**数位DP**

数位是指把一个数字按照个、十、百、千等等一位一位地拆开，关注它每一位上的数字。如果拆的是十进制数，那么每一位数字都是 0~9，其他进制可类比十进制。

数位 DP：用来解决一类特定问题，这种问题比较好辨认，一般具有这几个特征：

要求统计满足一定条件的数的数量（即，最终目的为计数）；

这些条件经过转化后可以使用「数位」的思想去理解和判断；

输入会提供一个数字区间（有时也只提供上界）来作为统计的限制；

上界很大（比如 10^{18}），暴力枚举验证会超时。

数位 DP 的基本原理：

考虑人类计数的方式，最朴素的计数就是从小到大开始依次加一。但我们发现对于位数比较多的数，这样的过程中有许多重复的部分。例如，从 7000 数到 7999、从 8000 数到 8999、和从 9000 数到 9999 的过程非常相似，它们都是后三位从 000 变到 999，不一样的地方只有千位这一位，所以我们可以把这些过程归并起来，将这些过程中产生的计数答案也都存在一个通用的数组里。此数组根据题目具体要求设置状态，用递推或 DP 的方式进行状态转移。

数位 DP 中通常会利用常规计数问题技巧，比如把一个区间内的答案拆成两部分相减（即 \mathit{ans}\_{[l, r]} = \mathit{ans}\_{[0, r]}-\mathit{ans}\_{[0, l - 1]}）

那么有了通用答案数组，接下来就是统计答案。统计答案可以选择记忆化搜索，也可以选择循环迭代递推。为了不重不漏地统计所有不超过上限的答案，要从高到低枚举每一位，再考虑每一位都可以填哪些数字，最后利用通用答案数组统计答案。

例题：M xor sum

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

ll n,m,k;

const int maxn = 25;

ll C[maxn][maxn];

map<ll,ll>dp[50][50];

int a[50];

const int mod = 1e9+7;

ll dfs(ll cur,ll s,ll lv)

{

if(lv<0)return 0;

if(cur<0)return lv == 0;

ll s1 = k/2,s2 = k - s1;

if(((1LL<<(cur + 1)) - 1)\*s1\*s2 < lv)return 0;

if(dp[cur][s].count(lv))return dp[cur][s][lv];

ll res = 0;

if(a[cur] == 1){

for(int i=0;i<=s;i++){

for(int j=0;j<=k-s;j++){

ll cnt = C[s][i]\*C[k-s][j]%mod;

ll val = (1LL<<cur)\*(i+j)\*(k-i-j);

res+=cnt\*dfs(cur - 1,i,lv - val)%mod;

res%=mod;

}

}

}

else{

for(int i=0;i<=0;i++){

for(int j=0;j<=k-s;j++){

ll cnt = C[s][i]\*C[k-s][j]%mod;

ll val = (1LL<<cur)\*(i+j)\*(k-i-j);

res+=cnt\*dfs(cur - 1,s,lv - val);

res%=mod;

}

}

}

return dp[cur][s][lv] = res;

}

void solve()

{

cin>>n>>m>>k;

ll tmp = m;

int pos = 0;

while(tmp){

a[pos++] = tmp%2;

tmp/=2;

}

pos--;

cout<<dfs(pos,k,n);

}

int main()

{

ios::sync\_with\_stdio(false);

cin.tie(0);

cout.tie(0);

int t = 1;

C[0][0] = 1;

for(int i=1;i<=20;i++){

C[i][0] = C[i][i] = 1;

for(int j=1;j<i;j++){

C[i][j] = C[i-1][j] + C[i-1][j-1];

}

}

while(t--){

solve();

}

return 0;

}