第五届广西省大学生程序设计大赛 题解

The 5th Guangxi Collegiate Programming Contest Tutorial

By Colin & his friends

Hangzhou Dianzi University

2022 年 6 月 26 日

Overview

	Easiest											Hardest
Idea	B	A	G	E	L	H	I	J	D	F	C	K
Coding	B	G	A	H	L	I	E	J	F	D	K	C
Summary	B	A	G	L	H	E	I	J	F	D	C	K
On Site	B	A	L	H	G	I	J	F	E	D	C	K

B. Bus

Description

n 个人坐车，每个车最多坐 m 个人，问最少需要几辆车。

- $1 \leq n, m \leq 4 \times 10^{15}$。

Solution

- 答案即为 $\lceil \frac{n}{m} \rceil = \lfloor \frac{n+m-1}{m} \rfloor$。
- 注意 long long 以及一辆车的时候输出格式不同。

A. Arrangement

Description

n 道题目，编号从 A 开始。

每个题目有两个难度 a_i, b_i，定义总难度为 $a_i \times b_i$。

按照总难度第一关键字、a_i 第二关键字、编号第三关键字排序。

- $1 \leq n \leq 26, 1 \leq a_i, b_i \leq 10^5$。

Solution

- 按要求实现 cmp 排序即可。
- 数据范围小，使用 $\mathcal{O}(n^2)$ 的排序算法也可通过。

G. Gambling

Description

从 n 道题目中等概率抽一道题目提问，Colin 只准备过一个题目。

Colin 准备过的题目可以拿 100 分，否则只能拿 50 分。

允许一次反悔的机会，可以把第一次抽到的题目扔掉，从剩下的 $n-1$ 个题目中等概率的再抽一道题目出来。不过第二次的成绩需要打八折。

问最优策略下期望得分是多少，输出答案乘 n。

- $1 \leq n \leq 2 \times 10^9$。

Solution

第一次抽到了：$P_1 = \frac{1}{n}$

第一次没抽到，第二次抽到了：$P_2 = \frac{n-1}{n} \times \frac{1}{n-1} = \frac{1}{n}$

第一次没抽到，第二次也没抽到：$P_3 = \frac{n-1}{n} \times \frac{n-2}{n-1} = \frac{n-2}{n}$

第一次没抽到不重抽第二次：$P_4 = \frac{n-1}{n}$

重抽的期望 $E_1 = P_1 \times 100 + 0.8 \times (P_2 \times 100 + P_3 \times 50)$

不重抽的期望 $E_2 = P_1 \times 100 + P_4 \times 50$

不难发现分母都为 n，于是我们只需要比较 nE_1 和 nE_2 的大小。

$nE_1 = 100 + 80 + 40(n-2) = 40n + 100$

$nE_2 = 100 + 50(n-1) = 50n + 50$

L. Lowbit

Description

给定一个二进制数 x

每次可以选择令 x 加或减 $\text{lowbit}(x)$

问最少操作多少次使 x 变为 0 。

- $0 \leq x \leq 2^{10^5}$

Solution

从后往前考虑：

- 如果单独的一个 1（前面是 0），使用一次 -lowbit 即可。
- 如果一段连续的 1，使用一次 $+$lowbit 操作变成一个 1。
- 循环此操作即可。

一个经典的错误：认为每一段长度超过 1 的段，都需要两次操作

例如 1110111 只需要三次操作即可（两次 $+$lowbit，一次-lowbit）

H. Homework

Description

有 n 份作业，第 i 份作业需要写 a_i 字，或者选择花费 c_i 的时间先从一份完成的作业 j 拷贝过来，然后补全不够的字数。

求写完所有作业需要的最少时间。

- $1 \leq n, a_i \leq 10^5, c_i < a_i$

Solution

首先，拷贝不会增加字数，所以 $\max a_i$ 的字数肯定要写完。

其次，题目保证了 $c_i < a_i$ ，所以我们肯定优先考虑拷贝作业。

一种错误的思路：先把 $\max a_i$ 的字数的文章写完，剩下的都拷贝。这样不一定是最优的，比如只有 $(10, 9)$ 和 $(20, 1)$。

假设最终的完成序列确定，从第二个开始都是先复制再补全，我们可以发现，需要自己写的字数总是 $\max a_i$ 。

因此将 c_i 最大的放到开头（这份作业不使用复制），最优解为 $\sum_{i=1}^{n} c_i - c_{max} + a_{max}$ 。

时间复杂度 $O(n \log_2 n)$ 或者 $O(n)$ 。

E. Envy-Freeness

Description

m 个物品分给两个人，第 i 人眼中第 j 个物品的价值为 $w_{i,j}$ 。

对于分配，优先级从高到低定义四种状态：

- EF：在每个人眼中，对方的物品价值和都不比自己的高
- EFX：在每个人眼中，对方至多去掉任意一个物品，剩余的物品价值都不比自己的高
- EF1：在每个人眼中，对方物品集中存在一个物品，去掉后剩余的物品价值都不比自己的高
- E：其他情况

每加入一个物品（指定分配给谁），求当前可能的最高优先级。

- $m, w_{i,j} \leq 10^6$

Solution

- 阅读理解，较长的题目不一定就是难题。
- 维护一下每个人眼中对应的 2 个人各自的价值，以及每个人眼中对方的物品里价值最大值、最小值
- EF：每个人眼中自己都比对方高
- EFX：每个人眼中自己的价值比（对方价值-对方最小价值）要高，即符合任意
- EF1：每个人眼中自己的价值比（对方价值-对方最大价值）要高，即符合存在
- E：其他情况

时间复杂度 $\mathcal{O}(n)$。

I. Infinity

Description

给定 a, b ，你可以初始化 $x = 0/1$ 。

可以进行多轮操作，每轮从下列选项中二选一：

- 令 $x = x \times a$
- 令 $x = x + b$

问是否存在一个 N 使得任何一个大于 N 的数字，都可以通过上述流程得到？

- $1 \leq a, b \leq 10^6$

Solution

根据游戏规则，如果 x 可表示，那么 $x+b$ 就可以表示。

由于只要求判断存在 N 使得对于任意 $n > N$ 都可以被表示，所以 $0 \cdots b-1$ 对于 $x \bmod b$ 都存在表示即可。

即 $a^k \bmod b$ 能取遍 $1 \cdots b-1$ 即可（0 可以直接初值取 0）。

这个条件等价于 a 是 b 的原根，由于题目范围只有 10^6，所以直接枚举 $k \in [1, b)$，判断 $a^k \bmod b$ 是否遍历 $1 \cdots b-1$ 即可。

J. Jump game

Description

给出长度为 n 的序列 $\{a_n\}$ ，从 1 开始跳跃。

如果 $i < j$ 并且 $\gcd(a_i, a_j) > 1$ 那么 i 可以跳到 j 。

保证任意一个 a_i **包含不超过 5 个质因子**。

求从 1 跳到 n 的方案数。

- $2 \leq n \leq 10^5, 1 \leq a_i \leq 10^5$

Solution

定义 f_i 表示从 1 跳到 i 的方案数，我们很容易写出 $O(n^2)$ 复杂度的 DP，但是 n 非常大，无法通过此题。

如果 x, y 含有至少一个相同的质因子，那么 $\gcd(x, y) > 1$ 。

根据这个性质，对于 i ，枚举 a_i 的质因子 p ，将所有含有质因子 p 的 a_j 对应的 $f_j (j < i)$ 累加起来。但是很显然，这会造成计数重复，比如 2 和 3 都会统计到 $a_j = 6$ 的那些 f_j 。

Solution - Cont'd

质因子不超过 5 个。不难想到用容斥来避免计数重复。

令 $sum(d) = \sum\{f_j | j < i \wedge a_j \bmod d = 0\}$ 即，是 d 倍数的 a_j 对应的 $f_j(j < i)$ 的和。

假设 a_i 含有 $p_1, p_2, \cdots, p_k$ 共 k 个质因子，S 是 $\{1, 2, \cdots, k\}$ 的非空子集，那么：

$$f_i = \sum_S (-1)^{|S|-1} sum(\prod_{s \in S} p_s)$$

每次维护出 f_i 之后更新所有 $sum(\prod_{s \in S} p_s)$ 。最后输出 f_n 即是答案。

时间复杂度 $\mathcal{O}(2^5 n)$ 。

F. Finding Stars

Description

给出 n 个范围在 $[1, 1000]$ 内的正整数 $\{a_n\}$，有 q 次询问。

每次有三种操作：

1. 每次修改一个位置上的数（每次修改后仍然满足 $1 \leq a_i \leq 1000$）

2. 每次询问一个区间 $[l, r]$ 中 $1, \cdots, 1000$ 出现的次数的奇偶性是否相同

3. 每次询问一个区间 $[l, r]$ 中 $1. \cdots, 1000$ 出现奇数次的数有多少种。

- $1 \leq n, q \leq 10^5, 1 \leq x_i \leq 10^3$。

Solution

用线段树，每个节点上维护一个 bitset<1000>，记录当前这个区间里，每一个权值出现次数的奇偶性，奇数记为 1，偶数记为 0.

发现合并区间的信息只需要将两个 bitset 异或起来即可。

每次查询的时候在线段树上不断合并区间，最后得到一个 bitset<1000> ans

对于 $op_i = 1$ 若 ans.all() || ans.none() 为真则输出 1 否则输出。

对于 $op_i = 2$，输出 ans.count() 即可。

现场部分队伍使用带修莫队也通过了此题。

D. Driving

Description

n 个点 m 条边的有向图，车从 S 到 T ，初始速度 $v = 1$ 。

对于一个长度为 w 的边，走过这条边需要 $\lceil \frac{w}{v} \rceil$ 的时间。

存在一些点上有维修站，可以在该点停留 c_i 的时间使速度翻倍(可以多次)。

存在一些边质量不好，经过之后速度就会回到 $v = 1$ 。

求到达 T 的最短时间。

- $2 \leq n \leq 2 \times 10^4, 1 \leq m \leq 7 \times 10^4, 1 \leq w, c_i \leq 10^6$

Solution

注意到当 $w < 2^k$ 时，$\lceil \frac{w}{2^k} \rceil = 1$，则车辆上累积的加速最多只有 $\log_2 w$ 次。

于是考虑建立一个 $\log_2 w$ 层的分层图，第 k 层的边权是 $\lceil \frac{w}{2^k} \rceil$。

如果一条边的道路状态是 G，就在同层的 u, v 点之间连边；

如果一条边的道路状态是 B，那就在第 k 层的 u 和第 0 层的 v 之间连边。

如果一个在 u 点，耗时为 c 的维修站，就在第 k 层和第 $k+1$ 层的 u 之间连一条长度为 c 的边。

然后在这个图上跑一个最短路即可，复杂度是 $\mathcal{O}(m \log w \log n)$ 。

C. Counting Strings

Description

给出 n 个模式串，q 组询问：

每次给出两个字符串 a 和 b，求每次有多少个模式串满足 a 为其前缀，且 b 为其后缀。

- $1 \leq \sum |s_i|, \sum(|a_i| + |b_i|) \leq 10^6$。

Solution 1

把模式串都插到 trie 树里头，对串正反分别建立一个 trie 树。

在 trie 树上按字典序 dfs ，对每一个节点标号。同时记录对每一个节点 u 记录 $(\text{low}_u, \text{high}_u)$

low_u 表示以 u 为根的子树内，最小的节点标号。

high_u 表示以 u 为根的子树内，最大的节点标号。

我们将每一个模式串结尾在 trie 树上的位置记录下来，对于正反两个 trie 树可以对每一个模式串 i 得到一个二元组 (s_i, t_i)

Solution 1 - Cont'd

每一次查询时，由于前缀和后缀性质类似，这里我们拿前缀举例：

对一个字符串 a ，我们首先在正串的 trie 树上定位这个字符串，假设我们当前定位到节点 p，那么显然的这个节点的子树里模式串都满足 a 是他的前缀。

从另一个视角来看就是满足 $\text{low}_p \leq s_i \leq \text{high}_p$

同样的对于字符串 b ，我们可以在反串的 trie 树上定位到节点 q ，同样的可以得到另一个限制 $\text{low}_q \leq t_i \leq \text{high}_q$

所以我们把问题转化到了数二维平面上矩形区域内有多少个点的问题。用可持久化线段树维护即可。

定义 L_1 为模式串总长，L_2 为查询串总长，时间复杂度为 $\mathcal{O}(L_1 \log(L_1) + q \log(L_1) + L_2)$ ，空间复杂度为 $\mathcal{O}(L_1 \log(L_1))$ 。

Solution 2

第二种做法比较直观。

我们重新拼接模式串，将其表示为 $s_i + @ + s_i$ 即中间用分隔符隔开。

我们将其插入到广义 SAM 中，对于每一个询问我们只需定位子串 $b + @ + a$，并输出其节点所包含的模式串大小即可。

时间复杂度为 $\mathcal{O}(27L_1 + L_2)$，空间复杂度为 $\mathcal{O}(27L_1)$

K. Kirbys Challenge

Description

给出一个 $4 \times n$ 的 01 网格，1 表示格点上有钥匙。只要卡比经过一个格点，该格点上的钥匙就会被收集。

卡比从左上角出发，每秒可以往上下右三个方向前进，也可以站在原地投掷回力镖，利用回力镖收集飞行路径上的钥匙。最终卡比需要达到右下角并收集到所有钥匙。

求达成条件需要的最少时间。

- $n \leq 100$

Solution 1

由于本题可以朝着上下右三个方向移动，因此不是很好考虑常规的网格DP。

观察网格以及回旋镖，我们发现网格行数不多，并且回旋镖往前前进的列数也不多，因此我们可以考虑状压DP。

为了方便，我们以列 j 作为第一维度，行 i 作为第二维度并且编号为 $0, 1, 2, 3$ 。然后将 j 列在内的后 4 列做如下编号：

```
   x 0  1  2  3
y ------------
0 | 0  4  8  12
1 | 1  5  9  13
2 | 2  6  10 14
3 | 3  7  11 15
```

在这个小网格中，(x, y) 对应的编号就是 $4x + y$ 。这样方便后面更新投掷回旋镖的状态。

定义 $f_{j,i,s}$ 表示目前走到了 (j, i) ，j 在内后 4 列的 16 个格点被覆盖的情况为 s（覆盖指的是途径过该格子，或者该格子被回旋镖经过，0 表示没被覆盖，1 表示被覆盖）所需要的最少秒数。

Solution 1 - Cont'd

这个DP细节比较多，因为同一列中，可能会经过某些格子多次，导致 s 没有转移到大于 s 的状态，产生后效性。

我们知道，后效性只会发生在 s 没有变化的情况中，也就是说，只要我们对相同的 s 的那些状态特殊处理，就可以解决这个问题。

根据贪心，我们显然不可能来回多次，即每个格子至多只会被经过两次，因此只要我们对于相同 s 之间的状态，进行正反两次更新，就可以解决后效性的问题。这个技巧是一个常见的解决上下行走导致的后效性的技巧。

```
void Fix(int j,int s){
  for (int i=1;i<=3;i++) // 正向更新
    if ((s>>i-1&1) && (s>>i&1)) // 在不改变s的情况中更新
      f[j][i][s]=min(f[j][i][s],f[j][i-1][s]+1);
  for (int i=2;i>=0;i--) // 反向更新
    if ((s>>i&1) && (s>>i+1&1))
      f[j][i][s]=min(f[j][i][s],f[j][i+1][s]+1);
}
```

Solution 1 - Cont'd

对于当前的 j 和 s，修正完成之后，就可以无后效性地利用 $f_{j,i,s}$ 去更新下一个状态（下一个状态要么 j 变大要么 s 变大，肯定没有后效性），更新分为 6 种：

1. $f_{j,i,s}+1 \to f_{j,i-1,s \vee (1<<i-1)}$：从 (j,i) 移动到 $(j,i-1)$。
2. $f_{j,i,s}+1 \to f_{j,i+1,s \vee (1<<i+1)}$：从 (j,i) 移动到 $(j,i+1)$。
3. $f_{j,i,s}+1 \to f_{j+1,i,s>>4 \vee (1<<i)}$：（只有 j 列钥匙全部捡完了才可以）从 (j,i) 移动到 $(j+1,i)$。
4. $f_{j,i,s}+1 \to f_{j,i,throw(s,i,-1)}$：在 (j,i) 向上投掷回旋镖，$throw(s,i,-1)$ 表示 s 状态在 (j,i) 向上投掷回旋镖之后的新状态。
5. $f_{j,i,s}+1 \to f_{j,i,throw(s,i,1)}$：在 (j,i) 向下投掷回旋镖，$throw(s,i,1)$ 表示 s 状态在 (j,i) 向下投掷回旋镖之后的新状态。
6. $f_{j,i,s}+2 \to f_{j,i,throw(s,i,1) \vee throw(s,i,-1)}$：在 (j,i) 向上向下都投掷回旋镖。

最后满足条件的 s 中 $\min\{f_{n,3,s}\}$ 就是答案。实现细节详见std。

时间复杂度 $O(2^{16} \cdot 4^2 n)$。

Solution 2

本题还有另一种做法，使用最短路来实现 DP 过程，这样就不需要考虑后效性的问题。

时间复杂度 $O(2^{16} \cdot 4^2 n \log_2(2^{16} \cdot 4n))$ 。

由于很多状态是用不到的，因此虽然时间复杂度算起来非常大，跑的还是很快。

Thank you!