**1. Train Rearrangement**

**문제:** T자형 철도가 있다고 가정하자. (a)에서 오는 열차가 (b)를 지나쳐서 (c)로 간다고 했을 때,

(a)로부터 오는 열차가 가는 순서를 increasing order로 재조정하려고 한다.

주어진 열차의 순서를 순서대로 재배열할 수 있는지 여부를 파악하는 method를 제안해라.

**설계:** 본인은 이동하는 순서를 재배열하기 위해서 (a)구간은 열차가 오는 순서를 저장한 배열로,

(b)구간은 잠시 열차가 멈추도록 하는 Stack으로 삼고, (c)는 현재 가야 하는 열차의 번호를

저장하고자 한다. 기차의 번호는 1~N까지 주어지기 때문에 (c)의 초기값은 1로 둔다.

또한 Stack은 LIFO구조이고 (b)에서 (c)로 가기 때문에 열차가 재배열되어 순서대로 들어가려면

Stack에 저장되는 원소들의 순서도 오름차순으로 저장되어야 한다.

먼저 기차의 번호와 Stack의 top을 비교하고 그 다음 Stack의 top과 (c)의 번호를 비교하고자 한다.

1. Stack에 top이 없다면 기차는 바로 Stack에 push해준다. 그 다음 (c)와 비교하여 그 기차의 번호가

(c)의 번호와 같다면 pop & (c)번호 증가, 아니면 다음 기차로 넘어간다.

2. 기차의 번호가 Stack의 top보다 작다면 Stack의 오름차순 규칙을 어기지 않으므로 Stack에

push한다.그 다음 (c)의 번호와 Stack의 top을 비교하여 같다면 pop과 (c)++을 해주는데, 이때

한번만 pop을 해주는 것이 아니라 (c)의 번호와 다를 때까지 pop과 (c)++을 반복한다.

한번만 pop하는 게 아니라 (c)의 번호와 다를 때까지 반복하는 이유는 Stack의 top이 충분히 (c)로

갈 수 있는 상황인데 다음 기차로 넘어가면 그 기차는 Stack에도 못 들어가고 그대로 재배치

불가로 결론이 나버릴 수 있기 때문이다.

도표, 디자인이(가) 표시된 사진

자동 생성된 설명

만약 기차의 번호가 Stack의 top보다 크다면 Stack의 오름차순 규칙을 어기므로 Stack에 push하지

못하고 따라서 (c)로도 못 가므로 그대로 기차 재배열 불가로 결론짓는다.

(c)로도 못 가는 이유는 이전 연산에서 (b)의 top과 (c)의 번호를 비교해서 pop을 한 결과를 가지고

현재 연산을 하는 건데, 현재 기차의 번호가 Stack의 top보다 크다는 의미는 당연히 (c)보다 기차의

번호가 크다는 의미이므로 더 이상의 연산은 의미가 없다.

도표, 라인, 디자인이(가) 표시된 사진

자동 생성된 설명 도표, 라인, 원, 디자인이(가) 표시된 사진

자동 생성된 설명

재배열이 불가능한 경우를 만나지 않는 이상 위의 3가지 과정을 (c)가 N이 될 때까지 반복한다.

**시간 복잡도**: (a)에서 (b)로의 이동은 기차번호와 Stack top의 비교와 Stack의 push 총 두번의 연산이

일어날 수 있다. 또한 push를 안 하는 상황은 재배열 불가 상황밖에 없으므로 최대 2n번의

연산이 가능하다. (b)에서 (c)로의 이동은 (c)의 번호와 Stack top의 비교와 Stack pop 총 두번의

연산이 발생한다. 만약 기차의 순서가 내림차순으로 되어 있다면 총 2n번의 연산이 이루어진다.

따라서 최종적인 시간 복잡도는 T(n) = 4n 🡪 O(n) 이다..

**공간 복잡도**: 기차의 번호를 저장할 n크기의 배열과 n크기의 Stack이 필요하고, 지나가야 할 기차 번호를

저장할 변수도 하나 필요하므로 최종적인 공간 복잡도는 T(n) = 2n+1 🡪 O(n) 이다.

**2. Agony of Engineer**

**문제:** 두 건물 A, B가 있을 때, 각 건물의 터미널이 N개가 있고 두 건물을 연결하는 N개의 전선이 있다.

건물의 모든 터미널은 해당 건물의 제어 상자에 있고, N은 2보다 크다고 한다.

한 엔지니어는 한 건물의 터미널이 전선으로 다른 건물과 어떻게 연결되는지 알아내려고 한다.

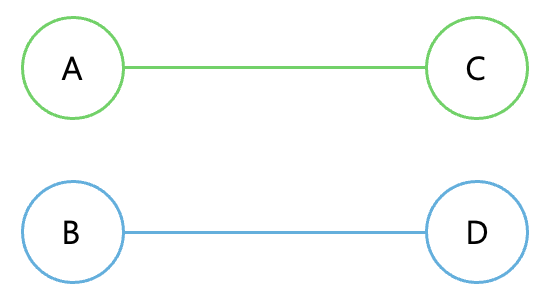
전구, 배터리, 전선을 사용하여 두 개 이상의 전선 간의 연결성을 확인할 수 있다.

건물 간 이동 횟수를 최소화하면서 터미널 간의 연결성을 찾는 알고리즘을 제안해라.

**설계:** 본인의 알고리즘은 먼저 전구에 불이 들어오는 후보 쌍들을 찾고, 찾은 후보 쌍들에 대해서 검증을

하는 방식을 취했다. 검증이 필요한 이유는 후보 쌍이 연결될 수 있는 경우의 수가 아래의 그림처럼

두개이기 때문이다.

1) 2) 원, 도표, 라인이(가) 표시된 사진

자동 생성된 설명

후보 쌍을 찾을 때는 먼저 한 건물에서 가장 가까운 두 터미널과 연결된 다른 건물의 터미널을

체크한다. 그 다음에는 이동한 건물에서 또 가장 가까운 두 터미널과 연결된 다른 건물의 터미널을

체크한다. 이렇게 아래의 그림처럼 모든 후보쌍들을 찾는다.

도표, 원, 라인, 디자인이(가) 표시된 사진

자동 생성된 설명 도표, 라인, 원, 디자인이(가) 표시된 사진

자동 생성된 설명

도표, 원, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명 도표, 원, 라인, 디자인이(가) 표시된 사진

자동 생성된 설명

모든 후보쌍들을 찾았다면 이제 그 연결이 유효한 연결인지 확인하는 절차가 필요하다.

후보 쌍에는 두 터미널의 연결이 존재하는데, 굳이 두 터미널의 연결을 모두 확인할 필요없이

한 터미널의 연결만 확인하면 다른 터미널의 연결도 자동으로 확정이 되기 때문에

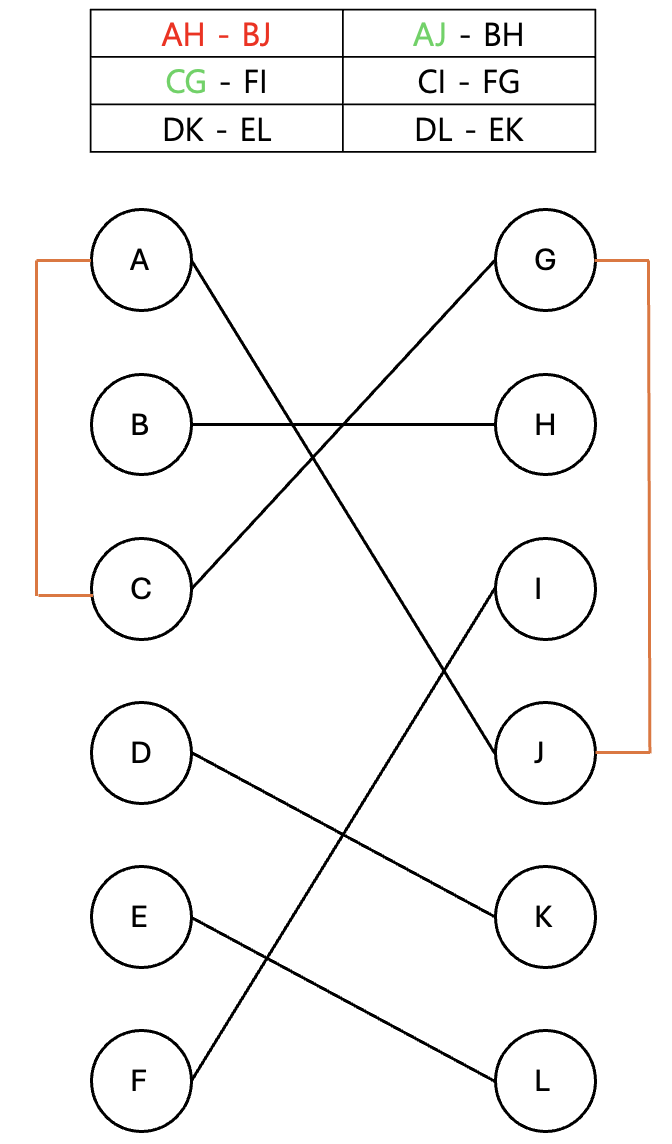
한 터미널의 연결만 검증하면 된다. 검증을 하기 위해서는 각 건물에 두 터미널을 연결하여

전구에 불이 들어오는지 확인해야 한다. 이때, 이미 후보쌍으로 짝지어져 있는 두 터미널을

연결하는 것은 검증의 의미가 없으므로 서로 다른 후보쌍의 두 터미널을 연결한 뒤 다른 건물로

가서 연결을 확인해야 한다.

도표, 라인, 스크린샷, 원이(가) 표시된 사진

자동 생성된 설명 

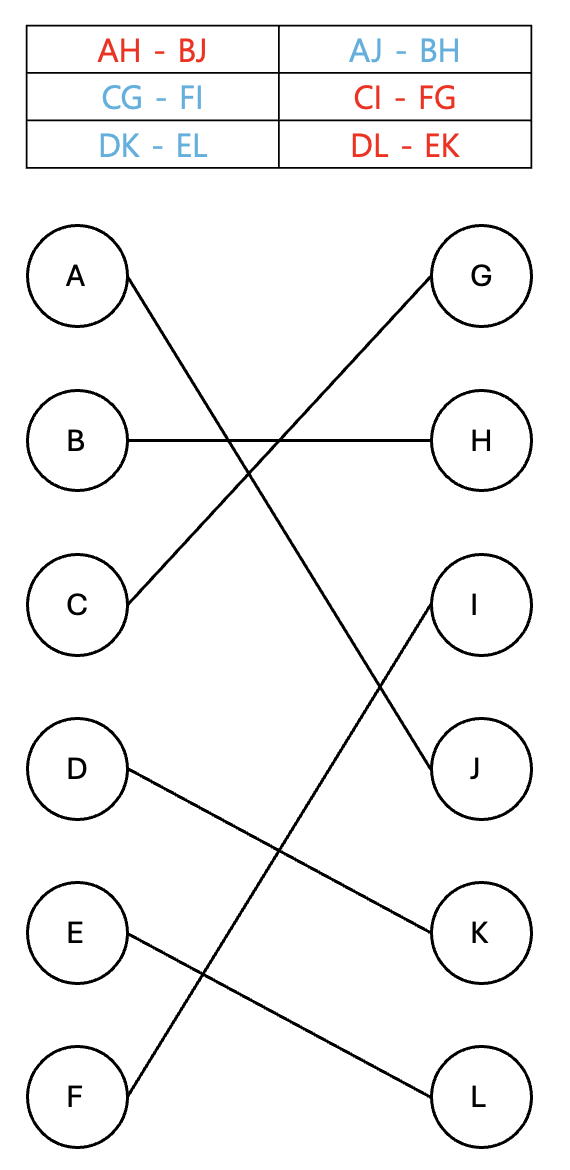
A와 C를 연결하고 다른 건물로 이동한 뒤 G와 H를 연결하여 전구에 불이 들어오는지 확인한다.

하지만 연결이 유효하지 않아 전구에 불이 들어오지 않기 때문에 첫번째 후보쌍의 다른 경우

A와 J의 연결을 확인한다. 이 연결은 유효한 경우이므로 전구에 불이 들어온다.

따라서 첫번째 후보쌍과 두번째 후보쌍의 연결 상태를 동시에 확정 지을 수 있게 됐다.

도표, 스크린샷, 라인, 원이(가) 표시된 사진

자동 생성된 설명 

그 다음 후보쌍도 검증하기 위해서 유효한 C, G연결은 그대로 두고 D, K 연결을 확인하기 위해

J에 연결되어 있던 전선을 K로 옮기고, 다른 건물로 이동해서 A에 연결되어 있던 전선을 D로

옮긴다. 이 연결도 유효한 경우가 되므로 전구에 불이 들어와 이 후보쌍의 연결 상태도 확인했다.

따라서 이렇게 모든 터미널 간의 연결 상태를 확인할 수 있게 됐다.

**시간 복잡도**: 먼저 후보쌍을 찾을 때, 단순히 생각해보면 N/2번 이동해야 모든 후보쌍을 찾을 수 있는

것처럼 보이지만 쌍을 찾아야 할 터미널이 두개 또는 한 개만 있다면 굳이 다음 건물로

넘어가서 확인할 필요없이 반대편 건물의 남은 터미널과 짝을 지어주면 된다.

따라서 후보쌍을 찾기 위해 건물을 이동해야 하는 횟수는 번이다.

검증을 하기 위해서는 현재 있는 건물에서 터미널을 연결하고, 다른 건물로 넘어가 터미널을

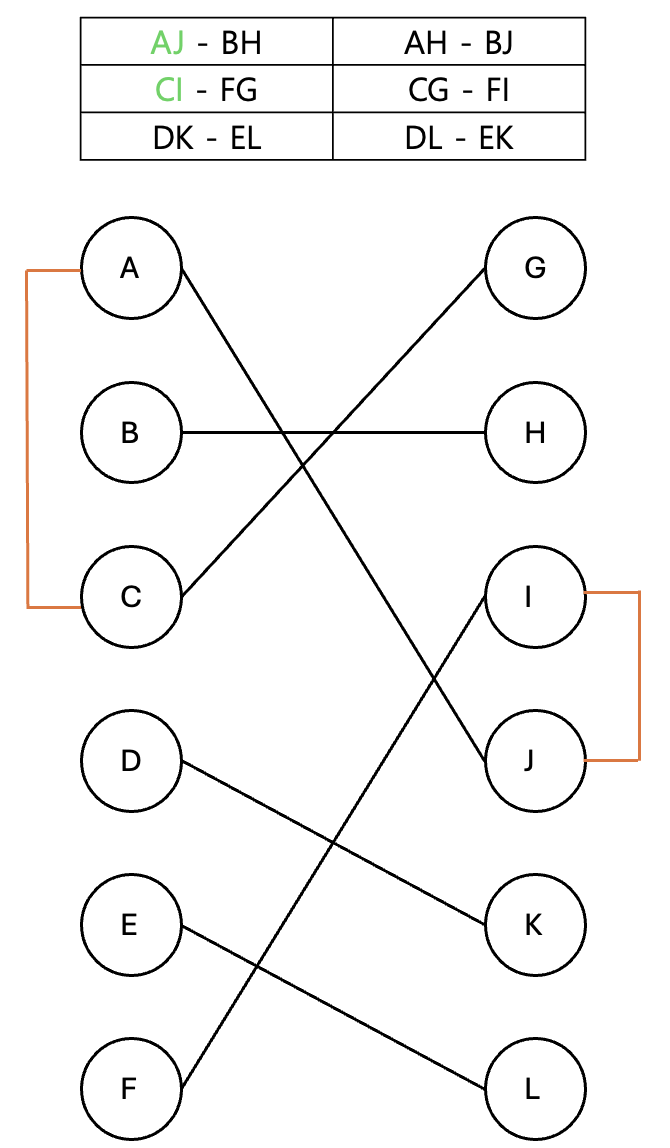
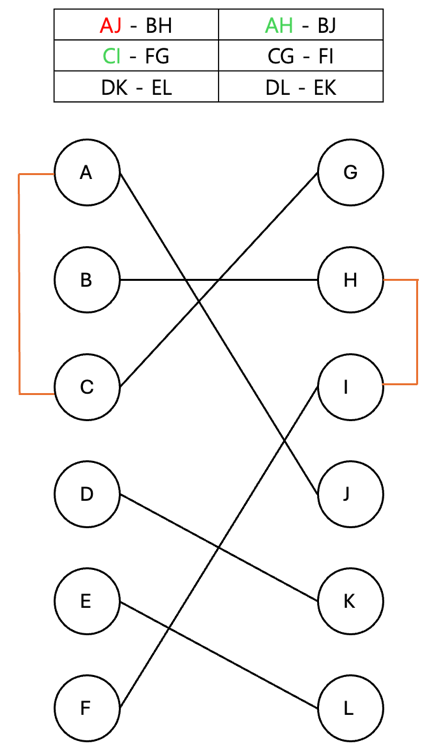
연결해야 한다. 만약 유효하지 않다면 건물을 이동하지 않고 바로 다른 경우에 대해서 검증을

한다. 만약 그 경우도 유효하지 않다고 한다면, 다른 경우를 확인하지 않은 후보쌍이 유효하지

않다는 의미이고, 그 말은 다른 경우가 유효한 경우임을 자연스레 알 수 있다. 이제 한 쪽에

대해서는 알았기 때문에 처음에 확인한 후보쌍이 유효하면 끝, 유효하지 않더라도

다른 경우가 유효하다는 것을 알 수 있다. 아래의 그림을 보면 더 이해하기 쉬울 것이다.

- A, J 연결과 C, I 연결 검증 - A, H 연결과 C, I 연결 검증

🡪 실패 🡪 실패

도표, 라인, 스크린샷, 그래프이(가) 표시된 사진

자동 생성된 설명 도표, 라인, 스크린샷, 그래프이(가) 표시된 사진

자동 생성된 설명

- 자동으로 C, G 연결 유효 확정 - 자동으로 A, J 연결 유효 확정

- A, H 연결과 C, G 연결 검증 🡪 **검증 성공**

🡪 실패

이를 통해 두 후보쌍을 검증하기 위해 건물 간 이동은 한번만 하면 된다는 것을 알 수 있다.

따라서 후보쌍을 검증하기 위해 건물을 이동해야 하는 횟수는 번이다.

\*건물의 터미널의 개수가 홀수라면 번이다. 왜냐하면 N-1개의 후보쌍을 검증한 뒤에는

한 쌍밖에 남지 않아서 자동으로 유효한 경우로 정해지기 때문이다.

따라서 최종적으로 모든 터미널의 연결 상태를 확인하기 위해서 건물간 이동해야 하는

최소 횟수는 짝수 : + 번 , 홀수 : 번이다.

**공간 복잡도**: 두 건물의 총 터미널 2n개를 저장할 공간이 필요하고, 한 후보쌍에 필요한 공간을 4라고

한다면 모든 후보쌍의 2가지 경우를 저장하기 위해서 4n 크기의 공간이 필요하다.

따라서 최종적인 공간 복잡도는 T(n) = 6n 🡪 O(n)이다.