**基础篇**

数位dp是一种计数用的dp，一般就是要统计一个区间[le,ri]内满足一些条件数的个数。所谓数位dp，字面意思就是在数位上进行dp咯。数位还算是比较好听的名字，数位的含义：一个数有个位、十位、百位、千位......数的每一位就是数位啦！

之所以要引入数位的概念完全就是为了dp。数位dp的实质就是换一种暴力枚举的方式，使得新的枚举方式满足dp的性质，然后记忆化就可以了。

两种不同的枚举：对于一个求区间[le,ri]满足条件数的个数，最简单的暴力如下：

**[cpp]** [view plain](https://blog.csdn.net/wust_zzwh/article/details/52100392) [copy](https://blog.csdn.net/wust_zzwh/article/details/52100392)

1. for(int i=le;i<=ri;i++)
2. if(right(i)) ans++;

然而这样枚举不方便记忆化，或者说根本无状态可言。

新的枚举：控制上界枚举，从最高位开始往下枚举，例如：ri=213，那么我们从百位开始枚举：百位可能的情况有0,1,2(觉得这里枚举0有问题的继续看)

然后每一位枚举都不能让枚举的这个数超过上界213（下界就是0或者1，这个次要），当百位枚举了1，那么十位枚举就是从0到9，因为百位1已经比上界2小了，后面数位枚举什么都不可能超过上界。所以问题就在于：当高位枚举刚好达到上界是，那么紧接着的一位枚举就有上界限制了。具体的这里如果百位枚举了2，那么十位的枚举情况就是0到1，如果前两位枚举了21，最后一位之是0到3(这一点正好对于代码模板里的一个变量limit 专门用来判断枚举范围)。最后一个问题：最高位枚举0：百位枚举0，相当于此时我枚举的这个数最多是两位数，如果十位继续枚举0，那么我枚举的就是以为数咯，因为我们要枚举的是小于等于ri的所以数，当然不能少了位数比ri小的咯！(这样枚举是为了无遗漏的枚举，不过可能会带来一个问题，就是前导零的问题，模板里用lead变量表示，不过这个不是每个题目都是会有影响的，可能前导零不会影响我们计数，具体要看题目)

由于这种新的枚举只控制了上界所以我们的Main函数总是这样：

**[cpp]** [view plain](https://blog.csdn.net/wust_zzwh/article/details/52100392) [copy](https://blog.csdn.net/wust_zzwh/article/details/52100392)

1. int main()
2. {
3. long long le,ri;
4. while(~scanf("%lld%lld",&le,&ri))
5. printf("%lld\n",solve(ri)-solve(le-1));
6. }

w\_w 是吧！统计[1,ri]数量和[1,le-1]，然后相减就是区间[le,ri]的数量了，这里我写的下界是1，其实0也行，反正相减后就没了，注意题目中le的范围都是大于等于1的(不然le=0,再减一就G\_G了)

在讲例题之前先讲个基本的动态模板(先看后面的例题也行)：dp思想，枚举到当前位置pos，状态为state(这个就是根据题目来的，可能很多，毕竟dp千变万化)的数量(既然是计数,dp值显然是保存满足条件数的个数)

**[cpp]** [view plain](https://blog.csdn.net/wust_zzwh/article/details/52100392) [copy](https://blog.csdn.net/wust_zzwh/article/details/52100392)

1. typedef long long ll;
2. int a[20];
3. ll dp[20][state];//不同题目状态不同
4. ll dfs(int pos,/\*state变量\*/,bool lead/\*前导零\*/,bool limit/\*数位上界变量\*/)//不是每个题都要判断前导零
5. {
6. //递归边界，既然是按位枚举，最低位是0，那么pos==-1说明这个数我枚举完了
7. if(pos==-1) return 1;/\*这里一般返回1，表示你枚举的这个数是合法的，那么这里就需要你在枚举时必须每一位都要满足题目条件，也就是说当前枚举到pos位，一定要保证前面已经枚举的数位是合法的。不过具体题目不同或者写法不同的话不一定要返回1 \*/
8. //第二个就是记忆化(在此前可能不同题目还能有一些剪枝)
9. if(!limit && !lead && dp[pos][state]!=-1) return dp[pos][state];
10. /\*常规写法都是在没有限制的条件记忆化，这里与下面记录状态是对应，具体为什么是有条件的记忆化后面会讲\*/
11. int up=limit?a[pos]:9;//根据limit判断枚举的上界up;这个的例子前面用213讲过了
12. ll ans=0;
13. //开始计数
14. for(int i=0;i<=up;i++)//枚举，然后把不同情况的个数加到ans就可以了
15. {
16. if() ...
17. else if()...
18. ans+=dfs(pos-1,/\*状态转移\*/,lead && i==0,limit && i==a[pos]) //最后两个变量传参都是这样写的
19. /\*这里还算比较灵活，不过做几个题就觉得这里也是套路了
20. 大概就是说，我当前数位枚举的数是i，然后根据题目的约束条件分类讨论
21. 去计算不同情况下的个数，还有要根据state变量来保证i的合法性，比如题目
22. 要求数位上不能有62连续出现,那么就是state就是要保存前一位pre,然后分类，
23. 前一位如果是6那么这意味就不能是2，这里一定要保存枚举的这个数是合法\*/
24. }
25. //计算完，记录状态
26. if(!limit && !lead) dp[pos][state]=ans;
27. /\*这里对应上面的记忆化，在一定条件下时记录，保证一致性，当然如果约束条件不需要考虑lead，这里就是lead就完全不用考虑了\*/
28. return ans;
29. }
30. ll solve(ll x)
31. {
32. int pos=0;
33. while(x)//把数位都分解出来
34. {
35. a[pos++]=x%10;//个人老是喜欢编号为[0,pos),看不惯的就按自己习惯来，反正注意数位边界就行
36. x/=10;
37. }
38. return dfs(pos-1/\*从最高位开始枚举\*/,/\*一系列状态 \*/,true,true);//刚开始最高位都是有限制并且有前导零的，显然比最高位还要高的一位视为0嘛
39. }
40. int main()
41. {
42. ll le,ri;
43. while(~scanf("%lld%lld",&le,&ri))
44. {
45. //初始化dp数组为-1,这里还有更加优美的优化,后面讲
46. printf("%lld\n",solve(ri)-solve(le-1));
47. }
48. }

相信读者还对这个有不少疑问，笔者认为有必要讲一下记忆化为什么是if(!limit)才行，大致就是说有无limit会出现状态冲突，举例：

约束：数位上不能出现连续的两个1(11、112、211都是不合法的)

假设就是[1,210]这个区间的个数

状态:dp[pos][pre]:当前枚举到pos位，前面一位枚举的是pre(更加前面的位已经合法了)，的个数(我的pos从0开始)

先看错误的方法计数，就是不判limit就是直接记忆化

那么假设我们第一次枚举了百位是0，显然后面的枚举limit=false，也就是数位上0到9的枚举，然后当我十位枚举了1，此时考虑dp[0][1],就是枚举到个位，前一位是1的个数，显然dp[0][1]=9;(个位只有是1的时候是不满足的)，这个状态记录下来，继续dfs，一直到百位枚举了2，十位枚举了1，显然此时递归到了pos=0,pre=1的层，而dp[0][1]的状态已经有了即dp[pos][pre]!=-1；此时程序直接return dp[0][1]了，然而显然是错的，因为此时是有limit的个位只能枚举0，根本没有9个数，这就是状态冲突了。有lead的时候可能出现冲突，这只是两个最基本的不同的题目可能还要加限制，反正宗旨都是让dp状态唯一

对于这个错误说两点：一是limit为true的数并不多，一个个枚举不会很浪费时间，所以我们记录下! limit的状态解决了不少子问题重叠。第二，有人可能想到把dp状态改一下dp[pos][state][limit]就是分别记录不同limit下的个数，这种方法一般是对的，关于这个具体会讲，下面有题bzoj3209会用到这个。

数位的部分就是这些，然后就是难点，dp部分，dp大牛的艺术，弱鸡只能看看+...+

既然从高位往低位枚举，那么状态一般都是与前面已经枚举的数位有关并且通常是根据约束条件当前枚举的这一位能使得状态完整(比如一个状态涉及到连续k位，那么就保存前k-1的状态，当前枚举的第k个是个恰好凑成成一个完整的状态，不过像那种状态是数位的和就直接保存前缀和为状态了)，不过必然有一位最简单的一个状态dp[pos]当前枚举到了pos位。dp部分就要开始讲例题了，不过会介绍几种常用防tle的优化。

**实战篇**

例一：[HDU 2089](http://acm.hdu.edu.cn/showproblem.php?pid=2089" \t "_blank) 不要62

入门题。就是数位上不能有4也不能有连续的62，没有4的话在枚举的时候判断一下，不枚举4就可以保证状态合法了，所以这个约束没有记忆化的必要，而对于62的话，涉及到两位，当前一位是6或者不是6这两种不同情况我计数是不相同的，所以要用状态来记录不同的方案数。

dp[pos][sta]表示当前第pos位，前一位是否是6的状态，这里sta只需要去0和1两种状态就可以了，不是6的情况可视为同种，不会影响计数。

**[cpp]** [view plain](https://blog.csdn.net/wust_zzwh/article/details/52100392) [copy](https://blog.csdn.net/wust_zzwh/article/details/52100392)

1. #include<iostream>
2. #include<cstdio>
3. #include<cstring>
4. #include<string>
5. using namespace std;
6. typedef long long ll;
7. int a[20];
8. int dp[20][2];
9. int dfs(int pos,int pre,int sta,bool limit)
10. {
11. if(pos==-1) return 1;
12. if(!limit && dp[pos][sta]!=-1) return dp[pos][sta];
13. int up=limit ? a[pos] : 9;
14. int tmp=0;
15. for(int i=0;i<=up;i++)
16. {
17. if(pre==6 && i==2)continue;
18. if(i==4) continue;//都是保证枚举合法性
19. tmp+=dfs(pos-1,i,i==6,limit && i==a[pos]);
20. }
21. if(!limit) dp[pos][sta]=tmp;
22. return tmp;
23. }
24. int solve(int x)
25. {
26. int pos=0;
27. while(x)
28. {
29. a[pos++]=x%10;
30. x/=10;
31. }
32. return dfs(pos-1,-1,0,true);
33. }
34. int main()
35. {
36. int le,ri;
37. //memset(dp,-1,sizeof dp);可优化
38. while(~scanf("%d%d",&le,&ri) && le+ri)
39. {
40. memset(dp,-1,sizeof dp);
41. printf("%d\n",solve(ri)-solve(le-1));
42. }
43. return 0;
44. }

入门就不多讲了，开始讲常用优化吧！

**第一:memset(dp,-1,sizeof dp);放在多组数据外面。**

这一点是一个数位特点，使用的条件是：约束条件是每个数自身的属性，而与输入无关。

具体的：上一题不要62和4，这个约束对每一个数都是确定的，就是说任意一个数满不满足这个约束都是确定，比如444这个数，它不满足约束条件，不管你输入的区间是多少你都无法改变这个数不满足约束这个事实，这就是数自身的属性（我们每组数据只是在区间计数而已，只能说你输入的区间不包含444的话，我们就不把它统计在内，而无法改变任何事实）。

由此，我们保存的状态就可以一直用(注意还有要limit，不同区间是会影响数位在有限制条件下的上限的)

这点优化就不给具体题目了，这个还有进一步的扩展。不过说几个我遇到的简单的约束：

1.求数位和是10的倍数的个数,这里简化为数位sum%10这个状态，即dp[pos][sum]这里10 是与多组无关的，所以可以memset优化，不过注意如果题目的模是输入的话那就不能这样了。

2.求二进制1的数量与0的数量相等的个数，这个也是数自身的属性。

3.。。。。。

还是做题积累吧。搞懂思想！

下面介绍的方法就是要行memset优化，把不满足前提的通过修改，然后优化。

介绍之前,先说一种较为笨拙的修改，那就是增加状态，前面讲limit的地方说增加一维dp[pos][state][limit]，能把不同情况下状态分别记录(不过这个不能memset放外面)。基于这个思想，我们考虑：约束为数位是p的倍数的个数，其中p数输入的，这和上面sum%10类似，但是dp[pos][sum]显然已经不行了，每次p可能都不一样，为了强行把memset提到外面加状态dp[pos][sum][p]，对于每个不同p分别保存对应的状态。这里前提就比较简单了，你dp数组必须合法，p太大就G\_G了。所以对于与输入有关的约束都可以强行增加状态(这并不代表能ac，如果题目数据少的话就随便你乱搞了)

**第二：相减。**

例题：[HDU 4734](http://acm.hdu.edu.cn/showproblem.php?pid=4734)

题目给了个f(x)的定义：F(x) = An \* 2n-1 + An-1 \* 2n-2 + ... + A2 \* 2 + A1 \* 1，Ai是十进制数位，然后给出a,b求区间[0,b]内满足f(i)<=f(a)的i的个数。

常规想：这个f(x)计算就和数位计算是一样的，就是加了权值，所以dp[pos][sum]，这状态是基本的。a是题目给定的，f(a)是变化的不过f(a)最大好像是4600的样子。如果要memset优化就要加一维存f(a)的不同取值，那就是dp[10][4600][4600]，这显然不合法。

这个时候就要用减法了，dp[pos][sum]，sum不是存当前枚举的数的前缀和(加权的)，而是枚举到当前pos位，后面还需要凑sum的权值和的个数，

也就是说初始的是时候sum是f(a),枚举一位就减去这一位在计算f(i)的权值，那么最后枚举完所有位 sum>=0时就是满足的，后面的位数凑足sum位就可以了。

仔细想想这个状态是与f(a)无关的(新手似乎很难理解)，一个状态只有在sum>=0时才满足，如果我们按常规的思想求f(i)的话，那么最后sum>=f(a)才是满足的条件。

**[cpp]** [view plain](https://blog.csdn.net/wust_zzwh/article/details/52100392) [copy](https://blog.csdn.net/wust_zzwh/article/details/52100392)

1. #include<cstdio>
2. #include<cstring>
3. #include<iostream>
4. #include<string>
6. using namespace std;
7. const int N=1e4+5;
8. int dp[12][N];
9. int f(int x)
10. {
11. if(x==0) return 0;
12. int ans=f(x/10);
13. return ans\*2+(x%10);
14. }
15. int all;
16. int a[12];
17. int dfs(int pos,int sum,bool limit)
18. {
19. if(pos==-1) {return sum<=all;}
20. if(sum>all) return 0;
21. if(!limit && dp[pos][all-sum]!=-1) return dp[pos][all-sum];
22. int up=limit ? a[pos] : 9;
23. int ans=0;
24. for(int i=0;i<=up;i++)
25. {
26. ans+=dfs(pos-1,sum+i\*(1<<pos),limit && i==a[pos]);
27. }
28. if(!limit) dp[pos][all-sum]=ans;
29. return ans;
30. }
31. int solve(int x)
32. {
33. int pos=0;
34. while(x)
35. {
36. a[pos++]=x%10;
37. x/=10;
38. }
39. return dfs(pos-1,0,true);
40. }
41. int main()
42. {
43. int a,ri;
44. int T\_T;
45. int kase=1;
46. scanf("%d",&T\_T);
47. memset(dp,-1,sizeof dp);
48. while(T\_T--)
49. {
50. scanf("%d%d",&a,&ri);
51. all=f(a);
52. printf("Case #%d: %d\n",kase++,solve(ri));
53. }
54. return 0;
55. }

例题 [**POJ 3252**](http://poj.org/problem?id=3252)

这题的约束就是一个数的二进制中0的数量要不能少于1的数量，通过上一题，这题状态就很简单了，dp[pos][num],到当前数位pos,0的数量减去1的数量不少于num的方案数，一个简单的问题，中间某个pos位上num可能为负数(这不一定是非法的，因为我还没枚举完嘛，只要最终的num>=0才能判合法，中途某个pos就不一定了)，这里比较好处理，Hash嘛，最小就-32吧(好像),直接加上32，把32当0用。这题主要是要想讲一下lead的用法，显然我要统计0的数量，前导零是有影响的。至于!lead&&!limit才能dp，都是类似的，自己慢慢体会吧。

**[cpp]** [view plain](https://blog.csdn.net/wust_zzwh/article/details/52100392) [copy](https://blog.csdn.net/wust_zzwh/article/details/52100392)

1. #pragma comment(linker, "/STACK:10240000,10240000")
2. #include<iostream>
3. #include<cstdio>
4. #include<cstring>
5. #include<string>
6. #include<queue>
7. #include<set>
8. #include<vector>
9. #include<map>
10. #include<stack>
11. #include<cmath>
12. #include<algorithm>
13. using namespace std;
14. const double R=0.5772156649015328606065120900;
15. const int N=1e5+5;
16. const int mod=1e9+7;
17. const int INF=0x3f3f3f3f;
18. const double eps=1e-8;
19. const double pi=acos(-1.0);
20. typedef long long ll;
21. int dp[35][66];
22. int a[66];
23. int dfs(int pos,int sta,bool lead,bool limit)
24. {
25. if(pos==-1)
26. return sta>=32;
27. if(!limit && !lead && dp[pos][sta]!=-1) return dp[pos][sta];
28. int up=limit?a[pos]:1;
29. int ans=0;
30. for(int i=0;i<=up;i++)
31. {
32. if(lead && i==0) ans+=dfs(pos-1,sta,lead,limit && i==a[pos]);//有前导零就不统计在内
33. else ans+=dfs(pos-1,sta+(i==0?1:-1),lead && i==0,limit && i==a[pos]);
34. }
35. if(!limit && !lead ) dp[pos][sta]=ans;
36. return ans;
37. }
38. int solve(int x)
39. {
40. int pos=0;
41. while(x)
42. {
43. a[pos++]=x&1;
44. x>>=1;
45. }
46. return dfs(pos-1,32,true,true);
47. }
48. int main()
49. {
50. memset(dp,-1,sizeof dp);
51. int a,b;
52. while(~scanf("%d%d",&a,&b))
53. {
54. printf("%d\n",solve(b)-solve(a-1));
55. }
56. return 0;
57. }

然后就是一些需要自己yy的题：

[**HDU 3709**](http://acm.hdu.edu.cn/showproblem.php?pid=3709)**这题就是要枚举中轴，然后数位dp**

[UVA 1305](http://acm.hust.edu.cn/vjudge/problem/38405) 这题我二分然后数位dp搞(好像不是正解，我水过的)

[**Hbzoj 1799**](http://acm.hust.edu.cn/vjudge/problem/19309)**这题讲一下：**

**（偷偷告诉你，这个oj是单组测试，然后memset什么的都是浮云了）**

**约束：一个数是它自己数位和的倍数，直接dp根本找不到状态，枚举数位和，因为总就162,然后问题就变成了一个数%mod=0，mod是枚举的，想想状态：dp[pos][sum][val]，当前pos位上数位和是sum,val就是在算这个数%mod,（从高位算  \*10+i），因为我们枚举的数要保证数位和等于mod，还要保证这个数是mod的倍数，很自然就能找到这些状态，显然对于每一个mod，val不能保证状态唯一，这是你要是想加一维dp[pos][sum][val][mod],记录每一个mod的状态(这里sum可以用减法，然而val不行，就只能加一维)，那你就想太多了，这样是会超时的(因为状态太多，记忆化效果不好)。这里直接对每一个mod，memset一次就能ac。下面的代码还把limit的当做了状态，因为每次都要初始化，所以能这样，memset在多组外面是不能这样的，不过奇葩的，这代码，如果不把limit当状态，还是在!limit 条件下记录dp，提交一发，时间竟然更短了，可能是每次memset的关系！！！**

**[cpp]** [view plain](https://blog.csdn.net/wust_zzwh/article/details/52100392) [copy](https://blog.csdn.net/wust_zzwh/article/details/52100392)

1. #include<cstdio>
2. #include<cstring>
3. #include<iostream>
4. #include<string>
6. using namespace std;
8. typedef long long ll;
10. ll dp[20][163][163][2];
11. int a[20];
12. ll dfs(int pos,int sum,int val,int mod,bool limit)
13. {
14. if(sum-9\*pos-9>0) return 0;//最坏的情况，这一位及后面的全部为9都不能达到0那就直接GG，这个剪枝不会影响ac
15. if(pos==-1) return sum==0 && val==0;
16. if(dp[pos][sum][val][limit]!=-1) return dp[pos][sum][val][limit];
17. int up=limit?a[pos]:9;
18. ll ans=0;
19. for(int i=0;i<=up;i++)
20. {
21. if(sum-i<0) break;
22. ans+=dfs(pos-1,sum-i,(val\*10+i)%mod,mod,limit && i==a[pos]);
23. }
24. dp[pos][sum][val][limit]=ans;
25. return ans;
26. }
27. ll solve(ll x)
28. {
29. int pos=0;
30. while(x)
31. {
32. a[pos++]=x%10;
33. x/=10;
34. }
35. ll ans=0;
36. for(int i=1;i<=pos\*9;i++)//上限就是每一位都是9
37. {
38. memset(dp,-1,sizeof dp);
39. ll tmp=dfs(pos-1,i,0,i,true);
40. ans+=tmp;
41. }
42. return ans;
43. }
44. int main()
45. {
46. //    cout<<18\*9<<endl;
47. ll le,ri;
48. //    memset(dp,-1,sizeof dp);
49. while(~scanf("%lld%lld",&le,&ri))
50. printf("%lld\n",solve(ri)-solve(le-1));
51. return 0;
52. }
53. /\*
54. 1 1000000000000000000
55. \*/

基本讲的差不多了。前段时间学了点新东西！！

**新的领域--计数转求和**

[HDU 4507](http://acm.hdu.edu.cn/showproblem.php?pid=4507)

这题麻烦就是要求数的平方和。

我们先考虑求和的问题，一个区间，数位dp能在一些约束下计数，现在要这些数的和。其实组合数学搞搞就可以了：如 现在枚举的某一位pos,我统计了这一位枚举i的满足条件的个数cnt，其实只要算i对总和的贡献就可以了，对于一个数而言第pos位是i，那么对求和贡献就是i\*10^pos,就是十进制的权值，然后有cnt个数都满足第pos位是i，最后sum=cnt\*i\*10^pos.原理就是这样平方和可以看做(a\*10^pos+b)^2,a是你当前pos位要枚举的，b其实是个子问题，就是pos之后的位的贡献值，把这个平方展开就可以了！

**[cpp]** [view plain](https://blog.csdn.net/wust_zzwh/article/details/52100392) [copy](https://blog.csdn.net/wust_zzwh/article/details/52100392)

1. #pragma comment(linker, "/STACK:10240000,10240000")
2. #include<iostream>
3. #include<cstdio>
4. #include<cstring>
5. #include<string>
6. #include<queue>
7. #include<set>
8. #include<vector>
9. #include<map>
10. #include<stack>
11. #include<cmath>
12. #include<algorithm>
13. using namespace std;
14. const double R=0.5772156649015328606065120900;
15. const int N=1e5+5;
16. const int mod=1e9+7;
17. const int INF=0x3f3f3f3f;
18. const double eps=1e-8;
19. const double pi=acos(-1.0);
20. typedef long long ll;
21. ll fact[20];
22. void init()
23. {
24. fact[0]=1;
25. for(int i=1;i<20;i++)
26. fact[i]=fact[i-1]\*10%mod;
27. }
28. struct node
29. {
30. ll cnt,sum,sqr;
31. node(ll cnt=-1,ll sum=0,ll sqr=0):cnt(cnt),sum(sum),sqr(sqr){}
32. }dp[20][7][7];
33. int a[20];
34. ll fac(ll x)
35. {
36. return x\*x%mod;
37. }
38. ll dfs(int pos,ll num,ll val,ll&cnt,ll&sum,bool limit)
39. {
40. if(pos==-1) {
41. if(num==0 || val==0)
42. return 0;
43. cnt=1;
44. return 0;
45. }
46. if(!limit && dp[pos][num][val].cnt!=-1) {
47. cnt=dp[pos][num][val].cnt;
48. sum=dp[pos][num][val].sum;
49. return dp[pos][num][val].sqr;
50. }
51. int up=limit?a[pos]:9;
52. ll sq=0;
53. for(int i=0;i<=up;i++)
54. if(i!=7)
55. {
56. ll cn=0,su=0;
57. ll tmp=dfs(pos-1,(num+i)%7,(val\*10+i)%7,cn,su,limit && i==a[pos]);
58. ll tm=i\*fact[pos]%mod;
59. tmp=(tmp+fac(tm)\*cn%mod+(tm\*su%mod)\*2%mod)%mod;//计数之后要更新sum,sqr
60. sum=(sum+su+(i\*fact[pos]%mod)\*cn%mod)%mod;
61. cnt=(cnt+cn)%mod;
62. sq=(sq+tmp)%mod;
63. }
64. if(!limit) dp[pos][num][val]=node(cnt,sum,sq);
65. return sq;
66. }
67. ll solve(ll x)
68. {
69. int pos=0;
70. while(x)
71. {
72. a[pos++]=x%10;
73. x/=10;
74. }
75. ll t1=0,t2=0;
76. return dfs(pos-1,0,0,t1,t2,true);
77. }
78. bool judge(ll x)
79. {
80. int sum=0;
81. int pos=0;
82. if(x%7==0) return false;
83. while(x)
84. {
85. if(x%10==7) return false;
86. sum+=x%10;
87. x/=10;
88. }
89. sum%=7;
90. return sum!=0;
91. }
92. int main()
93. {
94. init();
95. for(int i=0;i<20;i++)
96. for(int j=0;j<7;j++)
97. for(int k=0;k<7;k++)//memset
98. {
99. dp[i][j][k].cnt=-1;
100. dp[i][j][k].sum=0;
101. dp[i][j][k].sqr=0;
102. }
103. int T\_T;
104. scanf("%d",&T\_T);
105. while(T\_T--)
106. {
107. ll le,ri;
108. scanf("%I64d%I64d",&le,&ri);
109. ll ans=solve(ri)-solve(le-1);
110. ans=(ans%mod+mod)%mod;
111. printf("%I64d\n",ans);
112. }
113. return 0;
114. }