Bữa Tối Cuối Cùng

Leonardo rất hăng hái khi thực hiện tác phẩm Bữa Tối Cuối Cùng, tác phẩm tranh tường nổi tiếng nhất của ông: một trong những công việc hằng ngày đầu tiên của ông là quyết định các màu keo nào sẽ sử dụng trong ngày. Ông cần nhiều màu nhưng chỉ có thể giữ một số lượng hạn chế các màu trên giàn giáo. Trợ lý của ông là người đảm nhận, ngoài các nhiệm vụ khác, việc trèo lên giàn giáo để đưa màu cho ông và trèo xuống để đặt màu vào kệ phù hợp ở sàn nhà.

Trong bài này, bạn sẽ phải viết hai chương trình khác nhau để giúp đỡ người trợ lý. Chương trình thứ nhất sẽ nhận các yêu cầu của Leonardo (một chuỗi các màu mà Leonardo sẽ cần trong ngày), và tạo ra một xâu *ngắn* chứa các bit, được gọi là *chỉ dẫn*. Khi xử lý các yêu cầu của Leonardo trong ngày, người trợ lý sẽ không được truy cập đến các yêu cầu của Leonardo trong tương lai, mà chỉ được truy cập đến chỉ dẫn mà chương trình đầu tiên của bạn tạo ra. Chương trình thứ hai sẽ nhận chỉ dẫn này, và sau đó nhận và xử lý các yêu cầu của Leonardo theo hình thức online (nghĩa là từng yêu cầu một). Chương trình này phải hiểu được chỉ dẫn có ý nghĩa gì và sử dụng nó để đưa ra các lựa chọn tối ưu. Tất cả sẽ được giải thích cụ thể hơn sau đây.

Di chuyển màu giữa kệ và giàn giáo

Chúng ta sẽ xét một trường hợp đơn giản. Giả sử có N màu được đánh số từ 0 đến N - 1, và mỗi ngày Leonardo yêu cầu người trợ lý một màu mới đúng N lần. Coi C là một chuỗi gồm N yêu cầu lấy màu của Leonardo. Vì vậy, chúng ta có thể coi C là một chuỗi N số, mỗi số nằm từ 0 đến N - 1, tính cả 2 đầu mút. Lưu ý rằng một số màu có thể không xuất hiện trong C, và một số màu có thể xuất hiện nhiều lần trong C.

Giàn giáo lúc nào cũng đầy và chứa K màu trong số N màu với $K < N$. Ban đầu, giàn giáo chứa các màu từ 0 đến K - 1, gồm cả hai đầu mút.

Người trợ lý xử lý các yêu cầu của Leonardo lần lượt từng yêu cầu một. Khi màu được yêu cầu *đang sẵn có trên giàn giáo* thì người trợ lý có thể được nghỉ ngơi. Ngược lại, anh ta phải lấy màu được yêu cầu ở trên kệ và chuyển lên giàn giáo. Tất nhiên, nếu không có chỗ trên giàn giáo cho màu mới, người trợ lý phải chọn một trong các màu hiện có trên giàn giáo và mang nó xuống để trên kệ.

Chiến thuật tối ưu của Leonardo

Người trợ lý muốn được nghỉ ngơi càng nhiều lần càng tốt. Số lượng các yêu cầu mà anh ta được nghỉ ngơi phụ thuộc vào lựa chọn của anh ta trong quá trình thực hiện yêu cầu. Một cách chính xác hơn, mỗi khi người trợ lý phải bỏ một màu khỏi giàn giáo, các lựa chọn khác nhau có thể dẫn đến kết quả khác nhau trong tương lai. Leonardo giải thích cho anh ta làm cách nào mà anh ta có thể đạt được mục đích của mình khi đã biết C. Lựa chọn tốt nhất để bỏ một màu khỏi giàn giáo có thể

thu được bằng cách xem xét các màu đang ở trên giàn giáo, và các yêu cầu màu còn lại trong C. Một màu sẽ được lựa chọn trong các màu đang có trên giàn giáo theo các quy tắc sau:

- Nếu có một màu trên giàn giáo mà sẽ không cần tới trong tương lai, người trợ lý nên bỏ màu này khỏi giàn giáo.
- Ngược lại, màu cần loại bỏ khỏi giàn giáo là màu mà *sẽ cần tới muộn nhất trong tương lai*. (Nghĩa là, với mỗi màu đang có trên giàn giáo ta xác định thời điểm tiếp theo đầu tiên mà nó được yêu cầu. Màu được bỏ khỏi giàn giáo xuống kệ là màu sẽ được yêu cầu muộn nhất.)

Có thể chứng minh được rằng khi sử dụng chiến thuật của Leonardo, người trợ lý sẽ được nghỉ ngơi nhiều lần nhất có thể được.

Ví dụ 1

Với N = 4, chúng ta có 4 màu (đánh số từ 0 đến 3) và 4 yêu cầu. Xét chuỗi yêu cầu C = (2, 0, 3, 0). Giả sử K = 2. Tức là Leonardo sử dụng giàn giáo có thể chứa được 2 màu vào mỗi một thời điểm. Như đã mô tả ở trên, ban đầu giàn giáo có màu 0 và 1. Chúng ta sẽ ghi các màu trên giàn giáo như sau: [0, 1]. Một phương án người trợ lý có thể xử lý các yêu cầu được mô tả như sau.

- Màu đầu tiên được yêu cầu (số 2) không có trên giàn giáo. Người trợ lý mang nó lên giàn giáo và quyết định bỏ màu số 1 xuống khỏi giàn giáo. Trạng thái hiện tại của giàn giáo là [0, 2].
- Màu được yêu cầu tiếp theo (số 0) đang sẵn có trên giàn giáo, nên người trợ lý có thể nghỉ ngơi.
- Với yêu cầu thứ ba (màu số 3), người trợ lý bỏ màu 0, đổi trạng thái của giàn giáo thành [3, 2].
- Cuối cùng, màu được yêu cầu sau chót (số 0) phải được lấy từ kệ đưa lên giàn giáo. Người trợ lý quyết định bỏ màu 2 xuống, và trạng thái của giàn giáo là [3, 0].

Lưu ý rằng trong ví dụ trên, người trợ lý đã không đi theo chiến thuật tối ưu của Leonardo. Chiến thuật tối ưu sẽ bỏ màu 2 xuống tại bước thứ 3, như vậy người trợ lý sẽ được nghỉ ngơi ở bước cuối cùng.

Chiến thuật của người trợ lý khi trí nhớ bị hạn chế

Vào buổi sáng, người trợ lý đề nghị Leonardo viết C vào một tờ giấy, để anh ta có thể xác định và tuân theo chiến thuật tối ưu. Tuy nhiên, Leonard bị ám ảnh với việc muốn giữ bí mật các kỹ thuật của mình, nên ông ta từ chối đưa người trợ lý tờ giấy. Ông ta chỉ cho người trợ lý đọc C và phải nhớ nó.

Không may, người trợ lý có trí nhớ rất kém. Anh ta chỉ có thể nhớ không quá M bit. Nhìn chung, việc này có thể khiến anh ta không thể tái tạo lại toàn bộ chuỗi C. Chính vì vậy, người trợ lý phải sáng tạo ra một cách để tính toán chuỗi các bit mà anh ta sẽ ghi nhớ. Chúng ta gọi chuỗi này là *chuỗi chỉ dẫn* và đặt tên là A.

Ví dụ 2

Vào buổi sáng, người trợ lý có thể lấy tờ giấy ghi chuỗi C của Leonardo, đọc chuỗi này, và quyết

định các lựa chọn cần thiết. Một việc anh ta có thể làm là đánh giá trạng thái của giàn giáo sau mỗi yêu cầu. Ví dụ, khi sử dụng chiến thuật (không tối ưu) như đã cho trong Ví dụ 1, chuỗi trạng thái của giàn giáo sẽ là [0, 2], [0, 2], [3, 2], [3, 0]. (Nên nhớ rằng anh ta biết trạng thái đầu tiên của giàn giáo là [0, 1].)

Bây giờ giả sử chúng ta có M = 16, nên người trợ lý có thể nhớ đến 16 bit thông tin. Với N = 4, ta có thể lưu giữ mỗi màu sử dụng 2 bit. Cho nên 16 bit là đủ để lưu trữ chuỗi trạng thái giàn giáo ở trên. Vì vậy, người trợ lý có thể tính ra chuỗi chỉ dẫn sau: A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`).

Trong ngày, người trợ lý có thể giải mã chuỗi chỉ dẫn này và sử dụng nó để quyết định các lựa chọn của mình.

(Tất nhiên, với M = 16 người trợ lý có thể chọn để nhớ toàn bộ chuỗi C chỉ sử dụng 8 trong số 16 bit. Trong ví dụ này, chúng tôi chỉ muốn minh họa rằng anh ta có thể có các lựa chọn khác, mà vẫn cho kết quả tốt.)

Mô tả bài toán

Bạn phải viết *hai chương trình riêng biệt* sử dụng cùng một ngôn ngữ lập trình. Hai chương trình này sẽ được chạy theo thứ tự nối đuôi nhau, và chúng không được liên lạc với nhau trong khi chạy.

Chương trình đầu tiên sẽ được người trợ lý sử dụng vào buổi sáng. Chương trình này sẽ được cho biết chuỗi C, và nó phải tính ra chuỗi chỉ dẫn A.

Chương trình thứ 2 sẽ được người trợ lý sử dụng trong ngày. Chương trình này sẽ nhận được chuỗi chỉ dẫn A, và sau đó nó phải xử lý chuỗi C gồm các yêu cầu của Leonardo. Lưu ý rằng chuỗi C sẽ chỉ được đưa cho chương trình dưới dạng mỗi lần một yêu cầu, và mỗi yêu cầu phải được xử lý xong trước khi nhận được yêu cầu tiếp theo.

Nói một cách chính xác, trong chương trình thứ nhất bạn phải cài đặt một thủ tục duy nhất `ComputeAdvice`(C, N, K, M) có đầu vào là mảng C gồm N số nguyên (mỗi số nguyên nằm trong 0, …, N - 1), K là số màu trên giàn giáo, và M là số bit để lưu chỉ dẫn. Chương trình này phải tính ra chuỗi chỉ dẫn A sử dụng không quá M bit. Tiếp theo, chương trình phải đưa cho hệ thống chuỗi A bằng cách gọi thủ tục sau cho từng bit của A theo thứ tự:

- `WriteAdvice`(B) — Thêm bit B vào cuối chuỗi chỉ dẫn A. (Bạn có thể gọi thủ tục này nhiều nhất M lần.)

Ở chương trình thứ 2, bạn phải cài đặt một thủ tục duy nhất `Assist`(A, N, K, R). Đầu vào cho thủ tục này là chuỗi chỉ dẫn A, các số N và K như đã định nghĩa ở trên, và độ dài thực tế R của chuỗi A dưới dạng bit (R ≤ M). Thủ tục này sẽ thực hiện cho người trợ lý chiến thuật bạn đề xuất, sử dụng các thủ tục được cung cấp cho bạn như sau:

- `GetRequest`() — trả lại màu tiếp theo được Leonardo yêu cầu. (Không có thông tin nào về các yêu cầu trong tương lai được cung cấp.)
- `PutBack`(T) — Bỏ màu T từ giàn giáo xuống kệ. Bạn chỉ được gọi thủ tục này khi T là màu đang có trên giàn giáo.

Khi chạy, thủ tục `Assist` của bạn phải gọi `GetRequest` đúng N lần, mỗi lần sẽ nhận được một yêu cầu của Leonardo, theo thứ tự. Sau mỗi lần gọi `GetRequest`, nếu màu được yêu cầu trả về *không* nằm trên giàn giáo, bạn *phải* gọi `PutBack`(T) với lựa chọn của bạn cho T. Ngược lại, bạn *không được* gọi `PutBack`. Nếu bạn không làm như vậy thì sẽ bị coi như là lỗi và chương trình bạn sẽ bị kết thúc. Nên nhớ rằng ban đầu giàn giáo chứa các màu từ 0 đến K - 1, gồm cả hai đầu mút.

Một test sẽ được coi là giải xong nếu hai thủ tục của bạn tuân theo tất cả các yêu cầu, và tổng số lần gọi `PutBack` *đúng bằng* số lần gọi theo chiến thuật tối ưu của Leonardo. Lưu ý rằng nếu có nhiều chiến thuật với cùng số lần gọi `PutBack`, chương trình của bạn được phép sử dụng bất kỳ chiến thuật nào trong chúng. (Nghĩa là không nhất thiết phải theo chiến thuật của Leonardo, nếu có một chiến thuật tốt tương đương khác.)

Ví dụ 3

Tiếp theo Ví dụ 2, giả sử trong `ComputeAdvice` bạn tính ra A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`). Để đưa thông tin cho hệ thống, bạn phải thực hiện chuỗi các lệnh gọi sau: `WriteAdvice(0)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(0)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(1)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(1)`, `WriteAdvice(1)`, `WriteAdvice(0)`, `WriteAdvice(0)`.

Sau đó thủ tục thứ 2 của bạn `Assist` sẽ được chạy, nhận chuỗi A ở trên, và các giá trị N = 4, K = 2, and R = 16. Thủ tục `Assist` phải thực hiện đúng N = 4 lần gọi `GetRequest`. Sau một số lần gọi đó, `Assist` sẽ phải gọi `PutBack`(T) với lựa chọn phù hợp cho T.

Bảng dưới đây mô tả chuỗi các lần gọi tương ứng với các lựa chọn (không tối ưu) ở Ví dụ 1. Ký tự gạch ngang thể hiện không có việc gọi `PutBack`.

`GetRequest()`	Hành động
2	`PutBack`(1)
0	-
3	`PutBack`(0)
0	`PutBack`(2)

Subtask 1 [8 points]

- N ≤ 5 000.
- Bạn có thể sử dụng tối đa M = 65 000 bit.

Subtask 2 [9 points]

- N ≤ 100 000.
- Bạn có thể sử dụng tối đa M = 2 000 000 bit.

Subtask 3 [9 points]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Bạn có thể sử dụng tối đa M = 1 500 000 bit.

Subtask 4 [35 points]

- $N \leq 5\,000$.
- Bạn có thể sử dụng tối đa M = 10 000 bit.

Subtask 5 [up to 39 points]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Bạn có thể sử dụng tối đa M = 1 800 000 bit.

Điểm của subtask này phụ thuộc vào độ dài R của chuỗi chỉ dẫn mà chương trình bạn tính được. Nói một cách chính xác, nếu R_{max} là giá trị lớn nhất (trên tất các test con) của độ dài chuỗi chỉ dẫn mà thủ tục `ComputeAdvice` tính ra, điểm của bạn sẽ là:

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

Chi tiết cài đặt

Bạn phải nộp đúng hai file *sử dụng cùng một ngôn ngữ lập trình*.

File đầu tiên được gọi là `advisor.c`, `advisor.cpp` hoặc `advisor.pas`. File này phải cài đặt thủ tục `ComputeAdvice` như đã mô tả ở trên và có thể gọi thủ tục `WriteAdvice`. File thứ 2 được gọi là `assistant.c`, `assistant.cpp` hoặc `assistant.pas`. File này phải cài đặt thủ tục `Assist` như đã mô tả ở trên và có thể gọi các thủ tục `GetRequest` và `PutBack`.

Mẫu của các thủ tục như sau.

C/C++ programs

```
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 programs

```
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;
```

Các thủ tục này phải hoạt động như đã mô tả ở trên. Tất nhiên bạn có thể cài đặt thêm những thủ tục con khác để sử dụng nội bộ. Đối với các chương trình C/C++, các thủ tục nội bộ của bạn phải được khai báo `static`, vì sample grader sẽ link chúng với nhau. Một cách khác, chỉ đơn giản tránh có hai thủ tục (mỗi thủ tục ở một chương trình) trùng tên. Các lần giao nộp không được giao tiếp dưới bất cứ hình thức nào với input/output chuẩn, cũng như với bất cứ file nào khác.

Khi cài đặt chương trình của mình, bạn phải tuân theo các chỉ dẫn sau (template mẫu có sẵn trong môi trường thi đã thỏa mãn các yêu cầu dưới đây).

C/C++ programs

Ngay đầu lời giải của bạn, bạn phải include file `advisor.h` trong chương trình advisor và `assistant.h` trong chương trình assistant. Việc này được thực hiện bằng cách sử dụng các dòng lệnh sau trong chương trình nguồn:

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

hoặc

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

Hai file `advisor.h` và `assistant.h` sẽ được cung cấp cho bạn trong thư mục ở môi trường thi và cũng được cung cấp sử dụng giao diện Web của cuộc thi. Bạn cũng sẽ được cung cấp (vẫn dưới các hình thức như trên) code và scripts để biên dịch và test lời giải của mình. Cụ thể, sau khi copy lời giải của mình vào thư mục chứa các scripts này, bạn có thể chạy `compile_c.sh` hoặc `compile_cpp.sh` (phụ thuộc vào ngôn ngữ lập trình mà bạn sử dụng)

Pascal programs

Bạn phải sử dụng gói `advisorlib` trong chương trình advisor và `assistantlib` trong chương trình assistant. Việc này được thực hiện bằng cách sử dụng các dòng lệnh sau trong chương trình nguồn:

```
uses advisorlib;
```

hoặc

```
uses assistantlib;
```

Hai file `advisorlib.pas` và `assistantlib.pas` sẽ được cung cấp cho bạn trong thư mục ở môi trường thi và cũng được cung cấp sử dụng giao diện Web của cuộc thi. Bạn cũng sẽ được cung cấp (vẫn dưới các hình thức như trên) code và scripts để biên dịch và test lời giải của mình. Cụ thể, sau khi copy lời giải của mình vào thư mục chứa các scripts này, bạn có thể chạy `compile_c.sh`.

Sample grader

Sample grader sẽ nhận đầu vào với định dạng như sau:

- Dòng 1: N, K, M;
- Các dòng 2, …, N + 1: C[i].

Đầu tiên grader sẽ chạy thủ tục `ComputeAdvice`. Việc này sẽ tạo ra file `advice.txt`, bao gồm các bit của chuỗi chỉ dẫn, cách nhau bởi các dấu trống và kết thúc bởi một số 2.

Sau đó, nó sẽ tiếp tục chạy thủ tục `Assist` của bạn, và tạo ra đầu ra với mỗi dòng có định dạng "`R [number]`" hoặc "`P [number]`". Các dòng có định dạng theo kiểu đầu tiên tương ứng với các lệnh gọi `GetRequest()` và các trả lời nhận được. Các dòng có định dạng theo kiểu thứ hai tương ứng với các lệnh gọi `PutBack`() và các màu được lựa chọn để bỏ xuống. Đầu ra được kết thúc bởi một dòng có định dạng "`E`".

Lưu ý rằng với chương trình grader chính thức, thời gian chương trình bạn chạy sẽ khác một chút so với chạy trên máy tính của bạn. Sự khác biệt này không đáng kể. Bạn có thể sử dụng giao diện test để kiểm tra liệu lời giải của bạn có chạy trong khoảng giới hạn thời gian cho phép không.

Các giới hạn thời gian và bộ nhớ

- Giới hạn thời gian: 7 giây.
- Giới hạn bộ nhớ: 256 MiB.