הסעודה האחרונה

לאונרדו מצייר את הציור "הסעודה האחרונה" על איזו תקרה מפורסמת כשהוא עומד על סולם. איתו על הסולם יש סל (scaffold) שמכיל צבעים. על הרצפה יש ארון (shelf) שמכיל גם הוא צבעים. בסל יש מקום למספר צבעים מוגבל. בכל רגע, הצבעים שלא נמצאים בסל נמצאים בארון. ללאונרדו יש עוזר (assistant) שתפקידו לטפס בסולם ולהעביר מהארון לסל צבעים שלאונרדו מבקש.

המשימה היא לכתוב שתי תוכניות מחשב נפרדות עבור העוזר של לאונרדו. התוכנית הראשונה תקבל את סדרת בקשות הצבעים של לאונרדו ותייצר *עצה* (advice), שהיא מחרוזת *קצרה* של ביטים. התוכנית השנייה היא תוכנית אינטראקטיבית שתקבל בקשות צבעים מלאונרדו ותחליט איך להגיב עליהן. על התוכנית השנייה להגיב על כל בקשה מיד, לפני שהיא תקבל את הבקשה הבאה. נתונה לה העצה שנוצרה על-ידי התוכנית הראשונה, והיא יכולה להשתמש בה כדי להחליט איך להגיב על בקשות. שימו לב שסדרת הבקשות העתידיות לא נתונה לתוכנית השנייה מראש.

העברת צבעים בין הארון והסל

יש N צבעים שממוספרים מ- 0 עד N-1. לאונרדו יבקש N בקשות. לסדרת הבקשות של לאונרדו נקרא C. הסדרה C היא למעשה סדרה של N מספרים, שכל אחד מהם בין 0 לבין N-1. שימו לב שייתכן שאותו צבע יופיע בסדרה C יותר מפעם אחת וייתכן שלא יופיע כלל.

בסל יש מקום ל- K צבעים ($K < N$). בהתחלה, הסל מכיל את הצבעים 0 עד K-1 (כולל). שאר הצבעים נמצאים בהתחלה בארון.

כשלאונרדו מבקש צבע, יש שתי אפשרויות. אם הצבע המבוקש כבר נמצא בסל, העוזר נח. אם הצבע לא בסל (אלא בארון), העוזר לוקח את הצבע מהארון, מטפס בסולם, ושם אותו בסל. כדי לפנות מקום בסל, העוזר חייב להוציא מהסל צבע אחר, שכבר נמצא בו, ולהביא אותו לארון. העוזר חייב להיענות לבקשה. הוא יכול לבחור איזה מהצבעים שבסל הוא רוצה להוציא ולשים בארון. המטרה של העוזר היא לנוח כמה שיותר פעמים.

אסטרטגיה אופטימלית

אם העוזר היה יודע מראש מה היא כל סדרת הבקשות C, היתה לו אסטרטגיה אופטימלית פשוטה, שאותה נתאר עכשיו. קיימת הוכחה שזוהי אכן אסטרטגיה שממקסמת את מספר הפעמים שהעוזר נח.

האסטרגיה: בכל פעם שהעוזר צריך לבחור איזה צבע להוציא מהסל, עליו לפעול לפי החוקים הבאים.

- אם יש בסל צבע שלאונרדו לא יבקש שוב, על העוזר להוציא צבע כזה מן הסל.
- אחרת, העוזר צריך להוציא צבע שהפעם הבאה שלאונרדו יבקש אותו היא **כמה שיותר מאוחרת** בסדרה C. (כלומר, לכל צבע בסל, נבדוק מתי הפעם הבאה שלאונרדו יבקש אותו. נבחר בצבע שלאונרדו יבקש בעוד הכי הרבה זמן)

דוגמא 1

יהי N=4. אז יש 4 צבעים (ממוספרים 0 עד 3) ו- 4 בקשות. נניח שסדרת הבקשות היא C = (2, 0, 3, 0). בנוסף, נניח שגודל הסל הוא K=2. אז הסל מכיל בהתחלה את הצבעים 0 ו- 1. נסמן זאת כך: [0, 1]. נתאר דרך לא-אופטימלית שבה העוזר יכול לענות על הבקשות:

- הצבע הראשון שלאונרדו מבקש (מספר 2) אינו בסל. העוזר שם אותו בסל ומחליט להוציא את צבע מספר 1. הסל מכיל עכשיו [2, 0].
- הצבע הבא (מספר 0) כבר בסל, אז העוזר נח.
- בתגובה לבקשת השלישית (צבע מספר 3), העוזר מוציא את צבע מספר 0. הסל מכיל עכשיו [2, 3].
- בתגובה לבקשה הרביעית והאחרונה (צבע מספר 0), העוזר מוציא את צבע מספר 2 והסל מכיל עכשיו [0, 3].

בסך-הכל העוזר נח פעם אחת. אם הוא היה משתמש באסטרטגיה האופטימלית שתיארנו קודם, הוא היה מוציא בשלב השלישי את צבע מספר 2 והיה נח גם בשלב הרביעי. בסך הכל הוא היה נח פעמיים.

אסטרטגיה עבור עוזר עם זיכרון מוגבל

קודם דיברנו על תרחיש שבו העוזר יודע מראש את כל הסדרה C. בפועל, זה אינו המצב. העוזר מסתכל בבוקר על הרשימה C. הוא היה רוצה לזכור אותה, אבל לא בטוח שיש לו מספיק זיכרון (במוח). לכן, הוא מחשב מתוך C עצה קצרה שאותה הוא יכול לזכור להמשך היום. התוכנית הראשונה שתכתבו מקבלת את הרשימה C ומחזירה את העצה.

העצה שהעוזר זוכר יכולה להיות באורך של עד M ביטים. ייתכן שזה לא מספיק כדי לשחזר את כל הסדרה המקורית C. לכן, העוזר יצטרך להבין בחוכמה מה העצה שהוא רוצה לחשב. את העצה נסמן באות A. בהמשך היום, העוזר יצטרך להשתמש בעצה A כדי שמספר הפעמים שהוא נח יהיה המקסימום האפשרי שהוא היה יכול להגיע אליו אם היה יודע את כל הסדרה C מראש.

דוגמא 2

תהי C סדרת הבקשות מדוגמא מספר 1, עם סל באותו גודל (K=2). נניח שהעוזר מסתכל בבוקר על C ומחליט, מסיבה לא ברורה, שהוא מעוניין לבצע את סדרת ההחלפות הלא אופטימלית מדוגמא מספר 1. הוא יודע, כמובן, שהסל מתחיל במצב [0,1]. הוא מחשב מה תהייה סדרת המצבים הבאים של הסל. כפי שראינו בדוגמא מספר 1, הסדרה תהייה: [0, 2], [0, 2], [3, 2], [3, 0].

אם M=16, העוזר יכול להרשות לעצמו לקודד את סדרת המצבים של הסל (אחרי המצב ההתחלתי) בעזרת ארבעה ביטים לכל מצב. מכיוון ש- N=4, אפשר לקודד כל צבע באמצעות 2 ביטים ולכן אפשר לקודד כל מצב של הסל באמצעות 4 ביטים. העוזר מחשב את העצה הבאה: A = (0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0).

ברור שהעצה הזאת מכילה את כל האינפורמציה שהעוזר יזדקק לה במהלך היום כדי לממש את האסטרטגיה הלא אופטימלית הזו.

הדוגמא שנתנו עכשיו בכלל לא פותרת את הבעייה, כי מספר הפעמים שהעוזר נח הוא רק פעם אחת, בעוד שראינו שאסטרטגיה אופטימלית היתה מאפשרת לו לנוח פעמיים. באותו אופן שבו הוא קודד את אסטרטגית ההחלפות הלא אופטימלית, הוא יכול היה לקודד אסטרטגיה אופטימלית. זה כן היה פיתרון נכון לבעייה. עם זאת, קיימים פיתרונות טבעיים יותר. למשל, העוזר היה יכול לקודד את הסדרה C עצמה (זה היה אפשרי אפילו אם היה מתקיים M=8).

תיאור הבעייה

עליכם לכתוב *שתי תוכניות נפרדות* באותה שפת תכנות. התוכניות האלה יורצו זו לאחר זו. הן לא יוכלו לתקשר אחת עם השנייה, פרט לכך שהעצה שתחושב על-ידי התוכנית הראשונה תעבור כקלט לתוכנית השנייה.

התוכנית הראשונה היא זו שבה העוזר מקבל בבוקר את הסדרה C ומחשב את העצה A.

התוכנית השנייה היא זו שאומרת לעוזר מה לעשות במהלך היום. זו תוכנית אינטראקטיבית שתקבל בהתחלה את העצה A ותקבל אחת-אחת את הבקשות מתוך הסדרה C. נזכיר שהתוכנית תקבל כל בקשה רק לאחר שהגיבה על הבקשה הקודמת.

מבחינת מימוש, בתוכנית הראשונה עליכם לממש רק את הפונקציה `ComputeAdvice`(C, N, K, M). הקלט הוא המערך C שמכיל N מספרים שלמים (כל אחד בין 0 לבין N-1), המספר K שהוא גודל הסל והמספר M שהוא מספר הביטים המקסימלי בעצה. הפונקציה צריכה לחשב את העצה A שאורכה עד M. הפונקציה אומרת למערכת מה היא הסדרה A, על-ידי כך שהיא קוראת, עבור כל ביט ב- A לפי הסדר, לפונקציה הבאה:

- `WriteAdvice`(B) - הפונקציה הזאת אומרת למערכת להוסיף את הביט B לעצה A. (ניתן לקרוא לפונקציה עד M פעמים)

בתוכנית השנייה עליכם לממש רק את הפונקציה `Assist`(A, N, K, R). הפונקציה מקבלת את העצה A, את המספרים N ו- K שהוגדרו ואת המספר R, שהוא האורך של העצה ($R \leq M$). הפונקציה צריכה לממש אסטרטגית החלפות אופטימלית, על-ידי קריאה לפונקציות הבאות:

- `GetRequest`() - הפונקציה מחזירה את הבקשה הבאה של ליאונרדו. (הפונקציה לא נותנת אינפורמציה בנוגע לבקשות עתידיות)
- `PutBack`(T) - לפונקציה הזאת צריך לקרוא רק אם לאונרדו ביקש צבע שלא נמצא כרגע בסל. הפונקציה הזאת אומרת שהעוזר שם בסל את הצבע שלאונרדו ביקש, ומחזיר לארון את הצבע T, שנמצא כרגע בסל.

הפונקציה`Assist` צריכה לקרוא לפונקציה `GetRequest` בדיוק N פעמים. אחרי כל קריאה ל-`GetRequest`, אם הצבע המבוקש *אינו* בסל, *חייבים* לקרוא ל-`PutBack`(T). אחרת, *אסור* לקרוא ל-`PutBack`.

חשוב: פתרון ייחשב לנכון רק אם מספר הפעמים שהעוזר נח *שווה בדיוק* למספר הפעמים שהוא היה נח אם הוא היה יודע מראש את C ופועל באופן אופטימלי. מותר לו לפעול בכל אסטרטגית החלפות שמשיגה מספר אופטימלי זה ולאו דווקא באסטרטגיה האופטימלית שהצגנו בתחילת תיאור הבעייה.

דוגמא 3

Continuing Example 2, assume that in `ComputeAdvice` you computed A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`). In order to communicate it to the system, you would have to make the following sequence of calls: `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)`.

Your second routine `Assist` would then be executed, receiving the above sequence A, and the values N = 4, K = 2, and R = 16. The routine `Assist` then has to perform exactly N = 4 calls to `GetRequest`. Also, after some of those requests, `Assist` will have to call `PutBack`(T) with a suitable choice of T.

The table below shows a sequence of calls that corresponds to the (sub-optimal) choices from Example 1. The hyphen denotes no call to `PutBack`.

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

Subtask 1 [8 points]

- N ≤ 5 000.
- You can use at most M = 65 000 bits.

Subtask 2 [9 points]

- N ≤ 100 000.
- You can use at most M = 2 000 000 bits.

Subtask 3 [9 points]

- N ≤ 100 000.
- K ≤ 25 000.
- You can use at most M = 1 500 000 bits.

Subtask 4 [35 points]

- N ≤ 5 000.
- You can use at most M = 10 000 bits.

Subtask 5 [up to 39 points]

- N ≤ 100 000.
- K ≤ 25 000.
- You can use at most M = 1 800 000 bits.

The score for this subtask depends on the length R of the advice your program communicates. More precisely, if R_{max} is the maximum (over all test cases) of the length of the advice sequence produced by your routine `ComputeAdvice`, your score will be:

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

Implementation details

You should submit exactly two files *in the same programming language*.

The first file is called `advisor.c`, `advisor.cpp` or `advisor.pas`. This file must implement the routine `ComputeAdvice` as described above and can call the routine `WriteAdvice`. The second file is called `assistant.c`, `assistant.cpp` or `assistant.pas`. This file must implement the routine `Assist` as described above and can call the routines `GetRequest` and `PutBack`.

The signatures for all the routines follow.

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;
```

מעבר לפונקציות האלה, כמובן שמותר לכם לממש פונקציות עזר משלכם. כדי להימנע מהתנגשויות, אסור לכם להגדיר בקבצים השונים פונקציות עזר עם אותו שם. אם, משום מה, אתם רוצים בכל זאת להגדיר פונקציות עזר עם אותו שם בשני הקבצים, עליכם להוסיף להגדרה שלהן את המילה השמורה static בתחילת הפונקציה. כמובן שאסור לתוכנית שלכם להשתמש ב- standard input/output או לתקשר דרך קבצים.

When programming your solution, you also have to take care of the following instructions (the templates you can find in your contest environment already satisfy the requirements listed below).

C/C++ programs

At the beginning of your solution, you have to include the file `advisor.h` and `assistant.h`, respectively, in the advisor and in the assistant. This is done by including in your source the line:

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

or

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

The two files `advisor.h` and `assistant.h` will be provided to you in a directory inside your contest environment and will also be offered by the contest Web interface. You will 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_c.sh` or `compile_cpp.sh` (depending on the language of your code).

Pascal programs

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`.

Sample grader

The sample grader will accept input formatted as follows:

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

The grader will first execute the routine `ComputeAdvice`. This will generate a file `advice.txt`, containing the individual bits of the advice sequence, separated by spaces and terminated by a 2.

Then it will proceed to execute your `Assist` routine, and generate output in which each line is

either of the form "`R [number]`", or of the form "`P [number]`". Lines of the first type indicate calls to `GetRequest()` and the replies received. Lines of the second type represent calls to `PutBack`() and the colors chosen to put back. The output is terminated by a line of the form "`E`".

Please note that on the official grader the running time of your program may differ slightly from the time on your local computer. This difference should not be significant. Still, you are invited to use the test interface in order to verify whether your solution runs within the time limit.

Time and Memory limits

- Time limit: 7 seconds.
- Memory limit: 256 MiB.