# *Demo ticket*

**demo4D3P5N-DT9**
**Started on: 2013-01-05 14:42 UTC**
**Status: closed**
**Time limit: 30 min.**

Score:
**100**
of 100

## 1. prefix_set

Given a table A of N integers from 0 to N-1 calculate the smallest such index P, that that {A[0],...,A[N-1]} = {A[0],...,A[P]}.

**score: 100 of 100**

### Task description

A non-empty zero-indexed array A consisting of N integers is given. The *first covering prefix* of array A is the smallest integer P such that 0≤P<N and such that every value that occurs in array A also occurs in sequence A[0], A[1], ..., A[P].
For example, the first covering prefix of the following 5−element array A:

```
A[0] = 2  A[1] = 2  A[2] = 1
A[3] = 0  A[4] = 1
```

is 3, because sequence [ A[0], A[1], A[2], A[3] ] equal to [2, 2, 1, 0], contains all values that occur in array A.
Write a function

```
class Solution { public int ps(int[] A); }
```

that, given a zero-indexed non-empty array A consisting of N integers, returns the first covering prefix of A.
Assume that:

- N is an integer within the range [1..1,000,000];
- each element of array A is an integer within the range [0..N−1].

For example, given array A such that

```
A[0] = 2  A[1] = 2  A[2] = 1
A[3] = 0  A[4] = 1
```

the function should return 3, as explained above.
Complexity:

- expected worst-case time complexity is O(N);
- expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.



### Solution

**Time used: 14 min.**

**Notes:** correct functionality and scalability

### Task timeline

Use the controls below to see how the code changed during the test.

14:55:18 — 14:55:53

### Code: 14:55:53 UTC, java, final, score: 100.00

```
01. import java.util.HashSet;
02. import java.util.Set;
03. class Solution {
04.     public int ps (int[] A )
05.     {
06.         Set<Integer> input = new
               HashSet<Integer>();
07.         Set<Integer> prefix = new
               HashSet<Integer>();
08.         int i;
09.
10.         for(i=0;i<A.length;i++)
11.         {
12.             input.add(A[i]);
13.         }
14.
15.         for(i=0;i<A.length; i++)
16.         {
17.             prefix.add(A[i]);
18.             if(input.size()==prefix.size())
19.             {
20.                 return i;
21.             }
22.         }
23.         return -1;
24.     }
25. }
```

### Analysis

**Detected time complexity:**
**O(N) or O(N*log(N))**

| test | time | result |
|---|---|---|
| example1<br>example test | 0.270 s. | **OK** |
| extreme_single<br>1-element sequence | 0.270 s. | **OK** |

| | | |
|---|---|---|
| extreme_constant<br>constant sequence | 0.270 s. | **OK** |
| extreme_identity<br>identity permutation | 0.260 s. | **OK** |
| extreme_permutation<br>permutation | 0.270 s. | **OK** |
| simple1<br>very simple sequence | 0.270 s. | **OK** |
| simple2<br>simple sequence | 0.260 s. | **OK** |
| binary<br>binary sequence | 0.270 s. | **OK** |
| periodic<br>periodic pattern | 0.270 s. | **OK** |
| random_dec_100<br>random test 100 elements, 37 different values. | 0.280 s. | **OK** |
| random_dec_1000<br>random test 1000 elements, 34 different values. | 0.270 s. | **OK** |
| random_dec_10000<br>random test 10 000 elements, 30 different values. | 0.280 s. | **OK** |
| random_dec_100000<br>random test 100 000 elements, 27 different values. | 0.290 s. | **OK** |
| random_sqrt_100<br>random test 100 elements, 10 different values. | 0.270 s. | **OK** |
| random_sqrt_1000<br>random test 1000 elements, 31 different values. | 0.270 s. | **OK** |
| random_sqrt_10000<br>random test 10 000 elements, 100 different values. | 0.280 s. | **OK** |
| random_sqrt_100000<br>random test 100 000 elements, 316 different values. | 0.290 s. | **OK** |
| random_n_log_100<br>random test 100 elements and n/log_2 n values. | 0.270 s. | **OK** |
| random_n_log_1000<br>random test 1000 elements and n/log_2 n values. | 0.270 s. | **OK** |
| random_n_log_10000<br>random test 10 000 elements and n/log_2 n values. | 0.280 s. | **OK** |
| random_n_log_100000<br>random test 100 000 elements and n/log_2 n values. | 0.310 s. | **OK** |
| random_n_100<br>random test 100 elements and values. | 0.260 s. | **OK** |
| random_n_1000<br>random test 1000 elements and values. | 0.270 s. | **OK** |
| random_n_10000<br>random test 10 000 elements and values. | 0.280 s. | **OK** |
| random_n_100000<br>random test 100 000 elements and values. | 0.390 s. | **OK** |