メッセージ (Message)

Aisha と Basma は友人であり，連絡を取り合っている． Aisha は長さ S のビット列 (0 と 1 からなる列) で表されるメッセージ M を Basma に伝えようとしている． Aisha は **パケット** を作り Basma に送ることで Basma とコミュニケーションを取る． パケットは長さ 31 のビット列であり，パケットの各ビットには 0 番から 30 番までの番号がついている． Aisha はいくつかのパケットを Basma に送ることで，メッセージ M を Basma に伝える．

しかし，Cleopatra が Aisha と Basma の間の通信を盗聴しており，Cleopatra は Aisha が送ったパケットを **改ざん** することができる． Cleopatra は改ざんによって，各パケットについて特定の 15 個のビットを変更することができる． より正確には，長さ 31 のビット列 C があり，Cleopatra は以下のように改ざんを行う．

- $C[i] = 1$ のとき，Cleopatra はパケットの i 番のビットを変更することができる．このようなビットを，**Cleopatra にコントロールされたビット** と呼ぶことにする．
- $C[i] = 0$ のとき，Cleopatra はパケットの i 番のビットを変更することができない．

ビット列 C は 15 個の 1 と 16 個の 0 からなる． メッセージ M を送る間，すべてのパケットで，Cleopatra にコントロールされたビットの位置は同じである． Aisha は Cleopatra にコントロールされた 15 個のビットの位置を正確に知っている． 一方，Basma は Cleopatra にコントロールされたビットが 15 個であることは知っているが，それらの位置は知らない．

A を Aisha が送ったパケットとする (これを **元々のパケット** と呼ぶ)． B を Basma が受け取るパケットとする (これを **改ざんされたパケット** と呼ぶ)． $0 \le i < 31$ を満たす各 i について，

- Cleopatra が i 番のビットをコントロールしていないならば ($C[i] = 0$)， Basma は Aisha が送った i 番のビットをそのまま受け取り ($B[i] = A[i]$)，
- Cleopatra が i 番のビットをコントロールしているならば ($C[i] = 1$)， $B[i]$ の値は Cleopatra が決定する．

Aisha がパケットを Basma に送るたび，Aisha はそのパケットがどのように改ざんされたかを知ることができる．

Aisha が全てのパケットを送り終わった後，Basma は改ざんされたパケットを **送られた順に** すべて受け取り，元々のメッセージ M を復元しなければならない．

あなたの課題は，Aisha が Basma にパケットを送り，Basma が改ざんされたパケットからメッセージ M を復元するための 2 人の戦略を考え，実装することである． 特に，2 つの関数を実装しなければならない． 1 つ目の関数は Aisha の戦略を実装したものであり，メッセージ M とビット列 C を受け取っ

て，Basma にいくつかのパケットを送信しなければならない． 2 つ目の関数は Basma の戦略を実装したものであり，改ざんされたパケットを受け取って，元々のメッセージ M を復元しなければならない．

実装の詳細

あなたが実装する必要のある 1 つ目の関数は以下の通りである：

```
void send_message(std::vector<bool> M, std::vector<bool> C)
```

- M: 長さ S のビット列で，Aisha が Basma に送りたいメッセージを表す．
- C: 長さ 31 のビット列で，Cleopatra によってコントロールされたビットの位置を表す．
- この関数は，1 つのテストケースで **最大 2100 回** 呼ばれる可能性がある．

この関数内でパケットを送信するためには，以下の関数を呼び出さなければならない．

```
std::vector<bool> send_packet(std::vector<bool> A)
```

- A: 長さ 31 のビット列であり，Aisha によって送られた元々のパケットを表す．
- この関数の戻り値は改ざんされたパケット B であり，Basma が受け取るパケットを表す．
- `send_message` の各実行において，この関数を 100 回まで呼び出すことができる．

あなたが実装する必要のある 2 つ目の関数は以下の通りである：

```
std::vector<bool> receive_message(std::vector<std::vector<bool>> R)
```

- R: 改ざんされたパケットの列であり，1 回の `send_message` の実行で Aisha から送られて改ざんされたパケットが，**送られた順に** 並んでいる． R の各要素は長さ 31 のビット列であり，改ざんされたパケットを表す．
- この関数は，元々のメッセージ M に等しい長さ S のビット列を返さなければならない．
- この関数は，1 つのテストケースで **複数回** 呼ばれる可能性がある．特に，この関数は `send_message` の各呼び出しに対して **ちょうど** 1 **回** 呼び出される． **関数** `receive_message` **の呼び出しの順序** は，対応する `send_message` の呼び出しの順序と同じであるとは限らない．

採点システムでは，`send_message` と `receive_message` は **別々のプログラム** で呼び出されることに注意せよ．

制約

- $1 \leq S \leq 1024$
- C は長さ 31 のビット列で，16 個の 0 と 15 個の 1 からなる．

小課題と採点方法

いずれかのテストケースで send_packet の呼び出しが上記のルールに従っていなかった場合，または，いずれかの receive_message の呼び出しの戻り値が間違っていた場合，そのテストケースに対するあなたの提出の得点は 0 となる．

それ以外の場合，すべてのテストケースのすべての send_message の呼び出しにおける send_packet の呼び出し回数の最大値を Q とし， X を

- $Q \leq 66$ ならば $X = 1$，
- $66 < Q \leq 100$ ならば $X = 0.95^{Q-66}$，

と定義したとき，得点は以下のように計算される：

小課題	得点	追加の制約
1	$10 \times X$	$S \leq 64$
2	$90 \times X$	追加の制約はない．

採点プログラムは **適応的 (adaptive)** かもしれないことに注意せよ． 採点プログラムが適応的であるとは，send_packet の戻り値が与えられた引数だけでなく，入力や過去の send_packet の呼び出しにおける引数と戻り値，採点プログラムの疑似乱数などによって変化するかもしれないことを意味する．

採点プログラムは以下の意味で **決定的** である：異なる実行で同じようにパケットを送信したとき，採点プログラムは同じように改ざんを行う．

例

以下の呼び出しを考える．

```
send_message([0, 1, 1, 0],
             [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
              1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

Aisha が Basma に送ろうとしているメッセージは $[0, 1, 1, 0]$ である． Cleopatra はパケットの 16 番から 30 番までのビットを変更することができるが， 0 番から 15 番までのビットは変更することができない．

説明のため，この例では，Cleopatra がコントロールしているビットに交互に 0 と 1 を書き込む場合を考える． すなわち，Cleopatra はコントロールしている最初のビット (この例では 16 番のビット) に 0 を書き込み， コントロールしている次のビット (17 番のビット) に 1 を書き込み， コントロールしている次のビット (18 番のビット) に 0 を書き込み，$\cdots$ と行動する．

Aisha は，例えば，以下のように元のメッセージの 2 つのビットを 1 個のパケットに詰めて送ることができる： Aisha のコントロールする最初の 8 個のビットに送りたい 1 つ目のビットを書き込み，Aisha のコントロールする残りの 8 個のビットに送りたい 2 つ目のビットを書き込んで送る．

このとき，Aisha は以下のパケットを送る：

```
send_packet([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Aisha は Cleopatra がコントロールする後ろの 15 個のビットに任意の値を書き込むことができるが，これらのビットは Cleopatra によって書き換えられるかもしれないことに注意せよ． 上で仮定した Cleopatra の行動に基づくと，この関数は次の値を返す：
$[0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1]$

同様にして，Aisha が，メッセージの残る 2 つのビットをパケットに詰めて送ると，2 つ目のパケットは以下のようになる．

```
send_packet([1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

上で仮定した Cleopatra の行動に基づくと，この関数は次の値を返す：
$[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$

Aisha はさらにパケットを送ることもできるが，そうしなかった場合を考える．

その場合，採点プログラムは以下の呼び出しを行う：

```
receive_message([[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                  0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0],
                 [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                  0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]])
```

Basma は，以下のようにしてメッセージ M を復元することができる． 各パケットから，2 連続で同じ値が現れる最初の位置と最後の位置を見つけ，それぞれから 1 ビットずつ取り出す． このとき，Basma は 1 つ目のパケットから $[0,1]$ を取り出し，2 つ目のパケットから $[1,0]$ を取り出す． これらを繋げて，メッセージとして $[0,1,1,0]$ を復元すると，この `receive_message` の呼び出しの正しい戻り値となる．

上で仮定した Cleopatra の行動に基づくと，メッセージの長さが 4 であるとき，この方法によって Basma が C の値にかかわらず正しく M を復元できることを証明できる．

しかし，一般には正しく復元できない．

採点プログラムのサンプル

採点プログラムのサンプルは適応的 (adaptive) ではない．

その代わりに Cleopatra は，上の例で説明したように，自身がコントロールするビットに交互に 0 と 1 を書き込む．

入力形式：**入力の 1 行目には，シナリオの数を表す整数 T を与える．** その後，T 個のシナリオについて，以下の形式で入力を与える．

```
S
M[0]  M[1]  ...  M[S-1]
C[0]  C[1]  ...  C[30]
```

出力形式： 採点プログラムのサンプルは，T 個のシナリオそれぞれについて，以下の形式で出力する．

```
K L
D[0]  D[1]  ...  D[L-1]
```

ここで，K は `send_packet` の呼び出し回数，D は `receive_message` が返したメッセージ，L はその長さを表す．