야유회 (workshop)

N명의 마법사들이 모여 친목을 도모하고 서로의 마법을 공유하는 야유회를 가지려고 한다. 이 야유회에서는 하루 동안 재미있는 게임을 한다.

게임은 거대한 회의실에서 진행된다. 이 회의실에는 N개의 자리가 있는 원탁이 있다. 원탁의 각 자리는 모두 동일하게 생겼고, 마법사들 역시 모두 동일하게 생겨서 서로 구분할 수 없다. 마법사들은 게임이 진행되는 중 **아무 말도 할 수 없다.**

게임은 세 단계로 진행되는데, 각 단계는 다음과 같다.

- **아침**에 마법사들은 원탁에 둘러앉는다. 각 자리에는 빈 종이 한 장과, 0 이상 $N-1$ 이하의 정수가 적힌 종이 한 장이 놓여 있다. 각 종이에 적힌 정수는 **서로 다르다.** 마법사들은, 원탁에서 자신과 자신의 **오른쪽**에 앉은 사람의 종이에 적혀 있는 정수를 볼 수 있다. 이를 토대로, 마법사들은 빈 종이에 0 이상 10^9 미만의 정수를 쓴다. 이 정수는 자신만이 볼 수 있게 조심스럽게 적는다. 이후 방금 정수를 쓴 종이를 접어 자리 위에 놓고, 처음 자리에 놓여 있던 정수가 적힌 종이를 들고 회의실을 떠난다.

- **점심**에 마법사들은 원탁에 둘러앉는다. 각 자리에는 빈 종이 한 장과, 아침에 자신이 수를 적은 종이 한 장이 놓여 있다. 마법사들은, 원탁에서 자신과 자신의 **왼쪽과 오른쪽**에 앉은 사람의 종이에 적혀 있는 정수를 볼 수 있다. 이를 토대로, 마법사들은 빈 종이에 0 이상 10^9 미만의 정수를 쓴다. 이 정수는 자신만이 볼 수 있게 조심스럽게 적는다. 이후 방금 정수를 쓴 종이를 접어 자리 위에 놓고, 아침에 수를 적은 종이를 들고 회의실을 떠난다.

- **저녁**에 마법사들은 원탁에 둘러앉는다. 각 자리에는 빈 종이 한 장과, 점심에 자신이 수를 적은 종이 한 장이 놓여 있다. 마법사들은, 원탁에서 자신과 자신의 **왼쪽과 오른쪽**에 앉은 사람의 종이에 적혀 있는 정수를 볼 수 있다. 이를 토대로, 마법사들은 빈 종이에 0 이상 40 미만의 정수를 쓴다. 이 정수는 자신만이 볼 수 있게 조심스럽게 적는다. 이후 방금 정수를 쓴 종이를 접어 자리 위에 놓고, 점심에 수를 적은 종이를 들고 회의실을 떠난다.

회의실을 떠나면, 손에 들고 있는 종이가 마법의 힘으로 사라지며, 회의실에서 했던 모든 행동을 잊게 된다. 아침, 점심, 저녁에 각 마법사가 앉는 자리의 위치는 동일하다.

마법사들은 N을 모르지만, $10 \leq N \leq 100\,000$ 이라는 사실은 알고 있다.

게임이 끝난 후, 야유회 운영진들은 자리 위에 올려둔 종이들을 모두 편다. 마법사들이 게임을 승리하기 위해서는, 원탁 위의 인접한 자리에 있는 종이에 적힌 수들이 서로 달라야 한다. 또한, 게임이 끝났을 때 (즉, 저녁이 끝나고) 가능한 작은 수만이 종이에 적혀 있으면 좋다.

마법사들은 게임이 진행되는 중에는 아무 말도 할 수 없으나, 게임이 진행되기 전에 공통된 전략을 사용할 것을 합의할 수 있다. 당신은 마법사들을 대신하여 최종적으로 가능한 작은 수를 종이에 적는 전략을 고안해야 한다.

함수 목록 및 정의

여러분은 아래 함수들을 구현해야 한다.

```
void init()
```

- 이 함수는 단 한 번만 호출되며, 다른 모든 함수가 호출되기 전에 호출된다.
- 이후 함수 호출에 필요한 전처리나 전역 변수 설정이 있다면, 이 함수에 구현하면 된다.

```
int morning(int my_num, int right_num)
```

- `my_num`: 아침에 마법사의 자리에 있는 종이에 적힌 정수
- `right_num`: 아침에 마법사의 오른쪽 자리에 있는 종이에 적힌 정수
- 이 함수는 마법사가 아침에 빈 종이에 적을 정수를 반환해야 한다. 이 함수의 반환값은 0 이상 10^9 미만이어야 한다.
- 이 함수의 반환값은 `my_num`와 `right_num`의 값에만 의존해야 한다.
- 이 함수는 최대 $2\,000\,000$번 호출된다.

```
int afternoon(int left_num, int my_num, int right_num)
```

- `left_num`: 점심에 마법사의 왼쪽 자리에 있는 종이에 적힌 정수
- `my_num`: 점심에 마법사의 자리에 있는 종이에 적힌 정수
- `right_num`: 점심에 마법사의 오른쪽 자리에 있는 종이에 적힌 정수
- 이 함수는 마법사가 점심에 빈 종이에 적을 정수를 반환해야 한다. 이 함수의 반환값은 0 이상 10^9 미만이어야 한다.
- 이 함수의 반환값은 `left_num`, `my_num`, `right_num`의 값에만 의존해야 한다.
- 이 함수는 최대 $2\,000\,000$번 호출된다.

```
int evening(int left_num, int my_num, int right_num)
```

- `left_num`: 저녁에 마법사의 왼쪽 자리에 있는 종이에 적힌 정수
- `my_num`: 저녁에 마법사의 자리에 있는 종이에 적힌 정수
- `right_num`: 저녁에 마법사의 오른쪽 자리에 있는 종이에 적힌 정수
- 이 함수는 마법사가 저녁에 빈 종이에 적을 정수를 반환해야 한다. 이 함수의 반환값은 0 이상 40 미만이어야 한다.
- 이 함수의 반환값은 `left_num`, `my_num`, `right_num`의 값에만 의존해야 한다.
- 이 함수는 최대 $2\,000\,000$번 호출된다.

제출하는 소스 코드의 어느 부분에서도 입출력 함수를 실행해서는 안 된다.

`morning`, `afternoon`, `evening` 함수의 반환값은 **주어진 파라미터의 값에만 의존해야** 한다. 함수를 같은 파라미터 값으로 여러 번 호출했을 때 다른 값을 반환하면, 게임의 승패 여부와 무관하게 오답으로 처리된다.

각 테스트 케이스는 여러 개(1개 이상)의 독립적인 게임으로 이루어진다. `morning`, `afternoon`, `evening` 함수가 순서대로 호출된다는 보장은 하지 않지만, 함수의 반환값을 바탕으로 지문에서 제시된 방식대로 게임이 진행된다는 것은 보장된다.

각 테스트 케이스에서 여러분의 프로그램은 두 번 실행된다. 대회 시스템 상에서 수행 시간은 두 번 실행한 프로그램의 수행 시간의 합으로 측정되며, 메모리 사용량 또한 두 번 실행한 프로그램의 메모리 사용량의 합으로 측정된다. 시간 제한과 메모리 제한은 두 번의 실행 결과를 합친 것을 기준으로 한다. `morning`, `afternoon`, `evening` 함수의 호출 횟수 제약 조건 역시 두 번의 실행에서의 호출 횟수의 합을 기준으로 표시되어 있다.

실제 제출할 시, 최악의 경우 채점 프로그램이 기본적으로 소모하는 시간은 2초 이내라고 가정해도 좋다.

제약 조건

- $10 \leq N \leq 100\,000$
- 각 테스트 케이스에서 `morning` 함수는 최대 $2\,000\,000$번 호출된다.
- 각 테스트 케이스에서 `afternoon` 함수는 최대 $2\,000\,000$번 호출된다.
- 각 테스트 케이스에서 `evening` 함수는 최대 $2\,000\,000$번 호출된다.

부분문제

1. (5점) $N \leq 40$
2. (95점) 추가 제약 조건 없음

만약 테스트 케이스 중 어떤 경우에서든, `morning`, `afternoon`, `evening` 함수의 반환값을 바탕으로 게임을 진행했을 때 마법사들이 승리하지 못한다면, 이 부분문제에 대해서 당신은 0점을 받는다.

부분문제 2는 부분 점수가 있다. 이 서브태스크에 대해서 `evening` 함수의 반환값 중 **최댓값에 1을 더한 것**을 m이라 하자. 이 부분문제에서 당신의 점수는 다음 표와 같다.

조건	점수
$10 \leq m \leq 40$	$46 - m$
$m = 9$	38
$m = 8$	41
$m = 7$	46
$m = 6$	52
$m = 5$	64
$m = 4$	76
$m \leq 3$	95

예제

$N = 10$이고, 아침에 각 자리의 종이에 적힌 수가 반시계 방향으로 [3, 8, 7, 0, 2, 4, 5, 9, 6, 1]인 경우를 생각해 보자.

그레이더는 다음 함수를 호출한다.

```
morning(8, 7) = 1
morning(6, 1) = 4
morning(1, 3) = 6
morning(3, 8) = 0
morning(7, 0) = 2
morning(0, 2) = 3
morning(2, 4) = 2
morning(4, 5) = 3
morning(5, 9) = 1
morning(9, 6) = 2
```

이 경우 아침 이후 책상에 적혀 있는 수들은 [0, 1, 2, 3, 2, 3, 1, 2, 4, 6]이 된다. 이 값을 토대로 그레이더는 다음 함수를 호출한다.

```
afternoon(2, 3, 2) = 3
afternoon(1, 2, 3) = 2
afternoon(2, 4, 6) = 2
afternoon(3, 2, 3) = 1
afternoon(2, 3, 1) = 5
afternoon(0, 1, 2) = 4
afternoon(6, 0, 1) = 3
afternoon(3, 1, 2) = 3
afternoon(4, 6, 0) = 1
afternoon(1, 2, 4) = 0
```

이 경우 점심 이후 책상에 적혀 있는 수들은 [3, 4, 2, 3, 1, 5, 3, 0, 2, 1]이 된다. 이 값을 토대로 그레이더는 다음 함수를 호출한다.

```
evening(5, 3, 0) = 0
evening(4, 2, 3) = 0
evening(1, 3, 4) = 0
evening(3, 1, 5) = 0
evening(0, 2, 1) = 0
evening(2, 3, 1) = 1
evening(1, 5, 3) = 1
evening(3, 0, 2) = 1
evening(2, 1, 3) = 1
evening(3, 4, 2) = 1
```

이 경우 저녁 이후 책상에 적혀 있는 수들은 `[0, 1, 0, 1, 0, 1, 0, 1, 0, 1]`이다. 인접한 위치에 적힌 두 수가 항상 서로 다르기 때문에, 이 경우 마법사들이 게임을 이겼다. $m = 2$이다.

각 함수의 호출 순서가 실제 원탁의 배열 순서와 무관함을 관찰하면 좋다. 그레이더는 `morning` 함수의 파라미터가 고정된 순열이고, `afternoon`, `evening` 함수의 파라미터가 `morning`, `afternoon` 함수의 반환값이라는 것을 만족하는 하에 함수 호출 순서들을 임의로 섞을 수 있다. 이러한 방식 외에도 `morning`, `afternoon`, `evening` 함수들이 서로 통신하지 못하도록 다양한 장치들이 구현되어 있다.

Sample grader

Sample grader는 첫 줄에 테스트 케이스의 수 T를 입력받는다. 그 후 T회에 걸쳐 다음과 같은 정보를 입력받는다:

- Line 1: N
- Line 2: $A[0]$ $A[1]$ $\cdots$ $A[N-1]$

$A[i]$는 게임이 시작되기 전 원탁에 올려진 종이에 적힌 정수들을 임의의 위치에서부터 반시계 방향 순서대로 나열한 순열이다. $0 \leq A[i] \leq N-1$이고 모든 $A[i]$는 서로 다름을 가정한다. Sample grader는 입력의 정당성을 확인하지는 않는다.

Sample grader는 각 테스트 케이스에 대해서 다음을 출력한다.

- 만약 `morning` 함수의 반환값이 0 이상 10^9 미만이 아닌 경우 한 줄에 `Wrong Answer [1]`를 출력한다.
- 만약 `afternoon` 함수의 반환값이 0 이상 10^9 미만이 아닌 경우 한 줄에 `Wrong Answer [2]`를 출력한다.
- 만약 `evening` 함수의 반환값이 0 이상 40 미만이 아닌 경우 한 줄에 `Wrong Answer [3]`를 출력한다.
- 만약 `evening` 함수의 반환값이 원탁 위의 인접한 두 자리에 대해 같은 값을 반환한다면 한 줄에 `Wrong Answer [4]`를 출력한다.
- 그 외 경우 두 줄을 출력한다. 첫 번째 줄에는 `Correct!` 가 적혀 있다. 두 번째 줄에는 `m = 10` 와 같이 모든 테스트 케이스에 대한 m의 값이 적혀 있다.

`Wrong Answer`를 출력할 시, Sample grader는 즉시 종료된다.

Sample grader는 실제 채점에서 사용하는 그레이더와 다를 수 있음에 유의하라.