Problem A6

## Problem Statement

The permutations game is a simple solitaire game played on a rectangular board. The board is divided into N squares, arranged in a row. Each square of the board has a distinct number between 1 and N, inclusive, printed on it. Each square also has a position. The leftmost square has position 1, the rightmost square has position N, and positions are numbered consecutively from left to right.

```
+---+---+---+---+---+---+
| 3 | 2 | 4 | 1 | 6 | 5 |
+---+---+---+---+---+---+
  1   2   3   4   5   6
```

In this example for N=6 you can see the board with the numbers printed on it. Below each square is its position.

This game is played in the following way: First, the player chooses any square as his starting square and steps on it. At each step, he will look at the number printed on the square on which he is standing, and then move to the square at that position. He repeats this until he returns to his starting square.

In the board shown above, the player might start at position 3. He would see the 4 printed on that square and move to position 4. Then, he would see a 1 and move to the position 1. Finally, he would see a 3 and return to his starting square, and the game would end there. If he started at position 2, he would see the 2 printed on that square, move to position 2 (where he was already standing), and stop there. Depending on where he started, he would visit a different number of squares (3 in the first example, and 1 in the second example). The goal of the game is to select the starting square that allows you to visit the greatest number of squares possible.

The **board** will be given as a vector <int>, where the ith element is the value printed on the square at position i (i is a 1-based index). Return the maximum number of squares the player can visit if he selects his starting square optimally.

## Definition

| | |
|---|---|
| Class: | CyclesInPermutations |
| Method: | maxCycle |
| Parameters: | vector <int> |
| Returns: | int |
| Method signature: | int maxCycle(vector <int> board) |
| (be sure your method is public) | |

## Constraints

- **board** will contain between 1 and 50 elements, inclusive.
- **board** will contain exactly one occurrence of each integer between 1 and the number of elements in **board**, inclusive.

## Examples

0)

{3,2,4,1,6,5}

Returns: 3

The example from the problem statement.

1)

{1,2,3,4,5,6}

Returns: 1

No matter where the player starts, the first number he sees will always be his starting position, and therefore he will always visit just one square.

2)

{5,1,2,3,4}

Returns: 5

Each square takes you one square to the left, and the first square wraps around to the last one, so no matter where the player starts, he will always visit all the squares.

3)

{5,7,14,6,16,10,8,17,11,12,18,3,4,13,2,15,9,1,20,19}

Returns: 11