IOI2025 中国国家集训队集中培训

第三试

时间：2024 年 12 月 5 日 08:30 ~ 13:30

题目名称	比赛	绝顶之战	前往何方
题目类型	传统型	传统型	交互型
输入	标准输入	标准输入	标准输入
输出	标准输出	标准输出	标准输出
每个测试点时限	2.0 秒	1.0 秒	3.0 秒
内存限制	512 MiB	512 MiB	512 MiB
子任务数目	6	10	3
测试点是否等分	否	否	否

比赛（match）

【题目描述】

有 n 名选手，编号分别为 $1, 2, \cdots, n$。编号为 i 的选手的实力值为 i。所有选手分为红、蓝两队，其中编号为 i 的选手所在的队伍用字符 s_i 描述。s_i 为 `R` 表示他在红队，s_i 为 `B` 表示他在蓝队。

现在将所有选手排成一个环，每一对相邻且不属于同一队的选手会进行一场比赛，实力值较大的选手获胜，他所在的队伍的得分增加一。

然而蓝队的选手勾结了裁判，如果一场比赛中**红队选手**获胜，且他在蓝队选手的**顺时针方向**，则这场比赛**不计入得分**。

现在你想知道，对于每个 $k = -n, \cdots, -1, 0, 1, \cdots, n$，有多少种将选手排列的方法，使得红队得分恰好比蓝队得分大 k。两种方案不同，当且仅当存在某个选手，使得他顺时针方向的下一个选手在两个方案中不同。

由于答案很大，你只需要输出答案对 $998,244,353$ 取模后的值。

【输入格式】

从标准输入读入数据。

第一行包含一个整数 n，表示选手个数。

第二行包含一个长度为 n 的字符串 $s_1 s_2 \cdots s_n$，表示每个选手所属的队伍。

【输出格式】

输出到标准输出。

输出一行 $2n + 1$ 个整数，分别表示 $k = -n, \cdots, 0, \cdots, n$ 的方案数对 $998,244,353$ 取模后的值。

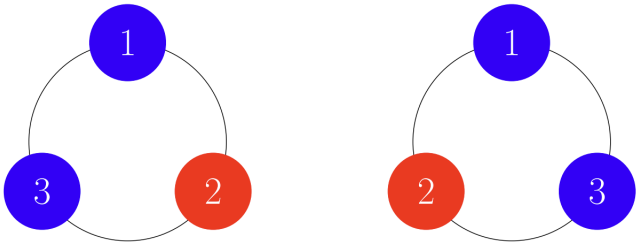
【样例 1 输入】

```
3
BRB
```

【样例 1 输出】

```
0 0 1 1 0 0 0
```

【样例 1 解释】



如图所示，共有两种排列的方法。

第一种排列中，选手 1 与 2 之间的比赛虽然选手 2 获胜，但他属于红队，且在选手 1 顺时针方向，故这场比赛不计入得分。而选手 2 和 3 之间的比赛蓝方获胜。因此红队得分为 0，蓝队得分为 1。

第二种排列中，共进行两场比赛，且均计入得分。红队得分与蓝队得分均为 1。

【样例 2 输入】

```
5
RBBRR
```

【样例 2 输出】

```
0 0 0 0 8 8 8 0 0 0 0
```

【样例 3】

见题目目录下的 ***3.in*** 与 ***3.ans***。

【子任务】

对于所有数据，满足 $3 \leq n \leq 3,000$，s 为由 `B` 和 `R` 构成的字符串。

子任务编号	分值	n
1	10	≤ 17
2		≤ 30
3		≤ 50
4		≤ 200
5	45	≤ 500
6	15	$\leq 3,000$

绝顶之战（struggle）

【题目描述】

有一个长度为 m 的一维空间，还有 n 个物品，第 i 个物品的长度为 a_i。现在按照编号从小到大的顺序依次将物品放入空间中，对于第 i 个物品：

- 如果当前空间中存在一段连续的长度至少为 a_i 的空余，则你**必须**将第 i 个物品放入一段连续的长度为 a_i 的空余空间中。
- 否则，第 i 个物品无法被放入，跳过它。

你需要输出：按照编号从小到大的顺序考虑完所有物品后，所有可能的**空间占用长度**，它的定义是所有被放入空间的物品的长度之和。

【输入格式】

从标准输入读入数据。

输入的第一行两个整数 m, n，分别表示空间长度和物品个数。

第二行 n 个整数 $a_1, \cdots, a_n$，依次表示每个物品的长度。

【输出格式】

输出到标准输出。

输出两行，第一行一个整数 S，表示可能的空间占用长度的数量。

第二行 S 个整数 $b_1, \ldots, b_S$，**从小到大**输出所有可能的空间占用长度。

注意，你需要保证 $b_1, \ldots, b_S$ 不重不漏。

【样例 1 输入】

```
8 4
3 4 1 2
```

【样例 1 输出】

```
4
4 6 7 8
```

【样例 1 解释】

下图分别展示了空间占用长度为 $4, 6, 7, 8$ 的一种可能方式：

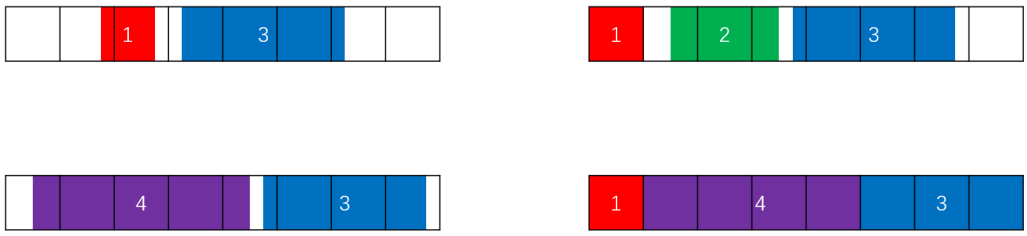


图 1: 样例解释 1

【样例 2】

见题目目录下的 ***2.in*** 与 ***2.ans***。

【子任务】

对于所有测试数据，$1 \le m, a_i \le 10^{16}$， $1 \le n \le 14$。

子任务编号	$n =$	$m, a_i \le$	分数
Subtask 1	5	10	15
Subtask 2	2	10^{16}	5
Subtask 3	3		
Subtask 4	5		
Subtask 5	7		
Subtask 6	9		
Subtask 7	11		10
Subtask 8	12		
Subtask 9	13		
Subtask 10	14		30

前往何方（wheretoreach）

这是一道交互题。

【题目描述】

有一棵 n 个点的树，你不知道其形态。你有一个初始为空的点集 S。你可以进行以下两种操作：

1. 向 S 中加入一个结点。
2. 从 S 中删除一个结点。

每次操作结束后，交互库将会返回树对 S 的导出子图（即仅保留 S 中的点和两端都在 S 中的边得到的图）的最大连通块大小。

你需要使用尽可能少的操作还原树的结构。

【实现细节】

请确保你的程序开头有 `#include "wheretoreach.h"`。

你不需要也不应该实现主函数。你需要实现以下函数：

```
void solve (int n);
```

- 其中 n 表示树的结点个数，结点从 1 至 n 编号。

你可以调用下列由交互库提供的函数：

```
int add (int x);
int remove (int x);
void report (int x,int y);
```

- `add(x)` 用于向 S 中加入一个结点 x，若 x 本身就在 S 中则什么也不做。
  - 你需要保证 $1 \le x \le n$。
  - 该函数返回值为操作后 S 的导出子图最大连通块大小。
- `remove(x)` 用于从 S 中删除一个结点 x，若 x 本身就不在 S 中则什么也不做。
  - 你需要保证 $1 \le x \le n$。
  - 该函数返回值为操作后 S 的导出子图最大连通块大小；
- `report(x,y)` 表示你找到了树的一条边 (x, y)。
  - 你需要恰好调用该函数 $n-1$ 次以提交树的所有边。边的顺序和方向任意。

你调用 `add` 与 `remove` 的总次数不得超过 2×10^6。

保证在满足题目条件和数据范围的情况下，最终测试时交互库的运行时间不会超过 2000 ms，运行空间不会超过 128 MiB。

交互库不是自适应的，即树形态固定，不会随着交互过程改变。

【测试程序方式】

试题目录下的 **implementer.cpp** 是我们提供的交互库参考实现。最终测试的交互库与样例交互库有一定不同，故你的实现不应该依赖样例交互库实现。

你需要在本题目录下使用如下命令编译得到可执行程序：

```
g++ implementer.cpp sample.cpp -o sample -O2 --std=c++14 -lm
```

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：
  - 第一行一个整数 n 表示树的点数。你需要保证 $1 \leq n \leq 10^4$。
  - 接下来 $n-1$ 行，每行两个整数 x, y 描述树上的一条边。边的顺序和方向是任意的。
- 读入完成之后，交互库将调用恰好一次函数 `solve`。
- 当你的实现不符合【实现细节】中的要求时，交互库只会在标准输出流输出一行一个整数 `-1`。具体地，以下情况发生时交互库会输出 `-1`：
  - 对 `add` 和 `remove` 的函数调用中传入了不满足 $1 \leq x \leq n$ 的 x；
  - 调用 `add` 和 `remove` 的次数和超过了 2×10^6；
  - `solve` 函数结束时 `report` 函数的调用次数不为 $n-1$。
- 否则，交互库会在标准输出流按照以下格式输出：
  - 第一行两个整数 Q, c，Q 表示你调用 `add` 和 `remove` 的总次数，$c = 1$ 表示你的答案正确，$c = 0$ 表示不正确。
  - 接下来若干行，为你调用 `add`，`remove` 和 `report` 的记录，其格式可以参考样例 1。

【样例 1 输入】

```
3
1 2
2 3
```

【样例 1 输出】

```
4 1
add(2);
add(3);
add(1);
remove(2);
```

```
report(3,2);
report(1,2);
```

【样例 1 解释】

该样例输出为下发的样例程序在该组样例下的输出。

【子任务】

对于所有测试数据，$1 \le n \le 10^4$。

Subtask 1 (10%)：$n = 500$。

Subtask 2 (20%)：$n = 2500$。

Subtask 3 (70%)：$n = 10^4$。

【评分方式】

本题首先会受到和传统题相同的限制，例如编译错误会导致整道题目得 0 分，运行时错误、超过时间限制、超过空间限制都会导致相应测试点得 0 分。选手只能在程序中访问自己定义的和交互库给出的变量或数据，及其相应的内存空间。尝试访问其他位置空间将可能导致编译错误或运行错误。

对于 Subtask 1,2 中的测试点，只要你的交互符合【实现细节】中的规则，且给出了正确答案，则可以获得满分。

对于 Subtask 3 中的测试点，在你的交互符合【实现细节】中的规则，且给出了正确答案的基础上，设你调用 `add` 和 `remove` 的总次数为 Q，你的得分为 $f(Q)$，其中

$$f(Q) = \begin{cases} 70 & (0.0 \times 10^6 \le Q \le 0.6 \times 10^6) \\ 70 - \frac{Q - 0.6 \times 10^6}{5000} & (0.6 \times 10^6 < Q \le 0.7 \times 10^6) \\ 50 - \frac{Q - 0.7 \times 10^6}{15000} & (0.7 \times 10^6 < Q \le 1.0 \times 10^6) \\ 30 - \frac{Q - 1.0 \times 10^6}{25000} & (1.0 \times 10^6 < Q \le 1.5 \times 10^6) \\ 10 - \frac{Q - 1.5 \times 10^6}{50000} & (1.5 \times 10^6 < Q \le 2.0 \times 10^6) \end{cases}$$

是一个连续的分段线性函数。你可以理解为：你的前 0.6×10^6 次调用是免费的；此后的 0.1×10^6 次调用中每 5000 次扣 1 分；随后的 0.3×10^6 次调用中每 15000 次扣 1 分；随后的 0.5×10^6 次调用中每 25000 次扣 1 分；随后的 0.5×10^6 次调用中每 50000 次扣 1 分。

你在每个子任务上获得的分数为在其中所有测试点上分数的最小值。

选手不应通过非法方式获取交互库的内部信息，如试图与标准输入、输出流进行交互。此类行为将被视为作弊。