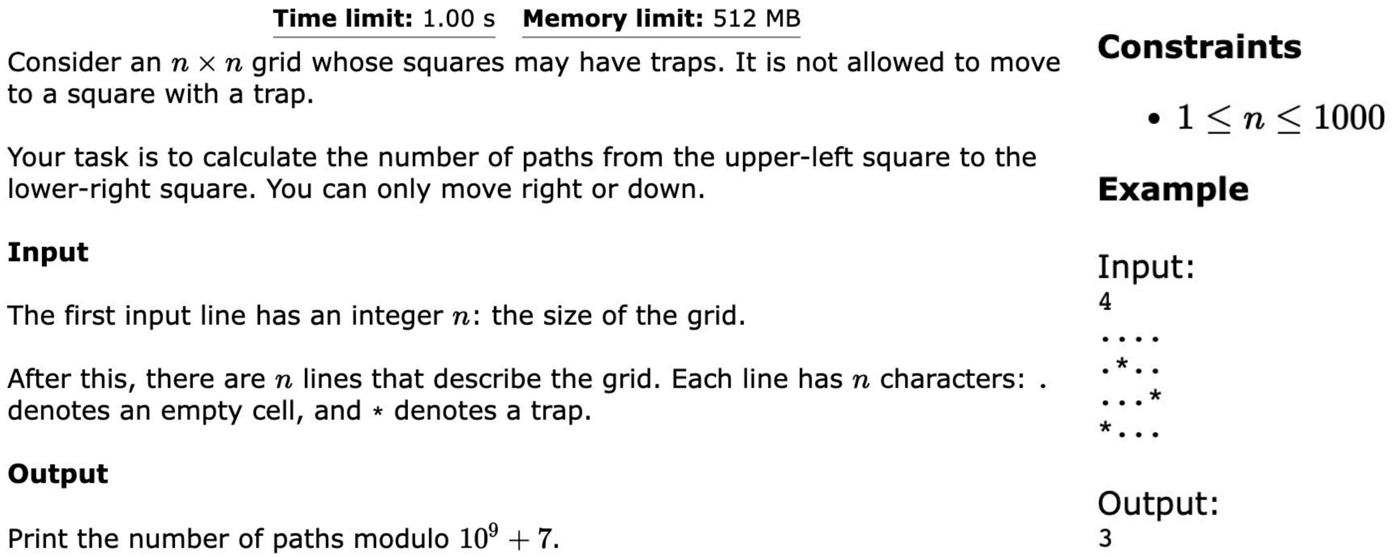
Competitive Coding @ Berkeley

Decal Lecture #9 - More Dynamic Programming

Grades are updated!

- The last pset is due this Wednesday
- Make sure that your student ID is on the sheet!
- Each attendance is worth 1 point, each pset is out of 2 points
- 30% attendance, 70% hw
- Let us know if you added the course late! The first few weeks can be waived
- GRADE SHEET
- Please submit questions/concerns/corrections here

Grid Paths Review



Time limit: 1.00 s **Memory limit:** 512 MB

Consider an $n \times n$ grid whose squares may have traps. It is not allowed to move to a square with a trap.

Your task is to calculate the number of paths from the upper-left square to the lower-right square. You can only move right or down.

Input

The first input line has an integer n: the size of the grid.

After this, there are n lines that describe the grid. Each line has n characters: . denotes an empty cell, and $*$ denotes a trap.

Output

Print the number of paths modulo $10^9 + 7$.

Constraints

- $1 \leq n \leq 1000$

Example

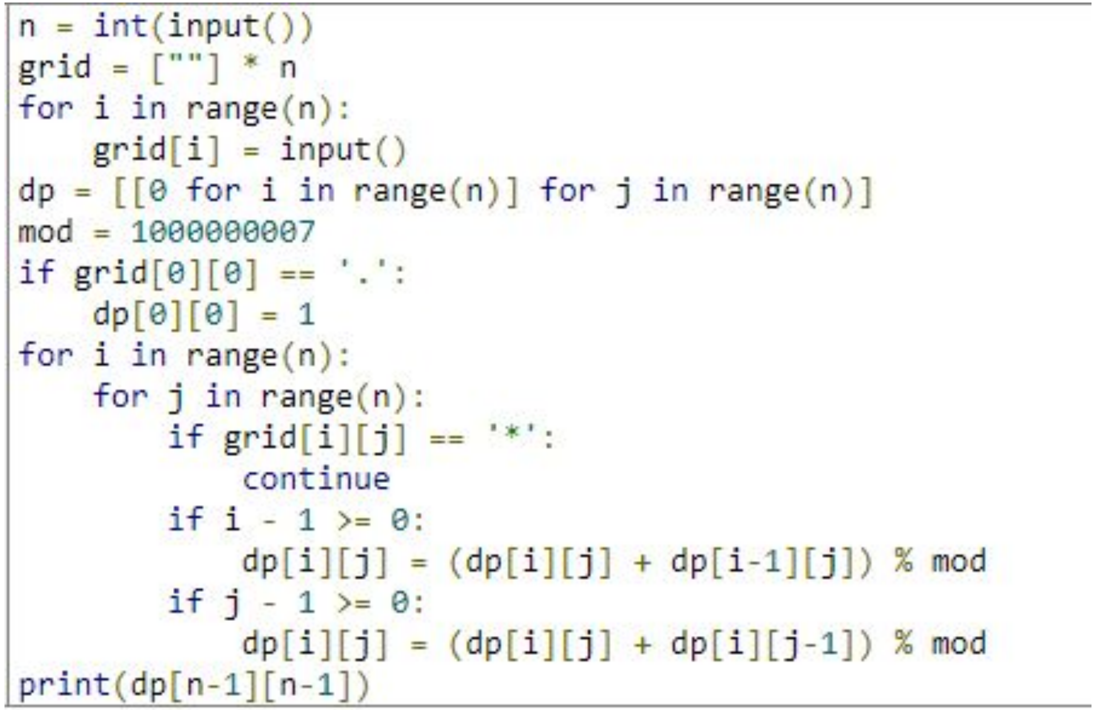
Input:

```
4
....
.*..
...*
*...
```

Output:

```
3
```

Grid Path Implementation

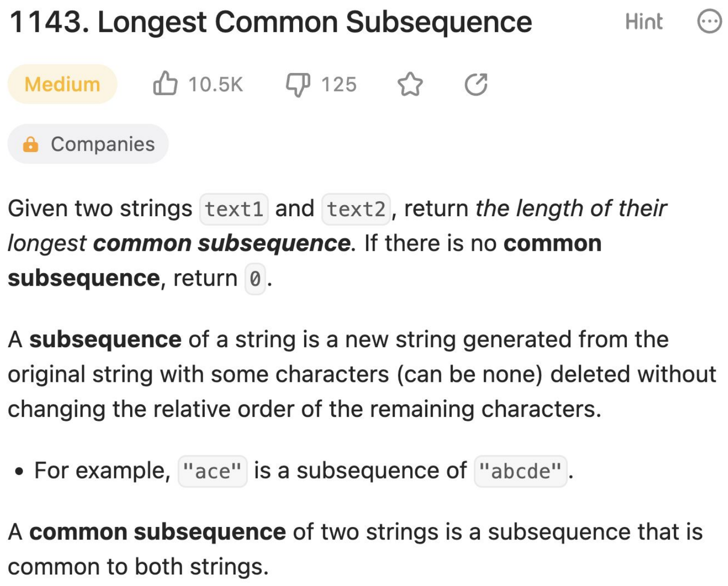


```python
n = int(input())
grid = [""] * n
for i in range(n):
    grid[i] = input()
dp = [[0 for i in range(n)] for j in range(n)]
mod = 1000000007
if grid[0][0] == '.':
    dp[0][0] = 1
for i in range(n):
    for j in range(n):
        if grid[i][j] == '*':
            continue
        if i - 1 >= 0:
            dp[i][j] = (dp[i][j] + dp[i-1][j]) % mod
        if j - 1 >= 0:
            dp[i][j] = (dp[i][j] + dp[i][j-1]) % mod
print(dp[n-1][n-1])
```

Grid Paths Applications

After understanding the solution to the basic grid path problem, there are a lot more problems that can be solved using the same technique, despite not appearing to involve a grid at all.

Longest Common Subsequence



1143. Longest Common Subsequence

Given two strings `text1` and `text2`, return *the length of their longest* ***common subsequence***. If there is no **common subsequence**, return `0`.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, `"ace"` is a subsequence of `"abcde"`.

A **common subsequence** of two strings is a subsequence that is common to both strings.

Longest Common Subsequence

- Like with the DP problems we solved last week, we should think about how to define a **subproblem**, and how to transition between them
- For example, for the Fibonacci problem seen last week, a subproblem was simply the function F(n) - the n’th Fibonacci number

Longest Common Subsequence

For this problem, we’ll define our subproblem F(i, j) to be the longest common subsequence of the first i characters of the first string, and the first j characters of the second string.

The answer to the problem will be F(n, m) (the longest common subsequence of the first and second string; what the problem is asking for)

Longest Common Subsequence

How can we solve for F(i, j) if we can use information from previous subproblems?

In other words, how can we define F(i, j) recursively?

Longest Common Subsequence

Base case: one of the strings is empty

- Return 0 in this case

Otherwise, we have three options for how to “reach” this state:

- F(i - 1, j) - if we ignore the rightmost character in the first string
- F(i, j - 1) - if we ignore the rightmost character in the second string
- F(i - 1, j - 1) - if we add the rightmost character in both strings to the answer. This is only valid if these characters are equal, and we’ll add one to the answer.

Longest Common Subsequence

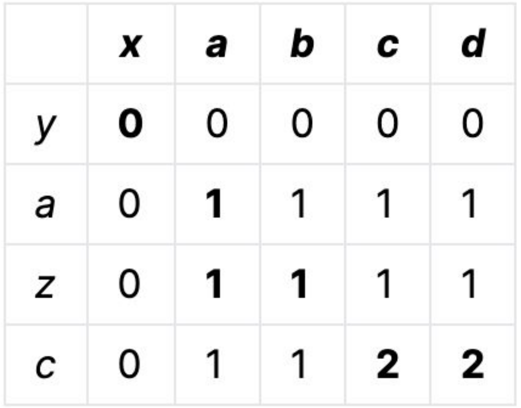
Now that we’ve established these different options for our transitions between subproblems, let’s make our recursive definition for each subproblem:

$F(i, j) = \max(F(i - 1, j), F(i, j - 1)\}$ [if $S[i] != T[j]$]

$F(i, j) = \max(F(i - 1, j), F(i, j - 1), F(i - 1, j - 1) + 1)\}$ [if $S[i] = T[j]$]

Connection to Grid Paths

We can think about the recursive definition we just wrote as a path through a grid: the first case corresponds to reaching the cell from the left, the second case corresponds to reaching the cell from above, and the third case corresponds to reaching the cell diagonally, up and to the left.



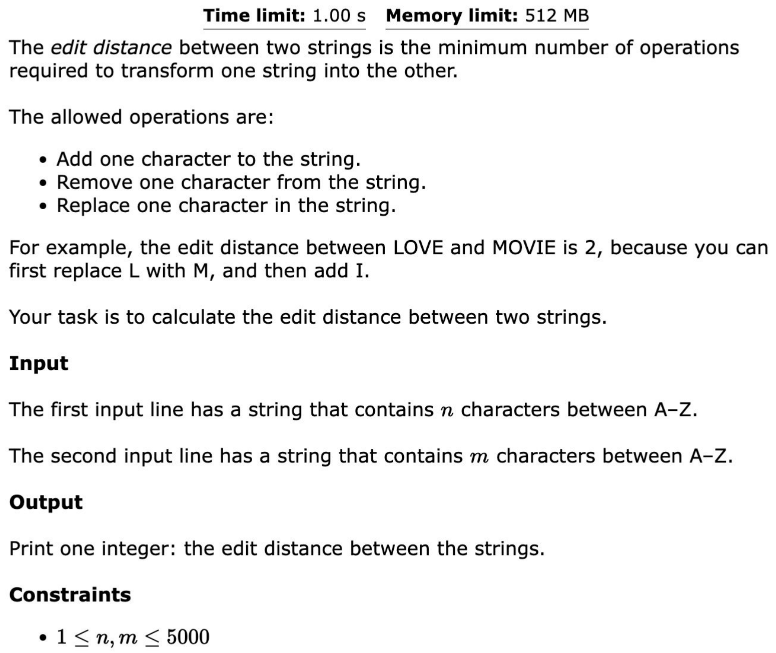
	x	***a***	***b***	***c***	***d***
y	**0**	0	0	0	0
a	0	**1**	1	1	1
z	0	**1**	**1**	1	1
c	0	1	1	**2**	**2**

Time Complexity

There are NM possible subproblems, and each one involves O(1) transitions

So the overall algorithm runs in O(NM).

More Applications of Grid Paths - CSES Edit Distance



Time limit: 1.00 s **Memory limit:** 512 MB

The *edit distance* between two strings is the minimum number of operations required to transform one string into the other.

The allowed operations are:

- Add one character to the string.
- Remove one character from the string.
- Replace one character in the string.

For example, the edit distance between LOVE and MOVIE is 2, because you can first replace L with M, and then add I.

Your task is to calculate the edit distance between two strings.

Input

The first input line has a string that contains n characters between A–Z.

The second input line has a string that contains m characters between A–Z.

Output

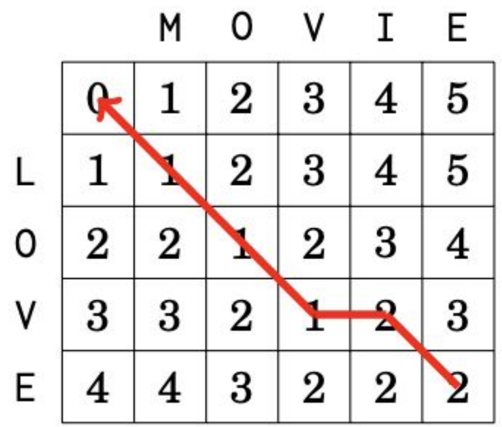
Print one integer: the edit distance between the strings.

Constraints

- $1 \le n, m \le 5000$

Edit Distance

We can use a similar technique to the previous problem - we’ll represent each subproblem as the minimum edit distance between a prefix of the first string and a prefix of the second string.



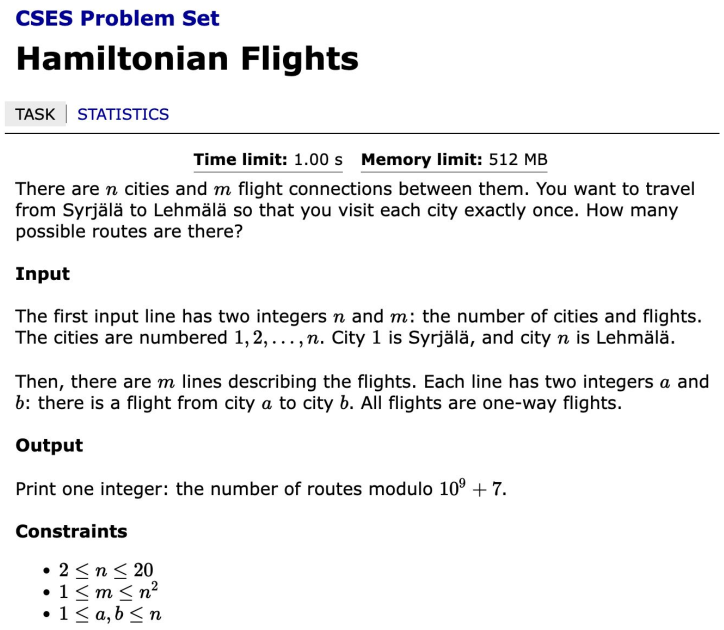
		M	O	V	I	E
	0	1	2	3	4	5
L	1	1	2	3	4	5
O	2	2	1	2	3	4
V	3	3	2	1	2	3
E	4	4	3	2	2	2

Bitmask DP

Now, we’ll look at a new type of DP problem: bitmask DP

These problems are often more difficult than other DP problems, and can often be recognized by unusually low constraints (i.e. N <= 20)

Motivating Problem - CSES Hamiltonian Flights



CSES Problem Set

Hamiltonian Flights

TASK | STATISTICS

Time limit: 1.00 s **Memory limit:** 512 MB

There are n cities and m flight connections between them. You want to travel from Syrjälä to Lehmälä so that you visit each city exactly once. How many possible routes are there?

Input

The first input line has two integers n and m: the number of cities and flights. The cities are numbered $1, 2, \dots, n$. City 1 is Syrjälä, and city n is Lehmälä.

Then, there are m lines describing the flights. Each line has two integers a and b: there is a flight from city a to city b. All flights are one-way flights.

Output

Print one integer: the number of routes modulo $10^9 + 7$.

Constraints

- $2 \le n \le 20$
- $1 \le m \le n^2$
- $1 \le a, b \le n$

Brute Force Solution

This problem can be solved with a complete search approach similar to many of the ones we discussed in Lecture 7:

We can iterate over all the possible orders of cities to visit, and see if an order is valid by checking if every edge on the route is in the graph.

Brute Force Solution

Unfortunately, this solution is too slow - it's $O(N!)$, which will only work for $N <= 11$.

Can we do better?

Dynamic Programming Solution

Like in the grid paths examples, we need to find a **subproblem**, which is slightly more difficult for this example.

Dynamic Programming Solution

We’ll define our subproblem to be the number of valid flight paths, given that we’ve already visited a certain subset of cities, and are at a certain city

Then, the answer will be F(empty set, city 1) - the number of valid flight paths at the start of the problem, when we haven’t visited any cities yet, and are at the start city.

Dynamic Programming Solution

The base case will be F(all cities, city n) = 1 - one way to visit every city if we’ve already visited all of them

We can transition between states by iterating over all possible edges we can take, adding the new city to the subset of cities.

How can we represent this subset of cities in a compact way?

Dynamic Programming Solution

Similar to our method for iterating over all subsets of a list (from Lecture 7, the Complete Search lecture), we can represent a subset of cities as a length-N binary number, with each digit being 1 if we've already visited this city, and 0 otherwise.

By doing this, there will be O(N * 2^N) states, and each transition will take O(N) time, because we have to check up to O(N) edges originating at the current vertex, so the overall time complexity is O(N^2 * 2^N)

Problemset Problem A

- https://codeforces.com/problemset/problem/1042/B

Berland shop sells n kinds of juices. Each juice has its price c_i. Each juice includes some set of vitamins in it. There are three types of vitamins: vitamin "A", vitamin "B" and vitamin "C". Each juice can contain one, two or all three types of vitamins in it.

Petya knows that he needs all three types of vitamins to stay healthy. What is the minimum total price of juices that Petya has to buy to obtain all three vitamins? Petya obtains some vitamin if he buys at least one juice containing it and drinks it.

Input

The first line contains a single integer n $(1 \le n \le 1\,000)$ — the number of juices.

Each of the next n lines contains an integer c_i $(1 \le c_i \le 100\,000)$ and a string s_i — the price of the i-th juice and the vitamins it contains. String s_i contains from 1 to 3 characters, and the only possible characters are "A", "B" and "C". It is guaranteed that each letter appears no more than once in each string s_i. The order of letters in strings s_i is arbitrary.

Output

Print -1 if there is no way to obtain all three vitamins. Otherwise print the minimum total price of juices that Petya has to buy to obtain all three vitamins.

Examples

input

```
4
5 C
6 B
16 BAC
4 A
```

output

```
15
```

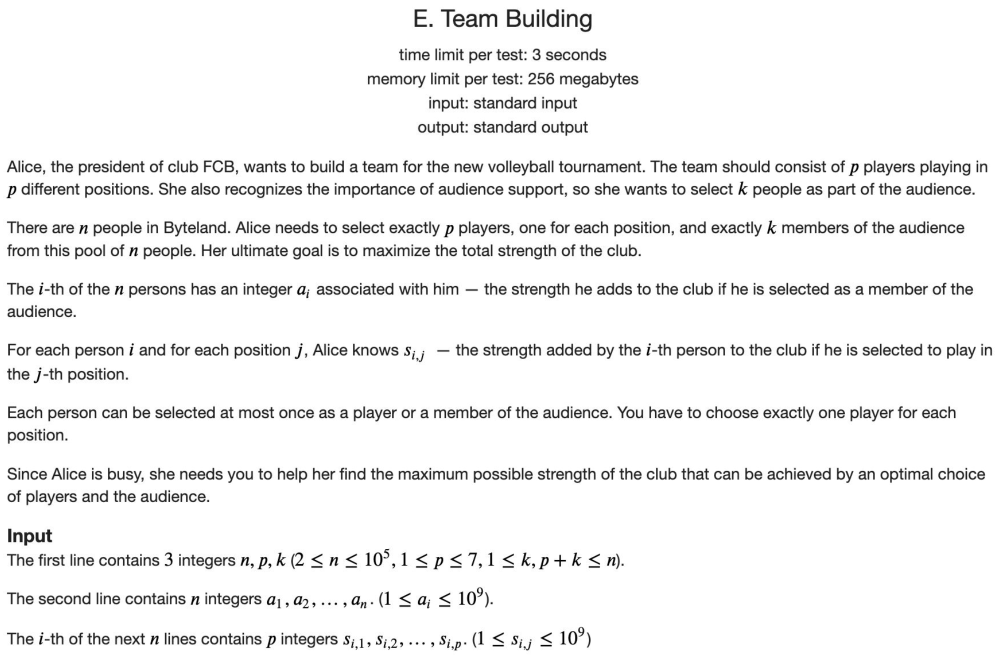
input

```
2
10 AB
15 BA
```

output

```
-1
```

Example Problem - CF Team Building



E. Team Building

time limit per test: 3 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

Alice, the president of club FCB, wants to build a team for the new volleyball tournament. The team should consist of p players playing in p different positions. She also recognizes the importance of audience support, so she wants to select k people as part of the audience.

There are n people in Byteland. Alice needs to select exactly p players, one for each position, and exactly k members of the audience from this pool of n people. Her ultimate goal is to maximize the total strength of the club.

The i-th of the n persons has an integer a_i associated with him — the strength he adds to the club if he is selected as a member of the audience.

For each person i and for each position j, Alice knows $s_{i,j}$ — the strength added by the i-th person to the club if he is selected to play in the j-th position.

Each person can be selected at most once as a player or a member of the audience. You have to choose exactly one player for each position.

Since Alice is busy, she needs you to help her find the maximum possible strength of the club that can be achieved by an optimal choice of players and the audience.

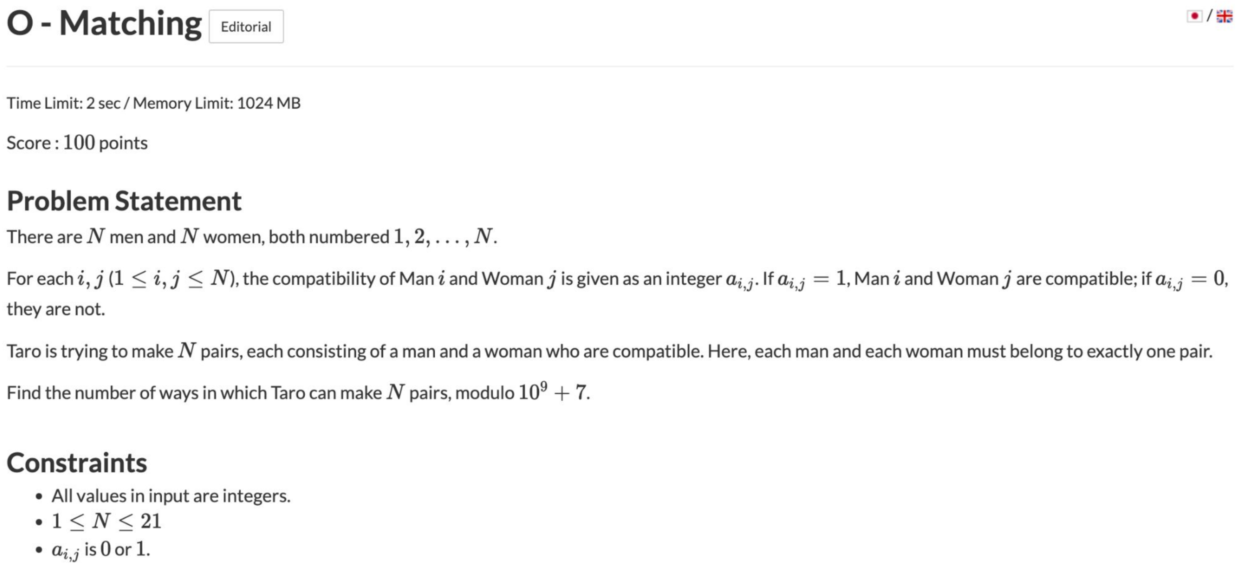
Input

The first line contains 3 integers n, p, k ($2 \le n \le 10^5, 1 \le p \le 7, 1 \le k, p + k \le n$).

The second line contains n integers $a_1, a_2, \ldots, a_n$. ($1 \le a_i \le 10^9$).

The i-th of the next n lines contains p integers $s_{i,1}, s_{i,2}, \ldots, s_{i,p}$. ($1 \le s_{i,j} \le 10^9$)

Example Problem - Matching



O - Matching

Editorial

Time Limit: 2 sec / Memory Limit: 1024 MB

Score : 100 points

Problem Statement

There are N men and N women, both numbered $1, 2, \ldots, N$.

For each i, j ($1 \leq i, j \leq N$), the compatibility of Man i and Woman j is given as an integer $a_{i,j}$. If $a_{i,j} = 1$, Man i and Woman j are compatible; if $a_{i,j} = 0$, they are not.

Taro is trying to make N pairs, each consisting of a man and a woman who are compatible. Here, each man and each woman must belong to exactly one pair.

Find the number of ways in which Taro can make N pairs, modulo $10^9 + 7$.

Constraints

- All values in input are integers.
- $1 \leq N \leq 21$
- $a_{i,j}$ is 0 or 1.