Codeforces

BITMASK

AJAY KUMAR (20JE0073)

Bitwise Operation

Function	Operation
__builtin_popcount(x)	Return the number of ones
__builtin_popcountll(x)	Same above but for long
__builtin_clz	Leading zeros
__builtin_ctz	Trailing zero

Compute the biggest power of 2 that is a divisor of x. Example

1<< __builtin_ctz(x)

Compute the smallest power of 2 that is not smaller than x Example

1<<(32- __builtin_clz(x-1))

But this is UB(undefined behavior for x<=1 so you'll often need an if for x == 1.

Motivation behind bitsets

Consider this problem:

There are N<=5000 workers.

- Each worker is available during some days of this month (which has 30 days).
- For each worker, you are given a set of numbers, each from interval [1,30] representing his/her availability. You need to assign an important project to two workers but they will be able to work on the project only when they are both available. Find two workers that are best for the job – maximize the number of days when both these workers are available.
- You can compute the intersection of two workers (two sets) in O(30) by using e.g. two pointers for two sorted sequences. Doing that for every pair of workers is $O(n^2 \times 30)$

- We can think of availablity of the worker as a binary string of lenth 30 and can be stored in single integer
- Int with this representation. We can count the intersection size in O(1) by using the __builtin_popcount(x[i] & x[j])
- The complexity become $O(N^2)$ fast enough

We can split D days into convenient parts of size 64 and store the availability of a single worker in an array of D/64 nsigned long longs. Then, the intersection can be computed in O(D/64) and overall complexity is o(N^2*D/64)

Code

```
Int num_workers;
cin>>num_workers;
Int mask=0;
for(int j=0;j<num_workers;++j){
   Int day;
   cin>>day;
   mask=(mask | (1<<day));
}
Int max_days=0;

for(int i=0;i<n;i++){
   for(int j=i+1;j<n;j++){
       Int intersection= (mask[I]& masks[j]);
       Int common_days= __buitin_popcount(intersections);
       max_days=max(max_days,common_days);
   }
}
```

Input

5

4

1 2 3 4

3

5 6 7

5

1 4 3 5 6

2

3 7

6

1 3 5 7 9 4

Predict the output

4

Instead of storing as a vector we are storing the mask of the vector by making the bit on of the array

Subset Generation

[2 4 5]
0→000-[]
1→001- [2]
2→010- [4]
3→011- [2 4]
4→100- [5]
5→101- [2 5]
6→110- [4 5]
7→111- [2 4 5]

```
vector<vector<int>>subsets(vector<int>&
nums){
    int n=nums.size();
    int subset_ct= (1<<n);
    vector<vector<int>>subsets;
    for(int mask=0;mask<subset_ct;++mask){
        vector<int>subset;
        for(int i=0;i<n;i++){
            if(mask&(1<<i)!=0){
                subset.push_back(nums[i]);
            }
        }
        subsets.push_back(subset);
    }
    return subsets;
}
```

Code explanation
For The array size we are making the bit array

For n=3 we are taking 111
To make the subset of we are iterating to the 8 and making the possible
//
Time complexity O(n*2^n)

// by recursion time complexity is O(2^n)

Problem Statement

Let there be N workers and N jobs. Any worker can be assigned to perform any job, incurring some cost that may vary depending on the work-job assignment. It is required to perform all jobs by assigning exactly one worker to each job and exactly one job to each agent in such a way that the total cost of the assignment is minimized.

Input Format

Number of workers and job: N

Cost matrix C with dimension N*N where C(j,i) is the cost incurred on assigning jth Person to ith Job.

Sample Input

4

[9 2 7 8

6 4 3 7

5 8 1 8

7 6 9 4]

Sample Output

13

Constraints

N <= 20

- Approach to this problem: Let $J_1, J_{2,} J_3, J_4$….. J_N be the jobs done by the $P_2, P_4, P_1, P_5, \ldots P_n \ldots$. P_6 in the order.
- if P_1 does the job J_x then total remaing jobs are J-1 and so second person have to chose one of remaining jobs and so on.
- So N*(N-1)*(N-2)*…..(1) so total will be N! ways for all jobs to be done.

- How to identify the subproblem
- Let $C_{j,i}$ is the cost incurred during the jth person does the ith job
- Think for a min

- Continued After the image

$$
\begin{tikzcd}
 & \left(1, \{P_1, P_2, \ldots, P_N\}\right) \arrow[dl, "C_{11}"'] \arrow[dr, "C_{21}"] & \\
\left(2, \{P_2 P_3 \ldots P_N\}\right) \arrow[dr, "C_{22}"'] & & \left(2, \{P_1 P_3 \ldots P_N\}\right) \arrow[dl, "C_{12}"] \\
 & \boxed{\left(3, \{P_3 P_4 \ldots P_N\}\right)} &
\end{tikzcd}
$$

- So we have identified the overlapping subproblem Now Our goal is clear we have to apply the dp here.But How we will represent our states ??
- Here we can see the people are of subsets so we can make a mask of the subsets of people
- So dp(j,mask): min Cost
- Job: 1 to n ≈ st people represented by mask

Start with dp(1,$(111111)_2$) -> (1<<(n)-1)

dp(i,mask)= c_{ji} +dp(i+1,mask | turn off the jth bit)

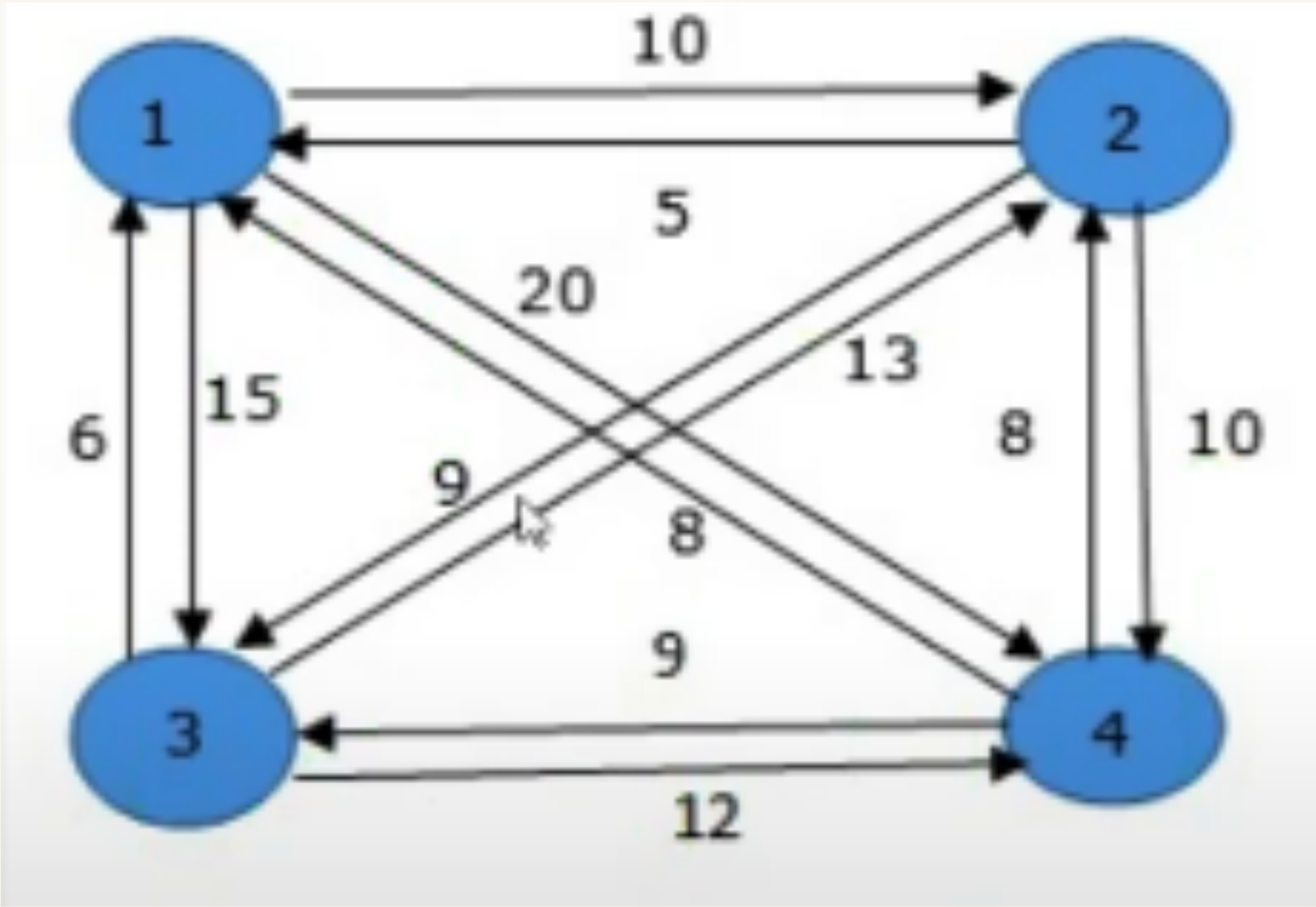
- For valid j i.e jth bit is on find the min
- So overall complexity is $O(N*2^N)$

```cpp
int cost[21][21];
int dp[21][(1<<21)];

int solve(int i,int mask,int& n){
    if(i==n+1)return 0;
    if(dp[i][mask]!=-1){
        return dp[i][mask];
    }
    int ans=INT_MAX;
    for(int j=0;j<n-1;j++){
        if(mask&(1<<j)){
            ans=min(ans,cost[j][i]+solve(i+1,(mask&(1<<j)),n);
        }
    }
    return dp[i][mask]=ans;
}
```

Travelling salesman Problem

Given a list of cities and the different between each pair of cities, what is shortest possible route that visits each cities and return to the origin city.



Problem

Read problems statements in Mandarin Chinese, Russian and Vietnamese as well.

Sereja has an undirected graph on **N** vertices. There are edges between all but **M** pairs of vertices.

A permutation **p** on the vertices of the graph is represented as **p[1], p[2], … , p[N]** such that for all **i**, **p[i]** is a vertex of the graph. A permutation is called connected if there is an edge between vertices **p[i]** and **p[i+1]** for all natural numbers **i** less than **N**. Sereja wants to know the number of connected permutations on the graph vertices.

Input

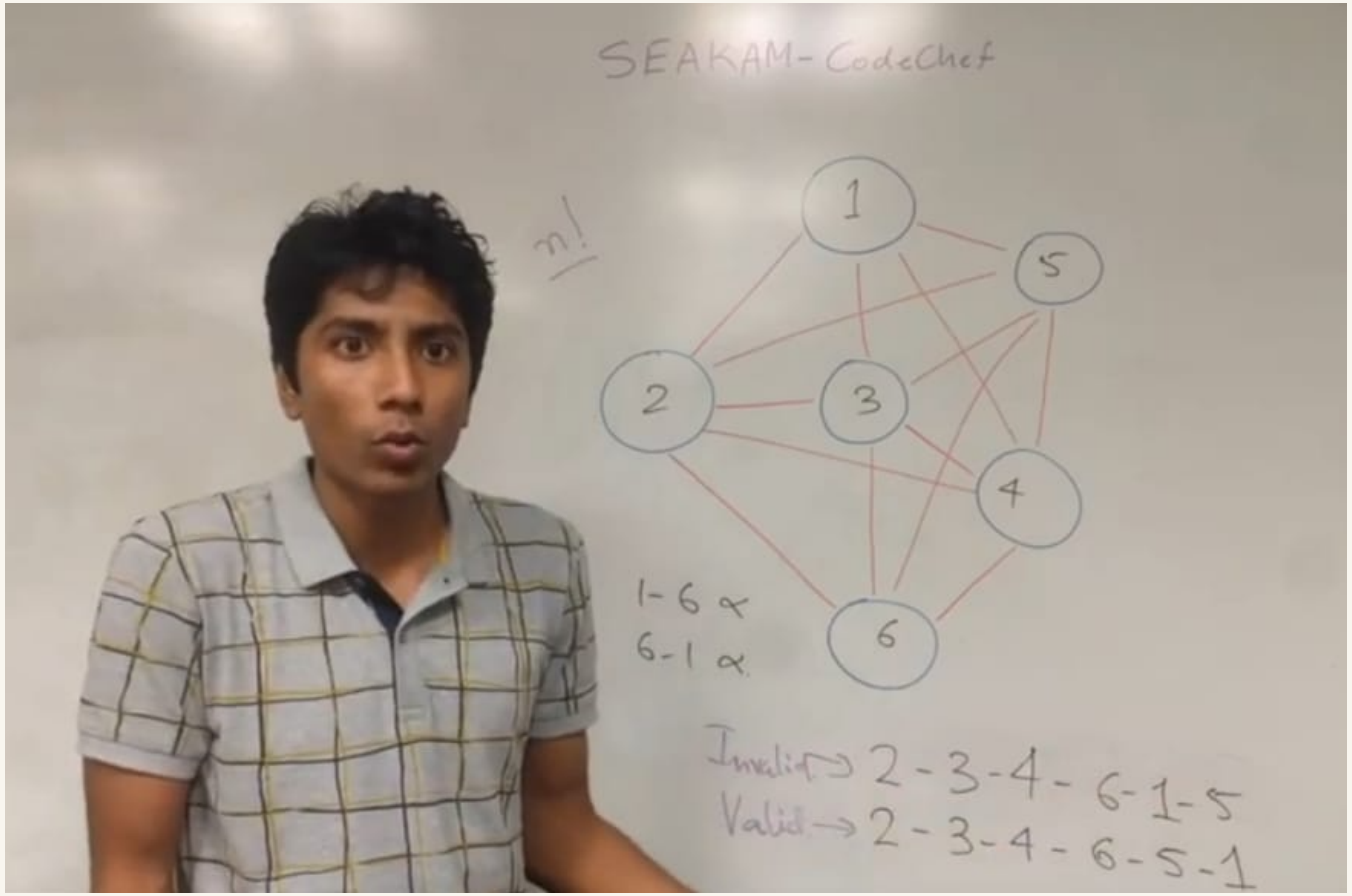
First line of input contains a single integer **T**, denoting the number of test cases. **T** tests follow. First line of each test case contains two integers, **N** and **M**. **M** lines follow, each containing a pair of indices of vertices, indicating that those vertices are **not** connected by an edge.

Output

For each test case, output one number — the answer for the problem modulo 10^9+7.

Constraints

- $1 \le T \le 10$
- $1 \le N \le 10^5$
- $0 \le M \le 7$



So we have to find the valid permutation of the graph. Where
2 3 4 6 1 5 is invalid permutation and
2 3 4 6 5 1 is valid permutation

By Inclusion-Exclusion Principle
1- 6 x
6- 1 x
p(g)= n!- p(1-6)-p(6-1)+p(1-6,6-1)
So it will pss the test but it is lot easier to solve with dp with bitmask

A blog on dp with bitmask

Hi guys, so someone asked me this problem from CSES problemset.

- While solving it I realized there are plenty of interesting ways to solve this problem and for a beginner who is starting with dynamic programming,
- it is a nice exercise for them to go through all the ways as solving this problem as it gives a decent amount of exposure to the standard techniques and one might be able to get a lot of value out of it if done sincerely.
- So I thought it would be a good idea to dot all these ideas/solutions in one place, so one doesn't have to search through a thousand places to learn them.
- it's just a collection of somewhat neatly written codes and ideas for beginners who are starting out with dp.

NOTATION:

- $$B_i(K) = \begin{cases} 1 & i\text{th bit of } K \text{ is set} \\ 0 & \text{otherwise} \end{cases}$$
- $C(K, L)$ → Number of ways to go from a mask K to a mask L on filling the unoccupied cells in mask K.
- F_i → i th Fibonacci number.
- ϕ → Golden Ratio

$\mathcal{O}(2^N NM)$ Solution:

$$DP[i][j][k] \rightarrow \text{ solution when we're currently at the point } (i, j) \text{ and first } i \text{ bits} \\ \text{of } k \text{ correspond to } j\text{th column and rest of the bits belong to } j-1 \text{ column}$$

$$DP[i][j][k] = \begin{cases} DP[i-1][j][k \oplus 2^i] + DP[i-1][j][k \oplus 2^{i-1}] & B_i(k) \text{ and } B_{i-1}(k) \\ DP[i-1][j][k \oplus 2^i] & \text{otherwise} \end{cases}$$

Final answer will be $DP[N-1][M][0]$. Since transitions a re straight forward I've dropped a couple of dimensions in the code. For details refer to USACO tutorial

```cpp
#include "bits/stdc++.h"
using namespace std ;
const int M =1e9+7 ;
int main(){
  int n,m ;
  cin >> n >> m ;
  vector<int>dp(1<<n) ;
  dp[(1<<n)-1]=1 ;
  for(int j=0;j<=m;j++){
    for(int i=0;i<n;i++){
      vector<int>DP(1<<n) ;
      for(int k=0;k<(1<<n);k++){
        (DP[k]+=dp[k^(1<<i)])%=M ;
        if(i&&(k>>i&1)&&(k>>(i-1)&1))
          (DP[k]+=dp[k^(1<<(i-1))])%=M ;
      }
      swap(dp,DP) ;
    }
  }
  cout<<dp[0] ;

}
```