Hacker Rank

☆ Splitting Pixels

A pixel color is defined as a 24 bit integer. Each of the 3 bytes making up the integer represents one of three colors: red, green or blue. The intensity of a pixel is proportional to its byte value which will range from *0* which results in none of that color, to *255*, the maximum intensity.

Determine which of the *pure* colors a series of pixels are nearest to. To do this, calculate the Euclidean distance of each of the RGB values of a pixel to the RGB values of a pure color. For the distance between two pixels having RGB values (r_1, g_1, b_1) and (r_2, g_2, b_2), it is calculated as follows:

$$d = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}$$

For reference, the RGB values are defined as follows:

Pure Color	R	G	B
Black	0	0	0
White	255	255	255
Red	255	0	0
Green	0	255	0
Blue	0	0	255

Given a *24*-bit binary string describing a pixel, identify which of these five colors the pixel is closest to using the Euclidean distance calculation. Then return the closest pure color: *Red, Green, Blue, Black, White* or if the pixel is equidistant from two or more colors, return *Ambiguous*.

For example, the pixel described by the binary string *000000001111111100000110* has the following three components:

1. *red* = $(00000000)_2 = (0)_{10}$
2. *green* = $(11111111)_2 = (255)_{10}$
3. *blue* = $(00000110)_2 = (6)_{10}$

This means the pixel's RGB value is *(0, 255, 6)*. Now, calculate its Euclidean distance to each color:

```
Pure Black: d = ((0 -   0)² + (255 -   0)² + (6 -   0)²)^1/2 =  65061^1/2 = 255.0705785
Pure White: d = ((0 - 255)² + (255 - 255)² + (6 - 255)²)^1/2 = 127026^1/2 = 356.4070706
Pure Red:   d = ((0 - 255)² + (255 -   0)² + (6 -   0)²)^1/2 = 130086^1/2 = 360.6743684
Pure Green: d = ((0 -   0)² + (255 - 255)² + (6 -   0)²)^1/2 =     36^1/2 = 6
Pure Blue:  d = ((0 -   0)² + (255 -   0)² + (6 - 255)²)^1/2 = 127026^1/2 = 356.4070706
```

The color with the smallest distance to the pixel is *Pure Green*, so the answer is *Green*.

Function Description

Complete the function *closestColor* in the editor below. The function must return an array of strings each representing the closest color for the pixels in the order presented.

closestColor has the following parameter(s):

pixels[pixels[0],...pixels[n-1]]: an array of 24-bit binary strings representing pixels as described

Constraints

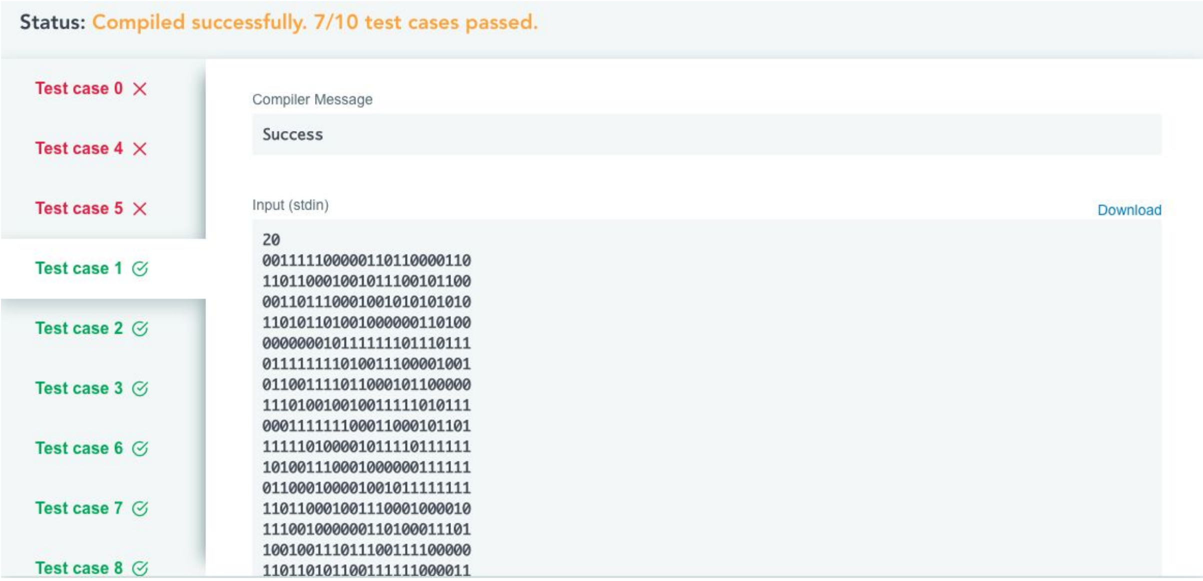
- $1 \le n \le 100$

  - The distance to pure *Blue* is 216.45784809056934.
  - The distance to pure *Red* is 258.3486016993318.
  - The distance to pure *Black* is 313.22356233208257.
  - The distance to pure *Green* is 333.33766663850037.

1. *110000010101011111101111* → *(193, 87, 239)* is closest to *White*:
   - The distance to pure *White* is 179.78876494375282.
   - The distance to pure *Blue* is 212.30638238168913.
   - The distance to pure *Red* is 261.7899921692959.
   - The distance to pure *Black* is 319.2788749666974.
   - The distance to pure *Green* is 350.13425996323184.
2. *100110101100111111101101* → *(154, 207, 237)* is closest to *White*:
   - The distance to pure *White* is 113.26517558367179.
   - The distance to pure *Blue* is 258.62907802488104.
   - The distance to pure *Green* is 286.6862396418775.
   - The distance to pure *Red* is 330.4829798945779.
   - The distance to pure *Black* is 350.334126228091.
3. *010111011010010110000011* → *(93, 165, 131)* is closest to *Green*:
   - The distance to pure *Green* is 184.14668066516975.
   - The distance to pure *White* is 222.97981971469974.
   - The distance to pure *Blue* is 226.38462845343543.
   - The distance to pure *Black* is 230.2932912613826.
   - The distance to pure *Red* is 265.7630523605567.
4. *000000001111111111111111* → *(0, 255, 255)* is equidistant from *White*, *Green*, and *Blue*, so it is *Ambiguous*:
   - The distance to pure *White* is 255.0.
   - The distance to pure *Green* is 255.0.
   - The distance to pure *Blue* is 255.0.

- The distance to pure *Black* is 360.62445840513925.
- The distance to pure *Red* is 441.6729559300637.

Return the array *["White", "White", "White", "Green", "Ambiguous"]* as the answer.




```
function closestColor (pixels){
   let nearestTo = [];
   let pureblack = [0,0,0];
   let purewhite = [255,255,255];
   let purered = [255,0,0];
   let puregreen = [0,255,0];
   let pureblue = [0,0,255];
   // Write your code here
   // for (let index = 0; index <= pixels.length-1; index++) {

   // }
   let results = pixels.map((value, index, number)=>{
       // let duplicateValues = [];
       let distance = [];
       const colors = pixels[index].split('');
       let red = colors.slice(0, 8);
       let green = colors.slice(8, 16);
       let blue = colors.slice(16, 24);
       // console.log(red);
       // console.log(green);
       // console.log(blue);
       let rednumber = parseInt(red.join(''), 2);
       let greennumber = parseInt(green.join(''), 2);
       let bluenumber = parseInt(blue.join(''), 2);
       // console.log(rednumber);
```

```
        // console.log(greennumber);
        // console.log(bluenumber);
        let blackdistance = EuclidianDistance(rednumber, greennumber, bluenumber,
pureblack[0], pureblack[1], pureblack[2]);
        let whitedistance = EuclidianDistance(rednumber, greennumber, bluenumber,
purewhite[0], purewhite[1], purewhite[2]);
        let reddistance = EuclidianDistance(rednumber, greennumber, bluenumber,
purered[0], purered[1], purered[2]);
        let greendistance = EuclidianDistance(rednumber, greennumber, bluenumber,
puregreen[0], puregreen[1], puregreen[2]);
        let bluedistance = EuclidianDistance(rednumber, greennumber, bluenumber,
pureblue[0], pureblue[1], pureblue[2]);
        distance.push({name: 'Black', value: blackdistance});
        distance.push({name: 'White', value: whitedistance});
        distance.push({name: 'Red', value: reddistance});
        distance.push({name: 'Green', value: greendistance});
        distance.push({name: 'Blue', value: bluedistance});

        let sortingdistance = distance.sort((a, b) => a.value > b.value);
        let duplicateValues_2 =
sortingdistance.reduce((b,c)=>((b[b.findIndex(d=>d.element===c.value)]||b[b.push({e
lement:c.value,count:0})-1]).count++,b),[]);
        // console.log(duplicateValues_2.find(a => a.count > 1));
        if(duplicateValues_2.find(a => a.count > 1)){
            nearestTo.push('Ambiguous');
        }else{
            nearestTo.push(sortingdistance[0].name);
        }

        // if(duplicateValues.length > 0){
        //     // console.log(duplicateValues.length);
        //     nearestTo.push('Ambiguous');
        // }else{
        //     nearestTo.push(sortingdistance[0].name);
        // }
    })

    return nearestTo;
    //RETURN STRING ARRAY
}
```

Probar rta

```
let array_binary = ['111111110000000010101010','010111011010010110000011',
'000000001111111111111111'];//red - green - ambiguous
console.log(array_binary);
console.log(closestColor(array_binary));
```

--

```
let array_binary = ['101111010110011011100100',
'110000010101011111101111',
'100110101100111111101101',
'010111011010010110000011',
'000000001111111111111111']
// White
// White
// White
// Green
// Ambiguous
console.log(array_binary);
console.log(closestColor(array_binary));
```