과제 #12 Palindrome Type - 201921438 조현태

1) 소스코드

```
#define _CRT_SECURE_NO_WARNINGS
#include <cmath>
#include <cstdio>
#include <iostream>
#include <string>
#include <string.h>
#include <algorithm>
#include <stdbool.h>
#include <stdlib.h>
#include <stack>
#include <queue>
#include <vector>
#include <utility>
#include <functional>
#include <map>
#include <stdio.h>
#include <unordered_set>
#include <set>
using namespace std;
// 초기 문자열
string main_word; // 5 <= n <= 100000

// 2차원 문자열
string arr[4][8]; // 세로 : type-k / 가로 : type-k별 문자열 저장


int main()
{
	cin >> main_word;

	// 초기값 설정
	arr[0][0] = main_word;

	// 초기값 판단
	int cnt = 0;
	for (int l = 0; l < (main_word.size() / 2); l++)
		// 앞뒤가 다르다면
		if (main_word[l] != main_word[main_word.size() - (l+1)])
			cnt = 1;
```

```
	if (cnt == 0)
	{
		cout << 0 << endl;
		return 0;
	}

	/                                                                                /
================================================================================
================================== //

	// k = 0,1,2,3, 총 type-k는 4번 가능하므로 4번 반복
	for (int i = 0; i < 4; i++)
	{
		int p = 0;
		// 앞뒤에서 제거되므로 -> 이진트리
		for (int j = 0; j < pow(2, i); j++)
		{
			// type-k별 문자열
			string word = arr[i][j];
			string another_word = word;

			int count = 0;
			for (int k = 0; k < (word.size() / 2); k++)
				// 앞뒤가 다르다면
				if (word[k] != word[word.size()-(k+1)])
				{
					// k = {-1, 0, 1, 2, 3}이므로
					if (i < 3)
					{
						// 앞에서 제거
						word.erase(word.begin() + k);
						// 뒤에서 제거

another_word.erase(another_word.begin() + word.size() - k);

						// 이차원 배열에 입력하기.
						arr[i + 1][p] = word;
						p++;
						arr[i + 1][p] = another_word;
						p++;
					}
					count = 1;
```

```
                                        break;
                                }

                        // 앞뒤가 모두 같은 경우
                        if (count == 0)
                        {
                                cout << i << endl;
                                return 0;
                        }
                }
        }
        // type-k의 모든 경우를 무사히 탐색한 경우 -> k의 범위를 벗어나므로 -1 출력
        cout << -1 << endl;

        return 0;
}
```

2) 문제 설명

이 문제는 회문을 찾는 문제로 일반적인 문제와 달리 type-k라는 것이 존재합니다.
k = {-1, 0, 1, 2, 3}까지 가능하고,
type-0은 원래 문자열에서 아무것도 제외하지 않아도 회문이 되는 경우,
type-1은 문자열에서 1개를 제외해서 회문이 되는 경우,
type-2, type-2도 마찬가지입니다.
마지막으로 type-(-1)은 3개의 문자를 제외해도 회문이 되지 않는 경우입니다,

회문은 start와 end, start+1과 end-1 이런식으로 중앙에 올때까지
모든 문자가 같아야하므로 아래와 같이 탐색했습니다.

```
for (int l = 0; l < (main_word.size() / 2); l++)
        // 앞뒤가 다르다면
        if (main_word[l] != main_word[main_word.size() - (l+1)])
                cnt = 1;
```

여기서 앞과 뒤, 둘 중에 어느 것을 제외하는가에서 분기점이 생기게 되는데
k의 개수가 4개이므로 1+2+4+8+16 = 31가지의 문자열이 생길 것입니다.
문자열의 길이는 5 ~ 10만이므로 전체 탐색을 해도 300만내외의 시간복잡도가 예상됩니다.

따라서 저는 트리는 만들어서 위에서 사용한 회문을 판단하는 반복문을 사용하여
모든 문자열의 탐색했습니다. 여기서 출력해야하는 것은 type-k의 숫자이므로
이차원 배열로 나타내어서 arr[type-k의 숫자][문자열 개수]의 형태로 표현했습니다.

예시)
word = “abcbab”
type_0 -> arr[0][0] = abcbab
type_1 -> arr[1][0] = abcba (이미 회문이므로 그냥 공백을 입력함), arr[1][1] = bcbab
type_2 -> arr[2][0] = “ ”, arr[2][1] = “ ”, arr[2][2] = bcba, arr[2][3] = 츄뮤
type_3 -> arr[3][0], arr[3][1], arr[3][2], … arr[3][7]

위의 배열을 모두 만든 후, 위에서 설명드린 회문을 판단하는 for문을 사용해서
모든 경우의 수를 탐색합니다.
type_k가 작은 수부터 탐색하므로 가장 작은 k값을 반환합니다.