1 数独

有些复杂的模拟题，这个题的初衷主要是考虑了 NOIP 2017 day1 第二题的情况。这道题目虽然看起来很啰嗦，但实现起来细节并不多，边界条件也很少，相信有一定代码能力的同学可以取得满分，大多数同学也能取得较高分数，即便是刚入门开始学信息学竞赛的同学，也不至于得零分。

2 分糖果

测试点 1 ~ 4。暴力枚举顺序，复杂度 $O(n\times n!)$或 $O(n!)$可以得到 20 分。

测试点 1 ~ 8。注意到获得最多糖果的小朋友，一定是最后一位小朋友，这个可以从糖果的定义中很容易发现 c_i 是单调递增的。考虑状态压缩的动态规划，记 $dp[S][i]$表示已经安排好的小朋友组成的集合为 S，站在队尾的小朋友为 i 位小朋友，第 i 位小朋友最少获得多少糖果。时间复杂度 $O(2^n n^2)$，可以得到 40 分。

测试点 1 ~ 8。从样例中我们可以发现，满足条件的排队方案并不唯一，因此对于前 40 分数据，只需要每次随机一个排列，多次随机并卡时取最优解，也可以通过。

测试点 9 ~ 10。任意一种排队方式都是最优解。

测试点 11 ~ 12。当 $b_i = a_i + 1$ 时，按照 a_i 递增的顺序排队可以得到最优解。

测试点 1 ~ 12。结合上面几部分算法，可以得到 60 分。

测试点 1 ~ 20。考虑 $n = 2$ 时的情况：假定两个人分别为(a, b)，(c, d)，则当且仅当 $\min(a,d) \le \min(b,c)$时，把(a, b)放在前面更优，否则把(c, d)放在前面更优，这里不给出证明。（具体证明可以参考 2014 年北京市高考理科数学第 20 题第 2 问）。实际上，$n = 2$ 的结论可以进行扩展。我们定义第 i 个小朋友比第 j 个小朋友小，当且仅当 $\min(a_i,b_j) < \min(a_j,b_i)$，以这个规则进行排序，时间复杂度 $O(n\log n)$。这样得到的新队伍一定是满足题目要求的最优解之一，当然还可能存在其它最优解。

3.1 题意简述

用 $\oplus$ 表示异或。有 x 和 y 两个非负整数。在 $[0, n)$ 内等概率随机一个数作为 x 的值，y 有 p 的概率取在 $[0, n)$ 中使得 $x \oplus y$ 最大的数，有 $1-p$ 的概率取在 $[0, n)$ 内等概率随机的一个数。求 $x \oplus y$ 的期望值。$n \le 10^{18}$。

3.2 分析解答

3.2.1 初步分析

定义 $f(a)$ 表示当 $x = a$ 时使得 $x \oplus y$ 最大的 y 的值。根据期望的定义，答案应为

$$(1-p) \cdot \frac{1}{n^2} \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} (x \oplus y) + p \cdot \frac{1}{n} \sum_{x=0}^{n-1} (x \oplus f(x))$$

这个式子分为两部分，前一部分中 y 为 $[0, n)$ 中等概率随机的一个数，后一部分中 $y = f(x)$。我们将分别讨论这两部分。

3.2.2 第一部分

根据答案的计算式，我们需要求出所有 $x \oplus y$ 的和。

- **算法 1**
  对于 $n \le 100$ 的部分，对于 $\forall x \in [0, n)$，从 0 到 $n-1$ 开始枚举 y，对于当前枚举到的 y，将答案加上 $x \oplus y$。最后将累加的结果除以 n^2 输出即可。

- **算法 2**
  当 $n = 2^k$ 时，可以观察到答案的值为；

  $$\frac{n-1}{2}$$

  这里不给出证明。

- **算法 3**
  通过一些观察，我们可以得到这样一个性质：

  我们设两个数进行 $\oplus$ 操作以后第 i 位（第 i 位表示 2^i，下同）为 1 为事件 A，第 j 位为 1 为事件 B，其中 $i \ne j$。由概率知识和位运算性质易知事件 A 和事件 B 独立，因此我们可以分别计算每一位对答案的贡献，然后累加即可。

  我们设从 $[0, n)$ 中选出一个数，第 i 位为 1 的概率为 p_i，则我们的答案就为：

  $$\sum_{i=0}^{\log n} 2p_i \times (1-p_i) \times 2^i$$

  因此我们只需要考虑如何计算 p_i。

对于一个 p_i，我们考虑在区间 $[0,n)$ 中的连续自然数，则在区间 $[0,2^i)$ 中第 i 位都为 0，区间 $[2^i,2^{i+1})$ 中第 i 位都为 1。一般地说，在区间 $[k\times 2^{(i+1)}, k\times 2^{(i+1)}+2^i)$ 中第 i 位为 0，在区间 $[k\times 2^{i+1}+2^i,(k+1)\times 2^{i+1})$ 中第 i 位为 1。因此在区间 $[0,n)$ 中第 i 位为 1 的数的个数为：

$$\left\lfloor \frac{n}{2^{i+1}} \right\rfloor + \max([n \bmod 2^{i+1}] - 2^i, 0)$$

由频率等于概率知：

$$p_i = \frac{\lfloor \frac{n}{2^{i+1}} \rfloor + \max([n \bmod 2^{i+1}] - 2^i, 0)}{n}$$

这个算法的复杂度为 $O(\log n)$。

3.2.3 第二部分

根据答案的计算式，我们需要求出所有 $x \oplus f(x)$ 的和。

- **算法 1**
  对于 $n \leq 100$ 的部分，对于 $\forall x \in [0,n)$，从 0 到 $n-1$ 开始枚举 y，找到一个使得 $x\oplus y$ 最大的 y 作为 $f(x)$ 的值，将答案加上 $x \oplus f(x)$。最后将累加的结果除以 n 输出即可。

- **算法 2**
  当 $n = 2^k$ 时，可以观察到答案的值为：

  $$n-1$$

  这里不给出证明。

- **算法 3**
  我们先考虑如何快速求出 $f(x)$。如果不存在 $f(x) \in [0,n)$ 的限制，那么 $f(x)$ 应为 x 按位取反后的值。考虑限制的话，我们贪心保留高位的 1 即可。如果保留某一位的 1 会导致 $f(x) \geq n$，那么改这一位为 0。

  我们用类似数位 DP 的方法从高位到低位考虑，计算前 i 位（最高的 i 位）都和 $n-1$ 的前 i 位相同的所有的 x 对答案的贡献。有三种情况：

  1. $n-1$ 的第 i 位为 0。这时 x 的这一位只能是 0，同时 $f(x)$ 的这一位也只能是 0。
  2. $n-1$ 的第 i 位为 1，且 x 的这一位也为 1。这时 $f(x)$ 的这一位可以取到 0，而且后面的位可以直接取 x 取反后的位而不会超出范围。
  3. $n-1$ 的第 i 位为 1，且 x 的这一位为 0。这时 $f(x)$ 的这一位可以取 1，但是这之后的位还存在限制。

  我们注意到，$n-1$ 的第 i 位为 0 的情况对答案没有影响。而 $n-1$ 的第 i 为为 1 的情况则分成了两部分，x 的这一位为 1 时对答案的贡献可以直接计算。假设后面还有 k 位，那么一个数的贡献就是 2^k-1。而 x 这一位为 0 时，我们相当于将当前计算的数的取值范围缩小了。

  这个算法的复杂度为 $O(\log n)$。