알고리즘 스터디 3주차 (백준 _ 10033, 13549, 2572번)

10033번. Fair Photography (https://www.acmicpc.net/problem/10033)

접근방식
- 문제 : 소 사진을 찍을 때, 얼룩소와 흰소의 수가 같아지는 최대크기찾기. - 얼룩소와 흰소 의 합은 **같아야된다-> 짝수.** **- 흰소 1, 얼룩소는 -1로 보았을 때, 부분합이 짝수가 되는 최대크기를 구한다** → 최대크기 이므로, 모든 것들의 합을 구하고, 1. 양수이고 짝수이면 최대값. 2. 양수이고 홀수, 가장 가까운 소를 제거 2. 음수 = X, 부분합이 0이되는 경우를 체크.

부분합이 음수인경우

마지막 0인 부분을 구한다.

으잌 모르겠당!

13549번. 숨바꼭질 3(https://www.acmicpc.net/problem/13549)

접근방식

- 문제가 되는 케이스. 이동혹은 순간이동식 위치가 **(0 ≤ N ≤ 100,000)**을 초과하는 경우
- **50,000이후는 순간이동이 안된다.**
- **아래부터 위로 가는 동적계획법 사용.**

(위에서부터 아래로 오는 케이스또한 괜찮을 것 같음 : ex) K가 홀수 인경우 +or -1로 해서 순간이동 위치 찾기.

- int[100001]은 메모리 제한 512MB 이하이므로, int arr[100001]사용

```
//배열과 인덱스가 들어오면, i 와 i+1 의 차가 1 보다 큰 경우 작은 배열의값 +1 로
변경하는 함수
    void walkingCheck(int location[], int i){
        int a = location[i];
        int b = location[i+1];

        if(a>b+1) location[i] = b+1;
        else if(a+1<b) location[i+1] = a+1;
    }
```

```
void walkingCheck(int location[],bool v[], int i){

        int a = location[i];
        int b = location[i+1];

        if(a>b+1) location[i] = b+1;
        else if(a+1<b) location[i+1] = a+1;
    }


    int main(){
        scanf("%d",&N);
        scanf("%d",&K);

        arr[N] = 0;

        //뒤로 걷는 좌표 체크
        for (int i = N-1; 0<=i;i--){
            arr[i] = arr[i+1]-1;
            visit[i] = true;
        }

        /*0 번 좌표부터 걸으면서 이동
        모든 순간이동마다 +1 , -1 좌표까지 확인해주기.
        50000 이하까지만 순간이동하기.*/
```

```
        for(int i =1; i<K; i++){
            walkingCheck(arr,i);
            //순간이동 체크
            for(int j = i; 2*j<10001; j=j*2){
                arr[j*2] = arr[j];
                walkingCheck(arr,2*j-1);
                walkingCheck(arr,2*j);
                walkingCheck(arr,2*j-1);
                }

            }

        }
```

더 알아보기.

- 모든 곳을 방문하는 케이스는 BFS로 풀자. 머리속으로는 다 탐색했다 생각했지만, 체크하지 못하는 부분이 존재 하는 것 같다. 위와 같은 문제 (모든곳을 탐색해야하는)경우 BFS 알고리즘을 이용하자.

2572번. 보드게임 (https://www.acmicpc.net/problem/2572)

접근방법

- 위에서 배웠던 BFS나 DFS를 쓰자.
- c++은 벡터를 이용해, 2차원 배열이라 생각하고 작성해야 될 것 같다.

뭔가 인공지능에서 배웠던것같은데

```
//#2572번_보드게임
#include <iostream>
#include <vector>
using namespace std;

int N, M, K, answer = 0;

vector<char> card;
vector<vector<bool>> map;
vector<vector<char>> road;

void input() {
	std::ios_base::sync_with_stdio(false);
	cin >> N;

	char temp;
	for (int i = 0; i < N; i++) {

		std::ios_base::sync_with_stdio(false);
		cin >> temp;
		card.push_back(temp);
	}

	std::ios_base::sync_with_stdio(false);
	cin >> M >> K;

	map.resize(M + 1, vector<bool>(M + 1, false));
	road.resize(M + 1, vector<char>(M + 1, 'C'));

	int i, j;
	char c;
	for (int k = 0; k < K; k++) {

		std::ios_base::sync_with_stdio(false);
		cin >> i >> j >> c;
		map[i][j] = true;
		map[j][i] = true;

		road[i][j] = c;
		road[j][i] = c;
	}
}

//재귀를 통한 DFS로 모든 트리 탐색
int DFS(int count, int row, int col, int max) {
```

```
        if (answer < max) answer = max;
        if (count == N) return 0;

        char C = card[count];

        for (int i = 1; i < M + 1; i++)
        {
                if (map[row][i]) {
                        if (C == road[row][i]) DFS(count + 1, i, row, max + 10);
                        else DFS(count + 1, i, row, max);
                }
        }

}

void output() {
        printf("%d", answer);
}

int main() {
        input();
        DFS(0,1,1,0);
        output();
}
```

시간초과 완전탐색의 경우 시간이 오버되나봄.

그럼 동적프로그래밍해보자.

접근방법

- 겹치는 경우를 찾아 제거해주자.
- n번째 카드에서, k마을에서 1등인경우를 체크하자.

그러면 벡터 DP 크기는 n*k+1; 형태는 vector<vector<int>>DP(n,vector<int>(k+1,0))

1카드로 갈수있는 경우 모두 체크해서 dp에 기입, 못가는 경우는 -1을 넣어줘서 따로 체크.

DP를 이용하여 2 카드로 가는 경우 체크, MAX값이 변경될 때만 기록. 이런식으로 하면..

총 실행횟수는 카드 수 * 마을 수로 된다.

이전에 완전탐색의 경우 실행횟수는 **마을수^카드수**

*** 코드에서 row, col로 구분하였지만 나타내는 의미가 배열이 아닌 길이므로 출발마을, 도착 마을등으로 구분해야된다.**

그리고 백준 ㅅㅂ __max함수가 안되서 당황했다. 왜안될까?

중요함수들

input

```
void input() {

    std::ios_base::sync_with_stdio(false);
    cin >> N;

    char temp;
    for (int i = 0; i < N; i++) {


        cin >> temp;
        card.push_back(temp);
    }


    cin >> M >> K;
    map.resize(M + 1, vector<bool>(M + 1, false));
    road.resize(M + 1, vector<char>(M + 1, 'C'));
    DP.resize(N + 1, vector< pair<int, bool> >(M + 1, make_pair(0, false)));
    int i, j;
    char c;
    for (int k = 0; k < K; k++) {


        cin >> i >> j >> c;
        map[i][j] = true;
        map[j][i] = true;

        road[i][j] = c;
        road[j][i] = c;
    }
}
```

DP

```
void DPsearch(int count, int start) {
    if(DP[count][start].second)
        for (int i = 1; i < M + 1; i++) {
        if (map[start][i]) {
            if (card[count] == road[start][i])//현재 카드와 해당 길이 같은
색이면, DP[count + 1][i]의 값과 DP[count][start]+10 의 값중 높은값을 입력.
                DP[count + 1][i].first = __max(DP[count + 1][i].first,
DP[count][start].first + 10);
            else DP[count + 1][i].first = __max(DP[count + 1][i].first,
DP[count][start].first);
            DP[count + 1][i].second = true;
        }
    }

}
```

DFS

```
//재귀를 통한 DFS로 모든 트리 탐색
int DFS(int count, int row, int col, int max ){
    if(answer < max) answer = max;
    if(count == N) retrun 0;

    char C = card[count];

    for(int  i = 0; i < M+1 ; i++)
    {
        if(map[row][i]){
            if(C == road[row][i]) DFS(count+1, row, i, max+10);
            else DFS(count+1, row, i, max);
        }
    }
}
```