10033번

문제 분석:

- 연속된 구간 안에 같은 수의 흰색과 점박이? 소가 있도록 하는 가장 긴 구간을 구하기
- 흰색은 점박이로 바꿀 수 있다

기본 접근 방법:

- 가장 기본적으로 구간 안에 있는 소의 수는 짝수가 되어야 한다!
- 항상 흰색 소가 점박이 새끼보다 0이상의 짝수 만큼은 더 많아야 된다 ex) (흰소, 점박이) = (4, 2), (2,2) 이때 (0,2)는 안됨!

방법 1

1. 들어오는 소들의 위치에 해당하는 번호와 품종?을 dictionary 형태로 저장
2. 들어올 때 위치에 해당하는 값을 index라는 list에 저장
3. dictionary는 자동 정렬되고 index는 sort를 사용하여 정렬하면 index 리스트의 0번째는 가장 왼쪽에 해당하는 소에 정보를 갖는다
4. index 리스트를 반복문을 통해 접근
5. index에 가리키고 있는 숫자를 포함 짝수개의 소를 포함 할 수 있는 인덱스를 가장 오른쪽 부터 접근
6. 모든 값으 합이 0 혹은 0의 짝수이면 max 값과 비교하여 더 큰 값을 저장하고 다음 index 리스트로 넘어간다.

In [ ]:

```
#from collections import OrderedDict

N = int(input())
index = []
cow = {}
cnt = 0

for i in range(N):
    cow_pos, cow_color = input().split()
    cow_pos = int(cow_pos)
    if cow_color == 'W':
        cnt +=1
        cow[cow_pos] = 1
    else:
        cnt -=1
        cow[cow_pos] = -1
    index.append(cow_pos)

index.sort()  ## 소의 위치를 index로 지정하기 위해서
maximum = 0
temp = 0

for i in range(len(index)):
    # j의 값 중 가장 큰 값을 구함 여기서부터 i+1 즉 i까지 내려가면 [j, i)로 가능 -2 씩 하강

    if i != 0: # 0번째를 제외하고 1번째부터 시작한다고하면 1번째 부터 시작이면 그 전 값을 빼줘야 함
        cnt -= cow[index[i-1]]

    temp = cnt
    ## j값 구하는 곳
    if i % 2 ==0: # i 가 짝수이면 j는 N-1 이하 중 가장 큰 홀수
        if N % 2 ==0:
            j = N-1
        else:
            j = N-2
    else: # i 가 홀수이면 j는 N-1 이하 중 가장 큰 짝수
        if N % 2 ==0: #N이 짝수
            j = N-2
        else:
            j = N-1

    if maximum > (index[j]-index[i]):  ## max의 길이보다 배열이 짧을 경우 그냥 끝냄
        break

    for ind in range(j,i,-2):
        if maximum > (index[ind] - index[i]):
            break

        if ind+1 < N: ## ind가 배열의 제일 오른쪽이면 그냥 temp 사용. ind가 배열 제일 오른쪽이 아니면 바로 오른쪽에 하나 값을 뺀다
            temp -= cow[index[ind+1]]

        if temp >= 0 and temp % 2 ==0:
            maximum = max(index[ind]-index[i],maximum)
            break
        temp -= cow[index[ind]] # 다음에는 자신을 제외하기 위해 자신을 뺀다.
print(maximum)
```

시간 초과 뜸

위 코드의 문제점:

- 결국 최악의 경우 N^2이다. 한번의 반복문 안에 충분한 정보를 구해야 한다.
- 이때 한번의 반복문으로는 각 위치에서 +1, -1로 인한 값들을 구할 수 있다. 그리고 특정 위치부터 특정 위치까지의 상대적인 값이 존재하고 이를 이용하면 될 것이다.

방법2

1. 흰 소는 +1, 점박이 소는 -1
2. 어떤 위치에서 시작하던 끝날때 합이 0 혹은 2 이상의 값이면 된다. 만약 i 위치 전까지 총 합한 값이 -2 이고 그 뒤로 나온 6 소의 값이 -2이면 결과적으로 i번째부터 i+6은 0으로 합리적인 이미지를 뜻한다. 이를 이용하자.
3. 반복문으로 첫 소부터 끝까지 돈다 이때 그 소의 값이 더해지기 전에 그 전까지 더해진 소들의 값을 각각 corresponding한 위치에 저장한다.
4. 그 소의 값을 더하고 그 수에 해당하여 알맞는 위치에 존재하는 소가 있는지 알아보고 있다면 그 길이를 구하고 맥시멈인지 확인한다.

    4.1. 예를 들어 위의 -2부터 시작하여 -2로 끝나면 만족하듯 -4로 시작하여 -4로 끝나면 그 사진은 만족하는 사진이 되고 그렇기 때문에 그 경우 길이를 구한다.
    4.2. 여기서 만약 -2로 시작하였는데 특정 위치까지의 값이 0이라면, -2까지의 길이가 아닌 0까지의 길이가 되어야 한다. 왜냐하면 -2에서 0으로 만들기 위해선 어떤 방법이던지 반드시 흰 소가 두마리 더 있어야 하고 0에서 -2까지 내려오기 위해서는 반드시 점박이소 두마리가 있어야 하기때문이다. 이와 같은 원리로 특정 소 앞에 까지의 값이 이 전에 저장된 값보다 작으면 저장할 필요가 없다. (0, -2, -4, -6, ...) 저장 된다.
    4.3. 위와 같이 반듯이 0, -2, -4, -6, .. 의 순서로 저장되기 때문에 특정 소까지 합한 값이 만약 -4라면 저장된 리스트의 abs(-4//2) 의 위치까지의 길이를 구하면 아주 쉽게 길이를 계산할 수 있다. (홀수의 경우 짝수완 다르게 1로 시작 할 수도, -1로 시작 할 수도 있음을 주의해라)
    4.4. 최대 길이는 그냥 맥스 메소드를 쓰자 귀찮으니깐.

In [ ]:

```
#from collections import OrderedDict

N = int(input())

cow = []
# cnt should be nonnegative even number.
for i in range(N):
    cow_pos, cow_color = input().split()
    cow_pos = int(cow_pos)
    if cow_color == 'W':
        cow.append((cow_pos, 1))
    else:
        cow.append((cow_pos, -1))

cow.sort()  ## sorting cow

maximum = 0
temp = 0

odd_num =[]  # cow with odd base
even_num =[] # cow with even base

base, maximum, index= 0, 0, 0
even = True

for i in range(N):
    if even:
        if (len(even_num)==0 or even_num[-1][0] > base):
            even_num.append((base, cow[i][0]))
    else:
        if (len(odd_num)==0 or odd_num[-1][0] > base):
            odd_num.append((base, cow[i][0]))

    base += cow[i][1]
    even = not even

    if even:
        ## even_num 중에 현재 base보다 작거나 같은 것 중 가장 앞에 것 찾아야 함.
        if( not(len(even_num)==0) and even_num[-1][0]<= base):
            if base >= 0 :
                index = 0
            else:
                index = abs(base//2)
            maximum = max(maximum, cow[i][0]-even_num[index][1])
    else:
        if (not(len(odd_num)==0) and odd_num[-1][0] <= base):
            if odd_num[0][0] == 1:
                if base >= 1:
                    index = 0
                else:
                    index = abs(base//2)
            else: #-1일때 입니다.
                if base >= -1:
                    index = 0
                else:
                    index = (abs(base//2)-1)
            maximum = max(maximum, cow[i][0] - odd_num[index][1])

print(maximum)
```

2572번

문제 분석: N번의 이동 후 최대 값을 구한다.

기본 접근 방법:

- 특정 시도에 도착하는 위치에 대한 점수를 2차원 배열에 저장한다.
- 똑같은 길을 또 들어 왔을 때 기존의 배열과 크기를 비교하고 현재 점수가 배열의 점수보다 작다며는 그 뒤에 PATH는 가볼 필요가 없다!

방법 1.

In [ ]:

```
N = int(input())

answer = input().split()

town, road = map(int, input().split())

connection = [[-1 for _ in range(town)] for _ in range(town)]
score = [[-1 for _ in range(town)] for _ in range(N)]

for _ in range(road):
    city_1, city_2, color = input().split()
    city_1, city_2 = int(city_1)-1, int(city_2)-1
    connection[city_1][city_2] = connection[city_2][city_1] = color

def longest(num_so_far, origin,score_now):
    if (num_so_far == N-1):
        if (score[num_so_far][origin]<score_now):
            score[num_so_far][origin]=score[num_so_far][origin] = score_now
        return
    if (score[num_so_far][origin]>score_now): #score 보다 높으면 그냥 끝내
        return
    score[num_so_far][origin]=score[num_so_far][origin] = score_now

    for destination, temp_color in enumerate(connection[origin]):
        if temp_color == -1:
            continue
        if temp_color == answer[num_so_far]:
            longest(num_so_far+1, destination,score_now+10)
        longest(num_so_far+1, destination,score_now)


for destination, temp_color in enumerate(connection[0]):
    if temp_color == -1:
        continue
    if temp_color == answer[0]:
        longest(0, destination,10)
    longest(0, destination, 0)

print(max(score[N-1]))
```

시간초과 --- 0.003633260726928711 seconds ---

위 코드의 문제점 1:

- 많이 반복을 줄이긴 했지만, 결국 매번 새로운 길을 갈때 업데이트가 되는 방식이다. 위에서 부터 내려가면서 계산을 한다. 이럴 경우, 각 값이 여러번 중복되면서 최대인지 확인이 필요하다. 그리고 만약 처음에 2번째로 3번 마을을 갈 때 점수가 10점이라고 저장 되었다가 추후에 20으로, 30으로 업데이트가 되면 그 쪽에서부터 아래로 또 다 들어가본다. 비효율적이다.
- 반대로 맨 아래부터 계산을 해서 올라오면 즉 마지막 도착한 값부터 계산을 해서 올라온다면, 최종 선택 전에 마지막 직전의 i부터 한번의 선택으로 최선의 선택을 하는 최대의 값을 고르는 형식으로 접근이 가능하다. 이렇게 되면, 여러번 반복해서 한 town에 가면서 값을 비교하는 것이 아니라 한번에 최대를 구할 수 있다.
- 정리하자면, 실질적으로 처음 재귀가 실행될 때는 dfs처럼 접근을 하지만 가장 밑바닥까지 도착한 순간 실질적으로 tree는 bfs형식으로 돌기 시작한다. (예, 한번 더 갈 수 있을 때 갈 수 있는 방향을 모두 가보고 그 위 레벨로 넘어감). 이럴 때, 맨 아래에서 위로 순차적으로 올라가는 경우에 맨 아래 레벨에서 최댓 값을 올리고, 그 다음에 두 번 째 아래에서 가능한 최대값을 올리는 것을 반복하면, 추후에 어떤 방식으로 그 노드에 도착하더라도, 그 값은 정해져있기 때문에 훨씬 짧게 계산이 가능하다.

위 코드의 문제점 2: (이게 쉬워서 방법 2 코드는 이 부분을 고쳤다.)

- -1인지 아닌지 확인하고 나오는 것이 1이라는 시간이긴 하지만 for문이 많아지면 이것 또한 번거로운 작업이다. 차라리 linked list 형식으로 만들어보자

방법 2(문제점 2 고친거)

In [ ]:

```
N = int(input())

answer = input().split()

town, road = map(int, input().split())

connection = [[] for _ in range(town)]
score = [[-1 for _ in range(town)] for _ in range(N)]

for _ in range(road): # 메모리 크기 오버 뜨면 여기서 바꾸셈 그냥 linked list 형식도 괜찮고 그냥 어레이에 tuple 넣는것도 ㄱㅊ ??
    city_1, city_2, color = input().split()
    city_1, city_2 = int(city_1)-1, int(city_2)-1
    connection[city_1].append((city_2, color))
    connection[city_2].append((city_1, color))

def longest(num_so_far, origin,score_now):
    if (num_so_far == N-1):
        if (score[num_so_far][origin]<score_now):
            score[num_so_far][origin]=score[num_so_far][origin] = score_now
        return

    if (score[num_so_far][origin]>score_now): #score 보다 높으면 그냥 끝내
        return

    score[num_so_far][origin]=score[num_so_far][origin] = score_now

    for destination, temp_color in connection[origin]:
        if temp_color == answer[num_so_far]:
            longest(num_so_far+1, destination,score_now+10)
        longest(num_so_far+1, destination,score_now)

for destination, temp_color in connection[0]:
    if temp_color == answer[0]:
        longest(0, destination,10)
    longest(0, destination, 0)

print(max(score[N-1]))
```

이것도 시간 초과 뜸 --- 0.0018780231475830078 seconds --- 그래도 약간의 시간 향상은 있었다

방법 3 (문제점 1 고친거)

In [ ]:

```
import time

N = int(input())

answer = input().split()

town, road = map(int, input().split())

connection = [[] for _ in range(town)]
score = [[-1 for _ in range(town)] for _ in range(N)]

for _ in range(road):
    city_1, city_2, color = input().split()
    city_1, city_2 = int(city_1)-1, int(city_2)-1
    connection[city_1].append((city_2, color))
    connection[city_2].append((city_1, color))

def longest(num_so_far, origin):
    if (num_so_far == N-1):
        return 0
    if score[num_so_far][origin] != -1:
        return score[num_so_far][origin]
    for destination, temp_color in connection[origin]:
        score[num_so_far][origin] = max(score[num_so_far][origin], longest(num_so_far+1, destina
tion)+ (10 if (temp_color==answer[num_so_far+1]) else 0))
    return score[num_so_far][origin]

start_time = time.time()

maximum = 0

for destination, temp_color in connection[0]:
    maximum = max(maximum, longest(0, destination) + (10 if (temp_color==answer[0]) else 0))

print(maximum)

print("start_time", start_time) #출력해보면, 시간형식이 사람이 읽기 힘든 일련번호형식입니다.
print("--- %s seconds ---" %(time.time() - start_time))
```

결과 성공!! 히득 --- 0.0057544708251953125 seconds --- 다른 코드에 비해서 코드가 짧을 땐 더 안 좋은 효율을 보이지만 커지면 더 효과적이다.