Het Laatste Avondmaal

Leonardo is lang bezig geweest met het maken van zijn meest bekende muurschildering: *Het Laatste Avondmaal*. Eén van de eerste dingen die hij 's ochtends moest doen, was het selecteren van de verfkleuren die hij de rest van die dag nodig had. Hij gebruikte altijd veel kleuren maar maar er konden maar een beperkt aantal kleuren verf aanwezig zijn op de steiger waarop hij aan het schilderij werkte. De leerling die hem hielp moest onder andere steeds de steiger opklimmen om hem de kleuren verf te brengen en zonodig kleuren verf naar beneden te halen. Deze kleuren verf die beneden stonden werden bewaard op een tafel.

In deze opdracht moet je twee verschillende programma's schrijven om de leerling te ondersteunen. Het eerste programma krijgt de instructies van Leonardo (een reeks van kleuren die Leonardo gedurende de dag nodig heeft) en maakt een *korte* string van bits die *advice* genoemd wordt. Gedurende de dag verwerkt de assistent de verzoeken van Leonardo, maar hij heeft geen toegang tot de verzoeken in de toekomst, alleen tot het advice van je eerste programma. Het tweede programma krijgt dit advice, en ontvangt en verwerkt vervolgens de verzoeken van Leonardo op een online manier (dus één voor één). Dit programma moet het advice begrijpen en dit gebruiken om optimale keuzes te maken. Hieronder staat alles nader beschreven.

Transport tussen steiger en de tafel

We beschouwen een eenvoudige situatie. Stel dat er N kleuren zijn genummerd van 0 tot en met N - 1 en dat iedere dag Leonardo zijn leerling precies N keer om een nieuwe kleur vroeg. Laat C de reeks van de N kleuren zijn die Leonardo voor die dag heeft verzocht. We mogen veronderstellen dat C een reeks is van N getallen van 0 tot en met N - 1. Merk op dat sommige kleuren in C niet hoeven voor te komen en sommige kleuren meerdere malen.

De steiger staat altijd vol met verf en bevat precies K van de N kleuren met $K < N$. Aan het begin van de dag staan alleen de kleuren 0 tot en met K - 1 op de steiger.

De leerling voert steeds één klusje van Leonardo tegelijkertijd uit. Wanneer een bepaalde kleur op de steiger aanwezig is kan de leerling uitrusten. Anders moet hij de gevraagde kleur verf van de tafel halen en naar boven brengen. Vanzelfsprekend is er geen ruimte om die erbij te zetten en neemt hij een van de aanwezige kleuren verf mee naar beneden en om die vervolgens weer op de tafel te zetten.

Leonardo's optimale strategie

De leerling wil zo vaak mogelijk uitrusten. Het aantal keren dat hij kan blijven uitrusten hangt af van de keuzes die hij telkens moet maken. Preciezer gezegd, elke keer dat de leerling een kleur van de steiger haalt, kan een andere keuze leiden tot een ander resultaat in de toekomst. Leonardo legt

hem uit hoe hij zijn doel kan bereiken als hij C kent. De beste keuze voor een kleur die moet worden weggehaald kan worden gekregen door de kleuren die nu op de steiger staan te onderzoeken, en tegelijk ook de kleurverzoeken die er nog over zijn in C. Van de kleuren op de steiger zou er één gekozen moeten worden op basis van de volgende regels:

- Als er een kleur op de steiger staat die niet meer nodig is in de toekomst dient de leerling zo'n kleur van de steiger te halen.
- Anders dient de kleur van de steiger te worden gehaald waarvan het gebruik pas zo ver mogelijk in de toekomst ligt. (Dat wil zeggen, hij bekijkt voor alle kleuren op de steijger wanneer ze weer het eerst nodig zijn. De kleur die wordt weggehaald is degene die het laatst weer zal worden gebruikt.)

Het kan worden bewezen dat de leerling op basis van deze strategie van Leonardo zo vaak mogelijk kan uitrusten.

Voorbeeld 1

Neem bijvoorbeeld N = 4, we hebben dus 4 kleuren genummerd van 0 tot en met 3 en we hebben 4 verzoeken. De reeks C = (2, 0, 3, 0). We nemen aan dat K = 2. Leonardo kan maar steeds 2 kleuren verf tegelijkertijd op de steiger hebben. Zoals beschreven staan de kleuren 0 en 1 op de steiger. Dit vatten we samen als: [0, 1]. Eén van de mogelijke oplossingen is de volgende.

De eerste gevraagde kleur (nummer 2) is niet op de steiger. De leerling brengt die naar boven en haalt kleur 1 van de steiger weg. De steiger bevat dan: [0, 2].

- De volgende gevraagde kleur (nummer 0) is aanwezig op de steiger dus kan de leerling uitrusten.

- Voor het derde verzoek (nummer 3) haalt de leerling kleur 0 weg. De steiger bevat nu: [3, 2].

- Tenslotte wordt de kleur 0 gevraagd en wordt naar boven gebracht. De leerling neem kleur 2 mee en de steiger bevat nu: [3, 0].

Merk op dat in het voorbeeld de leerling niet Leonardo's ideale oplossing heeft gebruikt. De optimale strategie wordt bereikt door kleur 2 weg te halen in het derde verzoek zodat hij bij het laatste verzoek kon blijven rusten.

De leerling's strategie indien zijn geheugen is gelimiteerd

De leerling vraagt 's ochtends aan Leonardo om C op te schrijven op een stuk papier zodat hij een optimale strategie kan toepassen. Helaas voor de leerling wil Leonardo zijn manier van werken geheim houden en weigert hij de leerling het stuk papier te geven. Hij staat alleen toe dat de leerling C mag lezen en dat hij het mag proberen te onthouden.

De leerling heeft helaas een slecht geheugen. Hij kan maximaal M bits onthouden. In het algemeen kan hij daardoor niet de hele reeks C reconstrueren. Daarom moet de leerling een slimme manier bedenken voor de bits die hij wil onthouden. We noemen deze reeks de *advice reeks*. Deze wordt aangegeven met A.

Voorbeeld 2

's Morgens kan de leerling het papiertje van Leonard met de reeks C pakken, lezen en de nodige keuzes maken. Wat hij zou kunnen doen is dat hij de toestand van de steiger omnderzoekt na elk van de verzoeken. Bijvoorbeeld wanneer hij de sub-optimale strategie van voorbeeld 1 gebruikt, is dat [0, 2], [0, 2], [3, 2], [3, 0]. (Merk op dat hij weet dat de eerste toestand van de steiger [0, 1] is.)

Stel dat we hebben M = 16, dus kan hij 16 bits informatie onthouden. Als N = 4 kunnen we elke kleur opslaan in 2 bits. Dus 16 bits zijn onvoldoende om de reeks van steigertoestanden te onthouden. Daarom berekent de leerling de volgende advice reeks: A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`).

Later die dag kan de leerling deze advice reeks decoderen en op basis hiervan zijn keuzes maken.

(Uiteraard kan de leerling met M = 16 ook de gehele reeks C coderen en onthouden door slechts 8 van de 16 bits te gebruiken. In dit voorbeeld willen we alleen aangeven dat hij meerdere mogelijkheden heeft, zonder hiermee een goede oplossing van het probleem weg te geven.)

Opdracht

Jij moet *twee aparte programma's* schrijven in dezelfde programmeertaal. Deze programma's worden achter elkaar uitgevoerd zonder dat deze met elkaar kunnen communiceren gedurende het uitvoeren van deze programma's.

Het eerste programma *advisor* is degene die de leerling 's ochtends gebruikt. Dit programma krijgt als invoer de reeks C en berekent de advice reeks A.

Het tweede programma is het programma dat de leerling later die dag gebruikt. Dit programma krijgt de advice reeks A en moet hierna de reeks C met verzoeken van Leonardo verwerken. Let op dat deze reeks C verzoek per verzoek wordt kenbaar gemaakt aan je programma, en dat elk verzoek verwerkt moet worden voordat het volgende gegeven wordt.

Beter gezegd: in het eerste programma moet je de routine `ComputeAdvice`(C, N, K, M) schrijven die als input heeft de reeks C van N integers (van 0 tot en met N - 1), het getal K van het aantal kleuren op de steiger en het aantal bits M dat beschikbaar is voor het advice. Dit programma moet de advice reeks A berekenen dat maximaal uit M bits bestaat. Het programma moet vervolgens de reeks A terug communiceren door bit voor bit de volgende routine aan te roepen:

- `WriteAdvice`(B) — voegt het bit B toe aan de al eerder doorgegeven advice reeks A. (Je mag deze routine maximaal M keer aanroepen.)

In *assistent* moet je de tweede routine `Assist`(A, N, K, R) implementeren. De input voor deze routine is de advice reeks A, de integer N en K zoals hierboven beschreven en de lengte R van de reeks A in bits ($R \leq M$). Deze routine moet jouw strategie uitvoeren die de leerling moet volgen. De volgende routines moeten in advisor gebruikt worden:

- `GetRequest`() — retourneert het volgende verzoek van Leonardo. (De rest van de verzoeken krijg je pas later.)

- `PutBack`(T) — haalt de kleur T van de steiger naar de tafel. Je mag alleen een kleur van de steiger afhalen die er daadwerkelijk op staat.

Wanneer deze routine uitgevoerd wordt, moet `Assist` `GetRequest` precies N keer aanroepen. Bij iedere aanroep krijg je dan een het volgende verzoek van Leonardo. Na iedere aanroep van `GetRequest` voor een kleur die *niet* op de steiger staat, *moet* jij daarna de routine `PutBack`(T) aanroepen met jouw keuze van de kleur T. Anders mag je `PutBack` *niet* aanroepen. Als je je hier niet aan houdt dan is dat een fout en wordt je programma beëindigd. Denk er alsjeblieft aan dat de steiger aan het begin van de dag de kleuren vanaf 0 tot en met K - 1 bevat.

Een specifiek testgeval wordt als opgelost gezien als jouw twee routines aan alle opgelegde voorwaarden voldoen, en het aantal aanroepen van `PutBack` *precies gelijk* is aan dat van Leonardo's optimale strategie. Let op: Als er verschillende strategieën zijn die hetzelfde aantal aanroepen van `PutBack` nodig hebben, dan mag je programma een willekeurige geven. (Het is dus niet noodzakelijk om precies de strategie van Leonardo te volgen als er een strategie is die net zo goed is).

Voorbeeld 3

Verder met voorbeeld 2, stel dat er in `ComputeAdvice` door jou is berekend A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`). Om deze met het systeem te communiceren moet je de volgende reeks aanroepen doen:

`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`).

Je tweede routine `Assist` wordt dan uitgevoerd, die de bovengenoemde reeks A ontvangt en de waarden N = 4, K = 2, en R = 16. De routine `Assist` moet dan precies N = 4 aanroepen doen van `GetRequest`. Ook moet na enkele van deze aanroepen `Assist` `PutBack`(T) aanroepen met een geschikte keuze voor T.

De tabel hieronder geeft een reeks aanroepen weer die overeenkomen met de (suboptimale) keuzes van Voorbeeld 1. Het streepje geeft aan dat er geen aanroep is van `PutBack`.

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

Subtask 1 [8 punten]

- $N \leq 5\,000$.

- Je kunt maximaal M = 65 000 bits gebruiken.

Subtask 2 [9 punten]

- $N \leq 100\,000$.
- Je kunt hoogstens M = 2 000 000 bits gebruiken.

Subtask 3 [9 punten]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Je kunt hoogstens M = 1 500 000 bits gebruiken.

Subtask 4 [35 punten]

- $N \leq 5\,000$.
- Je kunt hoogstens M = 10 000 bits gebruiken.

Subtask 5 [maximaal 39 punten]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Je kunt hoogstens M = 1 800 000 bits gebruiken.

De score voor deze subtask hangt af van de lengte R van het advice dat je programma communiceert. Preciezer gezegd, als R_{max} is het maximum (over alle testgevallen) van de lengte van de advice reeks die door jouw routine `ComputeAdvice` wordt geproduceerd, wordt je score:

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

Implementatie details

Je moet precies twee bestanden inzenden *in dezelfde programmeertaal*.

Het eerste bestand heet `advisor.c`, `advisor.cpp` of `advisor.pas`. Dit bestand moet de routine `ComputeAdvice` implementeren, zoals hierboven beschreven. Het kan de routine `WriteAdvice` aanroepen. Het tweede bestand heet `assistant.c`, `assistant.cpp` of

`assistant.pas`. Dit bestand moet de routine `Assist` implementeren zoals hierboven beschreven en kan de routines `GetRequest` en `PutBack` aanroepen.

De beschrijving van alle routines volgt hier.

C/C++ programma's

```
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 programma's

```
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;
```

De routines moeten zich gedragen zoals hierboven beschreven. Je mag natuurlijk zelf extra routines implementeren voor intern gebruik. Let op dat bij C/C++ je interne routines als `static` gedeclareerd moeten worden; de voorbeeld grader zal alles linken. Let erop dat je in je beide programma's geen twee routines (één in elk programma) met dezelfde naam gebruikt. Je inzending mag op geen enkele wijze interacteren met standard input/output, of met andere bestanden.

Wanneer je je oplossing schrijft moet je ook op de volgende instructies letten (de templates die je kunt vinden in de wedstrijd omgeving voldoen aan de hieronder genoemde eisen).

C/C++ programma's

Aan het begin van je oplossing moet je het bestand `advisor.h` en het bestand `assistant.h` includen, respectievelijk de *advisor* en de *assistant*. Die doe je door in je bestand de volgende regel in te voegen:

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

of

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

De beide bestanden `advisor.h` en `assistant.h` staan in een directory in je wedstrijdomgeving. Je kunt ze ook op de wedstrijd website vinden. Op dezelfde wijzen vind je ook de code en scripts om je inzending te compileren en testen. Als je je inzending in de directory met deze scripts hebt geplaatst moet je `compile_c.sh` of `compile_cpp.sh` aanroepen

(afhankelijk van de keuze van programmeertaal).

Pascal programma's

Gebruik de units `advisorlib` en `assistantlib`, voor respectievelijk de advisor en de assistant. Dit doe je door de volgende regel in je broncode te zetten:

```
uses advisorlib;
```

of

```
uses assistantlib;
```

De beide bestanden `advisorlib.pas` en `assistantlib.pas` staan in een directory in je wedstrijdomgeving. Je kunt ze ook op de wedstrijd website vinden. Op dezelfde wijzen vind je ook de code en scripts om je inzending te compileren en testen. Als je je inzending in de directory met deze scripts hebt geplaatst moet je `compile_pas.sh` aanroepen.

Voorbeeld grader

De voorbeeld grader verwacht invoer die er als volgt uitziet:

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

De grader voert eerst de routine `ComputeAdvice` uit. Dit genereert een bestand `advice.txt`, dat de individuele stukjes van de advice reeks bevat, gescheiden door spaties, en afgesloten met een 2.

Daarna wordt jouw routine `Assist` uitgevoerd, en wordt uitvoer gegenereerd waarbij elke regel of van de vorm "`R [number]`", of van de vorm "`P [number]`" is. Regels van de eerste vorm geven aanroepen van `GetRequest()` aan, met het bijhorende antwoord. Regels van het tweede type geven aanroepen van `PutBack`() aan en de gekozen kleuren om terug te plaatsen. De uitvoer wordt afgesloten met een regel die eruit ziet als "`E`".

Let op dat bij de officiële grader de runtime van je programma kan afwijken van die op je eigen computer. Dit verschil zou niet significant moeten zijn. Toch word je uitgenodigd om de test interface gebruiken om te controleren of je oplossing binnen de tijdslimiet blijft.

Tijds- en geheugenlimieten

- Tijdslimiet: 7 seconden.
- Geheugenlimiet: 256 MiB.