Solutions For Claris' Contest # 1

Claris

Hangzhou Dianzi University

2016 年 5 月 1 日

二进制的世界 (binary.c/cpp/pas)

维护一个集合，支持两种操作：

二进制的世界 (binary.c/cpp/pas)

维护一个集合，支持两种操作：

1. 加入一个数。

二进制的世界 (binary.c/cpp/pas)

维护一个集合，支持两种操作：

1. 加入一个数。

2. 查询和给定的数字进行某种位运算能得到的最大值以及达到最大值的方案数。

二进制的世界 (binary.c/cpp/pas)

维护一个集合，支持两种操作：

1. 加入一个数。

2. 查询和给定的数字进行某种位运算能得到的最大值以及达到最大值的方案数。

- 对于 20% 的数据，$n \leq 1000$。

二进制的世界 (binary.c/cpp/pas)

维护一个集合，支持两种操作：

1. 加入一个数。

2. 查询和给定的数字进行某种位运算能得到的最大值以及达到最大值的方案数。

- 对于 20% 的数据，$n \leq 1000$。
- 对于另外 20% 的数据，$n \leq 100000$，运算为 xor 运算。

二进制的世界 (binary.c/cpp/pas)

维护一个集合，支持两种操作：

1. 加入一个数。

2. 查询和给定的数字进行某种位运算能得到的最大值以及达到最大值的方案数。

- 对于 20% 的数据，$n \leq 1000$。
- 对于另外 20% 的数据，$n \leq 100000$，运算为 xor 运算。
- 对于另外 20% 的数据，$n \leq 100000$，不需要求方案数。

二进制的世界 (binary.c/cpp/pas)

维护一个集合，支持两种操作：

1. 加入一个数。

2. 查询和给定的数字进行某种位运算能得到的最大值以及达到最大值的方案数。

- 对于 20% 的数据，$n \leq 1000$。
- 对于另外 20% 的数据，$n \leq 100000$，运算为 xor 运算。
- 对于另外 20% 的数据，$n \leq 100000$，不需要求方案数。
- 对于 100% 的数据，$n \leq 100000$，$0 \leq a_i < 2^{16}$。

20 分做法 1

- 对于 20% 的数据，$n \leq 1000$。

20 分做法 1

- 对于 20% 的数据，$n \leq 1000$。
- 维护每个数字出现的次数，或者保存之前的每一个数字。

20 分做法 1

- 对于 20% 的数据，$n \leq 1000$。
- 维护每个数字出现的次数，或者保存之前的每一个数字。
- 查询时暴力扫描所有存在的数字即可。

20 分做法 1

- 对于 20% 的数据，$n \leq 1000$。
- 维护每个数字出现的次数，或者保存之前的每一个数字。
- 查询时暴力扫描所有存在的数字即可。
- 时间复杂度 $O(n^2)$。

20 分做法 2

- 对于另外 20% 的数据，$n \leq 100000$，运算为 xor 运算。

20 分做法 2

- 对于另外 20% 的数据，$n \leq 100000$，运算为 xor 运算。
- 建立一棵 Trie，同时记录数字个数。

20 分做法 2

- 对于另外 20% 的数据，$n \leq 100000$，运算为 xor 运算。
- 建立一棵 Trie，同时记录数字个数。
- 查询时从高位开始贪心往下走即可。

20 分做法 2

- 对于另外 20% 的数据，$n \leq 100000$，运算为 xor 运算。
- 建立一棵 Trie，同时记录数字个数。
- 查询时从高位开始贪心往下走即可。
- 时间复杂度 $O(n \log m)$。

20 分做法 3

- 对于另外 20% 的数据，$n \leq 100000$，不需要求方案数。

20 分做法 3

- 对于另外 20% 的数据，$n \leq 100000$，不需要求方案数。
- 考虑离线，将所有数字都插入 Trie 中，维护每个数字最早出现的时间。

20 分做法 3

- 对于另外 20% 的数据，$n \leq 100000$，不需要求方案数。
- 考虑离线，将所有数字都插入 Trie 中，维护每个数字最早出现的时间。
- 以 and 为例，先将 n 个数插入到 Trie 中，然后自底向上依次将 1 子树合并到 0 子树上去，这样每个点的左子树就变成了 0+1，右子树还是 1，询问 0 就跑左子树，询问 1 就跑右子树，那么查询的复杂度就是 $O(\log m)$。建树的复杂度为 $O((n+m)\log m)$。

20 分做法 3

- 对于另外 20% 的数据，$n \leq 100000$，不需要求方案数。
- 考虑离线，将所有数字都插入 Trie 中，维护每个数字最早出现的时间。
- 以 and 为例，先将 n 个数插入到 Trie 中，然后自底向上依次将 1 子树合并到 0 子树上去，这样每个点的左子树就变成了 0+1，右子树还是 1，询问 0 就跑左子树，询问 1 就跑右子树，那么查询的复杂度就是 $O(\log m)$。建树的复杂度为 $O((n+m)\log m)$。
- 时间复杂度 $O((n+m)\log m)$。

100 分做法

- 注意到位运算每一位是独立的。

100 分做法

- 注意到位运算每一位是独立的。
- 考虑将 16 位分为两部分：前 8 位和后 8 位。

100 分做法

- 注意到位运算每一位是独立的。
- 考虑将 16 位分为两部分：前 8 位和后 8 位。
- 设 $f[i][j]$ 表示前 8 位为 i 的数，与某个后 8 位是 j 的数进行位运算，后 8 位结果的最大值以及方案数。

100 分做法

- 注意到位运算每一位是独立的。
- 考虑将 16 位分为两部分：前 8 位和后 8 位。
- 设 $f[i][j]$ 表示前 8 位为 i 的数，与某个后 8 位是 j 的数进行位运算，后 8 位结果的最大值以及方案数。
- 那么加入一个数 x 的时候，设它前 8 位为 a，后 8 位为 b，只需要枚举 j，用 $j\ opt\ b$ 更新所有 $f[a][j]$。

100 分做法

- 注意到位运算每一位是独立的。
- 考虑将 16 位分为两部分：前 8 位和后 8 位。
- 设 $f[i][j]$ 表示前 8 位为 i 的数，与某个后 8 位是 j 的数进行位运算，后 8 位结果的最大值以及方案数。
- 那么加入一个数 x 的时候，设它前 8 位为 a，后 8 位为 b，只需要枚举 j，用 $j\ opt\ b$ 更新所有 $f[a][j]$。
- 查询 x 的时候，设它前 8 位为 a，后 8 位为 b，只需要枚举 i，用所有 $(i\ opt\ a) << 8|f[i][b]$ 更新答案。

100 分做法

- 注意到位运算每一位是独立的。
- 考虑将 16 位分为两部分：前 8 位和后 8 位。
- 设 $f[i][j]$ 表示前 8 位为 i 的数，与某个后 8 位是 j 的数进行位运算，后 8 位结果的最大值以及方案数。
- 那么加入一个数 x 的时候，设它前 8 位为 a，后 8 位为 b，只需要枚举 j，用 $j\ opt\ b$ 更新所有 $f[a][j]$。
- 查询 x 的时候，设它前 8 位为 a，后 8 位为 b，只需要枚举 i，用所有 $(i\ opt\ a) << 8|f[i][b]$ 更新答案。
- 时间复杂度 $O(n\sqrt{m})$。

可持久化字符串 (string.c/cpp/pas)

可持久地维护一个字符串 S，支持往末尾添加一个字符 c。

可持久化字符串 (string.c/cpp/pas)

可持久地维护一个字符串 S，支持往末尾添加一个字符 c。

输出每次操作之后的字符串的最短循环节长度。

可持久化字符串 (string.c/cpp/pas)

可持久地维护一个字符串 S，支持往末尾添加一个字符 c。输出每次操作之后的字符串的最短循环节长度。

- 对于 20% 的数据，$n \leq 100$，$m \leq 100$。

可持久化字符串 (string.c/cpp/pas)

可持久地维护一个字符串 S，支持往末尾添加一个字符 c。输出每次操作之后的字符串的最短循环节长度。

- 对于 20% 的数据，$n \leq 100$，$m \leq 100$。
- 对于另外 20% 的数据，$n \leq 300000$，$m \leq 300000$，所有操作形成一条链。

可持久化字符串 (string.c/cpp/pas)

可持久地维护一个字符串 S，支持往末尾添加一个字符 c。输出每次操作之后的字符串的最短循环节长度。

- 对于 20% 的数据，$n \leq 100$，$m \leq 100$。
- 对于另外 20% 的数据，$n \leq 300000$，$m \leq 300000$，所有操作形成一条链。
- 对于另外 30% 的数据，$n \leq 300000$，$m \leq 10$，允许离线。

可持久化字符串 (string.c/cpp/pas)

可持久地维护一个字符串 S，支持往末尾添加一个字符 c。输出每次操作之后的字符串的最短循环节长度。

- 对于 20% 的数据，$n \leq 100$，$m \leq 100$。
- 对于另外 20% 的数据，$n \leq 300000$，$m \leq 300000$，所有操作形成一条链。
- 对于另外 30% 的数据，$n \leq 300000$，$m \leq 10$，允许离线。
- 对于 100% 的数据，$n \leq 300000$，$m \leq 300000$。

20 分做法 1

- 对于 20% 的数据，$n \leq 100$，$m \leq 100$。

20 分做法 1

- 对于 20% 的数据，$n \leq 100$，$m \leq 100$。
- 一个长度为 t 的前缀是循环节当且仅当长度为 $n - t$ 的前后缀相等。

20 分做法 1

- 对于 20% 的数据，$n \leq 100$，$m \leq 100$。
- 一个长度为 t 的前缀是循环节当且仅当长度为 $n - t$ 的前后缀相等。
- 暴力枚举循环节长度然后检验即可。

20 分做法 1

- 对于 20% 的数据，$n \leq 100$，$m \leq 100$。
- 一个长度为 t 的前缀是循环节当且仅当长度为 $n - t$ 的前后缀相等。
- 暴力枚举循环节长度然后检验即可。
- 时间复杂度 $O(n^3)$。

20 分做法 2

- 对于另外 20% 的数据，$n \leq 300000$，$m \leq 300000$，所有操作形成一条链。

20 分做法 2

- 对于另外 20% 的数据，$n \leq 300000$，$m \leq 300000$，所有操作形成一条链。
- 考虑 KMP，那么 $ans = length - next(length)$。

20 分做法 2

- 对于另外 20% 的数据，$n \leq 300000$，$m \leq 300000$，所有操作形成一条链。
- 考虑 KMP，那么 $ans = length - next(length)$。
- 时间复杂度 $O(n)$。

30 分做法

- 对于另外 30% 的数据，$n \leq 300000$，$m \leq 10$，允许离线。

30 分做法

- 对于另外 30% 的数据，$n \leq 300000$，$m \leq 10$，允许离线。
- 将操作建成一棵树，并把相同儿子合并，得到一棵 Trie。

30 分做法

- 对于另外 30% 的数据，$n \leq 300000$，$m \leq 10$，允许离线。
- 将操作建成一棵树，并把相同儿子合并，得到一棵 Trie。
- 将 Trie 建成 AC 自动机，那么串 S 是串 T 的后缀等价于 T 沿着 *fail* 链可以走到 S。

30 分做法

- 对于另外 30% 的数据，$n \leq 300000$，$m \leq 10$，允许离线。
- 将操作建成一棵树，并把相同儿子合并，得到一棵 Trie。
- 将 Trie 建成 AC 自动机，那么串 S 是串 T 的后缀等价于 T 沿着 *fail* 链可以走到 S。
- 建出 *fail* 树，那么每个点的答案等于它深度最大 *fail* 树上的祖先，且这个祖先也是原树中的祖先。

30 分做法

- 对原树求出 DFS 序，然后遍历 *fail* 树，用一棵可持久化线段树维护每个点的答案。

30 分做法

- 对原树求出 DFS 序，然后遍历 *fail* 树，用一棵可持久化线段树维护每个点的答案。
- 每次将父亲的可持久化线段树传给儿子，然后根据原树 DFS 序信息进行区间赋值，单点查询即可。

30 分做法

- 对原树求出 DFS 序，然后遍历 *fail* 树，用一棵可持久化线段树维护每个点的答案。
- 每次将父亲的可持久化线段树传给儿子，然后根据原树 DFS 序信息进行区间赋值，单点查询即可。
- 时间复杂度 $O(n \log n)$。

100 分做法

- KMP 是均摊线性的，一旦加上可持久之后均摊分析就被破坏了，导致复杂度退化。

100 分做法

- KMP 是均摊线性的，一旦加上可持久之后均摊分析就被破坏了，导致复杂度退化。
- 考虑 KMP 慢在哪里，在于每次失配的时候要按着 *next* 数组不断往前跳，直到找到某个位置下一个字符可以匹配为止。

100 分做法

- KMP 是均摊线性的，一旦加上可持久之后均摊分析就被破坏了，导致复杂度退化。
- 考虑 KMP 慢在哪里，在于每次失配的时候要按着 *next* 数组不断往前跳，直到找到某个位置下一个字符可以匹配为止。
- 维护一个转移数组 $trans(x, y)$ 表示如果下一个字符为 y，那么从 x 开始按照 *next* 跳最终会跳到哪个点。

100 分做法

- 考虑加入一个字符 c 带来的影响，那么 $next(now) = trans(x, c)$，$trans(now)$ 与 $trans(next(now))$ 是一样的，$trans(x, c)$ 则修改为了 now。

100 分做法

- 考虑加入一个字符 c 带来的影响，那么 $next(now) = trans(x, c)$，$trans(now)$ 与 $trans(next(now))$ 是一样的，$trans(x, c)$ 则修改为了 now。
- 考虑用可持久化线段树来维护 $trans$，那么复制是 $O(1)$ 的，修改和查询都是 $O(\log m)$ 的。

100 分做法

- 考虑加入一个字符 c 带来的影响，那么 $next(now) = trans(x, c)$，$trans(now)$ 与 $trans(next(now))$ 是一样的，$trans(x, c)$ 则修改为了 now。
- 考虑用可持久化线段树来维护 $trans$，那么复制是 $O(1)$ 的，修改和查询都是 $O(\log m)$ 的。
- 因为整个字符串本身也要可持久化，所以还需要用另外一棵可持久化线段树来可持久地维护每个点 $trans$ 的线段树的树根编号。

100 分做法

- 考虑加入一个字符 c 带来的影响，那么 $next(now) = trans(x, c)$，$trans(now)$ 与 $trans(next(now))$ 是一样的，$trans(x, c)$ 则修改为了 now。
- 考虑用可持久化线段树来维护 $trans$，那么复制是 $O(1)$ 的，修改和查询都是 $O(\log m)$ 的。
- 因为整个字符串本身也要可持久化，所以还需要用另外一棵可持久化线段树来可持久地维护每个点 $trans$ 的线段树的树根编号。
- 时间复杂度 $O(n \log n)$。

反函数 (inverse.c/cpp/pas)

给定一棵有 n 个节点的无根树，每个节点是一个括号。

反函数 (inverse.c/cpp/pas)

给定一棵有 n 个节点的无根树，每个节点是一个括号。

定义 $S(x, y)$ 为从 x 开始沿着最短路走到 y，将沿途经过的点上的字符依次连起来得到的字符串。

反函数 (inverse.c/cpp/pas)

给定一棵有 n 个节点的无根树，每个节点是一个括号。

定义 $S(x, y)$ 为从 x 开始沿着最短路走到 y，将沿途经过的点上的字符依次连起来得到的字符串。

函数 $f(x, y)$ 等于对 $S(x, y)$ 进行划分，使得每一个部分都是平衡的括号串，能得到的最大的段数。

反函数 (inverse.c/cpp/pas)

给定一棵有 n 个节点的无根树，每个节点是一个括号。

定义 $S(x, y)$ 为从 x 开始沿着最短路走到 y，将沿途经过的点上的字符依次连起来得到的字符串。

函数 $f(x, y)$ 等于对 $S(x, y)$ 进行划分，使得每一个部分都是平衡的括号串，能得到的最大的段数。

m 次询问，每次输入一个 k，查询有多少点对的 f 值为 k。

反函数 (inverse.c/cpp/pas)

给定一棵有 n 个节点的无根树，每个节点是一个括号。

定义 $S(x, y)$ 为从 x 开始沿着最短路走到 y，将沿途经过的点上的字符依次连起来得到的字符串。

函数 $f(x, y)$ 等于对 $S(x, y)$ 进行划分，使得每一个部分都是平衡的括号串，能得到的最大的段数。

m 次询问，每次输入一个 k，查询有多少点对的 f 值为 k。

- 对于 10% 的数据，$n \leq 100$，$m \leq 50$。

反函数 (inverse.c/cpp/pas)

给定一棵有 n 个节点的无根树，每个节点是一个括号。

定义 $S(x, y)$ 为从 x 开始沿着最短路走到 y，将沿途经过的点上的字符依次连起来得到的字符串。

函数 $f(x, y)$ 等于对 $S(x, y)$ 进行划分，使得每一个部分都是平衡的括号串，能得到的最大的段数。

m 次询问，每次输入一个 k，查询有多少点对的 f 值为 k。

- 对于 10% 的数据，$n \leq 100$，$m \leq 50$。
- 对于 30% 的数据，$n \leq 5000$，$m \leq 2500$。

反函数 (inverse.c/cpp/pas)

给定一棵有 n 个节点的无根树，每个节点是一个括号。

定义 $S(x, y)$ 为从 x 开始沿着最短路走到 y，将沿途经过的点上的字符依次连起来得到的字符串。

函数 $f(x, y)$ 等于对 $S(x, y)$ 进行划分，使得每一个部分都是平衡的括号串，能得到的最大的段数。

m 次询问，每次输入一个 k，查询有多少点对的 f 值为 k。

- 对于 10% 的数据，$n \leq 100$，$m \leq 50$。
- 对于 30% 的数据，$n \leq 5000$，$m \leq 2500$。
- 对于另外 30% 的数据，$n \leq 50000$，$m \leq 10$。

反函数 (inverse.c/cpp/pas)

给定一棵有 n 个节点的无根树，每个节点是一个括号。

定义 $S(x, y)$ 为从 x 开始沿着最短路走到 y，将沿途经过的点上的字符依次连起来得到的字符串。

函数 $f(x, y)$ 等于对 $S(x, y)$ 进行划分，使得每一个部分都是平衡的括号串，能得到的最大的段数。

m 次询问，每次输入一个 k，查询有多少点对的 f 值为 k。

- 对于 10% 的数据，$n \leq 100$，$m \leq 50$。
- 对于 30% 的数据，$n \leq 5000$，$m \leq 2500$。
- 对于另外 30% 的数据，$n \leq 50000$，$m \leq 10$。
- 对于 100% 的数据，$n \leq 50000$，$m \leq 25000$。

10 分做法

- 对于 10% 的数据，$n \leq 100$，$m \leq 50$。

10 分做法

- 对于 10% 的数据，$n \leq 100$，$m \leq 50$。
- 判断一个串是否是合法括号序列：把 (看成 1，) 看成 -1，若任意时刻前缀和非负且总和为 0 则可行。

10 分做法

- 对于 10% 的数据，$n \leq 100$，$m \leq 50$。
- 判断一个串是否是合法括号序列：把 (看成 1，) 看成 -1，若任意时刻前缀和非负且总和为 0 则可行。
- 最大段数等于前缀和为 0 的个数。

10 分做法

- 对于 10% 的数据，$n \leq 100$，$m \leq 50$。
- 判断一个串是否是合法括号序列：把 (看成 1，) 看成 -1，若任意时刻前缀和非负且总和为 0 则可行。
- 最大段数等于前缀和为 0 的个数。
- 对于每个询问，暴力枚举 x, y，计算 f 之后累加答案即可。

10 分做法

- 对于 10% 的数据，$n \leq 100$，$m \leq 50$。
- 判断一个串是否是合法括号序列：把 (看成 1，) 看成 -1，若任意时刻前缀和非负且总和为 0 则可行。
- 最大段数等于前缀和为 0 的个数。
- 对于每个询问，暴力枚举 x, y，计算 f 之后累加答案即可。
- 时间复杂度 $O(n^3 m)$。

30 分做法

- 对于 30% 的数据，$n \leq 5000$，$m \leq 2500$。

30 分做法

- 对于 30% 的数据，$n \leq 5000$，$m \leq 2500$。
- 枚举起点 x，从这个点开始遍历整棵树，即可算出到每个 y 的答案。

30 分做法

- 对于 30% 的数据，$n \leq 5000$，$m \leq 2500$。
- 枚举起点 x，从这个点开始遍历整棵树，即可算出到每个 y 的答案。
- 预处理出每个询问的答案，每次查询查表即可。

30 分做法

- 对于 30% 的数据，$n \leq 5000$，$m \leq 2500$。
- 枚举起点 x，从这个点开始遍历整棵树，即可算出到每个 y 的答案。
- 预处理出每个询问的答案，每次查询查表即可。
- 时间复杂度 $O(n^2)$。

40 分做法

- 对于另外 30% 的数据，$n \leq 50000$，$m \leq 10$。

40 分做法

- 对于另外 30% 的数据，$n \leq 50000$，$m \leq 10$。
- 考虑如何快速回答单个询问。

40 分做法

- 对于另外 30% 的数据，$n \leq 50000$，$m \leq 10$。
- 考虑如何快速回答单个询问。
- 对这棵树进行树分治，求出重心到每个点的前缀和 s。

40 分做法

- 对于另外 30% 的数据，$n \leq 50000$，$m \leq 10$。
- 考虑如何快速回答单个询问。
- 对这棵树进行树分治，求出重心到每个点的前缀和 s。
- 对于两个点 i, j，假设是从 i 开始走到 j，那么它们的 s 互为相反数，且 i 的 s 是它到重心路径上最大的，j 的 s 则是最小的。

40 分做法

- 同时维护一下最值出现的个数，即可得到将两条链拼起来的链的 f 值。

40 分做法

- 同时维护一下最值出现的个数，即可得到将两条链拼起来的链的 f 值。
- 将所有点按 s 桶排序，对于每种 s 值单独计算，因为 k 固定，所以知道了某一项，另一项也知道，直接计数即可。

40 分做法

- 同时维护一下最值出现的个数，即可得到将两条链拼起来的链的 f 值。
- 将所有点按 s 桶排序，对于每种 s 值单独计算，因为 k 固定，所以知道了某一项，另一项也知道，直接计数即可。
- 需要根据 s 是不是 0 分两种情况讨论，细节比较多。

40 分做法

- 同时维护一下最值出现的个数，即可得到将两条链拼起来的链的 f 值。
- 将所有点按 s 桶排序，对于每种 s 值单独计算，因为 k 固定，所以知道了某一项，另一项也知道，直接计数即可。
- 需要根据 s 是不是 0 分两种情况讨论，细节比较多。
- 时间复杂度 $O(mn \log n)$。

100 分做法

- 考虑用一次树分治直接预处理出每个询问的答案。

100 分做法

- 考虑用一次树分治直接预处理出每个询问的答案。
- 注意到 40 分算法里最后的计数实际上可以转化成两个多项式的卷积，因此用 FFT 优化这个过程即可。

100 分做法

- 考虑用一次树分治直接预处理出每个询问的答案。
- 注意到 40 分算法里最后的计数实际上可以转化成两个多项式的卷积，因此用 FFT 优化这个过程即可。
- 对于每次计算，假设一共有 $size$ 个点，对于某个 s，有 t 个点，那么处理这个 s 的时候，多项式的次数上界不超过 t，因此每层分治的复杂度为 $O(size \log size)$。

100 分做法

- 考虑用一次树分治直接预处理出每个询问的答案。
- 注意到 40 分算法里最后的计数实际上可以转化成两个多项式的卷积，因此用 FFT 优化这个过程即可。
- 对于每次计算，假设一共有 $size$ 个点，对于某个 s，有 t 个点，那么处理这个 s 的时候，多项式的次数上界不超过 t，因此每层分治的复杂度为 $O(size \log size)$。
- 时间复杂度 $O(n \log^2 n)$。

Thank you!