数位 DP

经典题型

数位 DP 问题往往都是这样的题型，给定一个闭区间 $[l,r]$ 让你求这个区间中满足 **某种条件** 的数的总数。

> **例题** SCOI2009 windy 数
>
> 题目大意：给定一个区间 $[l,r]$ 求其中满足条件 **不含前导 0 且相邻两个数字相差至少为 2** 的数字个数

首先我们将问题转化成更加简单的形式。设 ans_i 表示在区间 $[1,i]$ 中满足条件的数的数量，那么所求的答案就是 ans_r-ans_{l-1}

分开求解这两个问题。

对于一个小于 n 的数，它从高到低肯定出现某一位，使得这一位上的数值小于 n 这一位上对应的数值。而之前的所有位和 n 上的对应位相等。

有了这个性质，我们可以定义 $f(i,st,op)$ 表示当前将要考虑的是从高到低的第 i 位，当前该前缀的状态为 st 且前缀和当前求解的数字的大小关系是 op （$op=1$ 表示等于 $op=0$ 表示小于）时的数字个数。在本题中，这个前缀的状态就是上一位的值，因为当前将要确定的位不能取哪些数只和上一位有关。在其他题目中，这个值可以是：前缀的数字和，前缀所有数字的 gcd 该前缀取模某个数的余数，也有两种或多种合用的情况。

写出 **状态转移方程** $$ f(i,st,op)=\sum^{maxx}_{k=1}f(i+1,k,op=1\ and\ k=maxx)\quad (\mid st-k\mid\ge 2) $$ 这里的 k 就是当前枚举的下一位的值，而 $maxx$ 就是当前能取到的最高位。因为如果 $op=1$ 那么你在这一位上取的值一定不能大于求解的数字上该位的值，否则则没有限制。

我们发现，尽管前缀所选择的状态不同，而 f 的三个参数相同，答案就是一样的。为了防止这个答案被计算多次，可以使用记忆化搜索的方式实现。

核心代码：

```
int dfs(int x, int st, int op)  // op=1 =;op=0 <
{
  if (!x) return 1;
  if (!op && ~f[x][st]) return f[x][st];
  int maxx = op ? dim[x] : 9, ret = 0;
  for (int i = 0; i <= maxx; i++) {
    if (abs(st - i) < 2) continue;
    if (st == 11 && i == 0)
      ret += dfs(x - 1, 11, op & (i == maxx));
    else
      ret += dfs(x - 1, i, op & (i == maxx));
  }
  if (!op) f[x][st] = ret;
  return ret;
}
int solve(int x) {
  memset(f, -1, sizeof f);
```

```
  dim.clear();
  dim.push_back(-1);
  int t = x;
  while (x) {
    dim.push_back(x % 10);
    x /= 10;
  }
  return dfs(dim.size() - 1, 11, 1);
}
```

习题

BZOJ 3679 数字之积

ZJOI2010 count 数字计数

Ahoi2009 self 同类分布

洛谷 P3413 SAC#1 - 萌数

HDU 6148 Valley Number

CF55D Beautiful numbers

CF628D Magic Numbers

参考链接

OI Wiki

From:
https://wiki.cvbbacm.com/ - **CVBB ACM Team**

Permanent link:
https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:lgwza:%E6%95%B0%E4%BD%8D_dp

Last update: **2020/08/05 23:09**