// 数位DP模板

typedef long long ll;

int a[20];

ll dp[20][state];//不同题目状态不同

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

{

//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

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 \*/

//第二个就是记忆化(在此前可能不同题目还能有一些剪枝)

if(!limit && !lead && dp[pos][state]!=-1) return dp[pos][state];

/\*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\*/

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

ll ans=0;

//开始计数

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

{

if() ...

else if()...

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

/\*这里还算比较灵活，不过做几个题就觉得这里也是套路了

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

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

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，

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\*/

}

//计算完，记录状态

if(!limit && !lead) dp[pos][state]=ans;

/\*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\*/

return ans;

}

ll solve(ll x)

{

int pos=0;

while(x)//把数位都分解出来

{

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

x/=10;

}

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

}

int main()

{

ll le,ri;

while(~scanf("%lld%lld",&le,&ri))

{

//初始化dp数组为-1,这里还有更加优美的优化,后面讲

printf("%lld\n",solve(ri)-solve(le-1));

}

}