Hello World ! ---数位DP

/\*

数位DP 好题 压缩空间

题意：统计区间内的beautiful number，beautiful number定义：一个正整数能整除它的每个位上的数字(a positive integer number is beautiful if and only if it is divisible by each of its nonzero digits.)

思路：容易想到要整除每位上的数字，即整除每位上的数字的最小公倍数。

比较难想到的一点是压缩：dp[pos][sum][lcm]表前面位的值为sum，最小公倍数的lcm的合法数。但是sum会超int，开不下这么大的数组。看题解知道需要压缩。下面讲压缩：

int MOD = LCM(1,2,..9) = 2520

按照定义，x为beautiful number：

x % LCM{digit[x]} = 0

即x % MOD % LCM{digit[x]} = 0

而在逐位统计时，假设到了pre\*\*\*(pre指前面一段已知的数字，而\*是任意变)

(preSum \* 10 + i) % MOD % LCM(preLcm, i)

= (preSum \* 10 % MOD + i % MOD) % LCM(preLcm, i)

另外，还把[1..2520]中能整除2520的数hash成一个下标，因为只有x能整除2520，才有可能在DP做LCM时算出LCM = x。这样，又把dp数组第三维压到了[48]

http://www.cppblog.com/Yuan/archive/2011/01/24/139201.html

\*/

//cf 55D

void preprocess()

{

int lcm = 1;

for(int i = 1; i <= 9; i++) lcm = LCM(lcm, i);

MOD = lcm;

int j = 0;

for(int i = 1; i <= MOD; i++) if(MOD % i == 0) idx[i] = j++;

}

LL dfs(int pos, int preSum, int preLcm, bool full)

{

if(pos == -1) return preSum % preLcm == 0;

if(!full && dp[pos][preSum][ idx[preLcm] ] != -1) return dp[pos][preSum][ idx[preLcm] ];

LL res = 0;

int end = full ? digit[pos] : 9;

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

int newSum = (preSum \* 10 + i) % MOD; //TODO 妙呀

int newLcm = (i == 0 ? preLcm : LCM(preLcm, i));

res += dfs(pos-1, newSum, newLcm, full && i == end);

}

if(!full) dp[pos][preSum][ idx[preLcm] ] = res;

return res;

}

LL calc(LL n)

{

int Index = init(n);

return dfs(Index-1, 0, 1, true);

}

/\*

数位DP

题意：[l,r]内的平衡数

平衡数的定义：以某位为中心，两边力矩相等。

力矩定义：假设中心位为mid，则左力矩为sum{digit[i]\*(mid-i)}, (i < mid)；右力矩同理。

思路：枚举中心点，对每个中心点都搞一次数位DP。

dp[i][mid][preMoment] 表现在处理到i位，中心点在mid位，前i位的力矩和为preMoment。注意，实现里面把右力矩作为负数来算，让左右力矩抵消。

//枚举中心位

//for(int i = 0; i < Index; i++) ans += dfs(Index-1, i, 0, true);

//状态转移

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

res += dfs(pos-1, central, sum + (pos-central)\*i, up\_limit && i == end);

}

\*/

long long dfs(int pos, int central, int sum, bool up\_limit)

{

if(sum < 0) return 0;

if(pos == -1) return sum == 0;

if(!up\_limit && dp[pos][central][sum] != -1) return dp[pos][central][sum];

long long res = 0;

int end = up\_limit ? digit[pos] : 9;

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

res += dfs(pos-1, central, sum + (pos-central)\*i, up\_limit && i == end);

}

if(!up\_limit) dp[pos][central][sum] = res;

return res;

}

long long calc(long long n)

{

int Index = init(n);

long long ans = 0;

for(int i = 0; i < Index; i++) ans += dfs(Index-1, i, 0, true);

return ans - Index + 1; //TODO

}

/\*

数位DP 好题

题目大意：给你两个数 a、b，F(x) = An \* 2n-1 + An-1 \* 2n-2 + ... + A2 \* 2 + A1 \* 1，A是a十进制各个数位上的数字，让你求出 0 ~ b 中f(x) 比f(a)小的数字个数。

思路：数位DP。。 d[ i ][ j ]表示没满的的时候前 i 位f(x)<=j的个数，d[ i ][ j ] = SIGMA(d[ i - 1][ j - num[ i ]\*c[ i ] ])。

挺水的数位DP啊，比赛的时候用for做的，因为我们不是预处理的那种，每次都算一遍，T很多，然后一直TLE，无语啊、。。T^T TLE原因如下：由于都是不满的，所以memset要放外面！

或者直接for，先预处理，不过要多一维表示该位上的数字是多少，即 d[ i ][ j ][ s ] 表示 前 i 位第 i 位上的数字是 j，<= s 的个数，具体代码看这里吧：http://blog.csdn.net/suvigo/article/details/11689709

我一开始定义dp[pos][sum]是pos位的数{x}中和为sum的合法数个数，但是会发现这样定义mem(dp,-1)得放在里面，则TLE，所以就需要改变sum的含义，改dp[i][j]为i位的数{ x } 中 F ( x ) 小于 j 的数的个数.

\*/

int F(int A)

{

int sum = 0, i = 0;

while(A) {

sum += (1 << i) \* (A % 10);

i++, A /= 10;

}

return sum;

}

int dfs(int pos, bool full, int sum)

{

if(sum < 0) return 0;

if(pos == -1) return 1;

if(!full && dp[pos][sum] != -1) return dp[pos][sum];

int res = 0;

int end = full ? digit[pos] : 9;

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

res += dfs(pos-1, full && i == end, sum - (i << pos));

}

if(!full) dp[pos][sum] = res;

return res;

}

/\*

数位DP 简单题

题意：设f(x)表示x的二进制有多少个1的右边还是1

12 = 1100B 有一个1的右边是1，所以f(12) = 1；f(15) = f(1111B) = 3；f(27) = f(11011B) = 2

计算[l,r]的所有f和.

思路：模板题

dp[pos][x][sum] 前pos个数，最近（上一个）数的x，f值和为sum的DP值.

\*/

//lightoj 1032

long long dfs(int pos, int preNum, int sum, bool full)

{

if(pos == -1) return sum;

if(!full && dp[pos][preNum][sum] != -1) return dp[pos][preNum][sum];

long long res = 0;

int end = full ? digit[pos] : 1;

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

res += dfs(pos-1, i, sum+(i==preNum && preNum==1), full && i == end);

}

if(!full) dp[pos][preNum][sum] = res;

return res;

}

/\*

数位统计DP

题意：将n分成k个不大于m的数的和有多少种方案，1,1,2与2,1,1是不同方案。

思路：dp[n][k][m] = Σ(dp[n-i][k-1][m]) (1 <= i <= m)

预处理出所有dp值即可。

我原来还用数位DP的dfs搞...TLE惨了...

\*/

//lightoj 1191

void initDP()//n个码，k个单元，一个单元最多m个码

{

for(int m = 1; m <= 50; m++) {

for(int n = 1; n <= m; n++) dp[n][1][m] = 1;

for(int n = 1; n <= 50; n++)

for(int k = 1; k <= 50; k++)

for(int i = 1; i <= m && i <= n; i++)

dp[n][k][m] += dp[n-i][k-1][m];

}

}

/\*

数位DP 好题 稍难有细节

题意：[l,r]内有多少个回文数？如12321, 7227

思路：http://blog.csdn.net/auto\_ac/article/details/8803952

开始想按hdu\_3709那样枚举中心点来搞，但是这种方法要记录前缀和preSum，而这里的前缀和太大了存不下...

LZ的写法挺少见的。他是像区间DP一样，两端l/r往中间移，边移边算。跟一般数位DP只有一个指针pos从左往右移复杂。由一端指针移动变到现在两端指针移动要改得也不多，改一下边界条件(l < r)，改一下移动方式(l--, r++)

这里是枚举长度搞，其实跟枚举中心位是一个道理，虽然枚举长度比枚举中心少见，不过个人感觉对于这道题来讲，枚举长度比较方便：

dfs()加了个参数preFit，这跟回文有关，就是记录已经遍历过的两端是否能够达到回文。

\*/

//lightoj 1205

long long dfs(int len, int l, int r, bool full, bool preFit) //preFit表s[len..l]与s[r..0]的的关系，preFit为true表示s[len..l+1]与s[r-1..0]可以匹配，否则需要当前digit[r] > i才能fit~~

{

if(l < r) return !full || preFit; //为什么写成‘=’等于就不对？

//估计跟preFit有关。因为虽然l == r之前preFit = false，但是如果a[l=r] > 0时，preFit是应该变为true的

if(!full && dp[len][l] != -1) return dp[len][l];

long long res = 0;

int end = full ? digit[l] : 9;

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

if(i == 0 && l == len-1) continue; //l == len-1表示当前是第1位。就是0啦~前导0真是麻烦

bool nowFit = preFit;

if(nowFit) nowFit = (digit[r] >= i); //跟回文性质有关

else nowFit = (digit[r] > i);

res += dfs(len, l-1, r+1, full && i == end, nowFit);

}

if(!full) dp[len][l] = res;

return res;

}

long long calc(long long n)

{

if(n <= 1) return n+1; //1还是0？

int Index = init(n);

long long res = 1; //估计1是表示0也是palindromic

for(int i = Index; i >= 1; i--) {

int len = i;

res += dfs(len, len-1, 0, len == Index, true); //第一次见这样搞的

}

return res;

}

/\*

数位统计 DP 推公式 组合数 好题 递推 染色 再加一个好题

思路：这些公式我哪会呀...不错啦...

\*/

void init()

{

for(int i = 1; i <= 391; 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]) % MOD;

}

for(int i = 1; i <= 391; i++) {

dp[1][i] = 1;

for(int j = 2; j <= 50; j++) {

dp[j][i] = (dp[j][i-1]\*j + dp[j-1][i-1]\*j) % MOD;

}

}

}

int main()

{

int Cas = 0, cases, m, n, k;

init();

scanf("%d", &cases);

while(cases--) {

scanf("%d%d%d", &m, &n, &k);

if(n == m && n == 0) {

printf("Case %d: %d\n", ++Cas, k);

continue;

}

m++, n++;

int x = m\*n/2 + ((m & 1) && (n & 1));

int y = m\*n - x;

long long ans = 0;

for(int i = 1; i < k; i++)

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

ans = (ans + (dp[i][x] \* dp[j][y]) % MOD \* C[k][i] % MOD \* C[k-i][j] % MOD) % MOD;

//ans += ((dp[i][x] + dp[k-i][y]) % MOD \* (C[k][i] % MOD)) % MOD;

printf("Case %d: %lld\n", ++Cas, ans);

}

return 0;

}

//dp[i][j] = dp[i-1][j]\*i + dp[i-1][j-1]\*i\*j

/\*

数位DP 有前导0处理

题意：[l,r]中的数转为二进制后0的个数不小于1的个数的数。

思路：常规数位DP，就是要注意前导 0

dp状态：dp[pos][one][zero]

注意点：

LL dfs(int pos, int one, int zero, bool full, bool first) {

res += dfs(pos-1, one+i, zero+(i==0 && !first), full && i == end, first && i == 0);

//zero表当前0的个数，那如果现在first为true的话zero即不应该增加！TODO

}

\*/

//poj 3252

LL dfs(int pos, int one, int zero, bool full, bool first)

{

if(pos == -1) {

if(first) return 1;

return zero >= one;

}

if(!first && !full && dp[pos][one][zero] != -1) return dp[pos][one][zero];

LL res = 0;

int end = full ? digit[pos] : 1;

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

res += dfs(pos-1, one+i, zero+(i==0 && !first), full && i == end, first && i == 0); //TODO

}

if(!full && !first) dp[pos][one][zero] = res;

return res;

}