최후의 만찬

레오나르도는 가장 유명한 벽화 '최후의 만찬'을 그릴때 매우 적극적이었다: 매일 작업을 시작할 때 하는 일 중 하나는 그 날에 사용할 벽화(템페라 그림)의 물감들을 결정하는 것이었다. 많은 물감이 필요했지만, 공사현장의 비계 (벽화를 그리기 위하여 사다리 위에 놓인 작업대) 위에는 제한된 수의 물감들만 놓을 수 있었다. 레오나르도의 조수는 바닥에 있는 선반에서 물감을 들고 비계로 올라가서 전달하고, 비계에 있는 물감을 들고 내려와서 바닥에 있는 선반에 놓아두는 책임을 맡았다.

이 문제에서는, 조수를 돕기위한 두 개의 별도의 프로그램을 작성하여야 한다. 첫번째 프로그램은, 레오나르도의 지시 (레오나르도가 그날에 필요한 물감의 순서)를 읽고, 비트 스트링으로 작성된*advice*라고 불리는 조수를 위한 *짧은* 힌트를 만든다. 두번째 프로그램에서는 레오나르도의 지시를 처리하는 과정에서, 조수는 레오나르도의 앞으로의 지시들을 알 수 없고, 단지 현재까지의 지시들과 첫번째 프로그램에서 생성된 advice만 참조할 수 있다. 즉, 두번째 프로그램은 advice를 읽고, 그런 다음 온라인 방식으로 (즉, 한번에 하나씩) 레오나르도의 지시를 읽어서 처리한다. 이 프로그램은 advice의 의미가 무엇인지를 이해할 수 있어야 하며, 이것을 사용하여 최적의 선택을 해야한다. 더 자세한 것은 아래에 설명이 되어 있다.

선반과 비계 사이의 물감 이동

간단한 예를 생각해보자. N가지 물감들이 0부터 N-1까지 번호가 매겨져 있고, 매일 레오나르도는 조수에게 새로운 물감을 정확히 N번 지시한다고 가정하자. C를 레오나르도가 지시하는 N개의 물감의 순서라고 하자. 따라서, C는 N개의 수의 순서라고 생각할 수 있고, 각각의 수는 0부터 N-1까지이다. 어떤 물감은 C에 전혀나타나지 않을수도 있고, 어떤것들은 여러번 나타날 수 있음에 유의하라.

비계는 항상 N개의 물감 중 K개의 물감으로 가득차있다. 단, $K < N$. 처음에 비계에는 0부터 K-1까지의 물감이 놓여져있다.

조수는 레오나르도의 지시를 한번에 하나씩 처리한다. 지시하는 물감이 *벽화용 비계에 이미* 존재할 경우, 조수는 쉴 수 있다. 그렇지 않으면, 선반에서 필요한 물감을 고른 후, 비계로 그것을 옮겨야한다. 물론, 비계에는 놓을 곳이 없을 것이므로, 조수는 비계에서 하나의 물감을 선택하여 그것을 다시 아래의 선반으로 가지고 내려와야 한다.

레오나르도의 최적 전략

조수는 가능한 많은 횟수를 쉬기 원한다. 레오나르도의 지시 횟수에 대해 그가 쉴 수 있는 횟수는 그의 선택에 달려있다. 좀더 자세히 말하면, 매번 조수가 어떤 물감을 가지고 내려오는가에 따라서 미래에 다른 결과를 만들어 낼 수도 있다. 레오나르도는 C를 알면 어떻게 목표를 성취할 수 있는지를 조수에게 설명했다. 비계에서 가지고 내려오는 물감의 최선의

선택은 현재 비계에 있는 물감과, C에서 지시하는 물감 중 비계에 남아있는것을 조사하는 것으로 얻어진다. 다음의 규칙에 따라 비계에 남아있는 물감 중 하나를 선택하여야 한다.

- 미래에 사용하지 않는 물감이 존재할 경우, 조수는 비계에서 그러한 물감을 가지고 내려와야 한다.
- 그렇지 않으면, *미래에 가장 나중에 사용될 물감*을 비계에서 가지고 내려와야 한다.(즉, 비계에 있는 각 물감에 대해, 미래에 처음으로 사용되는 시점을 찾는다. 가지고 내려와야 하는 물감은 가장 나중에 필요한 물감이다.)

레오나르도의 전략을 사용하는 경우에는 조수가 가능한 가장 많이 쉴 수 있음을 증명할 수 있다.

예제 1

N = 4 즉, 4가지 물감(0부터 3까지의 번호)과 4가지의 지시인 경우를 보자. 레오나르도의 지시는 C = (2, 0, 3, 0) 이다. 또한, K = 2 라 하자. 즉, 레오나르도는 비계에 최대 2개의 물감을 둘 수 있다. 앞에서 기술하였듯이, 비계 위에는 초기에 0과 1의 물감이 있다. 비계 위에 있는 물감은 [0, 1]로 표현한다. 레오나르도의 지시들에 대해 조수가 할 수 있는 한가지 가능한 처리 방법은 다음과 같다.

- 첫 번째 지시하는 물감(2번)은 비계 위에 없다. 조수는 2번 물감을 비계 위에 두고, 1번 물감을 비계 위에서 가지고 내려온다. 현재의 비계는 [0, 2]이다.
- 그 다음 지시하는 물감(0번)은 비계 위에 있으므로, 조수는 쉴 수 있다.
- 세 번째 지시하는 물감(3번)에 대해서 0번 물감을 비계 위에서 가지고 내려오면, 비계는 [3, 2]가 된다.
- 마지막으로, 네 번째 지시하는 물감(0번)은 선반으로부터 비계 위로 옮긴다. 조수가 2번 물감을 가지고 내려오면 비계는 [3, 0]이 된다.

위의 예제에서는, 조수가 레오나르도의 최적 전략을 사용하지 않았음에 유의하라. 최적 전략은 세번째 단계에서 물감 2를 가지고 내려오는 것이며, 그 경우 조수는 마지막 단계에서 한번 더 쉴 수 있다.

기억에 제약이 있는 경우의 조수의 전략

최적의 전략을 찾고 그것을 따라갈 수 있도록, 아침에 조수는 레오나르도에게 C를 종이에 적어달라고 부탁한다. 그러나, 레오나르도는 그의 작업 비밀 유지에 대한 강박감을 갖고 있어서 종이에 적어주는것을 거절한다. 레오나르도는 조수가 C를 읽고 외우는것만 허락했다.

불행하게도, 조수의 기억력은 매우 좋지않다. 조수는 최대 M 비트만 기억할 수 있다. 일반적으로, 이것은 조수가 전체 순서 C를 재구성 하는것을 방해할 수 있다. 그래서, 조수는 그가 기억할 수 있도록 비트의 순서를 계산하는 어떤 똑똑한 방법을 찾아야만 한다. 이 순서를 *advice 정보*라 부르고 이것을 A로 나타낸다.

예제 2

아침에 조수는 레오나르도의 순서 C가 적혀있는 종이를 받아서 읽고 모든 필요한 선택을

할 수 있다. 하나는, 각각의 지시후에 비계의 상태를 적는것을 선택할 수도 있다. 예를 들어, 예제 1에서 주어진 (차선의) 선택을 할 경우, 비계 상태의 순서는 [0, 2], [0, 2], [3, 2], [3, 0]이 될 것이다. (조수가 비계의 초기 상태 [0, 1]을 알고 있으므로 이것을 적을 필요는 없다.)

조수가 정보의 16비트만을 기억할 수 있는 경우, 즉 M = 16, 라고 가정해보자. N = 4 이므로, 각 물감을 저장하는데에는 2비트만 필요하다. 그러므로, 16 비트는 비계 상태의 순서를 저장하는데 충분하다. 그래서, 조수는 다음의 advice 정보를 계산할 수 있다: A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`).

그 날의 나중에, 조수는 그의 advice 정보를 해독하여 그의 선택에 사용할 수 있다.

(물론, M = 16 인 경우에 조수는 단지 16비트 중에서 8 비트만을 사용하여 C의 순서를 모두 기억할 수도 있다. 이 예제는, 좋은 솔루션 제시가 목적이 아니고, 단지 다른 선택들을 보여주기 위한것이다.)

해야 할 일

여러분이 할 일은 같은 프로그래밍 언어로 *두 개의 분리된 프로그램들*을 작성하는 것이다. 이 프로그램들은 실행되는 동안 서로 통신하지 않고, 각각 순서대로 실행될 것이다.

첫 번째 프로그램은 아침에 조수가 사용할 것이다. 이 프로그램은 순서 C가 주어졌을 때, advice 정보 A를 만들어 내야 한다.

두 번째 프로그램은 그 날 동안에 조수가 사용할 것이다. 이 프로그램은 advice 정보 A를 받아들인 다음, 레오나르도의 지시 순서 C를 처리해야 한다. 한 번에 오직 하나의 레오나르도의 지시를 받아낼 수 있으며, 각각의 지시는 새로운 지시를 받아내기 전에 처리하여야 함에 주목하라.

더 정확하게 말하면, 첫 번째 프로그램에 하나의 함수 `ComputeAdvice`(C, N, K, M) 를 구현해야 한다. 이 함수의 입력으로 N개의 정수로 이루어진 배열 C (정수는 각각 0 이상 N-1 이하임), 비계에 있는 물감의 수 K, 그리고 가능한 advice의 비트 수 M 이 주어진다. 이 프로그램은 반드시 최대 M 비트로 이루어진 advice 정보 A를 만들어 내야 하며, 만들어 낸 A를 시스템에 알려주기 위해서, A의 각 비트에 대해 순서대로 다음의 주어진 함수를 호출해야 한다:

- `WriteAdvice`(B) — 현재의 advice 정보 A의 마지막에 비트 B를 추가한다. (최대 M 번 이 함수를 호출할 수 있다.)

두 번째 프로그램에도 하나의 함수 `Assist`(A, N, K, R) 를 구현해야 한다. 이 함수의 입력으로 advice 정보 A, 위에서 정의한 정수 N, K, advice 정보 A의 실제 비트 수 R (R ≤ M) 이 주어진다. 이 함수는 조수를 돕기 위해서, 다음의 주어지는 함수들을 이용하여 여러분이 생각한 전략대로 처리해야 한다.

- `GetRequest`() — 레오나르도가 지시한 다음 물감을 반환한다. (그 이후의 지시에 대한 정보는 주어지지 않는다.)
- `PutBack`(T) — 비계에 놓여진 물감 T를 선반으로 내려 둔다. 현재 비계에 있는 물감 T에 대해서만 이 함수를 호출하여야 한다.

이 프로그램이 실행되었을 때 `Assist` 함수는 반드시, 레오나르도의 지시를 순서대로 하나씩 받아내는 `GetRequest` 함수를 정확히 N 번 호출해야 한다. 매번 `GetRequest` 함수를 호출 한 후 레오나르도가 지시한 물감이 비계에 *없으면*, 여러분이 선택한 T에 대해 `PutBack`(T) 함수를 *반드시* 호출해야 한다. 그렇지 않으면, `PutBack` 함수를 *절대로 호출하면 안된다*. 이 규칙을 지키지 못하면, 오류로 간주되어 프로그램이 종료된다. 맨 처음 비계에 0부터 K - 1까지의 물감이 있다는 것을 기억하라.

각 테스트 케이스는 여러분의 두 함수가 모든 주어진 조건을 만족하고, `PutBack` 함수의 호출 횟수가 레오나르도의 최적 전략과 *정확히 같은 경우* 답으로 인정될 것이다. 만약 같은 횟수의 `PutBack` 함수를 호출하는 여러 전략이 있다면, 그 중 아무 전략이나 실행해도 괜찮다. (즉, 레오나르도의 최적 전략과 똑같이 좋은 전략이 있는 경우, 레오나르도의 최적 전략을 사용할 필요는 없다.)

예제 3

예제 2에 이어, `ComputeAdvice` 함수에서 계산한 결과가 A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`) 라고 가정하자. 이 때, 이 정보를 시스템에게 알려주기 위해, 다음과 같이 함수들을 순서대로 호출하여야 한다: `WriteAdvice`(`0`), `WriteAdvice`(`0`), `WriteAdvice`(`1`), `WriteAdvice`(`0`), `WriteAdvice`(`0`), `WriteAdvice`(`0`), `WriteAdvice`(`1`), `WriteAdvice`(`0`), `WriteAdvice`(`1`), `WriteAdvice`(`1`), `WriteAdvice`(`0`), `WriteAdvice`(`1`), `WriteAdvice`(`1`), `WriteAdvice`(`0`), `WriteAdvice`(`0`).

두 번째 함수 `Assist` 는 위의 정보 A, 값 N = 4, K = 2, 그리고 R = 16 과 함께 실행된다. 함수 `Assist` 는 정확히 N = 4 번의 `GetRequest` 함수를 호출하여야 한다. 또한, 필요한 시점에 어떤 적절한 T에 대해서 `PutBack`(T) 함수를 호출해야 한다.

다음의 표는 예제 1에서 선택한 (차선의) 방법에 대해, 그에 상응하는 호출 순서를 보여준다. 하이픈(-)은 `PutBack` 함수를 호출하지 않았음을 의미한다.

`GetRequest()`	Action
2	`PutBack`(1)
0	-
3	`PutBack`(0)
0	`PutBack`(2)

서브태스크 1 [8점]

- N ≤ 5,000.
- 최대 M = 65,000 비트를 사용할 수 있다.

서브태스크 2 [9점]

- N ≤ 100,000.
- 최대 M = 2,000,000 비트를 사용할 수 있다.

서브태스크 3 [9점]

- $N \leq 100,000$.
- $K \leq 25,000$.
- 최대 $M = 1,500,000$ 비트를 사용할 수 있다.

서브태스크 4 [35점]

- $N \leq 5,000$.
- 최대 $M = 10,000$ 비트를 사용할 수 있다.

서브태스크 5 [최대 39점]

- $N \leq 100,000$.
- $K \leq 25,000$.
- 최대 $M = 1,800,000$ 비트를 사용할 수 있다.

이 서브태스크의 점수는 여러분의 프로그램이 만드는 advice 정보의 길이 R 에 따라 다르다. R_{max} 를 이 서브태스크에 포함된 모든 테스트 케이스에 대해, 여러분의 함수 `ComputeAdvice` 가 생성한 advice 정보의 길이 중 최대값으로 두면, 점수는 다음과 같다:

- $R_{max} \leq 200,000$ 이면 39점;
- $200,000 < R_{max} < 1,800,000$ 이면 $39 (1,800,000 - R_{max}) / 1,600,000$ 점;
- $R_{max} \geq 1,800,000$ 이면 0점.

구현 세부 사항

여러분은 *같은 프로그래밍 언어로 작성된* 두 개의 파일을 제출해야만 한다.

첫 번째 파일은 `advisor.c`, `advisor.cpp` 혹은 `advisor.pas` 이다. 이 파일은 위에 기술된 것처럼 `ComputeAdvice` 함수를 반드시 구현해야 하고, `WriteAdvice` 함수를 호출 할 수 있다. 두 번째 파일은 `assistant.c`, `assistant.cpp` 혹은 `assistant.pas` 이다. 이 파일은 위에 기술된 것처럼 `Assist` 함수를 반드시 구현해야 하고, `GetRequest` 와 `PutBack` 함수를 호출 할 수 있다.

모든 함수에 대한 선언은 다음과 같다.

C/C++ 프로그램

```
void ComputeAdvice(int *C, int N, int K, int M);
void WriteAdvice(unsigned char a);
```

```
void Assist(unsigned char *A, int N, int K, int R);
void PutBack(int T);
int GetRequest();
```

Pascal 프로그램

```
procedure ComputeAdvice(var C : array of LongInt; N, K, M : LongInt);
procedure WriteAdvice(a : Byte);
```

```
procedure Assist(var A : array of Byte; N, K, R : LongInt);
procedure PutBack(T : LongInt);
function GetRequest : LongInt;
```

이 함수들은 위에 기술한대로 동작하여야 한다. 물론, 내부적으로 사용되는 다른 함수들의 구현 내용은 자유롭게 하면 된다. C/C++ 프로그램에서, 내부적으로 사용되는 함수들은 예시 grader가 해당 함수들과 함께 링크되기 위해, 반드시 `static`으로 선언되어야만 한다. 또, 두 개의 프로그램에 같은 이름을 가진 함수를 작성하면 안된다. 제출한 프로그램은 어떠한 방식으로든 표준 입/출력뿐만 아니라, 다른 어떠한 파일과도 상호작용을 하여서는 안된다.

풀이를 작성할 때, 또한 여러분은 아래에 주어진 지시에 주의하여야 한다. (시험 환경에서 제공되는 템플릿은 이미 아래에 주어진 요건들을 만족한다.)

C/C++ 프로그램

풀이의 가장 앞에, `advisor.h` 와 `assistant.h` 를 각각 advisor 과 assistant 에 포함해야 한다. 여러분의 코드에 다음의 줄을 입력하면 된다:

```
#include "advisor.h"
```

또는

```
#include "assistant.h"
```

두 개의 파일 `advisor.h` 와 `assistant.h` 가 시험 환경에 있는 디렉토리 안에 제공되고, 대회용 웹 페이지에서도 얻을 수 있다. 또한, (앞의 파일들과 같은 방법으로) 여러분이 작성한 풀이를 컴파일하고 테스트해 볼 수 있는 코드와 스크립트도 제공될 것이다. 구체적으로 말하면, 여러분의 풀이를 이 스크립트들과 같은 디렉토리에 넣은 후, `compile_c.sh` 또는 `compile_cpp.sh` 를 작성한 코드의 프로그래밍 언어에 따라 실행시키면 된다.

Pascal 프로그램

You have to use the units `advisorlib` and `assistantlib`, respectively, in the advisor and in the assistant. This is done by including in your source the line:

```
uses advisorlib;
```

or

```
uses assistantlib;
```

The two files `advisorlib.pas` and `assistantlib.pas` will be provided to you in a directory inside your contest environment and will also be offered by the contest Web interface. You'll also be provided (through the same channels) with code and scripts to compile and test your solution. Specifically, after copying your solution into the directory with these scripts, you will have to run `compile_pas.sh`.

grader 예시

주어지는 grader는 다음의 형식에 맞는 입력을 받아들인다:

- line 1: N, K, M;
- lines 2, …, N + 1: C[i].

주어지는 grader는 제일 처음으로 함수 `ComputeAdvice`를 호출할 것이다. 또한, 이 grader는 각각의 advice 정보가 공백으로 분리되어 있고, 2 로 종료되는 파일 `advice.txt`를 생성할 것이다.

다음으로, 주어지는 grader는 당신의 `Assist` 함수를 실행하고, 각 줄이 `"R [number]"` 혹은 `"P [number]"`의 형태로만 이루어지는 출력 파일을 생성할 것이다. 첫 번째 타입(`"R [number]"`)은 `GetRequest()`의 함수 호출과, 그 함수의 반환값을 의미한다. 또한, 두 번째 타입(`"P [number]"`)은 `PutBack()` 함수의 호출과 내려 놓기로 선택한 물감을 의미한다. 이 출력 파일은 `"E"` 형태의 줄로 끝난다.

공식 grader에서의 실행 시간은 여러분의 로컬 컴퓨터에서 실행한 시간과 비교하여 약간 다를 수 있다. 이 차이는 별로 중요하지 않다. 그렇지만, 여러분은 제한 시간 안에 여러분의 풀이가 실행되는지 확인 하기 위해서 테스트 환경을 사용해 볼 수 있다.

시간 및 메모리 제한

- 시간 제한: 7 초.
- 메모리 제한: 256 MB.