Daily Coding Problem #189

Problem

This problem was asked by Google.

Given an array of elements, return the length of the longest subarray where all its elements are distinct.

For example, given the array `[5, 1, 3, 5, 2, 3, 4, 1]`, return 5 as the longest subarray of distinct elements is `[5, 2, 3, 4, 1]`.

Solution

The brute force solution here would be to test every possible subarray for distinctness, and keep track of the longest:

```
def is_distinct(arr):
    d = {}
    for e in arr:
        if e in d:
            return False
```

```
            d[e] = True

        return True

def distinct_subarray(arr):
    max_distinct_subarray = []
    for i in range(len(arr)):
        for j in range(i + 1, len(arr) + 1):
            subarray = arr[i:j]
            if is_distinct(subarray) and len(subarray) >
len(max_distinct_subarray):
                max_distinct_subarray = subarray
    return len(max_distinct_subarray)
```

This takes $O(n^3)$ time and $O(n)$ space, since we need to get $O(n^2)$ subarrays, and then iterate over each subarray which can be up to $O(n)$ in length.

We can make this faster by keeping track of the indices of the last occurring elements as well as the running longest distinct subarray. Thus, when we look at the element at the next index, there are two cases for the longest distinct subarry ending at that index:

- If the element doesn't exist in the dictionary, then the new longest distinct subarray is the same as the previous one with the current element appended
- If it does exist in the dictionary, then the longest distinct subarray starts from `d[i] + 1` to the current index.

```
def distinct_subarray(arr):
    d = {} # most recent occurrences of each element

    result = 0
    longest_distinct_subarray_start_index = 0
    for i, e in enumerate(arr):
        if e in d:
            # If d[e] appears in the middle of the current longest distinct
```

```
subarray
            if d[e] >= longest_distinct_subarray_start_index:
                result = max(result, i - longest_distinct_subarray_start_index)
                longest_distinct_subarray_start_index = d[e] + 1
        d[e] = i

    return max(result, len(arr) - longest_distinct_subarray_start_index)
```

This runs in O(n) time and O(1) space.
