Alice, Bob and Permutation

Alice and Bob are using computers for communication. Alice has a piece of data in her computer and she wants to send it to Bob.

However, things are not as simple as they imagined. Alice suddenly realized that her computer has something wrong, making it extremely difficult to send the information.

Specifically, Alice has a permutation P of length N satisfying $N \leq 20$. According to their original plan, she should send Bob $N+1$ integers:

- $N, P[0], ..., P[N-1]$

However, due to the computer malfunction, she can send **only one** 64-bit unsigned integer to Bob! This means that Alice must be meticulous and compress and encode all the information she needs to send into a 64-bit unsigned integer. Meanwhile, Bob must be able to process the information sent by Alice correctly to recover the original information that Alice originally intended to send.

This seems very difficult, fortunately Alice and Bob can seek your help. Please help each of Alice and Bob implement a program to complete the communication process they wish to achieve.

Implementation Details

You should implement the following functions:

Alice

```
unsigned long long Alice(const std::vector<int> P);
```

- In this function, you need to implement the encoding procedure on Alice's computer.
- P : The original permutation owned by Alice.
  - You can get its length by `P.size()`.
  - It is guaranteed that $P.size() \leq 20$ and each integers in $[0, P.size())$ appears exactly once in P.
  - Note that the subscripts and elements in P are both 0-indexed.
- This function should return a 64-bit unsigned integer, representing the encoding result to be passed to Bob.
- In a single test, the function will only be called exactly once.

Bob

```
std::vector<int> Bob(const unsigned long long x);
```

- In this function, you need to implement the decoding procedure on Bob's computer.
- x : The integer that Bob received from Alice.
- This function should return a permutation, representing the decoding result from the received information.
- In a single test, the function will only be called exactly once after calling the function `Alice()`.
- Your answer will be considered correct if and only if the permutation restored by Bob is exactly equal to the original one owned by Alice.

Example

Consider the following calls:

```
unsigned long long x = Alice({0, 1});
std::vector<int> p = Bob(x);
```

It represents the following scenario:

- The original permutation owned by Alice is $\{0, 1\}$ with length 2, as the parameter input for the encoding function `Alice()`.
  - A possible output for `Alice()` is 0.
- Then the above encoding result is passed to the decoding function `Bob()`.
  - The only correct output for this function is $\{0, 1\}$. Other output will be considered incorrect.

Constraints

For all test cases:

- $1 \leq N \leq 20$, where N is the length of the input permutation p of the function `Alice()`.
- The input permutation P of the function `Alice()` is valid, that means, each integer x satisfying $0 \leq x < N$ appears exactly once in P.
- `Alice()` and `Bob()` have a memory limit of 2048 MiB and a time limit of 1 seconds, respectively.

Subtasks

1. (10 points): $N \leq 8$.
2. (20 points): $N \leq 13$.

3. (30 points): $N \leq 18$.
4. (40 points): No additional constraints.

Sample Grader

The sample grader reads the input in the following format:

- Line 1: N
- Line 2: $P[0]\ P[1]\ \cdots\ P[N-1]$

The sample grader outputs in the following format:

- Line 1: a single string `Correct.` or `Incorrect answer.`
    - If the program finishes successfully and the decoding result of Bob is correct, the sample grader will output a string `Correct.` to stdout and exit.
    - Otherwise, the grader will output a string `Incorrect answer.` to stdout and exit.