Sort Colors

Given an array with *n* objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

Note:
You are not suppose to use the library's sort function for this problem.

click to show follow up.

Follow up:
A rather straight forward solution is a two-pass algorithm using counting sort.
First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.

Could you come up with an one-pass algorithm using only constant space?

Solution 1

The idea is to sweep all 0s to the left and all 2s to the right, then all 1s are left in the middle.

```
class Solution {
public:
    void sortColors(int A[], int n) {
        int second=n-1, zero=0;
        for (int i=0; i<=second; i++) {
            while (A[i]==2 && i<second) swap(A[i], A[second--]);
            while (A[i]==0 && i>zero) swap(A[i], A[zero++]);
        }
    }
};
```

written by yidawang original link here

Solution 2

```
// two pass O(m+n) space
void sortColors(int A[], int n) {
    int num0 = 0, num1 = 0, num2 = 0;

    for(int i = 0; i < n; i++) {
        if (A[i] == 0) ++num0;
        else if (A[i] == 1) ++num1;
        else if (A[i] == 2) ++num2;
    }

    for(int i = 0; i < num0; ++i) A[i] = 0;
    for(int i = 0; i < num1; ++i) A[num0+i] = 1;
    for(int i = 0; i < num2; ++i) A[num0+num1+i] = 2;
}

// one pass in place solution
void sortColors(int A[], int n) {
    int n0 = -1, n1 = -1, n2 = -1;
    for (int i = 0; i < n; ++i) {
        if (A[i] == 0)
        {
            A[++n2] = 2; A[++n1] = 1; A[++n0] = 0;
        }
        else if (A[i] == 1)
        {
            A[++n2] = 2; A[++n1] = 1;
        }
        else if (A[i] == 2)
        {
            A[++n2] = 2;
        }
    }
}

// one pass in place solution
void sortColors(int A[], int n) {
    int j = 0, k = n - 1;
    for (int i = 0; i <= k; ++i){
        if (A[i] == 0 && i != j)
            swap(A[i--], A[j++]);
        else if (A[i] == 2 && i != k)
            swap(A[i--], A[k--]);
    }
}

// one pass in place solution
void sortColors(int A[], int n) {
    int j = 0, k = n-1;
    for (int i=0; i <= k; i++) {
        if (A[i] == 0)
            swap(A[i], A[j++]);
        else if (A[i] == 2)
            swap(A[i--], A[k--]);
    }
}
```

written by shichaotan original link here

Solution 3

```
class Solution {
public:
//use counting sort
void sortColors(int A[], int n) {
int red = -1, white = -1, blue = -1;

for(int i = 0; i < n; i++){
    if(A[i] == 0){
        A[++blue] = 2;
        A[++white] = 1;
        A[++red] = 0;
    }
    else if(A[i] == 1){
        A[++blue] = 2;
        A[++white] = 1;
    }
    else if(A[i] == 2)
        A[++blue] = 2;
}
}
};
```

the clever thing is that use three variable to store the three colors' index position. When you face A[i] == 0, all the variables add 1 because 0 is former. Do the same thing to other 2 situation.

Ex: If you just face 2, just need to assign 2 to the A[++blue], and "++blue" will increase "blue" with 1. Next if you face 0, you will increase 3 variable and assign the number to A!

It will make sure you always get the right sorted array when you run the for loop.

written by autekroy original link here