消息篡改者（message）

Aisha 和 Basma 是两位互相通信的朋友。Aisha 有一条消息 M 想要发给 Basma，该消息是由 S 个比特（即若干 0 或 1）组成的序列。Aisha 通过发送**数据包**来跟 Basma 通信。一个数据包是由 31 个比特组成的序列，对应的位置从 0 到 30 编号。Aisha 想向 Basma 发送消息 M 时，会发送若干数据包。

然而第三个人 Cleopatra 在破坏 Aisha 和 Basma 之间的通信，能够**篡改**发送的数据包。在每个数据包中，Cleopatra 可以修改恰好 15 个位置的比特。具体来说，给定长度为 31 的数组 C，其中每个元素为 0 或 1，含义如下：

- $C[i] = 1$ 表示位置为 i 的比特可以被 Cleopatra 修改。我们称此类位置是被 Cleopatra **控制**的。
- $C[i] = 0$ 表示位置为 i 的比特不能被 Cleopatra 修改。

数组 C 恰好包含 15 个 1 和 16 个 0。当发送消息 M 时，Cleopatra 控制的位置集合对于所有数据包都是相同的。Aisha 清楚地知道哪 15 个位置被 Cleopatra 控制。Basma 只知道有 15 个位置被 Cleopatra 控制，但不知道是哪些位置。

令 A 为 Aisha 决定要发送的数据包（称之为**原始数据包**）。令 B 为 Basma 收到的数据包（称之为**篡改数据包**）。对每个在 $0 \leq i < 31$ 的 i 都有：

- 如果 Cleopatra 不能控制位置为 i 的比特（$C[i] = 0$），那么 Basma 将能收到 Aisha 发送的第 i 个比特（$B[i] = A[i]$），
- 否则，如果 Cleopatra 控制了位置为 i 的比特（$C[i] = 1$），那么 $B[i]$ 的值由 Cleopatra 决定。

每个数据包发送后，Aisha 会立刻知道被篡改后的数据包内容。

当 Aisha 发送完所有数据包后，Basma **按照发送顺序**接收到所有被篡改的数据包，她必须重建原始消息 M。

你的任务是制定并实现某种策略，使得 Aisha 给 Basma 发送消息 M 时，Basma 能够从篡改数据包中恢复 M。具体来说，你要实现两个函数，第一个函数进行 Aisha 的动作：给定消息 M 和数组 C，给 Basma 发送若干数据包来传输消息。第二个函数进行 Basma 的动作：给定若干篡改数据包，恢复原始消息 M。

实现细节

你要实现的第一个函数是：

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

- M：长度为 S 的数组，描述 Aisha 想要发给 Basma 的消息。
- C：长度为 31 的数组，标记 Cleopatra 控制的数据包中的位置。
- 每个测试用例中，该函数**最多**可被调用**2100次**。

该函数调用以下函数来发送数据包：

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

- A：原始数据包（长度为 31 的数组），表示 Aisha 发送的比特。
- 此函数返回篡改数据包 B，表示 Basma 接收到的比特。
- 此函数在 send_message 的一次调用过程中最多被调用 100 次。

你要实现的第二个函数是：

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

- R：描述若干篡改数据包的数组。这些数据包源自 Aisha 在一次 send_message 调用时发送的若干数据包，且按照 Aisha 的发送顺序排列。R 的每个元素是长度为 31 的数组，表示一个篡改数据包。
- 该函数应返回包含 S 个比特的数组，且与原始消息 M 相同。
- 每个测试用例中，该函数可能被调用**多次**。对于每次 send_message 的调用，对应地该函数要有**恰好一次**调用。函数 receive_message 的**调用顺序**不必与对应的 send_message 调用顺序一致。

注意在评测系统中，send_message 和 receive_message 两个函数是在**不同的程序**中来调用的。

约束条件

- $1 \leq S \leq 1024$
- C 恰好有 31 个元素，且其中 16 个为 0，15 个为 1。

子任务与评分

如果在任意的测试用例中，函数 send_packet 的调用不符合上述规则，或者某个函数 receive_message 的调用的返回值不正确，你的解答在该测试用例上得 0 分。

否则，令 Q 为所有测试用例中，每次 send_message 调用时调用函数 send_packet 的次数的最大值。令 X 等于：

- 1，如果 $Q \leq 66$
- 0.95^{Q-66}，如果 $66 < Q \leq 100$
- 0，如果 $100 < Q$

那么，得分将由以下式子计算获得：

子任务	分数	额外的约束条件
1	$10 \cdot X$	$S \leq 64$
2	$90 \cdot X$	没有额外的约束条件。

注意在某些测试用例中，评测程序的行为是**自适应的**。这意味着 send_packet 的返回值可能取决于它的输入参数和以前调用该函数的返回值。

例子

考虑以下调用。

```
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 试图发给 Basma 的消息是 $[0, 1, 1, 0]$。数据包的第 0 至第 15 个比特不能被 Cleopatra 修改，而第 16 至第 30 个比特可以被 Cleopatra 修改。

为便于解释这个例子，我们假设 Cleopatra 的行为是确定性的：她交替地用 0 和 1 填充所控制的比特，也就是她把她控制的第一个位置赋 0（例子中的第 16 位），把她控制的第二个位置赋 1（第 17 位），把她控制的第三个位置赋 0（第 18 位），以此类推。

Aisha 可以做出的一种决定是在一个数据包中发送原始消息中的两个比特，例如她是这样做的：通过她控制的前 8 个位置来发送第一个比特，通过她控制的接下来 8 个位置来发送第二个比特。

于是 Aisha 发送以下数据包：

```
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, 0])
```

由于 Cleopatra 可以更改最后 15 个比特，所以 Aisha 决定随意设置它们，因为它们可能会被覆盖。使用前面假定的 Cleopatra 的策略，该函数返回：

$[0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$。

Aisha 决定在第二个数据包中发送 M 的最后两个比特，与之前类似：

```
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, 0])
```

根据假定的 Cleopatra 的策略，该函数返回：

$[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$。

Aisha 还可以发送更多的数据包，但她没有这样做。

然后评测程序进行以下函数调用：

```
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 按照如下方式恢复消息 M。她从每个数据包中提取出第一个连续出现两次的比特，以及最后一个连续出现两次的比特。也就是说，她从第一个数据包提取出两个比特 $[0,1]$，从第二个数据包中提取出两个比特 $[1,0]$。把它们放在一起，她恢复了消息 $[0,1,1,0]$，这是对 `receive_message` 调用的正确返回值。

可以证明，在假设的 Cleopatra 的策略下，对于长度为 4 的消息，不管 C 的值是多少，Basma 这样做能够正确恢复 M。然而，一般情况下这并不正确。

评测程序示例

评测程序示例不具备自适应性，Cleopatra 的行为是确定性的，她交替地用 0 和 1 来填充她控制的比特，就像她在例子中所做的那样。

输入格式：**输入第一行包含一个整数 T，指定测试用例的数量。**接下来有 T 组测试用例，每组测试用例都按以下格式描述：

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

输出格式：评测程序示例按照输入的顺序，用以下格式输出 T 组测试用例的结果：

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

这里， K 是 `send_packet` 的调用次数，D 是 `receive_message` 返回的消息，L 是它的长度。