<https://leetcode.com/problems/can-place-flowers/description/>

You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in adjacent plots.

Given an integer array flowerbed containing 0's and 1's, where 0 means empty and 1 means not empty, and an integer n, return true if n new flowers can be planted in the flowerbed without violating the no-adjacent-flowers rule and false otherwise.

**Example 1:**

**Input:** flowerbed = [1,0,0,0,1], n = 1

**Output:** true

**Example 2:**

**Input:** flowerbed = [1,0,0,0,1], n = 2

**Output:** false

**Constraints:**

1 <= flowerbed.length <= 2 \* 10^4

flowerbed[i] is 0 or 1.

There are no two adjacent flowers in flowerbed.

0 <= n <= flowerbed.length

**Attempt 1: 2024-10-21**

**Solution 1: Greedy (30 min)**

class Solution {

    public boolean canPlaceFlowers(int[] flowerbed, int n) {

        int count = 0;

        int len = flowerbed.length;

        for(int i = 0; i < len; i++) {

            // Only if flower plant place is empty we will gonna perform this logic

            if(flowerbed[i] == 0) {

                // Creating two values previous & next. Are they 0 or 1

                int prev = (i == 0 || flowerbed[i - 1] == 0) ? 0 : 1;

                int next = (i == len - 1 || flowerbed[i + 1] == 0) ? 0 : 1;

                if(prev == 0 && next == 0) {

                    flowerbed[i] = 1;

                    count++;

                }

            }

        }

        return count >= n;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

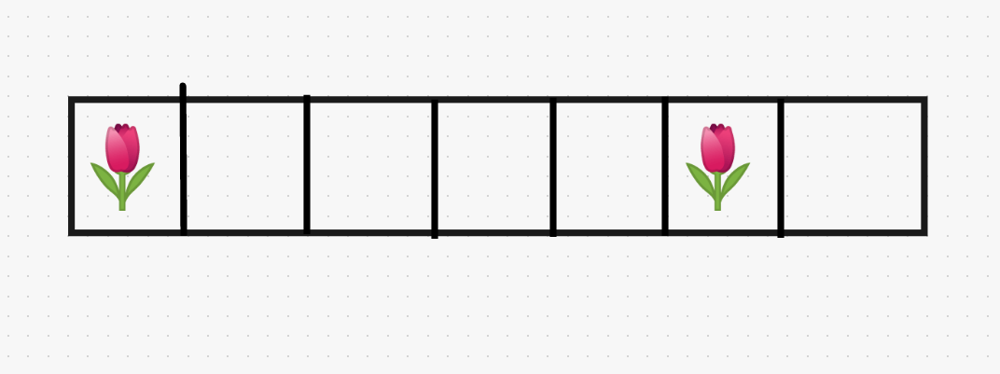
**Refer to**

<https://leetcode.com/problems/can-place-flowers/solutions/1698539/by-explanation-i-made-it-so-easy/>

Suppose [1,0,0,0,0,1,0] this is a flowerbed given to us.

Let's forget about how many flower we can plant into this & we will find the maximum no. of flower that we can plant.

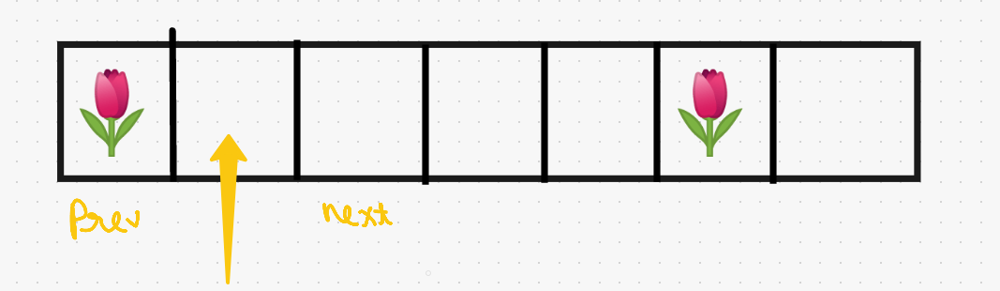
So, in order to find that what we can do is, if we see it visually we can see there are two flowers planted in the flower bed of 7 flowers



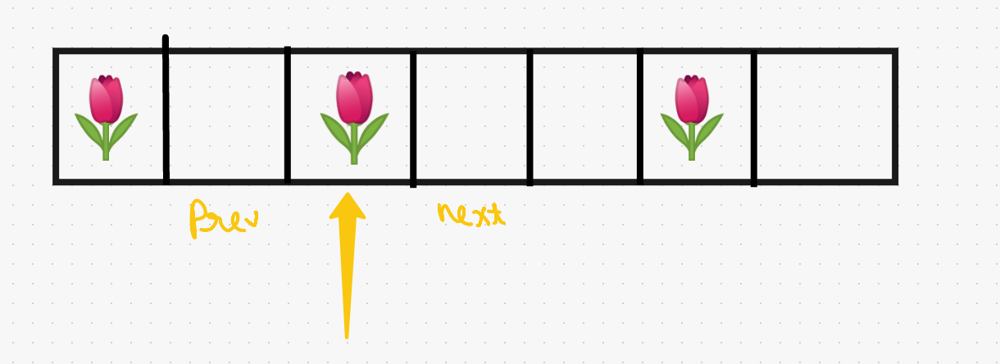
So, we need to **iterate** over all the **flowerbed** and we need to focus our attention on the fact where we can **plant flower**. The flower can only be **planted** where there is a **no flower yet**. So, we need to search for the position where value is **0**.

We **start** from **1st index** and we will move till we find a value i.e. **0** & going forward we find a place which has **no flower**. Now the condition says there **should not be any flower adjacent to this flower**. So, in order to place this flower we have to make sure that there is no adjacent flower.

So, we need to check these two position i.e. **previous position & next position**.



As we can see a flower exist in **previous position**, we can't plant a flower at this place. So, we move our pointer to **next index**. As we move here we again check for adjacent flowers. As there are no flowers present adjacent to this empty place, so **we will plant a flower**. And also keep a **count** of that we have planted a flower at this place.



We move ahead with same concept checking every two adjacent edges of empty place & move ahead **till we reach the end of the Array**.

**Now let's code it up**

**code each line explained :**

{

int count = 0; // created a count variable

for(int i = 0; i < flowerbed.length; i++){ // start looping from 1st position till end of the array

if(flowerbed[i] == 0){ // only if flower plant place is empty we will gonna perform this logic

// creating two values previous & next. Are they 0 or 1

int prev = (i == 0 || flowerbed[i - 1] == 0) ? 0 : 1;

int next = (i == flowerbed.length - 1 || flowerbed[i + 1] == 0) ? 0 : 1;

if(prev == 0 && next == 0){ // and only if these two values are 0

flowerbed[i] = 1; // will plant a flower

count++; // increment the count

}

}

}

return count >= n; // in the end we just need to check is count we get is greater or equals to n

**Java**

class Solution {

public boolean canPlaceFlowers(int[] flowerbed, int n) {

int count = 0;

for(int i = 0; i < flowerbed.length; i++){

if(flowerbed[i] == 0){

int prev = (i == 0 || flowerbed[i - 1] == 0) ? 0 : 1;

int next = (i == flowerbed.length - 1 || flowerbed[i + 1] == 0) ? 0 : 1;

if(prev == 0 && next == 0){

flowerbed[i] = 1;

count++;

}

}

}

return count >= n;

}

}

**Refer to**

[L735.Asteroid Collision (Ref.L2211)](note://WEB88a08f8ad07d5d7ac8a64605312c92a6)