Bitmasking and Dynamic Programming | Set 1 (Count ways to assign unique cap to every person)

Consider the below problems statement.

There 100 different types of caps each having a unique id from 1 to 100. Also, there 'n' persons each having a collection of variable number of caps. One day all of these persons decide to go in a party wearing a cap but to look unique they decided that none them will wear the same type of cap. So, count the total number of arrangements or ways such that none of them is wearing same type of cap.

Constraints: 1 <= n <= 10 Example:

```
First line contains value of n, next n lines contain collections
of all the n persons.
Input:
3
5 100 1     // Collection of first person.
2           // Collection of second person.
5 100       // Collection of second person.

Output:
4
Explanation: All valid possible ways are (5, 2, 100),  (100, 2, 5),
            (1, 2, 5) and  (1, 2, 100)
```

Since, number of ways could be large, so output modulo 1000000007

We strongly recommend you to minimize your browser and try this yourself first.
A **Simple Solution** is to try all possible combinations. Start by picking first element from first set, marking it as visited and recur for remaining sets. It is basically a Backtracking based solution.

A **better solution is to use Bitmasking and DP**. Let us first introduce Bitmasking.

What is Bitmasking?
Suppose we have a collection of elements which are numbered from 1 to N. If we want to represent a subset of this set than it can be encoded by a sequence of N bits (we usually call this sequence a "mask"). In our chosen subset the i-th element belongs to it if and only if the i-th bit of the mask I set i.e., it equals to 1. For example, the mask 10000101 means that the subset of the set [1... 8] consists of elements 1, 3 and 8. We

know that for a set of N elements there are total 2^N subsets thus 2^N masks are possible, one representing each subset. Each mask is in fact an integer number written in binary notation.

Our main methodology is to assign a value to each mask (and, therefore, to each subset) and thus calculate the values for new masks using values of the already computed masks. Usually our main target is to calculate value/solution for the complete set i.e., for mask 11111111. Normally, to find the value for a subset X we remove an element in every possible way and use values for obtained subsets X'_1, X'_2... ,X'_k to compute the value/solution for X. This means that the values for X'_i must have been computed already, so we need to establish an ordering in which masks will be considered. It's easy to see that the natural ordering will do: go over masks in increasing order of corresponding numbers. Also, We sometimes, start with the empty subset X and we add elements in every possible way and use the values of obtained subsets X'_1, X'_2... ,X'_k to compute the value/solution for X.

We mostly use the following notations/operations on masks:
bit(i,mask) – the i-th bit of mask
count(mask) – the number of non-zero bits in mask
first(mask) – the number of the lowest non-zero bit in mask
set(i, mask) – set the ith bit in mask
check(i, mask) – check the ith bit in mask

How is this problem solved using Bitmasking + DP?
The idea is to use the fact that there are upto 10 persons. So we can use a integer variable as a bitmask to store which person is wearing cap and which is not.

```
Let i be the current cap number (caps from 1 to i-1 are already
processed). Let integer variable mask indicates the the persons w
earing and not wearing caps.  If i'th bit is set in mask, then
i'th person is wearing a cap, else not.

                     // consider the case when ith cap is not included
                     // in the arrangement
countWays(mask, i) = countWays(mask, i+1) +

                    // when ith cap is included in the arrangement
                    // so, assign this cap to all possible persons
                    // one by one and recur for remaining persons.
                    ∑ countWays(mask | (1 << j), i+1)
                       for every person j that can wear cap i

Note that the expression "mask | (1 << j)" sets j'th bit in mask.
```

```
And a person can wear cap i if it is there in the person's cap list
provided as input.
```

If we draw the complete recursion tree, we can observer that many subproblems are solved again and again. So we use Dynamic Programming. A table dp[][] is used such that in every entry dp[i][j], i is mask and j is cap number.

Since we want to access all persons that can wear a given cap, we use an array of vectors, capList[101]. A value capList[i] indicates the list of persons that can wear cap i.

Below is the implementation of above idea.

C/C++

```
// C++ program to find number of ways to wear hats
#include<bits/stdc++.h>
#define MOD 1000000007
using namespace std;

// capList[i]'th vector contains the list of persons having a cap with id i
// id is between 1 to 100 so we declared an array of 101 vectors as indexing
// starts from 0.
vector<int> capList[101];

// dp[2^10][101] .. in dp[i][j], i denotes the mask i.e., it tells that
// how many and which persons are wearing cap. j denotes the first j caps
// used. So, dp[i][j] tells the number ways we assign j caps to mask i
// such that none of them wears the same cap
int dp[1025][101];

// This is used for base case, it has all the N bits set
// so, it tells whether all N persons are wearing a cap.
int allmask;

// Mask is the set of persons, i is cap-id (OR the
// number of caps processed starting from first cap).
long long int countWaysUtil(int mask, int i)
{
    // If all persons are wearing a cap so we
    // are done and this is one way so return 1
    if (mask == allmask) return 1;

    // If not everyone is wearing a cap and also there are no more
    // caps left to process, so there is no way, thus return 0;
    if (i > 100) return 0;

    // If we already have solved this subproblem, return the answer.
    if (dp[mask][i] != -1) return dp[mask][i];

    // Ways, when we don't include this cap in our arrangement
    // or solution set.
    long long int ways = countWaysUtil(mask, i+1);

    // size is the total number of persons having cap with id i.
    int size = capList[i].size();

    // So, assign one by one ith cap to all the possible persons
    // and recur for remaining caps.
```

```
    for (int j = 0; j < size; j++)
    {
        // if person capList[i][j] is already wearing a cap so continue as
        // we cannot assign him this cap
        if (mask & (1 << capList[i][j])) continue;

        // Else assign him this cap and recur for remaining caps with
        // new updated mask vector
        else ways += countWaysUtil(mask | (1 << capList[i][j]), i+1);
        ways %= MOD;
    }

    // Save the result and return it.
    return dp[mask][i] = ways;
}

// Reads n lines from standard input for current test case
void countWays(int n)
{
    //----------- READ INPUT -------------------------
    string temp, str;
    int x;
    getline(cin, str);  // to get rid of newline character
    for (int i=0; i<n; i++)
    {
        getline(cin, str);
        stringstream ss(str);

        // while there are words in the streamobject ss
        while (ss >> temp)
        {
            stringstream s;
            s << temp;
            s >> x;

            // add the ith person in the list of cap if with id x
            capList[x].push_back(i);
        }
    }
    //------------------------------------------------

    // All mask is used to check of all persons
    // are included or not, set all n bits as 1
    allmask = (1 << n) - 1;

    // Initialize all entries in dp as -1
    memset(dp, -1, sizeof dp);

    // Call recursive function count ways
    cout << countWaysUtil(0, 1) << endl;
}

// Driver Program
int main()
{
     int n;   // number of persons in every test case
     cin >> n;
     countWays(n);
     return 0;
}
```

Run on IDE