2026 亚洲和太平洋地区信息学奥林匹克中国区

APIO 2026 China

测试

时间：2026 年 5 月 9 日 09:00 ～ 14:00

题目名称	上升	蛋糕	集宝
题目类型	传统型	交互型	传统型
目录	`ascend`	`cake`	`gems`
每个测试点时限	3.5 秒	10.0 秒	2.0 秒
内存限制	1024 MiB	1024 MiB	1024 MiB
测试点数目	20	4	25
测试点是否等分	是	否	是
预测试点数目	20	4	25

提交源程序文件名

对于 C++ 语言	`ascend.cpp`	`cake.cpp`	`gems.cpp`

编译选项

对于 C++ 语言	`-O2 -std=c++14 -static`

注意事项（请仔细阅读）

1. 赛后正式测试时将以选手留在题目目录下的源代码为准。
2. 选手提交的程序源文件名必须使用英文小写。
3. 选手提交的程序源文件大小不得超过 100 KiB。
4. 程序可使用的栈空间内存限制与题目的内存限制一致。
5. 禁止在源代码中改变编译器参数（如使用 `#pragma` 命令），禁止使用系统结构相关指令（如内联汇编）或其他可能造成不公平的方法。
6. 因违反上述规定而出现的问题，申诉时一律不予受理。
7. 选手可使用快捷启动页面中的工具 selfEval 进行自测。选手需将待测程序的源文件置于相应题目目录下。每次自测时可选择全部或部分题目进行自测。注意：自测有次数限制，且自测结果仅供选手测试参考，不作为最终正式成绩。

上升（ascend）

【题目描述】

小 N 是一个喜欢事物呈上升趋势的女孩子，因为这往往意味着好事发生！

正是因为这样的爱好，对于一个 $1 \sim n$ 的排列 $q_1, q_2, \ldots, q_n$，小 N 也喜欢研究其上升的位置。具体地，她定义排列 q 中所有上升的位置构成的集合为 $S(q) = \{1 \le i < n \mid q_i < q_{i+1}\}$。

上升是一件幸运的事情，但具体有多幸运却是难以量化的。于是小 N 决定给排列里的每一个位置赋权，从而去量化幸运值。具体地，她给出了一个非负整数序列 $w_1, w_2, \ldots, w_{n-1}$，并定义排列 q 的**幸运值**为 $f(q) = \prod_{i \in S(q)} w_i$。特别地，若 $S(q) = \varnothing$，则 $f(q) = 1$。

小 C 是小 N 的好朋友，有一天，小 C 送了一个幸运的排列 $p_1, p_2, \ldots, p_n$ 给她。但由于种种意外，排列中有些位置的元素缺失了，这些缺失位置的值变成了 0。

收到礼物后，小 N 并没有为排列的不完整而感到难过，因为她惊奇地发现：排列中**所有未缺失位置上的元素**仍是单调递增的，即它们从左到右构成了一个单调递增的子序列。

小 N 顿时觉得自己是世界上最幸福的女孩子。同时，她也很好奇原来小 C 送的排列到底有多幸运。于是，她想要计算出符合 p 的所有排列 q 的幸运值 $f(q)$ 之和。其中排列 q 符合 p 当且仅当：对于所有 $1 \le i \le n$，均有 $p_i = 0$ 或 $q_i = p_i$。

你的任务是帮助小 N 求出，符合 p 的所有排列 q 的幸运值之和。

【实现细节】

选手不需要，也不应该实现 `main` 函数。

选手需要确保提交的程序包含头文件 `ascend.h`，即在程序开头加入以下代码：

```
#include "ascend.h"
```

选手需要在提交的程序源文件 `ascend.cpp` 中实现以下函数：

```
int ascend(int c, int n, int m, std::vector<int> p,
  std::vector<int> w);
```

- c, n 分别表示测试点编号、排列的长度。$c = 0$ 表示该测试点为样例。
- p 表示缺失了若干个位置后的小 C 的排列。对于 $0 \le i < n$，p_i 表示缺失后的排列的第 $i + 1$ 位的值。
- w 表示小 N 的权值序列。对于 $0 \le i < n - 1$，w_i 表示第 $i + 1$ 个位置的权值。
- 该函数需要返回符合 p 的所有排列 q 的幸运值 $f(q)$ 之和对 m 取模后的结果。
- 对于每个测试点，该函数会被交互库调用恰好 t 次。

【测试程序方式】

选手可以在本题目录下使用如下命令编译得到可执行程序：

```
g++ grader.cpp ascend.cpp -o ascend -O2 -std=c++14 -static
```

对于编译得到的可执行程序：

- 可执行文件将从标准输入读入以下格式的数据：
  - 输入的第一行包含两个非负整数 c, t，分别表示测试点编号和测试数据组数。
  - 接下来依次为每组测试数据，对于每组测试数据：
    - 第一行包含两个正整数 n, m。
    - 第二行包含 n 个非负整数 $p_1, p_2, \ldots, p_n$。
    - 第三行包含 $n-1$ 个非负整数 $w_1, w_2, \ldots, w_{n-1}$。
- 可执行文件将输出以下格式的数据至标准输出：
  - 对于每组测试数据，输出一行一个非负整数，表示 `ascend` 函数的返回值。

【样例 1 输入】

```
0 1
3 6
0 2 0
2 3
```

【样例 1 输出】

```
1
```

【样例 1 解释】

共有以下两个排列 q 符合排列 p：

1. $q = [1, 2, 3]$，$S(q) = \{1, 2\}$，$f(q) = w_1 \times w_2 = 2 \times 3 = 6$；
2. $q = [3, 2, 1]$，$S(q) = \varnothing$，$f(q) = 1$。

因此，符合 p 的所有排列的幸运值之和为 $6 + 1 = 7$，对 $m = 6$ 取模后的结果为 1。

【数据范围】

对于所有测试数据，均有：

- $1 \le t \le 5$；

- $2 \le n \le 500$，$2 \le m \le 10^9$；
- 对于所有 $1 \le i \le n$，均有 $0 \le p_i \le n$；
- 序列 p 中所有不为 0 的元素构成一个单调递增的子序列；
- 对于所有 $1 \le i \le n-1$，均有 $0 \le w_i < m$。

测试点编号	$n \le$	特殊性质
$1, 2$	10	无
$3, 4$	20	
$5 \sim 7$	500	A
$8 \sim 10$	50	B
$11 \sim 13$	150	
$14 \sim 18$	500	
$19, 20$		无

特殊性质 A：对于所有 $1 \le i \le n$，均有 $p_i = 0$。

特殊性质 B：$m \ge 5 \times 10^8$ 且 m 为素数。

蛋糕（cake）

这是一道交互题。

【题目描述】

Lavi 是一位很喜欢吃蛋糕的小星使。她认为每块蛋糕的美味度都可以定义为一个正整数。

有一天，她的好朋友 Sally 买来了一块蛋糕。这块蛋糕的美味度是一个不大于 W 的正整数 d，但 Lavi 此时只知道 W 的值。现在 Lavi 想要求出 d 的值。

运用星使的力量，Lavi 可以制作出至多 N 块美味度分别不超过 $W+200$ 的蛋糕并告知 Sally；然后 Sally 会偷偷地将买来的那块蛋糕混进这些蛋糕中，最后 Sally 会将这些蛋糕**按美味度升序排序**。由于这些不同美味度的蛋糕在外表上没有任何区别，所以 Lavi 暂时不能分辨出哪块蛋糕是 Sally 买来的。不过 Sally 有很强的记忆力，因此她清晰地记得排序后每块蛋糕的美味度。

将蛋糕排序后，Lavi 可以多次向 Sally 进行以下询问：

- Lavi 选出若干块蛋糕，将它们分为不相交的两组，然后 Sally 会告知 Lavi 两组蛋糕美味度之和的大小关系。

  形式化地，假设 Lavi 制作了 m 块蛋糕，记 $a_0, a_1, a_2, \ldots, a_m$ 为将 Sally 买来的蛋糕混入并排序后每块蛋糕的美味度，那么 Lavi 每次需要给出两个非空下标集合 S_1, S_2，满足 $S_1, S_2 \subseteq \{0, 1, 2, \ldots, m\}$ 且 $S_1 \cap S_2 = \varnothing$。Sally 会告知 Lavi $\sum_{i \in S_1} a_i$ 与 $\sum_{i \in S_2} a_i$ 的大小关系。

现在 Lavi 需要你的帮助！但她提醒你，询问次数过多会对 Sally 的记忆力提出很大的考验，因此 Sally 对询问次数做出了一定限制。具体地，Sally 会在 Lavi 制作蛋糕前给出一个正整数 K，如果 Lavi 作出了多于 K 次询问，那么你获得的分数会随询问次数增加而减少。

请协助 Lavi 决定制作蛋糕的策略以及随后的询问策略。

【实现细节】

选手不需要，也不应该实现 `main` 函数。

选手需要确保提交的程序包含头文件 `cake.h`，即在程序开头加入以下代码：

```
#include "cake.h"
```

选手需要在提交的程序源文件 `cake.cpp` 中实现以下两个函数：

```
std::vector<int> bake_cakes(int N, int W, int K);
```

- N 表示 Lavi 最多可制作的蛋糕数。
- W 表示 Sally 买来的蛋糕的美味度上界。
- K 表示询问次数阈值。

- 该函数需要返回一个正整数数组 c，表示 Lavi 制作出每块蛋糕的美味度，其中：
  - c 的长度 m 不能超过 N；
  - 对于所有 $0 \le i \le m-1$，均有 $1 \le c_i \le W + 200$。
- 对于每个测试点，该函数会被交互库调用恰好一次，且在所有 `find_tastiness` 函数调用之前。

```
int find_tastiness(int m, int W, int K);
```

- m 表示 Lavi 制作出的蛋糕数，也就是说，包含 Sally 买来的蛋糕后实际蛋糕数为 $m+1$。
- W 表示 Sally 买来的蛋糕的美味度上界。
- K 表示询问次数阈值。
- 该函数需要返回一个正整数 d，表示 Lavi 求出的 Sally 买来的蛋糕的美味度。
- 对于每个测试点，该函数可能会被交互库调用多次，每次调用之间是独立的。

在该函数中，你可以调用以下函数：

```
int compare_tastiness(std::vector<int> S1, std::vector<int>
  S2);
```

- S_1, S_2 分别为两组蛋糕的下标集合，你需要满足 S_1, S_2 非空且其中的元素互不相同，即 $S_1, S_2 \subseteq \{0, 1, 2, \ldots, m\}$ 且 $S_1 \cap S_2 = \varnothing$。
- 该函数会返回一个整数 $r \in \{-1, 0, 1\}$ 代表 $v_1 = \sum_{i \in S_1} a_i$ 与 $v_2 = \sum_{i \in S_2} a_i$ 的大小关系，其中 $r = -1$ 代表 $v_1 < v_2$，$r = 0$ 代表 $v_1 = v_2$，$r = 1$ 代表 $v_1 > v_2$。
- 在一次 `find_tastiness` 的调用中，你可以调用该函数至多 100 次。

评测程序不是适应性的，Sally 买来的蛋糕的美味度 d 在每次 `find_tastiness` 的调用前就已经确定。

注意：在任何情况下，交互库运行所需时间均不会超过 6 秒。

【测试程序方式】

选手可以在本题目录下使用如下命令编译得到可执行程序：

```
g++ grader.cpp cake.cpp -o cake -O2 -std=c++14 -static
```

对于编译得到的可执行程序：

- 可执行文件将从标准输入读入以下格式的数据：
  - 输入的第一行包含四个正整数 N, W, K, T。
  - 输入的第二行包含 T 个正整数 $d_1, d_2, \ldots, d_T$，分别表示每次调用 `find_tastiness` 时 Sally 买来的蛋糕的美味度。
- 可以通过在运行时启用 `-v` 或 `--verbose` 参数来输出更详细的交互流程。在未启用该参数的情况下，程序会在每次调用完 `find_tastiness` 后输出返回

值的正确性以及 compare_tastiness 的调用次数，并在所有调用完成后输出 compare_tastiness 的最大调用次数。启用 -v 或 --verbose 参数后，程序将会额外输出以下内容：

- 调用 bake_cakes 后函数的返回值。
- 每次 compare_tastiness 被调用时传递的参数，比较信息以及比较结果。

【样例 1 输入】

```
40 20 5 3
19 7 20
```

【样例 1 输出】

```
Correct: found 19 in 4 compares
Correct: found 7 in 5 compares
Correct: found 20 in 5 compares
Correct. Max compare count is 5
```

【样例 1 解释】

交互库将进行以下调用：

```
bake_cakes(40, 20, 5);
```

Lavi 可以制作至多 40 个蛋糕，Sally 买来的蛋糕的美味度上界为 20，询问次数阈值为 5。

一种可能的返回数组为 $[12, 1, 22, 9, 19, 1, 12, 12, 25]$。

接下来交互库将进行三次以下调用：

```
find_tastiness(9, 6, 5);
```

其中每次调用时 Sally 买来的蛋糕的美味度分别为 $19, 7, 20$。

- Sally 买来的蛋糕的美味度为 19 时，所有蛋糕排序后美味度分别为 $[1, 1, 9, 12, 12, 12, 19, 19, 22, 25]$。此时，
  - 若调用 compare_tastiness([0, 2, 4], [1, 3, 5])，则 S_1 中蛋糕美味度之和为 $1 + 9 + 12 = 22$，S_2 中蛋糕美味度之和为 $1 + 12 + 12 = 25$，因此该函数会返回 -1；
  - 若调用 compare_tastiness([8, 2, 6], [5, 0, 9])，则 S_1 中蛋糕美味度之和为 $22 + 9 + 19 = 50$，S_2 中蛋糕美味度之和为 $12 + 1 + 25 = 38$，因此该函数会返回 1；

  - 若调用 `compare_tastiness([0, 4, 7], [1, 3, 6])`，则 S_1 中蛋糕美味度之和为 $1+12+19=32$，S_2 中蛋糕美味度之和为 $1+12+19=32$，因此该函数会返回 0。

- Sally 买来的蛋糕的美味度为 7 时，所有蛋糕排序后美味度分别为 $[1,1,7,9,12,12,12,19,22,25]$。此时，
  - 若调用 `compare_tastiness([0, 1, 3], [6])`，则 S_1 中蛋糕美味度之和为 $1+1+9=11$，S_2 中蛋糕美味度之和为 19，因此该函数会返回 -1。

【数据范围】

对于所有测试数据，均有：

- $1 \le N \le 3 \times 10^3$，$1 \le W \le 10^9$，$1 \le K \le 100$，$1 \le T \le 2 \times 10^3$；
- 对于所有 $1 \le i \le T$，均有 $1 \le d_i \le W$。

测试点编号	分值	$N=$	$W=$	$K=$	$T \le$
1	7	$3,000$	10^2	10^2	10^2
2	8	3	3	1	3
3	30	40	10^9	30	$2,000$
4	55	$3,000$	$2,000$	7	

【评分方式】

在任意测试用例中，如果 `bake_cakes` 的返回值不符合实现细节中的约束条件，或者 `compare_tastiness` 的调用不符合实现细节中的约束条件，或者 `find_tastiness` 的返回值不正确，则该子任务得 0 分。

对于每个测试点，记 Q 为该测试点所有 `find_tastiness` 的调用中，`compare_tastiness` 调用次数的最大值，则程序得到的分数将按照以下方式计算：

- 在测试点 $1,2$ 中，若 $Q \le K$，则得分为该测试点的分值，否则得分为 0；
- 在测试点 3 中，得分为 $\max(30-3\cdot\max(Q-K,0),0)$；
- 在测试点 4 中，得分为 $\max(55-11\cdot\max(Q-K,0),0)$。

集宝（gems）

【题目描述】

第一千届收集宝石大赛开始了！

观众们都厌倦了发生在线性序列上的集宝问题，因此本次大赛相比历届有着明显的创新：参赛选手现在需要在一棵 n 个结点的无根树上收集宝石。

这棵树上一共有 m 颗宝石，其中第 i $(1 \le i \le m)$ 颗宝石拥有相应的二元组参数 (a_i, d_i)，表示当一名选手当前到达结点 a_i 的简单路径所含边数不超过 d_i 时，他就可以选择立刻收集这颗宝石（当然，也可以选择不收集）。

相应地，本次大赛的选手也热情高涨，总共有 q 名选手报名参赛。对于每名选手，他将会被分配一个出发结点 x 与一个收集宝石的区间 $[l, r]$，他需要完成以下任务：从结点 x 出发，不断选择当前结点的一条邻边并移动过去，依次收集第 l 至第 r 颗宝石，即按照 $l, l+1, \ldots, r$ 的顺序收集所有 $r-l+1$ 颗宝石。

因为每名选手在参赛过程中都遇到了或多或少的路线规划难题，为了给出合理的评分，主办方找到了你。请你对每一名选手，计算出该选手完成收集任务所需移动的最小总边数。

【实现细节】

选手不需要，也不应该实现 `main` 函数。

选手需要确保提交的程序包含头文件 `gems.h`，即在程序开头加入以下代码：

```
#include "gems.h"
```

选手需要在提交的程序源文件 `gems.cpp` 中实现以下两个函数：

```
void gems(int c, int n, int m, std::vector<int> u,
  std::vector<int> v, std::vector<int> a, std::vector<int>
  d);
```

- c, n, m 分别表示测试点编号、树的结点数与宝石的数量。$c = 0$ 表示该测试点为样例。
- 对于 $0 \le i < n-1$，u_i, v_i 表示树的一条边。
- 对于 $0 \le i < m$，a_i, d_i 表示第 $i+1$ 颗宝石的两个参数。
- 对于每个测试点，该函数会被交互库调用恰好一次，且在所有 `query` 函数调用之前。

```
long long query(int x, int l, int r);
```

- x, l, r 分别表示一名选手的出发结点与收集宝石的区间。

- 该函数需要返回该选手从结点 x 出发，依次收集第 l 至第 r 颗宝石所需移动的最小总边数。
- 对于每个测试点，该函数会被交互库调用恰好 q 次。

【测试程序方式】

选手可以在本题目录下使用如下命令编译得到可执行程序：

```
g++ grader.cpp gems.cpp -o gems -O2 -std=c++14 -static
```

对于编译得到的可执行程序：

- 可执行文件将从标准输入读入以下格式的数据：
  - 输入的第一行包含三个非负整数 c, n, m。
  - 输入的第 $1+i$ $(1 \le i \le n-1)$ 行包含两个正整数 u_i, v_i。
  - 输入的第 $n+1$ 行包含 m 个正整数 $a_1, a_2, \ldots, a_m$。
  - 输入的第 $n+2$ 行包含 m 个非负整数 $d_1, d_2, \ldots, d_m$。
  - 输入的第 $n+3$ 行包含一个正整数 q。
  - 输入的第 $n+3+i$ $(1 \le i \le q)$ 行包含三个正整数 x, l, r。
- 可执行文件将输出以下格式的数据至标准输出：
  - 输出共 q 行，每行一个非负整数，表示 `query` 函数的返回值。

【样例 1 输入】

```
0 5 4
1 2
1 3
2 4
2 5
4 1 5 3
1 0 1 2
2
2 2 4
4 1 2
```

【样例 1 输出】

```
2
2
```

【样例 1 解释】

对于第一名选手，该选手需要从结点 2 出发，收集第 $2 \sim 4$ 颗宝石。一种可能的收集路径为 $2 \to 1 \to 2 \to 2$，共经过 2 条边。

对于第二名选手，该选手需要从结点 4 出发，收集第 $1 \sim 2$ 颗宝石。一种可能的收集路径为 $4 \to 4 \to 1$，共经过 2 条边。

【数据范围】

对于所有测试数据，均有：

- $2 \le n \le 3 \times 10^5$，$1 \le m \le 3 \times 10^5$；
- 对于所有 $1 \le i \le n-1$，均有 $1 \le u_i, v_i \le n$，且所有的边构成一棵树；
- 对于所有 $1 \le i \le m$，均有 $1 \le a_i \le n$ 且 $0 \le d_i \le n$；
- $1 \le q \le 5 \times 10^5$；
- $1 \le x \le n$，$1 \le l \le r \le m$。

<table>
<tr><th>测试点编号</th><th>$n, m \le$</th><th>$q \le$</th><th>特殊性质</th></tr>
<tr><td>$1 \sim 3$</td><td>10^2</td><td>10^2</td><td rowspan="3">无</td></tr>
<tr><td>$4 \sim 7$</td><td rowspan="2">10^3</td><td>10^3</td></tr>
<tr><td>$8 \sim 10$</td><td>3×10^5</td></tr>
<tr><td>$11, 12$</td><td rowspan="3">3×10^5</td><td rowspan="3">5×10^5</td><td>A</td></tr>
<tr><td>$13, 14$</td><td>B</td></tr>
<tr><td>$15 \sim 17$</td><td>C</td></tr>
<tr><td>$18 \sim 21$</td><td>10^5</td><td>3×10^5</td><td rowspan="2">无</td></tr>
<tr><td>$22 \sim 25$</td><td>3×10^5</td><td>5×10^5</td></tr>
</table>

特殊性质 A：树的形态是一条链，即不存在度数超过 2 的结点。

特殊性质 B：对于所有 $1 \le i \le m$，均有 $d_i \ge n/2$。

特殊性质 C：$l = 1$。