

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

1 数位统计动态规划

- 理论
- 题目
 - Find a car
 - 两种 dp 数组求解的写法
 - K-wolf Number
 - 淘金

2 状态压缩动态规划

- 原理
- 题目
 - Stabilization
 - 炮兵阵地

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 $\{mat(1, j), mat(2, j), \dots, mat(i - 1, j), mat(i, 1), mat(i, 2), \dots, mat(i, j - 1)\}$ 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

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 $k \leq 2 \times 10^9$ 。

图 1: 矩阵左上角的 5 个元素

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

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

图 2: 异或

Find a car (Cont'd)

Codeforces Round #415 (Div. 1) C

二维前缀和

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

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

$$= sum(x_2, y_2) - sum(x_1 - 1, y_2) \quad (2)$$

$$- sum(x_2, y_1 - 1) + sum(x_1 - 1, y_1 - 1) \quad (3)$$

Find a car (Cont'd)

Codeforces Round #415 (Div. 1) C

数位 DP

问题转化成求 $\sum_{i=1}^x \sum_{j=1}^y [i \oplus j \leq k] (i \oplus j)$

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[i])  
    }  
}
```

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。

$L, R \leq 10^{18}; K \leq 5$

题目

K-wolf Number (Cont'd)

2016 Multi-University Training Contest 5 07

解法

转化为 $1, \dots, n$ 的 K-wolf 数个数的问題。

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。

题目

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。

题目

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。

具体实现见文件 k-wolf_1.cpp 与 k-wolf_2.cpp。

题目

淘金

SDOI 2013 Round 1 Day 2 Problem 3

题目

初始时，在 $1 \leq x \leq n, 1 \leq y \leq n$ 上的每个点上，都有一块金子。在一阵风过后， (x, y) 位置上的金子会到 $(f(x), f(y))$ 位置上，其中 $f(x)$ 等于 x 各个位上的数字乘积。

这阵风刮过之后，小 Z 希望依次走到 k 个点上进行金子的收集。请最大化收集到的金子总数。结果对 $10^9 + 7$ 取模。

$n \leq 10^{12}; k \leq \min(10^5, n^2)$

淘金 (Cont'd)

SDOI 2013 Round 1 Day 2 Problem 3

第一个问题

最后有多少个金子落到了 (x, y) 坐标上？

淘金 (Cont'd)

SDOI 2013 Round 1 Day 2 Problem 3

第一个问题

最后有多少个金子落到了 (x, y) 坐标上？

$$\left(\sum_{i=1}^n [f(i) = x]\right) \times \left(\sum_{j=1}^n [f(j) = y]\right)$$

所以为了简化，把 x 与 y 坐标分开考虑。

淘金 (Cont'd)

SDOI 2013 Round 1 Day 2 Problem 3

第二个问题

只考虑一维的话，什么样的坐标上会有金子？

淘金 (Cont'd)

SDOI 2013 Round 1 Day 2 Problem 3

第二个问题

只考虑一维的话，什么样的坐标上会有金子？
质因子只包括 2, 3, 5, 7（当然也可以不包括）。

$$\log_2 10^{12} \times \log_3 10^{12} \times \log_5 10^{12} \times \log_7 10^{12} \approx 244411$$

实际上，有金子的坐标数没那么 10^{12} 多。

淘金 (Cont'd)

SDOI 2013 Round 1 Day 2 Problem 3

数位 DP

设 $dp(i, l, c_2, c_3, c_5, c_7)$ 。 i 表示枚举到的位数， l 表示已枚举的位是否达到 n 的上限， c_2, c_3, c_5, c_7 分别表示已枚举的位上的数相乘得到的乘积中 2, 3, 5, 7 四种质因子的数目。

淘金 (Cont'd)

SDOI 2013 Round 1 Day 2 Problem 3

其实还有一个问题

给定序列 a_1, a_2, \dots, a_n (已经从大到小排好序), 在矩阵 M 中找到前 m 大的值的和。其中 $M_{i,j} = a[i] \times a[j]$ 。

淘金 (Cont'd)

SDOI 2013 Round 1 Day 2 Problem 3

其实还有一个问题

给定序列 a_1, a_2, \dots, a_n (已经从大到小排好序), 在矩阵 M 中找到前 m 大的值的和。其中 $M_{i,j} = a[i] \times a[j]$ 。

最大值可能是 $a_1 \times a_1, a_2 \times a_1, \dots, a_n \times a_1$, 然后呢? (显然 $a_1 \times a_1$ 最大。)

次大值可能是 $a_1 \times a_2, a_2 \times a_1, a_3 \times a_1, \dots, a_n \times a_1$ 。然后呢? (假设 $a_2 \times a_1$ 最大。)

第 3 大值可能是 $a_1 \times a_2, a_2 \times a_2, a_3 \times a_1, a_n \times a_1$ 。

维护矩阵每一行从左往右第一个没遍历的元素, 然后放入最大堆中, 每次从堆顶中取出 x , 此时 x 所在的行的第一个没遍历的元素变成了 x 右边的数, 把 x 弹出, 并把 x 右边的数加入堆中。

重复 m 次。时间复杂度 $O((m+n) \log n)$ 。

实际上还可以优化到 $O(m \log n)$ 。

使用二进制数进行状态压缩

使用二进制数表示状态的动机

- 很多问题中，每个元素都有两种状态（在集合中，不在集合中；被占据，没被占据）
- 计算机使用二进制存储数据，使用原生的数字运算便可实现很多集合运算

使用二进制数进行状态压缩 (Cont'd)

使用二进制数表示状态的方法

- ① $\&$: “与” 运算, 集合求交
- ② $|$: “或” 运算, 集合求并
- ③ \wedge : “异或” 运算, 集合对称差 $((A - B) \cup (B - A))$, 一般用作求补集 (也就是左操作数表示的集合包含右操作数表示的集合时使用)
- ④ \ll : “左移” 运算, $x \ll d$, 将 x 在二进制下的每一位向左移动 d 位, 最右边用 0 填充
- ⑤ \gg : “右移” 操作, $x \gg d$, 将 x 在二进制下的每一位向右移动 d 位, 最左边用 0 填充 (只考虑正数或无符号数)

例子: $((S \gg 5) \& 1)$ ——获取 S 状态下第 5 个元素的状态。

其他方法

视题目而定

有些题目中元素状态有若干种，例如连通性状态压缩动态规划中，插头的种类数就决定了每个元素的状态数目。

3 进制、4 进制、阶乘表示都有可能。

题目

Stabilization

2016 Multi-University Training Contest 6 06

题目

给定序列 a_1, a_2, \dots, a_n , 求一个 x , 使得

$$\sum_{i=1}^{n-1} |(a_{i+1} \oplus x) - (a_i \oplus x)|$$

最小。

$$n \leq 10^5; a_i < 2^{20}$$

题目

Stabilization (Cont'd)

2016 Multi-University Training Contest 6 06

第一个问题

异或 x 之后，相邻的数的大小关系会改变，怎么处理？

题目

Stabilization (Cont'd)

2016 Multi-University Training Contest 6 06

第一个问题

异或 x 之后，相邻的数的大小关系会改变，怎么处理？
因为异或的是同一个数 x 。所以两个数不同的二进制位异或之后仍然不同，相同的二进制位异或之后仍然相同。

题目

Stabilization (Cont'd)

2016 Multi-University Training Contest 6 06

第一个问题

异或 x 之后，相邻的数的大小关系会改变，怎么处理？

因为异或的是同一个数 x 。所以两个数不同的二进制位异或之后仍然不同，相同的二进制位异或之后仍然相同。

我们只需要看两个相邻的数的二进制表示中最高的不同的位，就可以知道相邻的两个数谁比谁大，谁减谁等于差的绝对值。

Stabilization (Cont'd)

2016 Multi-University Training Contest 6 06

解法

设 $S_i = \{j | a_j \text{ 与 } a_{j+1} \text{ 二进制表示中最高不同的位为 } i\}$ 。我们可以设

$$dp(x, i) = \sum_{j \in S_i} |(a_j \oplus x) - (a_{j+1} \oplus x)| \quad (4)$$

$$f(i) = |S_i| \quad (5)$$

$$g(i, k) = \sum_{j \in S_i} [a_j \text{ 和 } a_{j+1} \text{ 第 } k \text{ 位分别为 } 0 \text{ 和 } 1] \quad (6)$$

$$h(i, k) = \sum_{j \in S_i} [a_j \text{ 和 } a_{j+1} \text{ 第 } k \text{ 位分别为 } 1 \text{ 和 } 0] \quad (7)$$

Stabilization (Cont'd)

2016 Multi-University Training Contest 6 06

解法 (Cont'd)

设 x 的二进制表示中最高的为 1 的位为第 $highbit(x)$ 位。设 $y = x \oplus 2^{highbit(x)}$ ，此时假设我们算完了 $dp(y, 0 \dots 19)$ ，怎么算 $dp(x, 0 \dots 19)$ 呢？

题目

Stabilization (Cont'd)

2016 Multi-University Training Contest 6 06

解法 (Cont'd)

设 x 的二进制表示中最高的为 1 的位为第 $highbit(x)$ 位。设 $y = x \oplus 2^{highbit(x)}$ ，此时假设我们算完了 $dp(y, 0 \dots 19)$ ，怎么算 $dp(x, 0 \dots 19)$ 呢？

$dp(x, 0 \dots highbit(x) - 1) = dp(y, 0 \dots highbit(x) - 1)$ （这部分没有影响）

Stabilization (Cont'd)

2016 Multi-University Training Contest 6 06

解法 (Cont'd)

设 x 的二进制表示中最高的为 1 的位为第 $highbit(x)$ 位。设 $y = x \oplus 2^{highbit(x)}$ ，此时假设我们算完了 $dp(y, 0 \dots 19)$ ，怎么算 $dp(x, 0 \dots 19)$ 呢？

$dp(x, 0 \dots highbit(x) - 1) = dp(y, 0 \dots highbit(x) - 1)$ （这部分没有影响）

$dp(x, highbit(x)) = 2^{highbit(x)+1} \times f(highbit(x)) - dp(y, highbit(x))$

（原来是 $i - j$ ，因为最高不同的位取反，现在变成了

$2^{highbit(x)+1} + j - i = 2^{highbit(x)+1} - (i - j)$ ）

题目

Stabilization (Cont'd)

2016 Multi-University Training Contest 6 06

解法 (Cont'd)

设 x 的二进制表示中最高的为 1 的位为第 $highbit(x)$ 位。设 $y = x \oplus 2^{highbit(x)}$ ，此时假设我们算完了 $dp(y, 0 \dots 19)$ ，怎么算 $dp(x, 0 \dots 19)$ 呢？

$dp(x, 0 \dots highbit(x) - 1) = dp(y, 0 \dots highbit(x) - 1)$ （这部分没有影响）

$dp(x, highbit(x)) = 2^{highbit(x)+1} \times f(highbit(x)) - dp(y, highbit(x))$

（原来是 $i - j$ ，因为最高不同的位取反，现在变成了

$2^{highbit(x)+1} + j - i = 2^{highbit(x)+1} - (i - j)$ ）

$i: 0 \underline{1} 0 1 1 1 0$

$j: 0 \underline{0} 0 1 0 0 1$

$i: 0 \underline{0} 0 1 1 1 0$

$j: 0 \underline{1} 0 1 0 0 1$

图 3: 最高位取反前

图 4: 最高位取反后

题目

Stabilization (Cont'd)

2016 Multi-University Training Contest 6 06

解法 (Cont'd)

那 $dp(x, \text{highbit}(x) + 1 \dots 19)$ 呢？留作练习题。

注意：不要忘了用 $g(i, k)$ 和 $h(i, k)$ 。

启示

需要预处理的东西都不是一下子就能想到的，需要在实际进行计算的尝试时才能发现。

题目

炮兵阵地

NOI 2001

问题

给定一个 $n \times m$ 的网格图，需要在其上部署炮兵部队，网格图上的每个网格可能是平原（可以部署部队），或者是山地（不可以部署部队）。

问最多能部署多少支炮兵部队，使任意两个部队不在对方的攻击范围，炮兵的攻击范围见图5。

$n \leq 100; m \leq 10$

P♞	P♞	H♞	P♞	H♞	H♞	P♞	P♞	♞
P♞	H♞	P♞	H♞	P♞	H♞	P♞	P♞	♞
P♞	P♞	P♞	H♞	H♞	H♞	P♞	H♞	♞
H♞	P♞	H♞	P♞	P♞	P♞	P♞	H♞	♞
H♞	P♞	P♞	P♞	P♞	H♞	P♞	H♞	♞
H♞	P♞	P♞	H♞	P♞	H♞	H♞	P♞	♞
H♞	H♞	H♞	P♞	P♞	P♞	P♞	H♞	♞

图 5: 炮兵部队攻击范围

题目

炮兵阵地 (Cont'd)

NOI 2001

解法

按行转移做状态压缩动态规划。

由于上下攻击范围是相互的，所以我们只需要让上面的部队不要打到下面就可以。

设 $dp(i, S)$ 表示第 1 行到第 i 行已经确定，且第 i 行的状态为 S 的方案数。 S 实际上可以用 3 进制数来表示。

- S 的第 j 位为 0，意味着 (i, j) 与 $(i-1, j)$ 都没有放部队，此时 $(i+1, j)$ 可以放部队
- S 的第 j 位为 1，意味着 $(i-1, j)$ 放了部队，此时 $(i+1, j)$ 不可以放部队
- S 的第 j 位为 2，意味着 (i, j) 放了部队，此时 $(i+1, j)$ 也不可以放部队

题目

炮兵阵地 (Cont'd)

NOI 2001

解法 (Cont'd)

有了第 i 行的状态的表示，我们就能知道第 $i+1$ 行怎样枚举决策了，也就是说这一行放几支部队，分别放到哪。

题目

炮兵阵地 (Cont'd)

NOI 2001

解法 (Cont'd)

有了第 i 行的状态的表示，我们就能知道第 $i+1$ 行怎样枚举决策了，也就是说这一行放几支部队，分别放到哪。

第 i 行一共有 3^{10} 种状态，对于每个状态，第 $i+1$ 行都有 2^{10} 个决策（每个格子放或者不放）。

然而，事实上，有效的状态数并没有那么多，可以预处理一下有用的状态，加一些剪枝。比如，同一行不可能放两个相邻的炮兵部队（隔一个格子也不行）。

题目

多米诺骨牌覆盖问题

问题

多米诺骨牌的大小为 1×2 ，现在你要用多米诺骨牌去覆盖 $n \times m$ 大小的方格图。使得方格图中的每个方格被且仅被一个多米诺骨牌的一格覆盖。

问一共有多少覆盖方案？

$n \times m \leq 256$

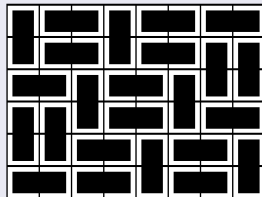


图 6: 6×8 网格图的多米诺骨牌覆盖

多米诺骨牌覆盖问题 (Cont'd)

初步分析

$n \times m \leq 256$ 的意思是说 $\min(n, m) \leq 16$, 不失一般性, 我们设 $m \leq n$ 。(n 为行数)

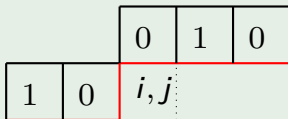
这样每行的格子数不多 (≤ 16)。

但不像上一个题里, 我们一行一行地决策, 这次我们一格一格地决策。

题目

多米诺骨牌覆盖问题 (Cont'd)

解法



设 $dp(i, j, S)$ 表示轮廓线状态为 S 时的方案数。 S 可以用一个二进制数表示，因为轮廓线上第 j 列的格子要么

- 0: 没被覆盖
- 1: 被覆盖了

图 7: 轮廓线示意图 (这图画起来好麻烦的)

		0	1	0
1	0	i, j		

设 $dp(i, j, S)$ 表示轮廓线状态为 S 时的方案数。 S 可以用一个二进制数表示, 因为轮廓线上第 j 列的格子要么

- 0: 没被覆盖
- 1: 被覆盖了

那么决策有哪几种呢？

- 放一个横着的骨牌（主动决策）
- 放一个竖着的骨牌（被动决策，要看上一行同列的方格有没有被覆盖）

多米诺骨牌覆盖问题 (Cont'd)

解法 (Cont'd)

代码写起来很简单。

```
#define _1(i, j) (((i)>>(j))&1)
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        swap(cur, lst);
        dp[cur].clear();
        for (auto it = dp[lst].begin(); it != dp[lst].end(); ++it) {
            int s = it->first,
                v = it->second;
            if (j && !_1(s, j) && _1(s, j - 1)) {
                addMod(dp[cur][s ^ (1<<(j-1))], v);
            }
            addMod(dp[cur][s ^ (1<<j)], v);
        }
    }
    addMod(f[i][m], dp[cur].count(0)?dp[cur][0]:0);
}
```