شام آخر

لئوناردو در حین خلق اثر «شام آخر» - معروفترین نقاشی دیواریش - خیلی فعال بود: یکی از فعالیتهای اولیه روزانه اش تصمیم گیری در مورد رنگهایی بود که میخواست آن روز استفاده کند. او رنگهای زیادی نیاز داشت، اما تعداد محدودی از آنها را میتوانست روی داربست نگه دارد. از جمله وظایف دستیارش آن بود که روی داربست رود و رنگهای مورد نیاز را به لئوناردو برساند و دوباره روی زمین برگردد و رنگهای اضافی را در قفسه مناسب روی زمین قرار دهد.

در این مساله شما باید دو برنامه مجزا بنویسید که به دستیار کمک کند. به اولین برنامه توالی رنگهایی که لئوناردو در طول روز نیاز دارد به عنوان ورودی داده میشود و باید یک رشته *کوتاه* از بیتها که *راهنما* نام دارد ایجاد کند. در طول روز، دستیار حین اجرای درخواستهای لئوناردو، به درخواستهای آینده او دسترسی ندارد و فقط راهنمای ایجاد شده توسط برنامه اول را در اختیار دارد. رشته راهنما به عنوان ورودی به برنامه دوم داده میشود و سپس این برنامه میبایست درخواستهای لئوناردو را به صورت برخط (Online) دریافت و پردازش کند (یعنی در هر زمان یک درخواست). این برنامه باید معنی رشته راهنما را بفهمد و از آن برای اتخاذ انتخابهای بهینه استفاده کند. همه چیز با جزییات بیشتری در زیر توضیح داده شده است.

انتقال رنگها بین قفسه و داربست

ما یک سناریوی ساده را در نظر میگیریم. فرض کنید N رنگ با شماره های 0 تا N−1 وجود دارد، و هر روز لئوناردو از دستیارش دقیقا N بار درخواست رنگ جدید میکند. فرض کنید C توالی N رنگی باشد که لئوناردو درخواست کرده است. پس میتوانیم C را توالی N عدد فرض کنیم که هریک از 0 تا N−1 هستند. بعضی از رنگها ممکن است اصلا در C وجود نداشته باشند و بعضی چند بار تکرار شوند.

داربست همیشه پر است و K تا از N رنگ در آن قرار دارد که $K < N$. در ابتدا رنگهای 0 تا K−1 در داربست قرار دارند.

دستیار، در هر زمان یکی از درخواستهای لئوناردو را انجام میدهد. اگر رنگ خواسته شده *قبلا در داربست باشد* دستیار استراحت میکند، وگرنه مجبور است آن رنگ را از قفسه برداشته به داربست منتقل کند. البته جایی در داربست برای رنگ جدید نیست، از این رو دستیار باید یکی از رنگهای قبلی موجود در داربست را انتخاب و آن را به قفسه برگرداند.

استراتژی بهینه لئوناردو

دستیار میخواهد تا حد ممکن به دفعات بیشتری استراحت کند. تعداد درخواستهایی که به ازای آن میتواند استراحت کند به انتخابهای او در طول فرایند بستگی دارد. به طور دقیق تر، هر بار که دستیار ناچار به حذف یک رنگ از داربست میشود، انتخابهای مختلف منجر به نتایج متفاوت در آینده خواهند شد. لئوناردو به او تشریح میکند که با دانستن C چگونه میتواند به هدفش برسد: بهترین انتخاب برای رنگی که باید از داربست حذف شود با چک کردن رنگهایی که در حال حاضر در داربست هستند، و رنگهای باقیمانده در فهرست درخواستهای C به دست می آید. از بین رنگهای موجود در داربست باید یک رنگ طبق قواعد زیر انتخاب شود:

- اگر رنگی در داربست هست که هیچ وقت در آینده مورد نیاز قرار نمیگیرد، دستیار باید آن رنگ را از داربست حذف کند.
- در غیر این صورت رنگی باید از داربست حذف شود که *دوباره در دیرترین هنگام در آینده مورد نیاز واقع شود*. (یعنی برای هر یک از رنگهای روی داربست، ما اولین نیاز بدان را در آینده می یابیم. رنگی باید به قفسه برگردانده شود که آخر همه مورد نیاز قرار میگیرد).

میتوان اثبات کرد که با استفاده از استراتژی لئوناردو، دستیار بیشترین دفعات ممکن استراحت خواهد کرد.

مثال ۱

فرض کنید $N = 4$، پس ما ۴ رنگ داریم (با شماره های ۰ تا ۳) و ۴ درخواست. فرض کنید توالی درخواستها $C = (2, 0, 3, 0)$ و نیز $K = 2$ است، یعنی داربست لئوناردو در هر لحظه جای نگهداری ۲ رنگ دارد. چنانچه شرح داده شد، در ابتدا رنگهای ۰ و ۱ در داربست هستند. ما رنگهای داربست را به صورت $[0, 1]$ نمایش میدهیم. دستیار ممکن است به ترتیب زیر درخواستها را انجام دهد.

- اولین درخواست (رنگ ۲) روی داربست نیست. دستیار آن را به داربست منتقل کرده، تصمیم میگیرد رنگ ۱ را از داربست حذف کند. وضعیت کنونی داربست [2 ,0] خواهد بود.
- درخواست بعدی (رنگ ۰) قبلا روی داربست هست. پس دستیار میتواند استراحت کند.
- برای انجام درخواست سوم (رنگ ۳) دستیار رنگ ۰ را از روی داربست برمیدارد، که باعث میشود داربست به [3, 2] تغییر وضعیت دهد.
- سرانجام، درخواست آخر (رنگ ۰) باید از قفسه به داربست منتقل شود. دستیار تصمیم به جایگزینی با رنگ ۲ میگیرد، و وضعیت کنونی داربست [3, 0] خواهد بود

توجه کنید در مثال فوق دستیار استراتژی بهینه لئوناردو را رعایت نکرد. طبق استراتژی بهینه میبایست در گام سوم، رنگ ۲ از داربست برداشته میشد و از این رو دستیار میتوانست در گام آخر مجددا استراحت کند.

استراتژی دستیار وقتی حافظه اش محدود است

صبح دستیار از لئوناردو میخواهد که C را روی یک تکه کاغذ بنویسد که او بتواند استراتژی بهینه را یافته و دنبال کند. اما لئوناردو میخواهد فوت و فن نقاشی به صورت یک راز باقی بماند، از این رو نمیگذارد دستیار کاغذ را نگاه دارد. او فقط اجازه داد دستیار C را خوانده و سعی کند که به خاطر بسپارد.

متاسفانه حافظه دستیار خیلی بد است. او فقط میتواند M بیت را به خاطر بسپرد. در حالت کلی این امر ممکن است باعث شود که او نتواند کل رشته C را به خاطر آورد. از این رو دستیار مجبور است طی یک روش هوشمندانه، رشته بیتهایی را که میخواهد به خاطر بسپرد محاسبه کند. او این رشته را *رشته راهنما* نامیده و با A مشخص میکند.

مثال ۲

صبح، دستیار میتواند کاغذ لئوناردو با توالی C روی آن را بگیرد، آن را بخواند، و همه انتخابهای لازم را انجام دهد. یک گزینه که او ممکن است انتخاب کند آن است که وضعیت داربست را بعد از هر درخواست بسنجد. به عنوان مثال اگر او استراتژی غیر بهینه مثال ۱ را در نظر بگیرد، توالی وضعیتهای داربست به صورت زیر خواهد بود (توجه کنید که او میداند وضعیت اولیه داربست [1 ,0] است، پس نیازی به نوشتن این نیست): [3, 0] ,[3, 2] ,[0, 2] ,[2 ,0]

حال فرض کنید M = 16، پس دستیار قادر است حداکثر ۱۶ بیت اطلاعات را حفظ کند. چون N = 4 ما میتوانیم هر رنگ را در ۲ بیت ذخیره کنیم. پس ۱۶ بیت برای ذخیره سازی توالی وضعیتهای داربست فوق کافی است، و دستیار رشته راهنمای زیر را محاسبه میکند:

A = (0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0)

بعدا در طول روز دستیار میتواند رشته راهنما را رمزگشایی (decode) کرده از آن برای انتخابهایش استفاده کند.

(البته با M = 16 دستیار میتواند به جای این، کل توالی C را با ۸ بیت از ۱۶ بیت موجود حفظ کند. در این مثال ما فقط خواستیم نشان دهیم او گزینه های دیگری هم دارد، بدون آن که پاسخ خوب را لو بدهیم).

شرح مساله

شما باید *دو برنامه* مجزا با زبان برنامه نویسی یکسان بنویسید. این برنامه ها یکی پس از دیگری، بدون آنکه در طول اجرا قادر به ارتباط با یکدیگر باشند، اجرا خواهند شد.

اولین برنامه همان است که دستیار صبح به کار میگیرد. این برنامه با گرفتن توالی C میبایست رشته راهنما (A) را محاسبه نماید.

برنامه دوم در طول روز توسط دستیار استفاده خواهد شد. این برنامه با گرفتن رشته راهنما، توالی C – درخواستهای لئوناردو – را پردازش میکند. توجه کنید در طول اجرای این برنامه هر دفعه یک درخواست از توالی C به برنامه داده میشود، و هر درخواست میبایست قبل از دریافت درخواست بعدی پردازش شود.

به طور دقیق تر، در برنامه اول شما باید یک رویه ComputeAdvice(C, N, K, M) بنویسید که به عنوان ورودی آرایه C از N عدد صحیح (هریک بین 0 تا 1 – N)، تعداد رنگهای روی داربست (K)، و تعداد بیتهای قابل استفاده توسط دستیار (M) را بگیرد. این برنامه میبایست یک رشته راهنمای A را که حداکثر از M بیت تشکیل شده محاسبه کند. سپس برنامه رشته A را با فراخوانی تابع زیر – برای هر بیت از A به ترتیب – به سیستم ارسال کند:

- `WriteAdvice(B)` — بیت B را به انتهای رشته راهنمای A اضافه کن. (میتوانید حداکثر M بار این رویه را فراخوانی کنید).

در برنامه دوم شما باید تنها یک رویه `Assist(A, N, K, R)` را پیاده سازی کنید. ورودی این رویه عبارت است از: رشته راهنمای A، اعداد صحیح N و K که در بالا تعریف شده اند، و نیز R – طول واقعی رشته A به صورت تعداد بیت ($R \le M$). این رویه میبایست استراتژی پیشنهادی شما برای دستیار را با استفاده از رویه های زیر که قبلا برای شما فراهم گردیده اند اجرا کند:

- `GetRequest()` — رنگ بعدی مورد درخواست لئوناردو را برگردان. (اطلاعاتی در مورد درخواستهای بعد از آن ارایه نخواهد شد).
- `PutBack(T)` — رنگ T از داربست را به قفسه منتقل کن. در فراخوانی این رویه رنگ T باید حتما در حال حاضر در داربست موجود باشد.

رویه `Assist` شما باید در حین اجرا رویه `GetRequest` را دقیقا N بار فراخوانی کند و هر دفعه یکی از درخواستهای لئوناردو را به ترتیب دریافت کند. بعد از هر فراخوانی `GetRequest` اگر رنگ بازگشتی در داربست نبود، شما *باید* رویه `PutBack(T)` را با مقدار T انتخابی خودتان نیز فراخوانی کنید. در غیر این صورت *نباید* رویه `PutBack` را فراخوانی کرد. اشتباه در رعایت این موضوع نیز یک خطا در نظر گرفته شده و باعث خاتمه اجرای برنامه شما خواهد شد. لطفا توجه کنید که در ابتدا رنگهای 0 تا K-1 در داربست قرار دارند.

یک داده تستی توسط برنامه شما «حل شده» تلقی میشود اگر دو رویه شما همه شرایط گفته شده را لحاظ کنند، و تعداد کل دفعات فراخوانی رویه `PutBack` *دقیقا برابر* با استراتژی بهینه لئوناردو باشد. توجه کنید اگر چندین استراتژی وجود داشته باشند که به تعداد یکسانی `PutBack` برسند، برنامه شما مجاز است هریک از آنها را به کار گیرد (یعنی اگر استراتژی دیگری به همین خوبی دارید حتما لازم نیست استراتژی بهینه لئوناردو را تبعیت کنید).

مثال ۳

در ادامه مثال ۲ فرض کنید که در `ComputeAdvice` شما A را به صورت زیر محاسبه کرده اید:

A = (0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0)

برای ارسال آن به سیستم شما باید توالی زیر از فراخوانی ها را اجرا کنید:

```
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).
```

پس از آن، رویه دوم شما (`Assist`)، با دریافت رشته A ی فوق و مقادیر N = 2، K = 4 و R = 16 اجرا خواهد شد. سپس رویه `Assist` میبایست دقیقا N = 4 بار رویه `GetRequest` را فراخوانی کند. همچنین بعد از بعضی از آن فراخوانی ها، `Assist` میبایست رویه `PutBack(T)` را با انتخاب مناسبی از T فراخوانی کند.

جدول زیر یک توالی از فراخوانی ها که مربوط به انتخابهای غیر بهینه مثال ۱ میباشد نمایش میدهد. خط تیره به معنی عدم فراخوانی `PutBack` است.

عمل	**`GetRequest()`**
`PutBack(1)`	2
-	0
`PutBack(0)`	3
`PutBack(2)`	0

زیرمساله ۱ [۸ نمره]

- $N \le 5\,000$

- شما میتوانید حداکثر $M = 65\,000$ بیت استفاده کنید.

زیرمساله ۲ [۹ نمره]

- $N \leq 100\,000$.
- شما میتوانید حداکثر $M = 2\,000\,000$ بیت استفاده کنید.

زیرمساله ۳ [۹ نمره]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- شما میتوانید حداکثر $M = 1\,500\,000$ بیت استفاده کنید.

زیرمساله ۴ [۳۵ نمره]

- $N \leq 5\,000$.
- شما میتوانید حداکثر $M = 10\,000$ بیت استفاده کنید.

زیرمساله ۵ [حداکثر ۳۹ نمره]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- شما میتوانید حداکثر $M = 1\,800\,000$ بیت استفاده کنید.

امتیاز این زیرمساله به طول رشته راهنمای ارسالی شما (R) بستگی دارد. به طور دقیق تر، اگر R_{max} حداکثر طول رشته راهنمای تولید شده توسط رویه `ComputeAdvice` شما بین همه نمونه تست ها (test case) باشد، نمره شما بدین صورت خواهد بود:

- 39 نمره اگر $R_{max} \leq 200\,000$؛
- $39\ (1\,800\,000 - R_{max})\ /\ 1\,600\,000$ نمره اگر $200\,000 < R_{max} < 1\,800\,000$؛
- 0 نمره اگر $R_{max} \geq 1\,800\,000$.

جزییات پیاده سازی

شما باید دقیقا دو فایل *با زبان برنامه نویسی یکسان* ارسال کنید.

اولین فایل `advisor.c`، `advisor.cpp` و یا `advisor.pas` نام دارد. این فایل باید رویه `ComputeAdvice` را همان گونه که توضیح داده شد پیاده سازی کند و میتواند رویه `WriteAdvice` را فراخوانی کند. فایل دوم، `assistant.cpp` `assistant.c` یا `assistant.pas` نام دارد. این فایل میبایست رویه `Assist` را به همان صورت که توضیح داده شد پیاده سازی کرده، میتواند رویه های `GetRequest` و `PutBack` را فراخوانی کند.

قالب همه رویه ها در زیر آمده است.

برنامه های C/C++

```
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();
```

برنامه های پاسکال

```
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;
```

این رویه ها باید به همان صورت که گفته شد رفتار کنند. البته شما مختارید رویه های دیگری برای استفاده داخلی بنویسید. در برنامه های C/C++، رویه های داخلی شما باید به صورت `static` تعریف شوند، چرا که سیستم ارزیابی نمونه، آنها را به یکدیگر ملحق (link) خواهد نمود؛ یا به جای این فقط کافی است از به کار بردن اسم یکسان برای دو رویه (یک رویه در هر یک از دو برنامه) خودداری کنید. ارسالهای شما نباید به هیچ صورت با ورودی/خروجی استاندارد یا هیچ فایل دیگری تعامل داشته باشند.

هنگام پیاده سازی راه حلتان، لازم است به دستورالعملهای زیر هم دقت داشته باشید (قالبهایی که برایتان در محیط مسابقه قرار دارند از قبل شرایط فهرست شده در زیر را برآورده میکنند).

برنامه های C/C++

در ابتدا شما میبایست فایلهای `advisor.h` و `assistant.h` را به ترتیب در advisor و assistant لحاظ (include) کنید. این امر با لحاظ کردن سطر زیر در برنامه تان محقق خواهد شد:

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

یا

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

دو فایل `advisor.h` و `assistant.h` در شاخه ای درون محیط مسابقه برای شما فراهم شده اند و همچنین در واسط وب مسابقه موجود هستند. از طریق همین درگاه ها به شما کد و script مورد نیاز برای کامپایل و تست برنامه داده میشود. به طور مشخص، بعد از کپی کردن راه حل در شاخه ای که حاوی این script ها است، شما میبایست `compile_c.sh` یا `compile_cpp.sh` را (بسته به زبان برنامه نویسی) اجرا کنید.

برنامه های پاسکال

شما میبایست از کتابخانه های `advisorlib` و `assistantlib` به ترتیب در برنامه های advisor و assistant استفاده کنید. بدین منظور سطر زیر را در برنامه لحاظ کنید:

```
uses advisorlib;
```

یا

```
uses assistantlib;
```

دو فایل `advisorlib.pas` و `assistantlib.pas` در شاخه ای درون محیط مسابقه برای شما فراهم شده اند و همچنین

در واسط وب مسابقه موجود هستند. از طریق همین درگاه ها به شما کد و script مورد نیاز برای کامپایل و تست برنامه داده میشود. به طور مشخص، بعد از کپی کردن راه حل در شاخه ای که حاوی این script ها است، شما میبایست `compile_pas.sh` را اجرا کنید.

مصحح نمونه

مصحح نمونه، ورودی با قالب زیر را قبول خواهد کرد:

- سطر ۱: اعداد N, K, M
- سطرهای ۲ تا ۱ + N: عدد C[i].

مصحح ابتدا رویه `ComputeAdvice` را اجرا میکند. این امر فایل `advice.txt` را ایجاد میکند که شامل بیتهای مجزای رشته راهنما که با فاصله از هم جدا شده و با عدد ۲ خاتمه یافته اند خواهد بود.

سپس رویه `Assist` را اجرا خواهد کرد، و خروجی که هر سطر آن به صورت "`R [number]`" یا "`P [number]`" است ایجاد میکند. سطرهای نوع اول مشخص کننده فراخوانی `GetRequest()` و مقدار بازگشتی هستند. سطرهای نوع دوم مشخص کننده `PutBack()` و رنگهای انتخاب شده برای برگرداندن به قفسه هستند. خروجی با یک سطر به صورت "`E`" پایان مییابد.

لطفا توجه کنید که در سیستم نمره دهی اصلی، زمان اجرای برنامه شما ممکن است کمی با رایانه خودتان تفاوت کند. این تفاوت محسوس نخواهد بود، با این وجود بهتر است از واسط تست جهت کنترل زمان اجرای برنامه تان استفاده کنید.

محدودیتهای زمان و حافظه

- محدودیت زمان: ۷ ثانیه.
- محدودیت حافظه: 256MiB.