动态规划进阶（二）

数位统计动态规划与状态压缩动态规划

马玉坤

哈尔滨工业大学计算机科学与技术学院

2017 年 8 月 18 日

数位统计动态规划

一般形式

给定一个区间 $[L, R]$，求在此区间内满足某个条件限制（特性）的数的个数或者加和。一般有多组询问。

一般套路

1. 先将问题 $[L, R]$ 转化为 $[1, L-1]$ 和 $[1, R]$ 的问题

数位统计动态规划

一般形式

给定一个区间 $[L, R]$，求在此区间内满足某个条件限制（特性）的数的个数或者加和。一般有多组询问。

一般套路

1. 先将问题 $[L, R]$ 转化为 $[1, L-1]$ 和 $[1, R]$ 的问题
2. dp 状态有两部分：

数位统计动态规划

一般形式

给定一个区间 $[L, R]$，求在此区间内满足某个条件限制（特性）的数的个数或者加和。一般有多组询问。

一般套路

1. 先将问题 $[L, R]$ 转化为 $[1, L-1]$ 和 $[1, R]$ 的问题
2. dp 状态有两部分：
    1. 跟枚举有关的部分：已枚举到的位数，以及前若干位数是不是已经到了上限（范围的限制）

数位统计动态规划

一般形式

给定一个区间 $[L, R]$，求在此区间内满足某个条件限制（特性）的数的个数或者加和。一般有多组询问。

一般套路

1. 先将问题 $[L, R]$ 转化为 $[1, L-1]$ 和 $[1, R]$ 的问题
2. dp 状态有两部分：
   1. 跟枚举有关的部分：已枚举到的位数，以及前若干位数是不是已经到了上限（范围的限制）
   2. 跟特性有关的部分：已枚举的位上的数已经到达的状态（特性的限制）

数位统计动态规划 (Cont'd)

对解题的忠告

1. 数位 DP 较容易对拍，可以本地测试，减少 WA

数位统计动态规划 (Cont'd)

对解题的忠告

1. 数位 DP 较容易对拍，可以本地测试，减少 WA
2. 耐心，细心

数位统计动态规划 (Cont'd)

对解题的忠告

1. 数位 DP 较容易对拍，可以本地测试，减少 WA
2. 耐心，细心
3. 重构/重写

Find a car

Codeforces Round #415 (Div. 1) C

题目

给定一个无穷大的矩阵，其中第 i 行第 j 列的元素 $mat(i,j)$ 为最小的不在集合 $\{mat(1,j), mat(2,j), \ldots, mat(i-1,j), mat(i,1), mat(i,2), \ldots, mat(i,j-1)\}$ 中出现过的正整数。

1	2	3	4	5
2	1	4	3	6
3	4	1	2	7
4	3	2	1	8
5	6	7	8	1

图 1: 矩阵左上角的 5 个元素

然后会有 $Q(\leq 10^4)$ 个询问，每个询问给定 x_1, y_1, x_2, y_2, k，回答子矩阵 $(x_1, y_1) - (x_2, y_2)$ 内所有小于小于等于 k 的数的和。$x_1, y_1, x_2, y_2 \leq 10^9$ 且 $k \leq 2 \times 10^9$。

Find a car (Cont'd)

Codeforces Round #415 (Div. 1) C

第一个问题

第 i 行第 j 列，即 $mat(i,j)$ 该怎么求？能不能用一个简单的式子来表示？

Find a car (Cont'd)

Codeforces Round #415 (Div. 1) C

第一个问题

第 i 行第 j 列，即 $mat(i,j)$ 该怎么求？能不能用一个简单的式子来表示？
先把矩阵里的每个元素都减 1，行数和列数从 1 开始变为从 0 开始。

Find a car (Cont'd)

Codeforces Round #415 (Div. 1) C

第一个问题

第 i 行第 j 列，即 $mat(i,j)$ 该怎么求？能不能用一个简单的式子来表示？

先把矩阵里的每个元素都减 1，行数和列数从 1 开始变为从 0 开始。

此时可转化为两堆石子（分别为 i 和 j)，做 NIM 游戏。

$mat(i,j) =$ 各个子状态的 SG 函数 $= i \oplus j$ ($\oplus$ 为异或)。对于原矩阵就有 $mat(i,j) = (i-1) \oplus (j-1) + 1$。

不懂原理也没关系，今天我们的重点不在这。

Find a car (Cont'd)

Codeforces Round #415 (Div. 1) C

异或

$$
\begin{array}{cccccccccc}
 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & & \text{(left)} \\
= & \oplus & \oplus & \oplus & \oplus & \oplus & \oplus & \oplus & & \\
 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & & \text{(right)} \\
= & 0 & 0 & 1 & 1 & 1 & 1 & 0 & &
\end{array}
$$

图 2: 异或

Find a car (Cont'd)

Codeforces Round #415 (Div. 1) C

二维前缀和

设 $sum(x,y)=\sum_{i=1}^{x}\sum_{j=1}^{j}a(i,j)$，那么我们有

$$\sum_{i=x_1}^{x_2}\sum_{j=y_1}^{y_2}a(i,j) \tag{1}$$

$$=sum(x_2,y_2)-sum(x_1-1,y_2) \tag{2}$$

$$-sum(x_2,y_1-1)+sum(x_1-1,y_1-1) \tag{3}$$

Find a car (Cont'd)

Codeforces Round #415 (Div. 1) C

数位 DP

问题转化成求 $\sum_{i=1}^{x}\sum_{j=1}^{y}[i\oplus j\le k](i\oplus j)$
设 $f(di,dx,dy,dk)$ 为这个状态下的数字个数，$g(di,dx,dy,dk)$ 为数字和。其中 di 表示从最高位开始已枚举到的位数，dx,dy 分别表示已枚举的 x 与 y 的前缀是否到达上限，dk 表示已枚举的 x 和 y 的前缀的异或是否达到 k 的上限。
详见代码。

Find a car (Cont'd)

Codeforces Round #415 (Div. 1) C

数位 DP (Cont'd)

为什么这样设计状态？举个例子。
比如 $x = 1010, y = 0101, k = 1110$，枚举完了高三位，此时已枚举的 $x' = 101$，$y' = 010$，且它们的异或值 $k' = 111$，这时候的状态应该是 $f(0, 1, 1, 1)$（枚举到第 0 位，且 x, y, k 前缀都达到限制）。
那么我们枚举低位的时候就不能随便枚举 x 和 y，否则会超出 x 和 y 的限制。同时，我们还要注意已枚举的前缀是否达到 k 的上限，如果达到了 k 的上限，同样不能随便枚举 x 和 y。枚举第 0 位时，我们实际上 x 只能取 0，y 只能取 0 或 1，但 y 不能取 1，因为这时候 $x \oplus y = 1111$，超出了范围。

Find a car (Cont'd)

Codeforces Round #415 (Div. 1) C

总结

四个状态中前三个状态（位数，x 前缀，y 前缀）都是关于枚举的范围的，而第四个状态可以理解成枚举的范围，也可以理解成数的特性。在这里我们使用了布尔变量，表示前缀是否达到上限，从而使得我们枚举后面的位时知道是否需要注意范围的限制。

递推求动态规划

```
for (int i = 0; i < n; i++) {
  for (int j = volume[i]; j <= capacity; j++) {
    dp[i][j] = max(dp[i-1][j], dp[i-1][j-volume[i]] + value
  }
}
```

递归求动态规划（记忆化搜索）

```
int dfs(int i, int j) {
  if (i == 0) {
    return 0;
  }
  if (vis[i][j]) {
    return dp[i][j];
  }
  vis[i][j] = 1;
  return dp[i][j] = max(dfs(i-1, j-1),
                        dfs(i-1, j-volume[i]) + value[i]);
}
```

两种方法的比较

- 递推的优势

两种方法的比较

- 递推的优势
  1. 常数小

两种方法的比较

- 递推的优势
  1. 常数小
  2. 不需递归，不怕爆栈

两种方法的比较

- 递推的优势
  1. 常数小
  2. 不需递归，不怕爆栈
- 递归的优势

两种方法的比较

- 递推的优势
  1. 常数小
  2. 不需递归，不怕爆栈
- 递归的优势
  1. 对有效状态数较少的问题可大大提高效率，避免无效状态的搜索

两种方法的比较

- 递推的优势
  1. 常数小
  2. 不需递归，不怕爆栈
- 递归的优势
  1. 对有效状态数较少的问题可大大提高效率，避免无效状态的搜索
  2. 实现简单，更符合人类思维

K-wolf Number (Cont'd)

2016 Multi-University Training Contest 5 07

问题

K-wolf 数字定义为：十进制表示下，任意相邻的 k 位数字都各不相同。
给定 L, R, K，求区间 $[L, R]$ 内的 K-wolf 数字的个数。
$L, R \leq 10^{18}; K \leq 5$

K-wolf Number (Cont'd)

2016 Multi-University Training Contest 5 07

解法

转化为 $1, \ldots, n$ 的 K-wolf 数个数的问题。
设 $dp(i, j, k)$ 表示从最高位枚举到第 i 位，j 表示已枚举的位有没有达到 n 的上限，k 存的是已枚举的位中后 k-1 位的值。这样我们就能知道。

K-wolf Number (Cont'd)

2016 Multi-University Training Contest 5 07

解法 (Cont'd)

注意到一个问题：刚才的题目 (Find a car) 中，我们实际上将所有数都加上了前导零，补成了 31 位。然而在这个题目中，如果我们仍然这么操作，从第 8 位开始枚举，补上前导零，把所有数都变成 18 位数，那么像 29 这样的数就会变成 000000000000000029，不符合 K-wolf 数的定义 (有连续的 0)，所以我们不能算上前导零。

K-wolf Number (Cont'd)

2016 Multi-University Training Contest 5 07

解法 (Cont'd)

注意到一个问题：刚才的题目 (Find a car) 中，我们实际上将所有数都加上了前导零，补成了 31 位。然而在这个题目中，如果我们仍然这么操作，从第 8 位开始枚举，补上前导零，把所有数都变成 18 位数，那么像 29 这样的数就会变成 000000000000000029，不符合 K-wolf 数的定义 (有连续的 0)，所以我们不能算上前导零。

一种方案是：在 dp 状态中，加入一维指示之前枚举的位是否全是 0。

另一种方案是：手动枚举数的位数以及最高位的数字，然后剩下再 dp。

K-wolf Number (Cont'd)

2016 Multi-University Training Contest 5 07

Stabilization

2016 Multi-University Training Contest 6 06