最後の晩餐 (Last Supper)

レオナルド (Leonardo) は，彼の最も有名な壁画である「最後の晩餐 (the Last Supper)」に取り組む際にとても活動的であった．彼の日課は，まず，その日の作業に使うテンペラ絵の具 (tempera colors) の色を決めることであった．彼は多くの色を必要としていたが，限られた数の絵の具しか足場に置くことができなかった．彼の助手は，他の仕事に加えて，足場に登って絵の具を彼に届けた後，絵の具を足場から降ろして作業場の適切な棚に片づけることを担当した．

この課題では，助手を助けるためにあなたは 2 つのプログラムを書くことになる．1 つ目のプログラムは，レオナルドの指示 (レオナルドが日中に必要とする絵の具の列) を受け取り，助手のために**補助** (advice) と呼ばれる**短い**ビット列を作る．日中にレオナルドの要求を処理していく間，助手はレオナルドの未来の要求を知ることはできず，あなたの 1 つ目のプログラムによって作られた補助だけを参照できる．2 つ目のプログラムは，補助を受け取り，レオナルドの要求を 1 個ずつオンラインに処理していく．このプログラムは補助の意味を読み取り，最適な選択を行うためにそれを利用しなければならない．詳細は以下に記述されている．

棚と足場の間での絵の具の移動 (Moving colors between shelf and scaffold)

単純化された状況を考える．0 から N - 1 までの番号をつけられた N 種類の絵の具があり，毎日レオナルドは助手に新しい絵の具をちょうど N 回要求する．C をレオナルドが要求した N 個の絵の具の列とする．C は 0 以上 N - 1 以下の N 個の数の列と考えることができる．C に 1 回も現れない絵の具や複数回現れる絵の具があるかもしれない．

足場は常に埋まっており，N 個の絵の具のうちのいずれか K 個が置かれる (K < N)．初め，足場には絵の具 0 から絵の具 K - 1 までが置かれている．

助手はレオナルドの要求を 1 個ずつ処理する．要求された絵の具が**既に足場にある**ときは，助手は休むことができる．そうでないときは，助手は棚から要求された絵の具を取り，足場に運ばなければならない．もちろん，新しい絵の具を置く場所が足場にはないので，助手は足場にある絵の具を 1 個選んで足場から棚に片づけなければならない．

レオナルドの最適な戦略 (Leonardo's optimal strategy)

助手は可能な限り多くの回数休みたい．助手が休める回数は彼の選択による．より正確には，助手が足場から絵の具を片づけるにあたって，異なる選択は未来に異なる結果をもたらすかもしれない．レオナルドは，助手が C を知っていたとするときの最適な戦略を教える：足場から片づける絵の具としての最もよい選択は，現在足場にある絵の具と列 C 中の残りの絵の具の要求を調べることによってわかる．足場にある絵の具のうちどれを選ぶべきかは，以下のルールによって定まる：

- 今後要求されることがない絵の具が足場にある場合，助手はそのような絵の具のうち 1

個を片づけるべきである.

- そうでない場合は，片づける絵の具は，**次に要求されるのが最も遠い未来であるようなものであるべきである** (つまり，足場にある絵の具それぞれに対して，今後初めて要求される時点を考えることができるが，それが最も遅いような絵の具が棚に片づけられるべきである).

レオナルドの戦略に従った場合，助手は可能な限り多くの回数休めることが証明できる.

例 1 (Example 1)

N = 4 とする．よって，番号 0 から番号 3 までの 4 種類の絵の具があり，4 回の要求がある．要求の列を C = (2, 0, 3, 0) とする．また，K = 2，すなわち，同時に足場に置ける絵の具は 2 個であるとする．上記の通り，最初は足場には絵の具 0 と 1 が置かれている．足場の状態を [0, 1] のように書く．助手が要求を処理する方法の 1 つは次の通りである.

- 最初に要求される絵の具 2 は足場にない．助手は絵の具 2 を置いて，絵の具 1 を足場から片づけることにする．現在の足場の状態は [0, 2] である.
- 次に要求される絵の具 0 は既に足場にあるので，助手は休める.
- 次の絵の具 3 の要求に対しては，助手は絵の具 0 を片づけることにし，足場の状態は [3, 2] となる.
- 最後の絵の具 0 の要求に対しては，助手は絵の具 2 を片づけることにし，足場の状態は [3, 0] となる.

上の例では助手はレオナルドの最適な戦略に従っていないことに注意せよ．助手の最適な戦略は 3 番目のステップで絵の具 2 を片づけ，最後のステップで休めるようにすることである.

助手の記憶が限られているときの戦略 (Assistant's strategy when his memory is limited)

朝，助手は最適な戦略をとれるよう，レオナルドに C を紙に書くことを頼む．しかし，レオナルドは作品の技術をどうしても秘密にしたいので，助手にその紙を渡さず，彼は助手に対して，C を読んで覚えようとすることのみを許した.

残念ながら，助手の記憶力はとても悪い．彼は M ビットまでを覚えることができる．一般に，彼は列 C を再構成することができないかもしれないので，覚えるビット列を決めるよい方法を思いつかなければならない．この覚えるビット列を**補助列** (advice sequence) と呼び，A と書く.

例 2 (Example 2)

朝，助手はレオナルドの列 C を書いた紙を読み，必要なすべての選択を行う．彼ができる 1 つのこととしては，各要求の後の足場の状態を調べることである．例えば，例 1 の (最適ではない) 戦略を使うと，足場の状態の列は [0, 2], [0, 2], [3, 2], [3, 0] となる (彼は最初の状態 [0, 1] を知っているので，これを書く必要はない).

ここで，M = 16 とする (助手は 16 ビットまでの情報を覚えることができる)．N = 4 なの

で，1 個の絵の具を 2 ビットを使って覚えることができる．したがって，上の足場の状態の列を覚えるのに 16 ビットで十分であり，助手は次の補助列を求める： A = (0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0).

後になって，助手はこの補助列をデコードして，彼の選択を決めるのに使うことができる．

(もちろん，M = 16 のときは 8 ビットを使って列 C の全体を覚えることもできる．この例は，よい解答を示さずに，他の方法もあることを説明しようとしたものである．)

問題文 (Statement)

あなたは同じプログラミング言語で **2 つの独立なプログラム**を書かなければならない．2 つのプログラムは互いに情報を渡すことなく順に実行される．

1 つ目のプログラムは，助手が朝に使うものである．このプログラムは，列 C が与えられたとき，補助列 A を求めなければならない．

2 つ目のプログラムは，助手が日中に使うものである．このプログラムは補助列 A を受け取り，レオナルドの要求の列 C を処理しなければならない．列 C 中の要求は 1 個ずつこのプログラムに明かされ，各要求は次の要求を受け取る前に処理しなければならない．

より正確には，1 つ目のプログラムでは，1 つのルーチン `ComputeAdvice`(C, N, K, M) を実装しなければならない．C は 0, …, N - 1 の範囲にある N 個の整数の配列，K は足場に置かれる絵の具の個数，M は補助列に使えるビット数である．このプログラムは長さ M 以下の列 A を求め，A の各ビットに対して順番に以下のルーチンを呼び出すことによってシステムに列 A を伝えなければならない：

- `WriteAdvice`(B) — 現在の補助列 A の末尾にビット B を加える (このルーチンは高々 M 回呼び出すことができる).

2 つ目のプログラムでは，1 つのルーチン `Assist`(A, N, K, R) を実装しなければならない．このルーチンへの入力は，補助列 A，上で定義された整数 N と K，補助列 A の長さ R (R ≤ M) である．このルーチンは，提供される以下のルーチンを使って，あなたの戦略を実行しなければならない：

- `GetRequest`() — レオナルドが次に要求する絵の具を返す (未来の要求についての情報は明かされない).
- `PutBack`(T) — 絵の具 T を足場から棚に片づける．T は現在足場にある絵の具のうちの 1 つでなければならない.

実行時，ルーチン `Assist` は `GetRequest` をちょうど N 回呼び出し，レオナルドの要求を順番に受け取らなければならない．`GetRequest` を呼び出すたびに，絵の具が足場に**ない**場合，T を選んで `PutBack`(T) を呼び出さ**なければならない**．そうでない場合は，`PutBack` を呼び出し**てはならない**．この条件に従わないと，エラーとみなされプログラムの実行は終了される．初め，足場には絵の具 0 から絵の具 K - 1 までが置かれていることに注意せよ．

各テストケースは，あなたの 2 つのルーチンがすべての制約を満たし，`PutBack` を呼び出した

回数の合計がレオナルドの最適な戦略と**ちょうど等しい**とき，正解とみなされる．そのような方法が複数あるときは，どの方法でもよい (すなわち，レオナルドの戦略とは異なる，同等によい戦略であってもよい).

例 3 (Example 3)

例 2 に引き続き，ComputeAdvice の中で補助列 A = (0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0) を求めたとする．この列をシステムに伝えるためには，次のようにルーチンを呼び出さなければならない： 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).

次に，2 つ目のルーチン Assist が，上の補助列 A と値 N = 4, K = 2, R = 16 を受け取って実行される．ルーチン Assist は GetRequest をちょうど N = 4 回呼び出さなければならない．また，一部の要求の後には，Assist は適切な T を選んで PutBack(T) を呼び出さなければならない.

以下の表は例 1 の (最適ではない) 選択に対応する呼び出しの列を示している．ハイフンは PutBack を呼び出さないことを示す.

GetRequest()	動作
2	PutBack(1)
0	-
3	PutBack(0)
0	PutBack(2)

小課題 1 [8 点]

- N ≤ 5 000.
- 高々 M = 65 000 ビットを使うことができる.

小課題 2 [9 点]

- N ≤ 100 000.
- 高々 M = 2 000 000 ビットを使うことができる.

小課題 3 [9 点]

- N ≤ 100 000.
- K ≤ 25 000.

- 高々 M = 1 500 000 ビットを使うことができる.

小課題 4 [35 点]

- N ≤ 5 000.
- 高々 M = 10 000 ビットを使うことができる.

小課題 5 [最大で 39 点]

- N ≤ 100 000.
- K ≤ 25 000.
- 高々 M = 1 800 000 ビットを使うことができる.

この小課題の得点はあなたのプログラムが生成した補助列の長さ R によって決まる．より正確には，R_{max} を (すべてのテストケースでの) ComputeAdvice の返した補助列の長さの最大値とすると，あなたの得点は以下のように決まる：

- R_{max} ≤ 200 000 ならば 39 点,
- 200 000 < R_{max} < 1 800 000 ならば 39 (1 800 000 - R_{max}) / 1 600 000 点,
- R_{max} ≥ 1 800 000 ならば 0 点.

実装の詳細 (Implementation details)

あなたは**同じプログラミング言語で** 2 つのファイルを提出しなければならない.

1 つ目のファイルは advisor.c, advisor.cpp または advisor.pas という名前である．このファイルはルーチン ComputeAdvice を上で説明されたように実装しなければならず，ルーチン WriteAdvice を呼び出すことができる．2 つ目のファイルは assistant.c, assistant.cpp または assistant.pas という名前である．このファイルはルーチン Assist を上で説明されたように実装しなければならず，ルーチン GetRequest と PutBack を呼び出すことができる.

ルーチンのシグネチャは以下の通りである.

C/C++ プログラム

```
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 プログラム

```
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;
```

これらのルーチンは上で説明された通りに動作しなければならない．もちろん，内部での使用のために他のルーチンを実装することは自由である．C/C++ プログラムにおいては，内部のルーチンは `static` で宣言されなければならない (採点プログラムのサンプルが共にリンクするためである)．あるいは，advisor と assistant の中に同じ名前を持つルーチンがあることを避けよ．あなたの提出は標準入力・標準出力，あるいは他のファイルといかなる方法でもやりとりしてはならない．

プログラムを書くときは，以下の指示にも従わなければならない (コンテスト環境にあるテンプレートはすでにこの条件を満たしている)．

C/C++ プログラム

あなたのプログラムの最初において，ファイル `advisor.h` と `assistant.h` を それぞれ advisor と assistant にインクルードしなければならない．これはソースコードに以下の行を入れることで実現できる：

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

あるいは

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

2 つのファイル `advisor.h` と `assistant.h` は，コンテスト環境のディレクトリの中およびコンテストのウェブインターフェースにおいて提供される．あなたの解答のコンパイルやテストのためのコードやスクリプトも同じ場所で提供される．具体的には，あなたの解答をスクリプトと同じディレクトリの中に入れ，`compile_c.sh` または `compile_cpp.sh` のいずれか (プログラミング言語による) を実行する．

Pascal プログラム

あなたはユニット `advisorlib` と `assistantlib` を，それぞれ advisor と assistant 中で使わなければならない．これはソースコードに以下の行を入れることで実現できる：

```
uses advisorlib;
```

あるいは

```
uses assistantlib;
```

2 つのファイル advisorlib.pas と assistantlib.pas は，コンテスト環境のディレクトリの中およびコンテストのウェブインターフェースにおいて提供される．あなたの解答のコンパイルやテストのためのコードやスクリプトも同じ場所で提供される．具体的には，あなたの解答をスクリプトと同じディレクトリの中に入れ，compile_pas.sh を実行する．

採点プログラムのサンプル (Sample graders)

採点プログラムのサンプルは以下の書式の入力を読み込む：

- 1 行目： N, K, M.
- 2 行目から N + 1 行目まで： C[i].

採点プログラムは最初にルーチン ComputeAdvice を実行し，ファイル advice.txt を生成する．このファイルには補助列の各ビットが空白を区切りとして書かれており，2 で終了する．

次に，ルーチン Assist を実行し，"R [number]" または "P [number]" と書かれた行からなる出力を生成する．前者は GetRequest() が呼び出されたことと受け取った返り値を示す．後者は PutBack() が呼び出されたことと，片づけるために選ばれた絵の具を示す．出力は "E" と書かれた行で終わる．

公式の採点プログラムにおいては，プログラムの実行時間はあなたのローカルのコンピュータでの実行時間と少し異なるかもしれないことに注意せよ．この差は非常に大きくはならない．しかし，あなたの解答が実行時間制限内で実行されるかどうかを確認するために，テストインターフェースを使うことが推奨される．

時間とメモリの制限 (Time and Memory limits)

- 時間制限： 7 秒.
- メモリ制限： 256 MiB.