    dp[w+1][tc]                                    =
max(dp[w+1][tc],dp[w][sc]+days[tc][w]);
                }
            }
        }
    }
    int ans = -1;
    for(int ii=0;ii<N;ii++)
    {
        ans = max(dp[K][ii],ans);
    }
    return ans;
}

int main(int argc,char *argv[])
{
    //[[0,1,1],[1,0,1],[1,1,0]]
    vector<vector<int>> flights = {{0,1,1},{1,0,1},{1,1,0}};
    vector<vector<int>> days = {{7,0,0},{0,7,0},{0,0,7}};

    vector<vector<int>> flights2 = {{0,0,0},{0,0,0},{0,0,0}};
    vector<vector<int>> days2 = {{1,1,1},{7,7,7},{7,7,7}};

    //flights     =     [[0,1,1],[1,0,1],[1,1,0]],     days     =
[[1,3,1],[6,0,3],[3,3,3]]
    vector<vector<int>> flights3 = {{0,1,1},{1,0,1},{1,1,0}};
    vector<vector<int>> days3 = {{1,3,1},{6,0,3},{3,3,3}};
    int ans1 = maxVacationDays(flights,days);
    int ans2 = maxVacationDays(flights2,days2);
    int ans3 = maxVacationDays(flights3,days3);
    cout << ans1 << " "<<ans2 << " "<< ans3 << endl;
```

```
    return 0;
}
```