과일 게임 (fruitgame)

과일 게임은 여러 가지 종류의 과일들을 합쳐 크기가 큰 종류의 과일을 만드는 게임이다. 과일 게임의 게임판은 수열 $X[0], X[1], \cdots, X[K-1]$로 표현할 수 있다. 이 때 각 수는 과일의 종류에 따른 번호를 나타내며, 번호가 클수록 과일의 크기가 크다는 것을 의미한다.

이 때 플레이어는 종류가 같으며 인접한 두 과일을 합쳐서 크기가 큰 과일을 만들 수 있는 ***합치기*** 연산을 수행할 수 있다. 이 연산은 다음과 같이 정의된다.

합치기: $X[0], X[1], \cdots, X[K-1]$로 표현되는 게임판에서 정수 $0 \le i \le K-2$를 골라, $X[i] = X[i+1]$을 만족한다면 게임판을 $X[0], \cdots, X[i-1], X[i]+1, X[i+2], \cdots, X[K-1]$로 바꾼다.

플레이어의 목표는 초기 게임판이 주어지면 ***합치기*** 연산을 0회 이상 사용하여 크기가 큰 과일을 만드는 것이다.

예를 들어서 게임판이 $X = [2, 1, 1, 3, 2]$인 경우, $X[1] = X[2]$이기 때문에 $i = 1$을 선택하여 ***합치기*** 연산을 수행하면 게임판이 $X = [2, 2, 3, 2]$로 바뀌게 된다. 또 $X[0] = X[1]$이기 때문에 $i = 0$을 선택하여 ***합치기*** 연산을 수행하면 게임판이 $X = [3, 3, 2]$로 바뀌게 된다. 마지막으로 $X[0] = X[1]$이기 때문에 $i = 0$을 선택하여 ***합치기*** 연산을 수행하면 게임판이 $X = [4, 2]$로 바뀌게 된다. 이렇게 하면 번호가 4인 과일을 만들 수 있고, 이것이 얻을 수 있는 가장 큰 과일의 번호이다.

여러분에게 길이 N의 수열 A가 주어진다. 이 때 A의 원소는 중간에 변경될 수 있으며 이 변화는 누적된다. 여러분은 $0 \le l \le r \le N-1$을 만족하는 정수 순서쌍 (l, r)이 주어질 때마다 $A[l], \cdots, A[r]$로 표현되는 게임판에서 얻을 수 있는 가장 큰 과일의 번호를 구하는 프로그램을 작성해야 한다. 수열의 원소가 변하거나 순서쌍이 주어지는 횟수는 총 Q번이다.

함수 목록 및 정의

여러분은 아래 함수들을 구현해야 한다.

```
void prepare_game(std::vector<int> A)
```

- A: 크기가 N인 정수 배열.
- 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 play_game(int l, int r)
```

- 이 함수는 $A[l], \cdots, A[r]$로 이루어진 게임판에서 얻을 수 있는 가장 큰 과일의 번호를 반환해야 한다.
- 이 함수는 한 번 이상 호출된다.

```
void update_game(int p, int v)
```

- $A[p]$의 값을 v로 변경해야 한다.

`play_game` 함수가 호출되거나 `update_game` 함수가 호출되는 횟수는 총 Q번이다.

제출하는 소스 코드의 어느 부분에서도 입출력 함수를 실행해서는 안 된다.

참고

$A[i], \cdots, A[j]$는 수열 A의 i번 원소부터 j번 원소까지로 구성된 길이가 $j - i + 1$인 수열이다. 예를 들어 $A = [3, 5, 7, 2, 9]$일 때, $A[1], \cdots, A[3]$은 $[5, 7, 2]$이고, $A[4], \cdots, A[4]$는 $[9]$이다.

제약 조건

- $1 \leq N, Q \leq 100\,000$
- 모든 i에 대해 $1 \leq A[i] \leq 10$ $(0 \leq i \leq N - 1)$
- 모든 `play_game` 호출에 대해 $0 \leq l \leq r \leq N - 1$
- 모든 `update_game` 호출에 대해 $0 \leq p \leq N - 1, 1 \leq v \leq 10$

부분문제

1. (5점)
    - $N \leq 10$
    - $Q \leq 10$
2. (6점)
    - $N \leq 600$
    - $Q \leq 600$
3. (8점)
    - $N \leq 4\,000$
    - $Q \leq 4\,000$
    - 모든 i에 대해 $A[i] \leq 2$ $(0 \leq i \leq N - 1)$
    - 모든 `update_game` 호출에 대해 $v \leq 2$
4. (15점)
    - $N \leq 4\,000$
    - $Q \leq 4\,000$
5. (12점)
    - 모든 i에 대해 $A[i] \leq 2$ $(0 \leq i \leq N - 1)$
    - 모든 `update_game` 호출에 대해 $v \leq 2$
6. (14점)
    - `update_game`이 호출되지 않는다.
7. (40점)
    - 추가적인 제약 조건이 없다.

예제 1

$N = 5$, $A = [2, 1, 1, 3, 4]$인 경우를 생각해 보자.

그레이더는 다음 함수들을 순서대로 호출한다.

```
prepare_game([2, 1, 1, 3, 4])
play_game(0, 4) = 5
update_game(2, 3)
play_game(2, 4) = 5
update_game(1, 2)
play_game(0, 2) = 4
```

예제 2

$N = 7$, $A = [1, 1, 1, 1, 2, 2, 2]$인 경우를 생각해 보자.

그레이더는 다음 함수들을 순서대로 호출한다.

```
prepare_game([1, 1, 1, 1, 2, 2, 2])
play_game(0, 6) = 4
play_game(2, 4) = 3
update_game(6, 4)
play_game(4, 6) = 4
play_game(0, 6) = 5
```

Sample grader

Sample grader는 아래와 같은 형식으로 입력을 받는다.

- Line 1: N
- Line 2: $A[0]$ $A[1]$ $\cdots$ $A[N-1]$
- Line 3: Q
- Line $3 + i (1 \le i \le Q)$: `play_game` 함수가 호출될 경우 1 l r, `update_game` 함수가 호출될 경우 2 p v

Sample grader는 다음을 출력한다.

- Line i: i번째로 호출한 `play_game` 함수가 반환한 값

Sample grader는 실제 채점에서 사용하는 그레이더와 다를 수 있음에 유의하라.