575. Distribute Candies

May 6, 2017 | 38.2K views

Average Rating: 4.21 (34 votes)

Given an integer array with **even** length, where different numbers in this array represent different **kinds** of candies. Each number means one candy of the corresponding kind. You need to distribute these candies **equally** in number to brother and sister. Return the maximum number of **kinds** of candies the sister could gain.
Example 1:

```
Input: candies = [1,1,2,2,3,3]
Output: 3
Explanation:
There are three different kinds of candies (1, 2 and 3), and two candies for each kind.
Optimal distribution: The sister has candies [1,2,3] and the brother has candies [1,2,3].
The sister has three different kinds of candies.
```

Example 2:

```
Input: candies = [1,1,2,3]
Output: 2
Explanation: For example, the sister has candies [2,3] and the brother has candies [1,1].
The sister has two different kinds of candies, the brother has only one kind of candies.
```

Note:

1. The length of the given array is in range [2, 10,000], and will be even.
2. The number in given array is in range [-100,000, 100,000].

Solution

Approach 1: Brute Force

Algorithm

The brute force approach is really simple. We can generate all the permutations of the given $nums$ array representing the candies and determine the number of unique elements in the first half of the generated array.

In order to determine the number of unique elements in the first half of the array, we put all the required elements in a set and count the number of elements in the set. We count such unique elements in the first half of the generated arrays for all the permutations possible and return the size of the largest set.

Java

```
public class Solution {
  int max_kind = 0;
  public int distributeCandies(int[] nums) {
    permute(nums, 0);
    return max_kind;
  }
  public void permute(int[] nums, int l) {
    if (l == nums.length - 1) {
      HashSet < Integer > set = new HashSet < > ();
      for (int i = 0; i < nums.length / 2; i++) {
        set.add(nums[i]);
      }
      max_kind = Math.max(max_kind, set.size());
    }
    for (int i = l; i < nums.length; i++) {
      swap(nums, i, l);
      permute(nums, l + 1);
      swap(nums, i, l);
    }
  }
  public void swap(int[] nums, int x, int y) {
    int temp = nums[x];
    nums[x] = nums[y];
    nums[y] = temp;
  }
}
```

Complexity Analysis

- Time complexity : $O(n!)$. A total of $n!$ permutations are possible for $nums$ array of size n.
- Space complexity : $O(n)$. The depth of the recursion tree can go upto n.

Approach 2: Better Brute Force

Algorithm

Before looking into the idea behind this approach, firstly we need to observe one point. The maximum no. of unique candies which the girl can obtain could be atmost $n/2$, where n refers to the number of candies. Further, in case the number of unique candies are below $n/2$, to maximize the number of unique candies that the girl will obtain, we'll assign all the unique candies to the girl. Thus, in such a case, the number of unique candies the girl gets is equal to the total number of unique candies in the given $candies$ array.

Now, let's look at the idea behind this approach. We need to find the total number of unique candies in the given $candies$ array. One way to find the number of unique candies is to traverse over the given $candies$ array. Whenever we encounter an element, say $candies[j]$, we can mark all the elements which are the same as $candies[j]$ as invalid and increment the count of unique elements by 1.

Thus, we need to do such markings for all the elements of $candies$ array. At the end, $count$ gives the required number of unique candies that can be given to the girl. Further, the value to be returned is given by: $\min(\frac{n}{2}, count)$. Instead of finding the $\min$, we can stop the traversal over the given $candies$ array as soon as the $count$ exceeds $\frac{n}{2}$.

Java

```
public class Solution {
    public int distributeCandies(int[] candies) {
        int count = 0;
        for (int i = 0; i < candies.length && count < candies.length / 2; i++) {
            if (candies[i] != Integer.MIN_VALUE) {
                count++;
                for (int j = i + 1; j < candies.length; j++) {
                    if (candies[j] == candies[i])
                        candies[j] = Integer.MIN_VALUE;
                }
            }
        }
        return count;
    }
}
```

Complexity Analysis

- Time complexity : $O(n^2)$. We traverse over all the elements of $candies$ for every new element found. In the worst case, we do so for every element of $candies$ array. n refers to the size of $candies$ array.
- Space complexity : $O(1)$. Constant space is used.

Approach 3: Using Sorting

Algorithm

We can sort the given $candies$ array and find out the elements which are unique by comparing the adjacent elements of the sorted array. For every new element found(which isn't the same as the previous element), we need to update the $count$. At the end, we can return the required result as $\min(n/2, count)$, as discussed in the previous approach.

Java

```
public class Solution {
    public int distributeCandies(int[] candies) {
        Arrays.sort(candies);
        int count = 1;
        for (int i = 1; i < candies.length && count < candies.length / 2; i++)
            if (candies[i] > candies[i - 1])
                count++;
        return count;
    }
}
```

Complexity Analysis

- Time complexity : $O(n \log n)$. Sorting takes $O(n \log n)$ time.
- Space complexity : $O(1)$. Constant space is used.

Approach 4: Using Set

Algorithm

Another way to find the number of unique elements is to traverse over all the elements of the given $candies$ array and keep on putting the elements in a set. By the property of a set, it will contain only unique elements. At the end, we can count the number of elements in the set, given by, say $count$. The value to be returned will again be given by $\min(count, n/2)$, as discussed in previous approaches. Here, n refers to the size of the $candies$ array.

Java

```
public class Solution {
    public int distributeCandies(int[] candies) {
        HashSet < Integer > set = new HashSet < > ();
        for (int candy: candies) {
            set.add(candy);
        }
        return Math.min(set.size(), candies.length / 2);
    }
}
```

Complexity Analysis

- Time complexity : $O(n)$. The entire $candies$ array is traversed only once. Here, n refers to the size of $candies$ array.
- Space complexity : $O(n)$. set will be of size n in the worst case.

Rate this article:

Previous Next

Comments: 25

Sort By

Type comment here... (Markdown is supported)

Preview Post

m-guo ★ 17 April 14, 2018 7:15 PM

Following is my python solution:
candyType = set(candies)
return min(len(candyType), len(candies)//2)

10 Share Reply

vashchyshyn ★ 6 October 9, 2018 1:58 AM

one line solution:

```
public int distributeCandies(int[] candies) {
        return Math.min((int)Arrays.stream(candies).boxed().distinct().count(), c
andies.length/2);
}
```

Read More

6 Share Reply

SHOW 1 REPLY

radistao ★ 35 April 2, 2018 4:28 AM

Also this could be improved:

1. use BitSet instead of Set (more memory consumption, but much faster)
2. finish fast when set.size() == candies.length / 2. No need to count after this condition is achieved.

3 Share Reply

leetcode_deleted_user ★ 275 January 22, 2018 5:47 PM

class Solution(object):
def distributeCandies(self, candies):
return min(len(set(candies)),int(len(candies)/2))

2 Share Reply

shenhaichilong ★ 502 August 30, 2018 8:23 AM

my solution:

```
/**
 * beats 60%
 * @param candies
 * @return
 */
```

Read More

0 Share Reply

ThisRay ★ 0 August 14, 2018 12:56 PM

return min(len(set(candies)), int(len(candies)/2))

0 Share Reply

eden_work ★ 0 March 14, 2018 7:42 AM

Approach 1: If the array length is greater than 10, the speed will be very slow，but you can learn the idea of all permutations in it

0 Share Reply

leetcode_deleted_user ★ 275 January 17, 2018 7:15 AM

class Solution {
public int distributeCandies(int[] candies) {
Map<Integer,Integer> maps = new HashMap<Integer,Integer>();
for(int i=0;i<candies.length;i++){
maps.put(candies[i],

Read More

0 Share Reply

hezhichaoleetcode ★ 1 January 14, 2018 1:49 PM

class Solution:
def distributeCandies(self, candies):
"""
:type candies: List[int]
:rtype: int

Read More

0 Share Reply

sai_mun ★ 0 December 20, 2017 12:19 PM

class Solution:
def distributeCandies(self, candies):
freq = set()
for i in candies:
freq.add(i)

Read More

0 Share Reply