CS325 - Group Assignment 1
Group: 123gogogo
Group members:
Ziwen Tong
Yu Chien, Lin
Xiaoru Chen
2020-10-12

We solved this problem in two steps:

STEP 1: Locate the position of the majority element in the input array by iterating through the input array and canceling out two elements when we see a pair of two different elements appear.

2 variables we use:
majority_index: the index of our current possible majority element. Starts from 0.
count: a number to tell how many times the current possible majority element (candidate) has not been canceled out. Starts from 0.

If the length of the array is 1 or 2, then the size of the majority party is the length of the array. Suppose that it is legal to have only one party in the convention.

for i in range(n):
    if ***count*** is 0, that means the current possible majority element has not been canceled out, and we need to start a new comparison with a new current possible majority element :
        1. Set the majority element's index number as the current index. Now the ***majority_index*** is the position of the current possible majority element.
        2. Set ***count*** = 1, ready for comparing the next 2 elements.

    else if ***same_party*** returns true:
        If ***count*** is not 0, that means we need to keep looking for whether there are different elements to cancel out the extra number(s) of the current possible majority element.
            Pick the next element and compare it with the current possible majority element by calling the ***same_party*** function. Increase the times that current possible majority element has not been canceled out by 1, ***count*** + 1;
    else:
        This means that ***count*** is not 0 and the ***same_party*** return false, so decrease the ***count*** by 1:
            Decrease the times that the current possible majority element has not been canceled out by 1, ***count*** – 1. Note that ***count*** will never be negative because when it reaches 0, we set it to 1 and ready for comparing the next 2 elements again.

After completing the iteration, the position (index number) of the majority element from the input array is the position of the last current possible majority element.

STEP 2: We count the number of the majority element from the input array.
Variable we use:
size: represents the numbers of the majority element. Starts from 0.

for i in range(n):
    Compare each element from the input with the majority element we found from step 1 by calling the ***same_party*** function.
    Increase ***size*** by 1, every single time when ***same_party*** returns true.

Finally, ***size*** is the answer to the size of the largest party.

Running time analysis:

Following the assignment instruction, the running time here is the number of pairs of delegates that they introduce to each other.

- Best case: $O(1)$, constant running time when $n = 1\ or\ n = 2$.
- Worst case: $T(n) = O(n-1) + O(n) = O(2n-1)$

The worst case of our algorithm calls the ***same_party*** function $2n-1$ times.

Prove correctness of the algorithm:

Because the size of the whole party is n and the majority party size should be greater than half of n, then we have two situations:
1. The size of the whole party is even:
If so, the majority party size must be greater than (n/2)+1and the non-majority party size will be smaller than (n/2)-1. The "neutralization" idea of our algorithm is basically performing a Mathematical subtraction：

$$\left(\frac{n}{2}+1\right)-\left(\frac{n}{2}-1\right)=2$$

It tells us that if n is even, then there must be at least 2 majority elements that could not be canceled by all the non-majority elements.
Therefore, the ***majority_index*** will be the majority Party element in the array (Party).

2. The size of the whole party is odd:
If so, the majority party size must be greater than (n/2)+0.5 and the non-majority party size will be smaller than (n/2)-0.5. The "neutralization" idea of our algorithm is basically performing a Mathematical subtraction：

$$\left(\frac{n}{2}+0.5\right)-\left(\frac{n}{2}-0.5\right)=1$$

It tells us that if n is odd, then there must be at least 1 majority elements that could not be canceled by all the non-majority elements.

Therefore, the ***majority_index*** will be the majority Party element in the array (Party).