Bitmask DP

In some problems, you need to know which elements are taken till now.
↳ Demand of the transition.

ways
↳ keep the whole moves till now
↳ $O(N!)$ states

↳ keep a bitmask of elements taken / free
→ $O(2^N)$ states

$N \leq 20$ we can use bitmask DP

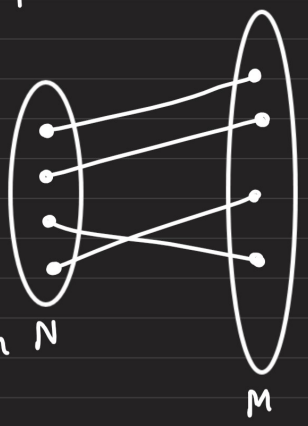
Bitmasks

	a_0	a_1	a_2	a_3	a_4	a_5
		X	X			
mask:	1	0	0	1	1	

dp (mask, _)
↳ dp (i, mask) / dp (mask)
↳ dp (mask, k) [say you need to take exactly k elements]
↳ dp ($mask_1$, $mask_2$)

Application form I

Bipartite matching



Q1. There are N Ranks and M students in a class. The i^{th} student gets happy[i][j] on getting rank j. Atmost one student can be assigned to a rank.

Total happiness is the sum of happiness of each student in a class. Find maximum total happiness.

$1 <= N <= 15$

$1 <= M <= 50$

$0 <= happy[i][j] <= 10^5$

soln:

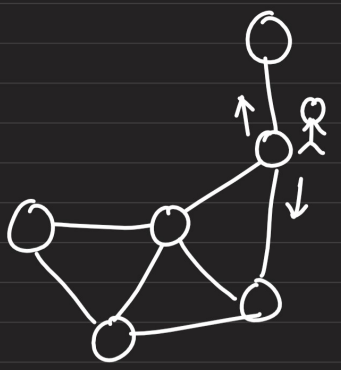
```
int n,m;
vector<vector<lli>> happy;
int dp[52][2000];

//no. of states = (M * 2^N)
//no. of transitions = N
//Time complexity is O( (M *2^N) * n) = 50 * 1024 * 10
lli rec(int i, lli mask) //i = student number, initially mask = 1111.. all ranks are available to be distributed
{
    if(i==m+1) return 0;
    if(mask==0) return 0;

    if(dp[i][mask]!=-1) return dp[i][mask];

    lli ans = rec(i+1,mask); //with out giving any rank
    for(int j=0;j<n;j++)
    {
        if(mask&(1<<j)){
            ans = max( ans,happy[i][j] + rec( i+1,(mask^(1<<j)) ) );
        }
    }
    return dp[i][mask] = ans;
}
```

```
void solve()
{
    cin>>n>>m;
    memset(dp,-1,sizeof(dp));
    //n = ranks, and m = no. of students
    happy.assign(m+1,vector<lli>(n+1,0));
    for(int i=0;i<m;i++)
    {
        for(int j=0;j<n;j++)
        {
            cin>>happy[i][j];
        }
    }
    lli mask = (1<<n) - 1;
    cout<<rec(0,mask)<<'\n';
}
```

Application Form - II

Hamiltonian Walk

you need to visit every node exactly once starting from some node, say i

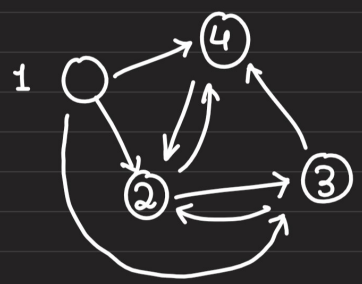


Q A city is modelled using a graph with N nodes & K edges. The nodes are numbered from 1 to N.

There is a person standing at node 1, wants to visit a friend at city N. He wants to visit every city exactly once. Find the number of paths that exists of this form.

$N <= 20$

Soln: paths available here



1, 2, 3, 4
1, 3, 2, 4
} Ans = 2

Recursive

```
int ways (int cur_node, vector<int> already_visited)
{
    if (path.len == N)
                    ——→ check valid

    // Try all moves
}
```

↳ give no. of ways to move from cur. node to N without again visiting the visited nodes.

Bitmask DP (mask of visited node)

Varients

1. Shortest hamiltonian walk from 1 → N

```
int n,m; //n<=20
vector<vector<ii>> g;
vector<vector<lli>> dp;
//O( (N+M) * (1<<N) )
//Hamiltonian graph problems cannot be solved in polynomial type
lli shortest(lli nn, lli mask)
{
    //if we want to get shortest for any hamiltonian path from i, we can just change this base case
    if(mask+1==(1<<n))
    {
        //return 0 always if last node is not restructed to node n
        if(nn==n) return 0;
        else return INF;
    }
    if(dp[nn][mask]!=-1) return dp[nn][mask];

    lli ans = INF;
    for(auto v: g[nn])
    {
        if(mask&(1<<(v.F-1))) continue;
        ans = min(ans , v.S + shortest( v.F, (mask|(1<<(v.F-1))) ) );
    }
    return dp[nn][mask] = ans;
}
```

```
void solve()
{
    cin>>n>>m;
    g.resize(n+1);
    dp.assign(n+1,vector<lli>((1<<n)+5,-1));
    int u,v,w;
    for(int i=0;i<m;i++)
    {
        cin>>u>>v>>w;
        g[u].push_back({v,w});
    }
    cout<<shortest(1,1)<<'\n'; //starting from 1 node ans 1 node is already visited
}

int main()
{
    solve();
}
```

2. No. of Hamiltonian walks [from any vertex i to any vertex j]

```cpp
int n,m; //n<=20
vector<vector<lli>> g,dp;

lli ways(int nn, int mask) //no. of hamiltonian walks from nn with visited as mask
{
    if(mask+1==(1<<n)) return 1; //as we have reached all node

    if(dp[nn][mask]!=-1) return dp[nn][mask];

    lli ans = 0;
    for(auto v: g[nn])
    {
        if(mask&(1<<(v-1))) continue;
        ans += ways( v, (mask|(1<<(v-1))) );
    }
    return dp[nn][mask] = ans;
}
```

3. Number of Hamiltonian cycles

Hamiltonian cycle from $X \to X$ is equal to no. of
" " " $Y \to Y$

```cpp
using lli = long long int;
using ii = pair<lli,lli>;

int n,m, init; //n<=20
vector<vector<lli>> g,dp;
map<ii,int> freq;

//Find number of hamiltonian Cycles
lli ways(int nn, int mask) //no. of hamiltonian walks from nn to 0 with visited as mask
{
    if(mask+1==(1<<n)){
        //if there is an edge from current node to initial node, then return 1
        if(freq[{nn,init}]) return 1; //as we have reached all node
        else return 0;
    }

    if(dp[nn][mask]!=-1) return dp[nn][mask];

    lli ans = 0;
    for(auto v: g[nn])
    {
        if(mask&(1<<(v-1))) continue;
        ans += ways( v, (mask|(1<<(v-1))) );
    }
    return dp[nn][mask] = ans;
}
```

4. No. of simple path

$$\frac{1}{2} \sum_{i=1,\ \underbrace{count(mask) \geq 2}_{\text{atleast 2 node in a single path}}}^{n} Dp(i, mask)$$

Dp(i, mask)
↳ number of simple path from i with mask as visited.

* { divided by $\frac{1}{2}$ as for undirected graph, every path will be calculated twice
  $x \to z \to y$ } counted in DP(x, {z, y}) & DP(y, {z, z})

// Base case

If all nodes are visited, $\underbrace{dp(nn, mask) = 0}_{\text{we want atleast two nodes}}$

// Transitions

Dp(i, mask) ⟶ __builtin_popcountll(mask) ≥ 2
- → we can either stop at node i
- → or we can move

ans = 1 + dp(neigh, updated mask)

5. No. of Simple cycles

```
lli n,m;
vector<vector<int>> g;
vector<vector<lli>> dp;
lli edges[21][21];
lli init = 1;
lli count_cycles(lli nn, lli mask)
{

    if(dp[nn][mask]!=-1) return dp[nn][mask];

    lli ans = ( (__builtin_popcountll(mask) > 2) && (edges[nn][init]) ) ? 1 : 0;

    for(auto v: g[nn])
    {
        if(mask&(1<<(v-1))) continue;
        if(v>init) ans += count_cycles(v,mask|(1<<(v-1))); //to prevent multicounting
    }
    return dp[nn][mask] = ans;
}
```

```
void solve()
{
    cin>>n>>m;
    g.resize(n+1);
    memset(edges,0,sizeof(edges));
    for(int i=0;i<m;i++)
    {
        int a,b; cin>>a>>b;
        g[a].push_back(b);
        g[b].push_back(a);
        edges[a][b] = edges[b][a] = 1;
    }
    lli ans = 0;
    for(int i=1;i<=n;i++)
    {
        dp.assign(n+1,vector<lli>((1<<20),-1));
        init = i;
        ans += count_cycles(i,(1<<(i-1)));
    }
    cout<<ans/2;
}

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    solve();
}
```

Application Form - III (Intraset Matching)

There are 2*N Chess players and N Boards in a chess tournament. The rating of the Chess Player is given by the array A. Every player can play only with one player.

There is a N board pairing to be done, and if you match Player i and Player j in the board K, then the Happiness Score increases by K*abs(A[i]-A[j])*gcd(A[i],A[j]).

Find the maximum total happiness by ideal pairing

$N \leq 10$ & $A[i] \leq 10^9$

Soln:

```
lli n;
vector<lli> val;
vector<vector<lli>> dp;

//Time complexity = 10*(1024*1024)*(20*20)
//it turns out that time complexity is better than this as many are rejected

lli rec(lli level,lli mask) //mask tells players already chosen
{
    if(level==n+1) return 0;
    if(mask+1==(1<<(2*n))) return 0;

    if(dp[level][mask]!=-1) return dp[level][mask];

    lli ans = 0;
    for(int i=0;i<2*n;i++)
    {
        for(int j=0;j<2*n;j++)
        {
            lli temp = ((1<<i)|(1<<j));
            if( (mask&temp) == 0)
            {
                lli happy = (level)*abs(val[i]-val[j])*__gcd(val[i],val[j]) ;
                ans = max(ans,happy + rec(level+1,(mask|temp)));
            }
        }
    }
    return dp[level][mask] = ans;
}
```

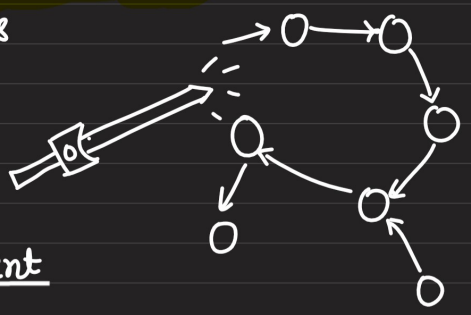
```
void solve()
{
    cin>>n;
    val.resize(2*n);
    dp.assign(n+1,vector<lli>((1<<(2*n)),-1));

    for(int i=0;i<2*n;i++) {
        cin>>val[i];
    }
    cout<<rec(1,0)<<'\n';
    val.clear();
}
```

Now, suppose if the happiness value is abs(a[i]-a[j])*gcd(a[i], a[j]) which is independent of board number

So, order in which pairing is done does not matter, we can always pair the first person who is unpaired with others (obv. using mask)

Application form - IV

Breaking cycles

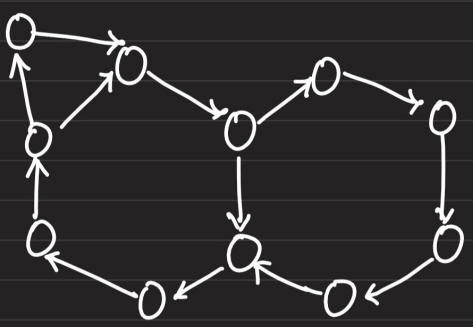


Problem Statement

Given a directed graph, find the minimum number of edges to delete from a graph, so that it becomes a DAG.

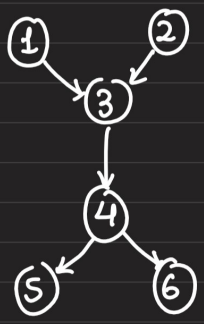
$N <= 20$

Soln:



DAG → atleast 1 topological ordering

Topological ordering



TO

delete this

1	2	3	4	5	6
2	1	3	4	5	6
2	1	3	4	6	5
1	2	3	4	6	5

all permutations

every edge is forward in topological ordering.

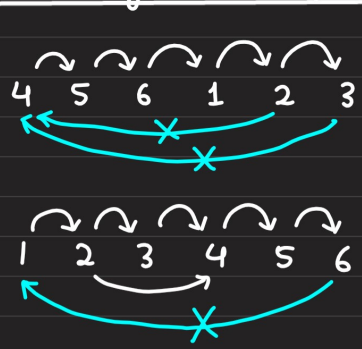
What we can do is that, we can generate all possible permutation & delete all unwanted edges from each permutation.

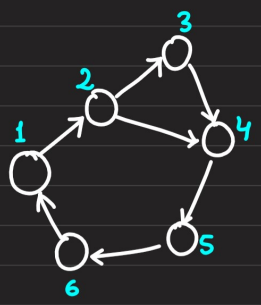
Break Cycle ≡ Create a Topolo

↑

Tough to handle cycles

Forming the Topo Order



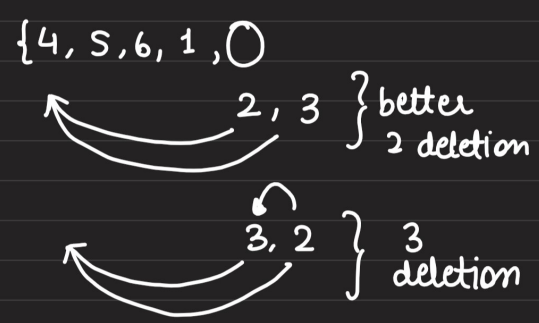


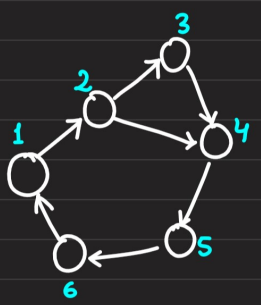
Ans = 3

One way is to generate all 20! permutation & then check for backedges & delete them.

$O(N^2 \cdot N!)$ → inefficient (would work for $N \le 10$)

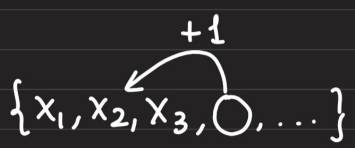
Finding the best efficiently





DP State

DP (pos, mask)
↳ of already placed



↳ DP (pos + 1, mask | (1 << j)) + cost (mask, j)

$j \notin mask$

```cpp
lli n;
vector<vector<lli>> g;
vector<lli> dp;
lli edges[21][21];

//pos = #set bits in the mask so we can keep dp[mask], space reduction
lli rec(lli pos, lli mask)
{
    if(pos==n) return 0;

    if(dp[mask]!=-1) return dp[mask];

    lli ans = 10000;
    for(int i=1;i<=n;i++) //finding bits not set in mask
    {
        if(mask&(1<<(i-1))) continue;

        //ith node is inserted
        lli temp = 0;
        for(int j=1;j<=n;j++)
        {
            if(mask&(1<<(j-1))){
                temp += edges[i][j];
            }
        }
        ans = min(ans,temp + rec(pos+1,mask|(1<<(i-1))));
    }
    return dp[mask] = ans;
}
```

```cpp
void solve()
{
    cin>>n;
    g.resize(n+1);
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        {
            cin>>edges[i][j];
            if(edges[i][j]) g[i].push_back(j);
        }
    }
    lli ans = 10000;
    for(int i=1;i<=n;i++)
    {
        dp.assign(1<<n,-1);
        ans = min(ans,rec(1,(1<<(i-1))));
    }
    cout<<ans<<'\n';
    g.clear();
}

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int t; cin>>t; while(t--)
    solve();
}
```

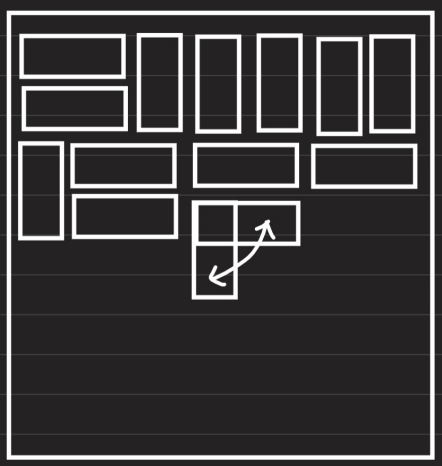
Application Form - V

Profile DP

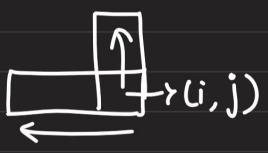
Given a Rectangular Board of NxM board, Find the number of ways of Tiling the Board with Dominos

1 <= N, M <= 12

You cannot solve this if you haven't already



Domino



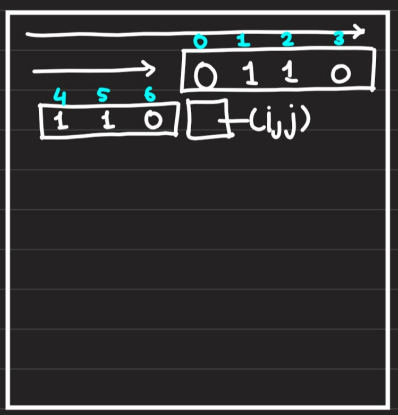
Dp (i, j, mask)

↳ # of ways to tile (i, j) to (N, M) with mask before

1 → filled
0 → empty

mask → 0 1 1 0 1 1 0
0 1 2 3 4 5 6

required bits



If we move from i,j to i+1,j , we need to change the mask to sliding mask

Case I : (i, j)

0 1 2 3 4 5 6 } bit number

0 0 0 1 0 1 0 → MSB

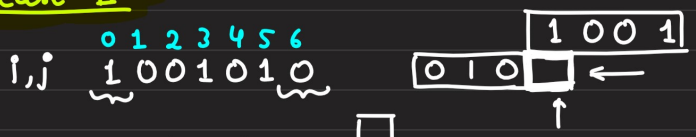
upper cell is empty } → force to fill the cell like 

(i, j+1)

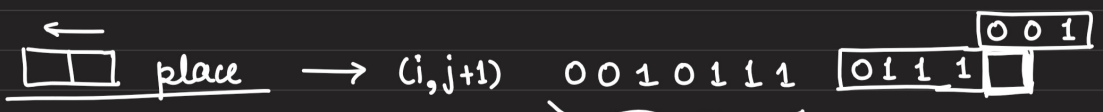
0 1 2 3 4 5 6

new mask : 0 0 1 0 1 0 1 (mask >> 1) | (1 << (M-1))

Case II



we cannot place  , we can either leave it empty (that will be filled later) or we can place a domino like 

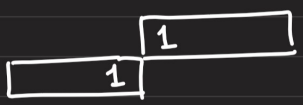


new_mask = (mask >> 1) | (1 << (M-1)) | (1 << (M-2))

<u>if kept empty</u>

(i, j+1) new-mask = mask >> 1

Case III



then we have only one choice

(i, j+1) new-mask = mask >> 1

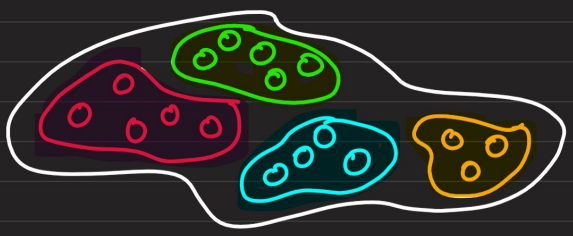
<u>NOTE:</u> If $j+1 > M$ } move to next row.

Also if $j = 1$, then only 2 choice  or keep empty

Time complexity = $O(NM2^M)$ = no. of states as #transitions = 1

```cpp
int n,m;
long long int dp[12][12][(1<<12)];
long long int rec(int i, int j, int mask) //mask will contain 0,1,2,..,m-1 bits
{
    //base case
    if(i==n)
    {
        if(mask+1==(1<<m)) return 1;
        else return 0;
    }
    //memoization
    if(dp[i][j][mask]!=-1) return dp[i][j][mask];

    //transitions
    long long int ans = 0;
    int fbit = (mask&1); //upper is filled or not
    int lbit = (mask&(1<<(m-1))) ? 1 : 0;
    if(i==0)
    {
        if(j==0){
            ans = rec(i + (j+1==m), (j+1)%(m), mask>>1);
        }
        else{
            ans = rec(i + (j+1==m), (j+1)%m, mask>>1);
            if(!lbit) ans += rec(i + (j+1==m), (j+1)%m, ((mask)>>1)|(1<<(m-1))|(1<<(m-2)));
        }
    }
    else
    {
        if(j==0){
            if(!fbit) ans = rec(i + (j+1==m), (j+1)%m, ((mask)>>1)|(1<<(m-1)));
            else ans = rec(i + (j+1==m), (j+1)%(m), mask>>1);
        }
        else
        {
            if(!fbit) ans = rec(i + (j+1==m), (j+1)%m, ((mask)>>1)|(1<<(m-1)));
            else
            {
                ans = rec(i + (j+1==m), (j+1)%m, (mask>>1));
                if(!lbit) ans += rec(i + (j+1==m), (j+1)%m, ((mask)>>1)|(1<<(m-1))|(1<<(m-2)));
            }
        }
    }
    return dp[i][j][mask] = ans;
}
void solve()
{
    cin>>n>>m;
    if(m>n){
        swap(m,n);
    }
    memset(dp,-1,sizeof(dp));
    cout<<rec(0,0,0)<<'\n';
}

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);

    int t;cin>>t;while(t--)
    solve();
}
```

Application form - 6 : Submask Decomposition



We have a class of N students and they are asked to form team of any size. If a student X and Y are in the same team, they contribute Happy[X][Y] score to the final of the decomposition.

Find the best way to form the teams so that maximum happiness can be generated. Its given that Happy[X][Y] = Happy[Y][X]

$N <= 15$ and $-10^9 \leq Happy[i][j] \leq 10^9$

Soln: DP State

DP(mask) → Best way to decompose people in mask.

0110101

$$DP(mask) = \max_{submask \leq mask} \begin{pmatrix} DP(mask \wedge submask) \\ + Happiness(submask) \end{pmatrix}$$

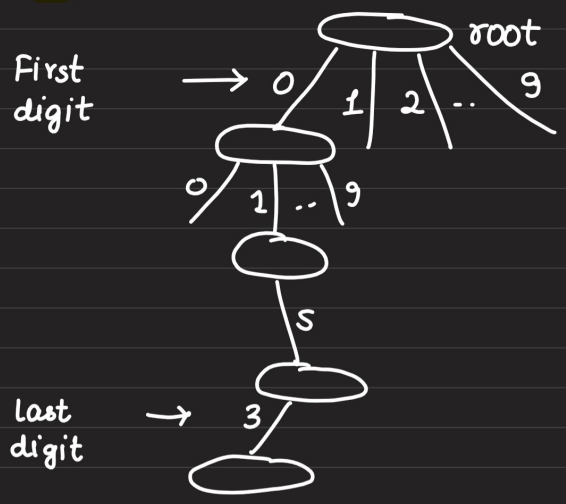
for eg: 0110101
0010100 } one psbl. submask.

```cpp
int n;
int happy[15][15];
long long int happy_sm[(1<<15)]; //just an oprimization
long long int dp[(1<<15)];

long long int happy_grp(int submask)
{
    long long int ans = 0;
    for(int i=0;i<n;i++)
    {
        for(int j=i+1;j<n;j++)
        {
            int si = (submask&(1<<i)) ? 1 : 0;
            int sj = (submask&(1<<j)) ? 1 : 0;
            if(si && sj){
                ans += happy[i][j];
            }
        }
    }
    return ans;
}
```

```cpp
//no. of states = o(2^N)
//no. of transitions = O(submask)
//Time Complexity = O(3^N)
long long int maxHappiness(int mask)
{
    if(mask==0) return 0;

    if(dp[mask]!=-1) return dp[mask];

    long long int ans = 0;
    for(int submask = mask;submask;submask = (submask-1)&mask)
    {
        ans = max(ans,happy_sm[submask] + maxHappiness(mask^submask));
    }
    return dp[mask] = ans;
}
```

```cpp
void solve()
{
    cin>>n;
    memset(dp,-1,sizeof(dp));
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            cin>>happy[i][j];
        }
    }
    int mask = (1<<n)-1;
    for(int submask = 0; submask < (1<<n); submask++)
    {
        happy_sm[submask] = happy_grp(submask);
    }
    cout<<maxHappiness(mask)<<'\n';
}
```

Digit DP ⟶ solve for [L, R]



Constraints are something we use to design the dp state

Q Find the number of odd numbers in the range [L, R]

Ans :

$$\frac{\left[R - (R+1)\%2\right] - \left[L + (L+1)\%2\right]}{2} + 1$$ ← Ans

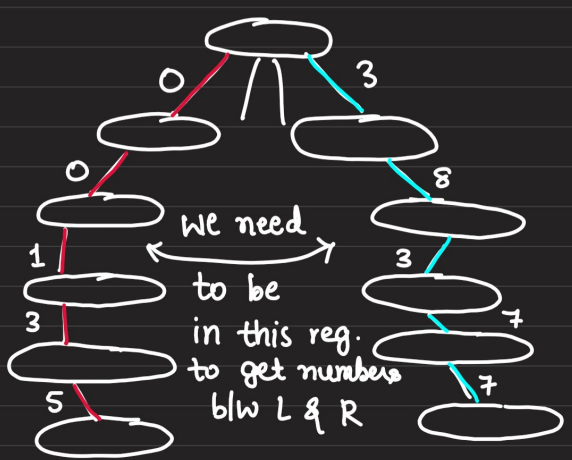
$$\boxed{0 \le L \le R \le 10^{18}}$$

or $\left\lceil \frac{R}{2} \right\rceil - \left\lceil \frac{L-1}{2} \right\rceil$

Digit DP

$0 \le L \le R < \underbrace{10^{18}}_{\text{18 digit}}$

say L = 00135... } 18 digits
R = 38377... } 18 digits



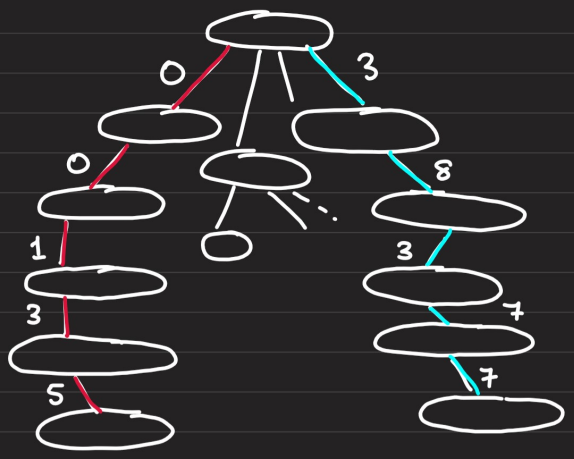
4 possibility for a node

i) Both Blue & red edge originating from it

ii) Only Blue edge originating from it

iii) Only Red edge originating from it

iv) Non coloured edge originating



4 possibility for a node

i) Both Blue & red edge originating from it

ii) Only Blue edge originating from it

iii) only Red edge originating from it

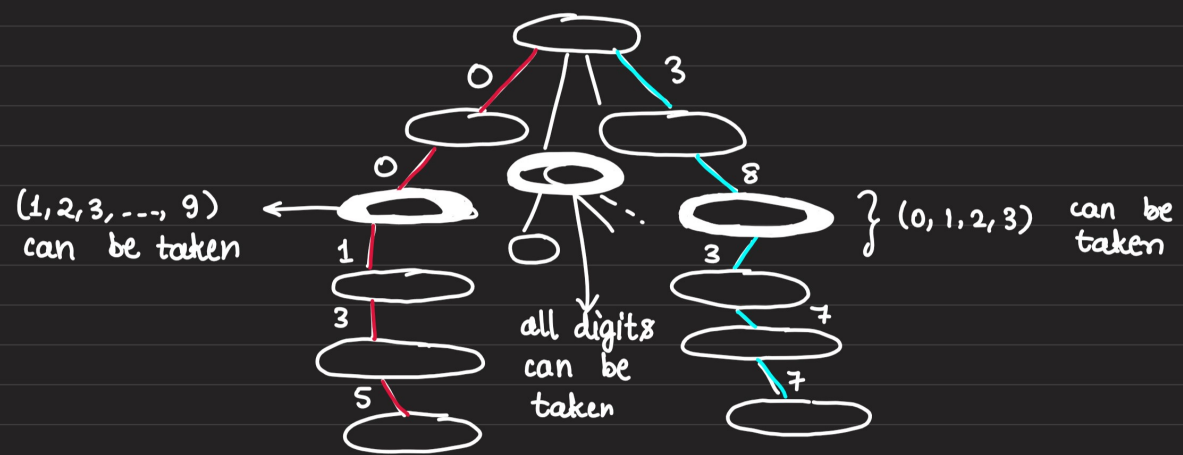
iv) Non coloured edge originating

For non-coloured we can take any digit

For Blue coloured we can only take $\leq$ digit

For Red " , we can only take $\geq$ digit

For both Red & Blue, we can only take = digit



It is not possible to build the whole trie due to large count of numbers b/w [L, R]

We are going to do DP on height (or digit)

dp(node) = number of leaf that are odd in value

Note: $dp(i)_{i \in \text{node at height } h}$ = constant

as 36 - - - - - ... } for both we will get the same answer.
37 - - - - - ... }

dp(level, red, blue) → have a unique answer

↓ ↙ ↓ ↙ ↓

height 0 1 0 1

One you become non-red / non-blue, you can never become red/blue again.

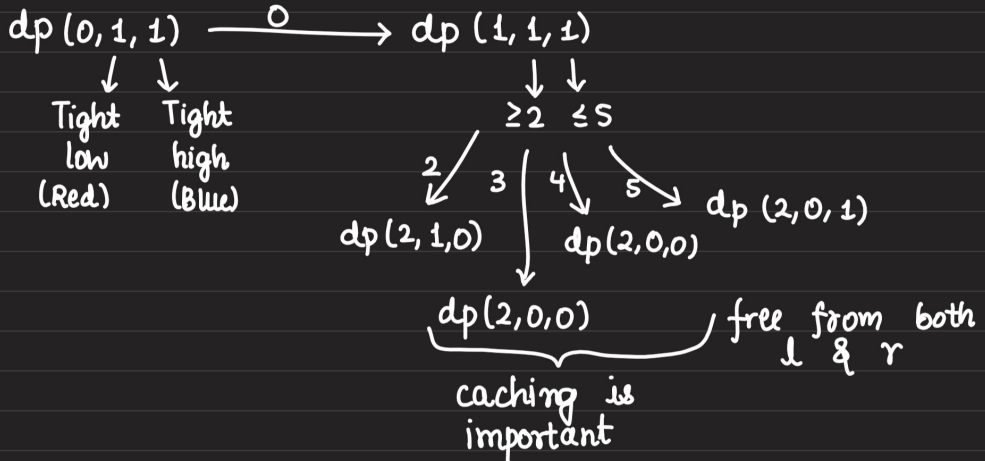
eg: L = 0257
R = 0593
} We will be solving problems using Form 1 DP

soln:

_ _ _ _

↑

0



Form 1 with 2 extra parameters, tight low & tight high.

* { In the last step only, we will choose only odd numbers

S - Digit Sum Atcoder

Problem Statement

Find the number of integers between 1 and K (inclusive) satisfying the following condition, modulo $10^9 + 7$:

- The sum of the digits in base ten is a multiple of D.

Constraints

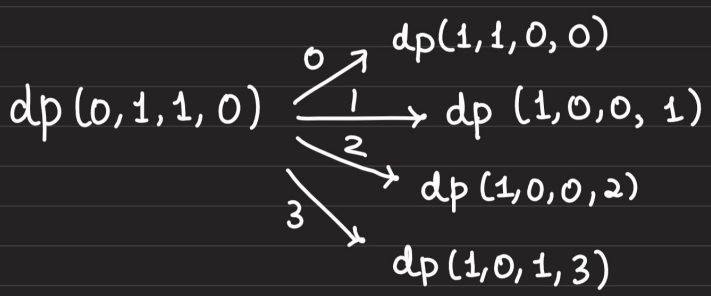
- All values in input are integers.
- $1 \leq K < 10^{10000}$
- $1 \leq D \leq 100$

soln: no. of digits = 10000

maintain sum of digits mod D till now

eg: L = 0 1 5 6
R = 3 5 7 8 } D = 100

dp (level, Ltight, Rtight, sumofDigit %D) — — — — %D



Whenever, we reach the last digit & sumofDigit == 0 then return 1.

```cpp
#include<bits/stdc++.h>
using namespace std;
using lli = long long int;

int n,d,mod = 1e9 + 7;
string l,r;
int dp[10001][2][2][101];

lli rec(int level,int Ltight, int Htight, int sumD)
{
    if(level==n){
        if(sumD==0) return 1;
        else return 0;
    }
    if(dp[level][Ltight][Htight][sumD]!=-1) return dp[level][Ltight][Htight][sumD];

    lli ans = 0, lo = 0, hi = 9;
    if(Ltight) lo = l[level]-'0';
    if(Htight) hi = r[level]-'0';

    for(int i=lo;i<=hi;i++)
    {
        int nLti = Ltight, nHti = Htight;

        if(i!=l[level]-'0') nLti = 0;
        if(i!=r[level]-'0') nHti = 0;

        ans += rec(level+1,nLti,nHti,(sumD+i)%d);
        ans %= mod;
    }
    return dp[level][Ltight][Htight][sumD] = ans%mod;
}

void solve()
{
    //cin>>l;
    l = "1";
    cin>>r; cin>>d;
    n = r.size();

    //making l and r of the same length by padding '0's
    int diff = r.size() - l.size();
    string pre = "";
    for(int i=0;i<diff;i++){
        pre.push_back('0');
    }
    l = pre + l;

    memset(dp,-1,sizeof(dp));
    cout<<rec(0,1,1,0)<<'\n';
}

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    solve();
}
```

Description

A number N is said to be D-Magic if digit D appears in the decimal representation of the number on only even positions. You have to tell the number of D-Magic numbers in the range A to B(both inclusive) that are multiple of M. Since the answer can be large, print the number of integers modulo $10^9 + 7$.

It is given that the number of digits in A and B is the same and A ≤ B.

Constraints

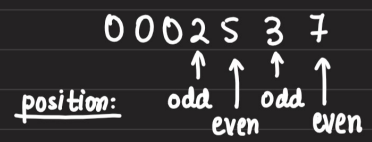
$1 \le T \le 5$

$1 \le M \le 2000$

$0 \le D \le 9$

$1 \le A \le B \le 10^{2000}$

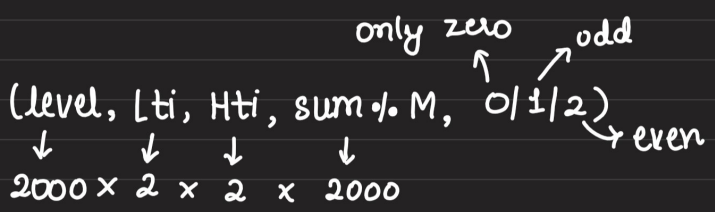
NOTE: Even positions are decided from left to right



And D should be placed at all even positions

Soln:

: Dp state




```
int modulo(string num, int m)
{
    int res = 0;
    //num contains the large number stored as a string
    for (int i = 0; i < num.length(); i++) {
        res = (res*10 + (num[i] - '0')) % m;
    }
    return res;
}
```

for the problem, they have given that the length of a & b are same, so no leading zeros.

```cpp
const int mod = 1000000007;
string l,r;
int n,d,m;
int dp[2000][2][2][2000][2];
```

```cpp
int rec(int level, int lti, int hti, int sum, int type) //type = 1 means odd else type=0 means even, assuming no leading zeros
{
    if(level==n)
    {
        if(sum==0) return 1;
        else return 0;
    }
    if(dp[level][lti][hti][sum][type]!=-1) return dp[level][lti][hti][sum][type];

    int lo = 0, hi = 9;
    if(lti) lo = l[level]-'0';
    if(hti) hi = r[level]-'0';

    int ans = 0;
    for(int i=lo;i<=hi;i++)
    {
        if(type==1 && i==d) continue;
        if(type==0 && i!=d) continue;

        int nlti = lti, nhti = hti, ntype = type;

        if(type==1) ntype=0;
        else ntype=1;

        if(i!=lo) nlti = 0;
        if(i!=hi) nhti = 0;

        int nsum = (10*sum + i)%m; //note how to take modulo
        ans += rec(level+1,nlti,nhti,nsum,ntype);
        ans%=mod;
    }
    return dp[level][lti][hti][sum][type] = ans;
}
```

```cpp
void solve()
{
    cin>>m>>d;
    cin>>l>>r;
    n = r.size();

    memset(dp,-1,sizeof(dp));
    cout<<rec(0,1,1,0,1)<<'\n'; //starting with odd position
}

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);

    int t;cin>>t;while(t--)
    solve();
}
```

Maximum XOR

Description

You are given two integers L and R. You have to pick any two integers(they may be the same) in the range L to R(both inclusive) such that the xor of those two integers is maximum.

Constraints

$1 \le T \le 10^4$

$1 \le L \le R \le 10^{18}$

```cpp
//Digit DP but using Binary Format instead of Decimal
lli max_xor(int level,int lt1,int ht1,int lt2,int ht2)
{
    if(level==-1) return 0;

    if(dp[level][lt1][ht1][lt2][ht2]!=-1) return dp[level][lt1][ht1][lt2][ht2];

    int l1 = 0, h1 = 1;
    int l2 = 0, h2 = 1;

    int ls = (l&(1ll<<level)) ? 1 : 0;
    int rs = (r&(1ll<<level)) ? 1 : 0;

    if(lt1) l1 = ls;
    if(ht1) h1 = rs;

    if(lt2) l2 = ls;
    if(ht2) h2 = rs;

    lli ans = 0;

    //atmost 4 transitions
    for(int i=l1;i<=h1;i++)
    {
        for(int j=l2;j<=h2;j++)
        {
            int nlt1 = lt1, nlt2 = lt2, nht1 = ht1, nht2 = ht2;
            if(i!=l1) nlt1 = 0;
            if(i!=h1) nht1 = 0;
            if(j!=l2) nlt2 = 0;
            if(j!=h2) nht2 = 0;
            ans = max(ans,(i^j)*(1ll<<level) + max_xor(level-1,nlt1,nht1,nlt2,nht2));
        }
    }
    return dp[level][lt1][ht1][lt2][ht2] = ans;
}

void solve()
{
    cin>>l>>r;
    memset(dp,-1,sizeof(dp));
    cout<<max_xor(60,1,1,1,1)<<'\n';
}

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);

    int t;cin>>t;while(t--)
    solve();
}
```

Description

String P is given, consisting of uppercase alphabets. You have to find the number of strings Q, such that string Q is lexicographically larger than string P and reverse of string Q is lexicographically larger than the reverse of string P. Since the answer can be large, print the number of integers modulo $10^9 + 7$.

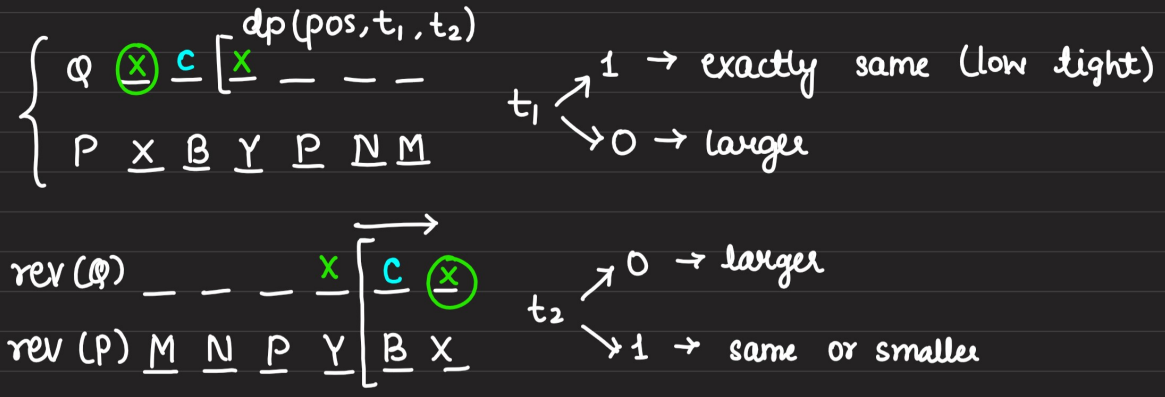
Constraints

$1 \le T \le 100$

$1 \le |P| \le 10^5$ where $|P|$ is the length of string P.

It is guaranteed that the sum of $|P|$ for all test cases is not greater than 10^5.

soln:



This is the way we will maintain the states.

```
const lli mod = 1000000007;
int n; string l;
lli dp[100100][2][2];
lli cnt(int level,int t1, int t2)
{
    if(level==n){ //last we will assign number greater than l[n-1]
        if(t1==0 && t2==0) return 1;
        else return 0;
    }
    if(dp[level][t1][t2]!=-1) return dp[level][t1][t2];

    int lo = 'A', hi = 'Z';
    if(t1) lo = l[level];

    lli ans = 0;
    for(int i=lo;i<=hi;i++)
    {
        int nt1 = t1;
        if(i!=lo) nt1 = 0;

        int nt2;
        if(i<l[level]) nt2 = 1;
        else if(i==l[level]) nt2 = t2;
        else nt2 = 0;

        ans += cnt(level+1,nt1,nt2);
        ans%=mod;
    }
    return dp[level][t1][t2] = ans;
}
```

```
void solve()
{
    cin>>l;
    n = l.size();
    for(int i=0;i<=n;i++){
        for(int j=0;j<2;j++){
            dp[i][j][0] = dp[i][j][1] = -1;
        }
    }
    cout<<cnt(0,1,1)<<'\n';
}
```