과제 #2 double rainbow - 201921438 조현태

1) 소스코드

```
#include <stdio.h>
#include <vector>
#include <iostream>
#include <algorithm>

using namespace std;

int n, k; // n:배열의 크기, k:색깔의 종류
vector <int> P; // 벡터 P
vector <int> color_P; // P에 대한 색깔의 수 벡터
vector <int> color_Pl; // P'에 대한 색깔의 수 벡터
vector <int> answer; // double rainbow후보를 저장하기 위한 정답 벡터

// 색깔의 개수를 계산하는 함수
int count_color(vector <int> arr)
{
	// 이진탐색을 위한 정렬
	sort(arr.begin(), arr.end());
	// color[0] = 0으로 기본값으로 가지고 있으므로 arr.begin()+1
	if (binary_search(arr.begin()+1, arr.end(), 0))
		return 0;
	// 색깔 벡터에 모든 색깔이 있을 경우 -> O
	else
		return k;
}

// double rainbow를 탐색하는 함수
void double_rainbow(void)
{
	// 벡터의 왼쪽, 오른쪽 변수
	int left = 0;
	int right = 0;

	// left, right가 n을 넘지 않을 때까지 반복
	while ((left < n) && (right < n))
	{
		// P'이 k가지 색깔을 만족할 때
		if (count_color(color_Pl) == k)
		{
			// P-P'이 만족할 때 -> 정답 배열에 저장한다.
```

```
                        if (count_color(color_P) == k)
                                // 아래 else문에서 right += 1을 color배열에 반영시킨 후 적용했기때문에 (right - 1) - left + 1 => right - left
                                answer.push_back(right - left);

                        // P'의 왼쪽을 조인다. -> P'이 만족하지 못할 때까지
                        left += 1;
                        color_P[P[left-1]] += 1;
                        color_Pl[P[left-1]] -= 1;
                }
                // P'이 k가지 색깔을 만족하지 못할 때
                else
                {
                        // P'을 오른쪽으로 확장한다.
                        color_P[P[right]] -= 1;
                        color_Pl[P[right]] += 1;
                        right += 1;
                }
        }
        // 위의 조건을 모두 만족하지 못할 때 -> double rainbow가 없을 경우
        if (answer.size() == 0)
                // 최솟값 출력을 위해 answer에 0추가.
                answer.push_back(0);
}


// 메인 함수
int main(void)
{
        cin >> n >> k;

        // 벡터 초기화
        color_P.assign(k + 1, 0);
        color_Pl.assign(k + 1, 0);

        // P 벡터 받기 + color1의 색깔 수 받기
        for (int i = 0; i < n; i++)
        {
                int a;
                cin >> a;
                P.push_back(a);
                // 색깔의 종류와 인덱스를 동일시해서 색깔의 개수 카운트한다. -> 색깔은 1부터이므로 0번 인덱스는 없는셈치고 진행해야함.
```

```
            // color벡터은 k+1개로 (0,a,b,c,d)로 표현할 수 있음. (a,b,c,d는 색깔의 개수)
            color_P[a] += 1;
        }

        // 함수 실행
        double_rainbow();

        // answer벡터(정답 후보)중 최솟값 출력
        int min = *min_element(answer.begin(), answer.end());
        cout << min << endl;

        return 0;
}
```

2) 문제 설명

길이가 n이고 k 종류의 색깔로 칠해진 배열 P에서 double rainbow를 찾는 문제입니다.
double rainbow란 **P'이라는 임의의 배열과 P-P'이라는 배열 모두**
배열 안에 k 종류의 색깔을 적어도 하나 이상 포함하고 있어야 합니다.

저는 일반적으로 배열을 다룰 때 길이를 정하고 시작점만 움직이는 방식을 많이 사용했었는데
위의 방식으로 코드를 작성하니 **길이도 증가하고 시작점도 증가하고 P'의 내부도 확인해야 하니**
시간복잡도가 거의 **O(n^3)** 가까이 나왔습니다. (n<10000이므로 O(n^2)까지를 목표로!)
그래서 배열에 대한 정보를 찾아보니까 굳이 길이를 정하고 시작점만 움직이지 않고
배열의 양 끝점을 모두 제어하는 방법이 있다는 것을 알게 되었고 코드에 적용했습니다.

먼저, 색깔의 개수를 저장하는 color 벡터들을 만들었고,
P 벡터을 받음과 동시에 color_P 벡터에 **인덱스와 색깔의 종류 동일시**해서
색깔의 개수를 카운트했습니다. **ex) color_P = (0,3,2,3,2) -> 1번 색깔: 3개, 2번 색깔: 2개, ...**
이때, 색깔은 1부터이므로 color 벡터의 **0번 인덱스를 제외**하고 진행했습니다.

```
color_P.assign(k + 1, 0);
color_Pl.assign(k + 1, 0);
for (int i = 0; i < n; i++)
{
        int a;
        cin >> a;
        P.push_back(a);
        color_P[a] += 1;
}
```

저는 먼저 **P'의 양 끝점인 left와 right**를 변수로 두고
오른쪽 인덱스값인 **right를 키워서 P'이 k 종류의 색깔을 모두 가지고 있을 때,**
right를 멈추고 P'의 왼쪽 인덱스값인 left를 줄여나가면서 P-P'도 k 종류의 색깔을
가지고 있을 때, 값을 answer에 저장하는 함수인 double_rainbow를 만들었습니다.

```
void double_rainbow(void)
{
        int left = 0; / int right = 0;
        while ((left < n) && (right < n))
                { ... }
}
```

위에서 color_P는 P를 받을 때 카운트했으므로 (0,3,2,3,2)이고 color_Pl은 (0,0,0,0,0)입니다.
여기서 **P'에 P의 0번 값의 색깔을 추가하고 반대로 P-P'은 P의 0번 값의 색깔을 빼야합니다.**
따라서 color_P = (0,2,2,3,2), color_Pl = (0,1,0,0,0)이 됩니다. (sample input 1번 기준)
이런 식으로 right를 늘려주다가 진행하다가 **color_Pl의 종류가 k개를 만족할 때는**
right를 멈추고 이번엔 left를 늘리며 P'에는 색깔을 하나 빼고, P-P'에는 색깔을 하나 더해줍니다.
이때, color_P의 종류가 k를 만족하는 값은 모두 answer에 저장합니다.

계속 left를 늘려가다가 마지막엔 answer에 최솟값을 저장하고 P'이 k 종류의 색깔을
만족하지 못하는 경우가 생기는데 이 경우, 다시 right를 늘리면서 위의 과정들을 반복하게 됩니다.

```
if (count_color(color_Pl) == k)
        if (count_color(color_P) == k)
                answer.push_back(right - left);
        left += 1;
        color_P[P[left-1]] += 1;
        color_Pl[P[left-1]] -= 1;
else
{
color_P[P[right]] -= 1;
color_Pl[P[right]] += 1;
right += 1;
}
```

while문을 모두 진행했는데도 answer에 아무 값이 없을 경우는 double rainbow가 없다는 의미로
문제에서 제시한 대로 0을 출력하기 위해서 answer에 0을 저장합니다.

```
if (answer.size() == 0)
        answer.push_back(0);
```

이런 식으로 계산하기 위해서는 **color_P와 color_Pl가 k 종류의 색깔을 가지는지를**
계산하는 함수가 필요하므로 color 벡터 내 0이 있으면 0을 0이 없이 모든 값이 존재한다면
k를 반환하는 함수인 count_color를 만들었습니다.
0번 인덱스를 제외한 나머지에 0이 있는지를 **이진 탐색**을 통해 계산했습니다.

```
int count_color(vector <int> arr)
{
        // 이진탐색을 위한 정렬
        sort(arr.begin(), arr.end());
        // color[0] = 0으로 기본값으로 가지고 있으므로 arr.begin()+1
        if (binary_search(arr.begin()+1, arr.end(), 0))
                return 0;
        else
                return k;
}
```

마지막으로 함수를 호출하고 double rainbow후보들을 가지고 있는
answer벡터의 최솟값을 구해 출력합니다.

```
double_rainbow();
int min = *min_element(answer.begin(), answer.end());
cout << min << endl;
```