C. Hashing Trees

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Sasha is taking part in a programming competition. In one of the problems she should check if some rooted trees are isomorphic or not. She has never seen this problem before, but, being an experienced participant, she guessed that she should match trees to some sequences and then compare these sequences instead of trees. Sasha wants to match each tree with a sequence $a_0, a_1, ..., a_h$, where h is the height of the tree, and a_i equals to the number of vertices that are at distance of i edges from root.

Unfortunately, this time Sasha's intuition was wrong, and there could be several trees matching the same sequence. To show it, you need to write a program that, given the sequence a_i, builds two non-isomorphic rooted trees that match that sequence, or determines that there is only one such tree.

Two rooted trees are isomorphic, if you can reenumerate the vertices of the first one in such a way, that the index of the root becomes equal the index of the root of the second tree, and these two trees become equal.

The height of a rooted tree is the maximum number of edges on a path from the root to any other vertex.

Input

The first line contains a single integer h ($2 \le h \le 10^5$) — the height of the tree.

The second line contains $h+1$ integers — the sequence $a_0, a_1, ..., a_h$ ($1 \le a_i \le 2 \cdot 10^5$). The sum of all a_i does not exceed $2 \cdot 10^5$. It is guaranteed that there is at least one tree matching this sequence.

Output

If there is only one tree matching this sequence, print "`perfect`".

Otherwise print "`ambiguous`" in the first line. In the second and in the third line print descriptions of two trees in the following format: in one line print integers, the k-th of them should be the parent of vertex k or be equal to zero, if the k-th vertex is the root.

These treese should be non-isomorphic and should match the given sequence.

Examples

input

```
2
1 1 1
```

output

```
perfect
```

input

```
2
1 2 2
```

output

```
ambiguous
0 1 1 3 3
0 1 1 3 2
```

Note

The only tree in the first example and the two printed trees from the second example are shown on the picture: