알고리즘2 (2024-2)

5. 복합 문제 연습 1

국립금오공과대학교 컴퓨터공학과

김 경 수

학습 목표

- 지금까지 배운 지식을 활용하여 다양한 난이도를 갖는 문제들을 제한된 시간 내에 직접 풀어볼 수 있다.

문제 1. 폰켓몬

문제 1. 폰켓몬 (1/4)

제한시간: 30분

- 당신은 폰켓몬을 잡기 위한 오랜 여행 끝에, 홍 박사님의 연구실에 도착했습니다.
- 홍 박사님은 당신에게 자신의 연구실에 있는 총 N 마리의 폰켓몬 중에서 N/2마리를 가져가도 좋다고 했습니다.
- 홍 박사님 연구실의 폰켓몬은 종류에 따라 번호를 붙여 구분합니다. 따라서 같은 종류의 폰켓몬은 같은 번호를 가지고 있습니다.
- 예를 들어 연구실에 총 4마리의 폰켓몬이 있고, 각 폰켓몬의 종류 번호가 [3번, 1번, 2번, 3번] 이라면 이는 3번 폰켓몬 두 마리, 1번 폰켓몬 한 마리, 2번 폰켓몬 한 마리가 있음을 나타냅니다.
- 이때, 4마리의 폰켓몬 중 2마리를 고르는 방법은 다음과 같이 6가지가 있습니다.

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/1845

문제 1. 폰켓몬 (2/4)

제한시간: 30분

- 첫 번째(3번), 두 번째(1번) 폰켓몬을 선택
- 첫 번째(3번), 세 번째(2번) 폰켓몬을 선택
- 첫 번째(3번), 네 번째(3번) 폰켓몬을 선택
- 두 번째(1번), 세 번째(2번) 폰켓몬을 선택
- 두 번째(1번), 네 번째(3번) 폰켓몬을 선택
- 세 번째(2번), 네 번째(3번) 폰켓몬을 선택

- 이때, 첫 번째(3번) 폰켓몬과 네 번째(3번) 폰켓몬을 선택하는 방법은 한 종류(3번 폰켓몬 두 마리)의 폰켓몬만 가질 수 있지만, 다른 방법들은 모두 두 종류의 폰켓몬을 가질 수 있습니다.
- 따라서 위 예시에서 가질 수 있는 폰켓몬 종류 수의 최댓값은 2가 됩니다.
- 당신은 최대한 다양한 종류의 폰켓몬을 가지길 원하기 때문에, 최대한 많은 종류의 폰켓몬을 포함해서 N/2마리를 선택하려 합니다.

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/1845

문제 1. 폰켓몬 (3/4)

제한시간: 30분

- N마리 폰켓몬의 종류 번호가 담긴 배열 nums가 매개변수로 주어질 때, N/2마리의 폰켓몬을 선택하는 방법 중, 가장 많은 종류의 폰켓몬을 선택하는 방법을 찾아, 그때의 폰켓몬 종류 번호의 개수를 return 하도록 **solution1 함수**를 완성해주세요.

- **제한사항**
  - nums는 폰켓몬의 종류 번호가 담긴 1차원 배열입니다.
  - nums의 길이(N)는 1 이상 10,000 이하의 자연수이며, 항상 짝수로 주어집니다.
  - 폰켓몬의 종류 번호는 1 이상 200,000 이하의 자연수로 나타냅니다.
  - 가장 많은 종류의 폰켓몬을 선택하는 방법이 여러 가지인 경우에도, 선택할 수 있는 폰켓몬 종류 개수의 최댓값 하나만 return 하면 됩니다.

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/1845

문제 1. 폰켓몬 (4/4)

제한시간: 30분

- **입출력 예제 설명**

nums	result
[3,1,2,3]	2
[3,3,3,2,2,4]	3
[3,3,3,2,2,2]	2

➢ **입출력 예 #1**

- 문제의 예시와 같습니다.

➢ **입출력 예 #2**

- 6마리의 폰켓몬이 있으므로, 3마리의 폰켓몬을 골라야 합니다.
- 가장 많은 종류의 폰켓몬을 고르기 위해서는 3번 폰켓몬 한 마리, 2번 폰켓몬 한 마리, 4번 폰켓몬 한 마리를 고르면 되며, 따라서 3을 return 합니다.

➢ **입출력 예 #3**

- 6마리의 폰켓몬이 있으므로, 3마리의 폰켓몬을 골라야 합니다.
- 가장 많은 종류의 폰켓몬을 고르기 위해서는 3번 폰켓몬 한 마리와 2번 폰켓몬 두 마리를 고르거나, 혹은 3번 폰켓몬 두 마리와 2번 폰켓몬 한 마리를 고르면 됩니다.
- 따라서 최대 고를 수 있는 폰켓몬 종류의 수는 2입니다.

★ **문제 출처:** https://school.programmers.co.kr/learn/courses/30/lessons/1845

문제 1. 문제 분석

- **문제의 목표**
  - N/2 마리의 폰켓몬을 선택하는 방법 중 가장 많은 종류의 폰켓몬 종류 수를 반환한다.

- **문제 해결 시 유의 사항**
  - 동일한 종류의 폰켓몬을 뽑는 경우 하나로 처리함

문제 1. 문제 해결 전략 수립

- 폰켓몬 종류가 담긴 배열의 크기가 N일 때, N/2 = k 개만큼 폰켓몬을 선택함
  - 만약 k 값이 중복을 제거한 nums의 길이보다 작다면, k값을 정답으로 반환함
    - 이유: 중복을 제거한 nums보다 k가 작으므로 k개만큼 뽑을 수 있으므로...
  - K값이 중복을 제거한 nums의 길이보다 크다면, 중복을 제거한 nums의 길이를 정답으로 반환함
    - 이유: k값이 아무리 커도 중복을 제거한 nums를 초과하면 의미가 없으므로...
  - 이때 **중복 제거를 위해서 사용 가능한 자료구조 → 집합(Set)**

문제 1. 구현 및 검증

```python
def solution1(nums):
    num_set = set(nums)
    n = len(nums) # 총 폰켓몬의 수
    k = n // 2    # 선택할 폰켓몬의 수

    # 중복을 제거한 폰켓몬의 종류의 수와 선택할 폰켓몬의 갯수 중 작은 값을 반환
    return min(k, len(num_set))
```

테스트 케이스

```python
nums1 = [3, 1, 2, 3]
print( solution1(nums1) )

nums2 = [3, 3, 3, 2, 2, 4]
print( solution1(nums2) )
```

문제 2. 섬 연결하기

문제 2. 섬 연결하기 (1/2)

제한시간: 70분

- n개의 섬 사이에 다리를 건설하는 비용(costs)이 주어질 때, 최소의 비용으로 모든 섬이 서로 통행 가능하도록 만들 때 필요한 최소 비용을 return 하도록 **solution2 함수**를 완성하세요.
- 이때 다리를 여러 번 건너더라도, 도달할 수만 있으면 통행 가능하다고 봅니다.
- 예를 들어 A 섬과 B 섬 사이에 다리가 있고, B 섬과 C 섬 사이에 다리가 있으면 A 섬과 C 섬은 서로 통행 가능합니다.
- **제한사항**
  - 섬의 개수 n은 1 이상 100 이하이고, costs의 길이는 ((n-1) * n) / 2이하입니다.
  - 임의의 i에 대해, costs[i][0] 와 costs[i] [1]에는 다리가 연결되는 두 섬의 번호가 들어있고, costs[i] [2]에는 이 두 섬을 연결하는 다리를 건설할 때 드는 비용입니다.
  - 같은 연결은 두 번 주어지지 않습니다. 또한 순서가 바뀌더라도 같은 연결로 봅니다. 즉 0과 1 사이를 연결하는 비용이 주어졌을 때, 1과 0의 비용이 주어지지 않습니다.
  - 모든 섬 사이의 다리 건설 비용이 주어지지 않습니다. 이 경우, 두 섬 사이의 건설이 불가능한 것으로 봅니다.
  - 연결할 수 없는 섬은 주어지지 않습니다.

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/42861

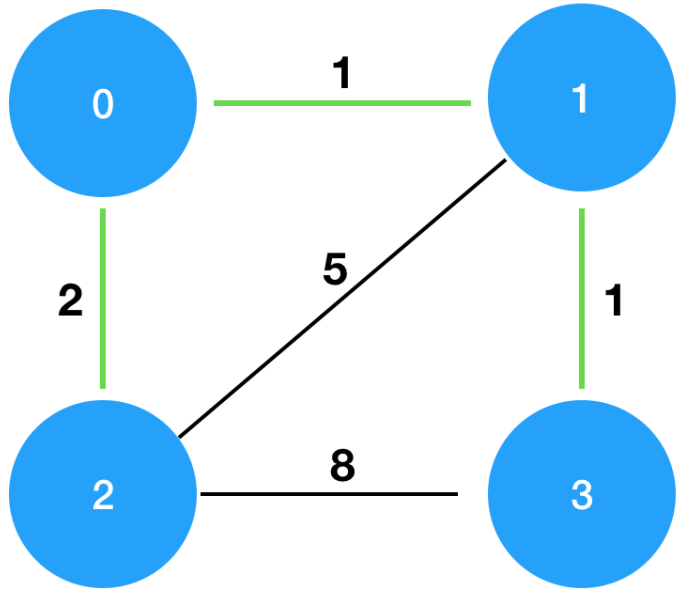
문제 2. 섬 연결하기 (2/2)

제한시간: 70분

- **입출력 예제 설명**

n	costs	return
4	[[0,1,1],[0,2,2],[1,2,5],[1,3,1],[2,3,8]]	4

➢ costs를 그림으로 표현하면 다음과 같으며, 이때 초록색 경로로 연결하는 것이 가장 적은 비용으로 모두를 통행할 수 있도록 만드는 방법입니다.



★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/42861

문제 2. 문제 분석

- **문제의 목표**
  - ① 모든 섬을 연결해야 한다.
  - ② 연결한 다리들을 건설하는 비용의 합이 최소가 되어야 한다.
- **문제 해결 절차 기술**
  - ➢ 각 섬 사이의 다리를 건설하는 비용을 오름차순으로 정렬
  - ➢ 비용이 작은 다리부터 선택해 섬을 연결 **(← Greedy algorithm)**
  - ➢ N-1개의 다리가 선택될 때까지 위 두 과정을 반복함 (이유: N개의 섬을 연결하기 위해서는 N-1개의 다리가 필요함)
- **유의사항**
  - ➢ 건설 비용을 최소화하기 위해 다리를 추가할 때 **사이클이 형성되지 않도록 함**.

문제 2. 문제 해결 전략 수립

- 이 문제는 **최소 비용 신장 트리(minimum cost spanning tree)**로 모델링 하여 해결할 수 있다.

- **최소 비용 신장 트리를 사용하는 문제의 특성**
  ① 그래프 자료구조로 모델링이 가능함
  ② "최단 거리", "최소 비용"과 같은 키워드가 등장함
  ③ 간선에 가중치가 존재함
  ④ "사이클이 형성되면 안됨"이라는 제한 조건이 직/간접적으로 등장함

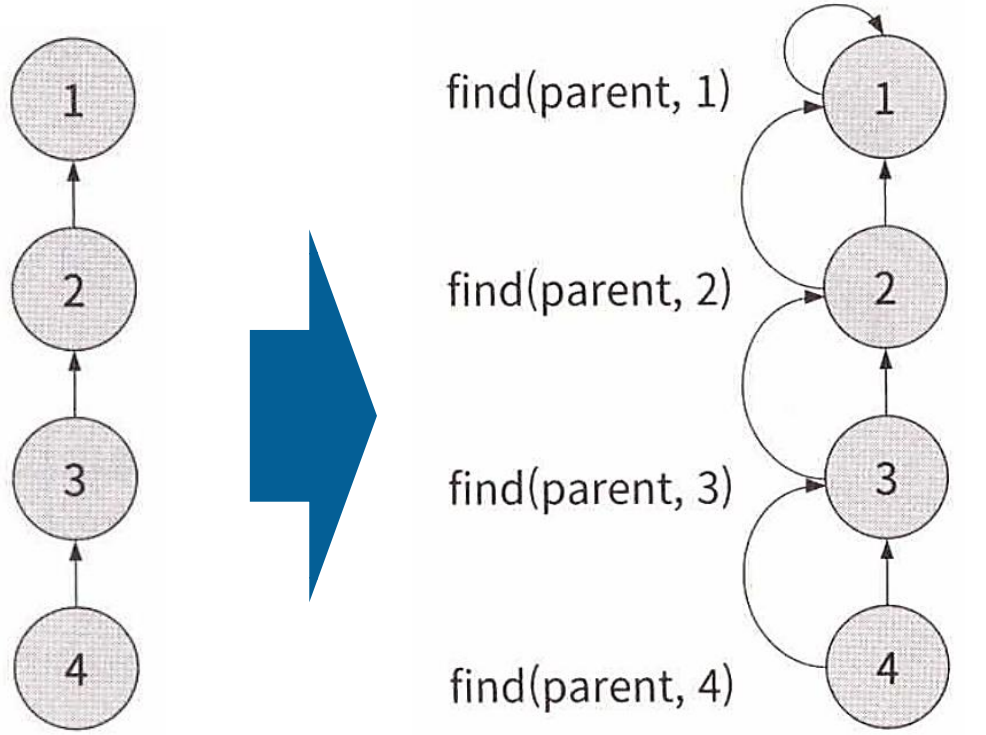
문제 2. 문제 해결 전략 수립

- **앞서 문제 목표 분석 단계에서 특히 중요한 부분**
  - ➢ 다리를 추가할 때 사이클이 형성되지 않도록 해야 한다.
    - ✓ 사이클이 형성되면 최소 비용을 달성할 수 없기 때문…
  - ➢ 사이클이 형성되는지 여부를 확인하는 방법
    - ✓ Kruskal algorithm을 사용하는 경우: Union-Find
    - ✓ Prim algorithm을 사용하는 경우: 별도로 사이클 형성 여부를 체크할 필요 없음

- **Union-Find**
  - ➢ 분리-집합(disjoint set)을 표현하기 위한 트리 형태의 자료구조
  - ➢ 두 집합에 대한 합집합 연산인 "**union**" 연산과 특정 원소가 어떤 집합에 속하는지 반환하는 "**find**" 연산으로 구성됨.

[참고] Union-Find 연산

```
def find(parent, i):
    # 원소 i가 속한 집합의 루트 노드를 찾는다.
    if parent[i] == i:
        return i

    # 원소 i를 루트 노드의 자식으로 설정한다.
    parent[i] = find(parent, parent[i])
    return parent[i]
```

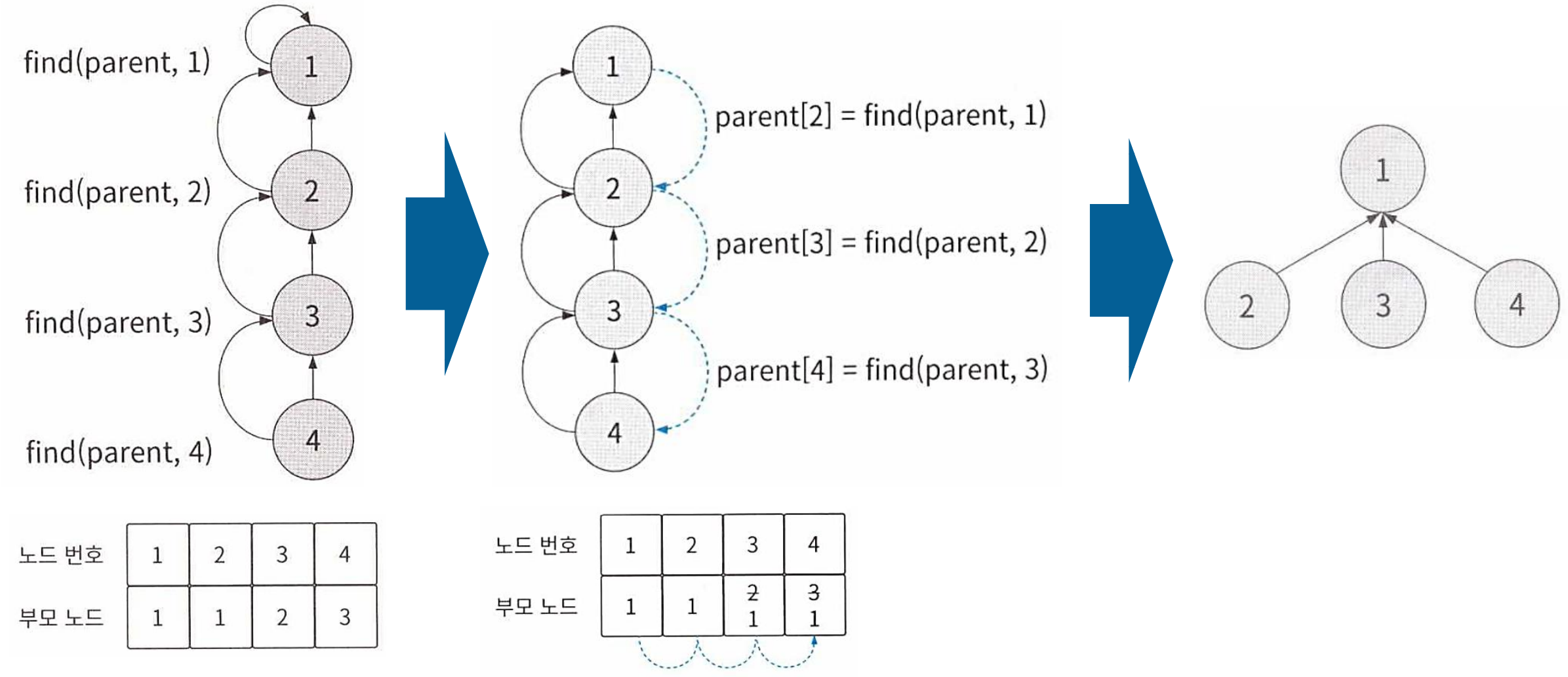


노드 번호	1	2	3	4
부모 노드	1	1	2	3

노드 번호	1	2	3	4
부모 노드	1	1	2	3

▲ (예) find(parent, 4)를 호출하였을 때 경로 압축 과정

[참고] Union-Find 연산



노드 번호	1	2	3	4
부모 노드	1	1	2	3

노드 번호	1	2	3	4
부모 노드	1	1	~~2~~ 1	~~3~~ 1

▲ (예) find(parent, 4)를 호출하였을 때 경로 압축 과정

[참고] Union-Find 연산

```
def union(parent, rank, x, y):
    # 두 원소가 속한 집합의 루트 노드를 찾는다.
    xroot = find(parent, x)
    yroot = find(parent, y)

    # 랭크를 기준으로 두 집합을 합친다.
    # 작은 랭크의 트리를 큰 랭크의 트리 아래에 연결
    if rank[xroot] < rank[yroot]:
        parent[xroot] = yroot
    elif rank[xroot] > rank[yroot]:
        parent[yroot] = xroot
    else:
        # 랭크가 같은 경우 한 트리를 다른 트리에 붙이고 랭크를 증가시킴
        parent[yroot] = xroot
        rank[xroot] += 1
```

- 높이가 더 높은 트리가 높이가 낮은 트리 밑으로 합쳐지는 경우 트리의 깊이가 점점 증가할 수 있음.
- 따라서, **높이가 더 낮은 트리가 높은 트리 밑으로 합쳐져야 함.**
- 이를 위해 트리의 **rank**가 사용됨.

  (※ 트리의 rank: 트리의 높이 또는 사이즈)

문제 2. 구현 및 검증

```
def solution2(n, costs):
    # 비용을 기준으로 간선을 오름차순으로 정렬
    costs.sort(key=lambda x: x[2])

    # 각 노드의 부모를 추적하는 parent 배열 생성
    parent = [i for i in range(n)]

    # 각 노드의 트리의 랭크를 추적하는 rank 배열 생성
    rank = [0] * n

    min_cost = 0
    edges = 0
```

문제 2. 구현 및 검증

```
for edge in costs:
    if edges == n-1:  # n-1개의 다리가 건설되었으면 중단한다.
        break

    # 현재 간선의 두 노드가 속한 집합의 루트 노드 x, y
    x = find(parent, edge[0])
    y = find(parent, edge[1])

    # 현재 간선의 두 노드가 서로 다른 집합에 속한 경우 두 집합을 합친다.
    if x!= y:
        union(parent, rank, x, y)
        min_cost += edge[2]  # 현재 간선의 비용을 최소 비용에 추가한다.
        edges += 1  # 포함된 간선(다리)의 개수를 증가시킨다.

return min_cost
```

문제 2. 구현 및 검증

테스트 케이스

```
n = 4
costs = [[0,1,1], [0,2,2], [1,2,5], [1,3,1], [2,3,8]]
print( solution2(n, costs) )
```

문제 3. 카드 뭉치

문제 3. 카드 뭉치 (1/3)

제한시간: 40분

- 코니는 영어 단어가 적힌 카드 뭉치 두 개를 선물로 받았습니다. 코니는 다음과 같은 규칙으로 카드에 적힌 단어들을 사용해 원하는 순서의 단어 배열을 만들 수 있는지 알고 싶습니다.
  - 원하는 카드 뭉치에서 카드를 순서대로 한 장씩 사용합니다.
  - 한 번 사용한 카드는 다시 사용할 수 없습니다.
  - 카드를 사용하지 않고 다음 카드로 넘어갈 수 없습니다.
  - 기존에 주어진 카드 뭉치의 단어 순서는 바꿀 수 없습니다.
- 예를 들어 첫 번째 카드 뭉치에 순서대로 ["i", "drink", "water"], 두 번째 카드 뭉치에 순서대로 ["want", "to"]가 적혀있을 때 ["i", "want", "to", "drink", "water"] 순서의 단어 배열을 만들려고 한다면 첫 번째 카드 뭉치에서 "i"를 사용한 후 두 번째 카드 뭉치에서 "want"와 "to"를 사용하고 첫 번째 카드뭉치에 "drink"와 "water"를 차례대로 사용하면 원하는 순서의 단어 배열을 만들 수 있습니다.
- 문자열로 이루어진 배열 cards1, cards2와 원하는 단어 배열 goal이 매개변수로 주어질 때, cards1과 cards2에 적힌 단어들로 goal를 만들 있다면 "Yes"를, 만들 수 없다면 "No"를 return하는 **solution3 함수**를 완성해주세요.

★ **문제 출처:** https://school.programmers.co.kr/learn/courses/30/lessons/159994

문제 3. 카드 뭉치 (2/3)

제한시간: 40분

- **제한사항**
  - 1 ≤ cards1의 길이, cards2의 길이 ≤ 10
    - 1 ≤ cards1[i]의 길이, cards2[i]의 길이 ≤ 10
    - cards1과 cards2에는 서로 다른 단어만 존재합니다.
  - 2 ≤ goal의 길이 ≤ cards1의 길이 + cards2의 길이
    - 1 ≤ goal[i]의 길이 ≤ 10
    - goal의 원소는 cards1과 cards2의 원소들로만 이루어져 있습니다.
  - cards1, cards2, goal의 문자열들은 모두 알파벳 소문자로만 이루어져 있습니다.

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/159994

문제 3. 카드 뭉치 (3/3)

제한시간: 40분

- **입출력 예제 설명**

cards1	cards2	goal	result
["i", "drink", "water"]	["want", "to"]	["i", "want", "to", "drink", "water"]	"Yes"
["i", "water", "drink"]	["want", "to"]	["i", "want", "to", "drink", "water"]	"No"

➢ **입출력 예 #1**

- 문제의 설명과 같음

➢ **입출력 예 #2**

- cards1에서 "i"를 사용하고 cards2에서 "want"와 "to"를 사용하여 "i want to"까지는 만들 수 있지만 "water"가 "drink"보다 먼저 사용되어야 하기 때문에 해당 문장을 완성시킬 수 없습니다.
- 따라서 "No"를 반환합니다.

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/159994

문제 3. 문제 분석

- 문제에 주어진 핵심 키워드 파악
  - "앞에서부터 순서대로 …"
  - "앞의 카드를 반드시 사용해야 한다."
- 순차 로직에서 고려해야 할 자료구조
  - ✓ **배열 인덱스 활용**: 단일 인덱스, 투 포인터, 슬라이딩 윈도우
  - ✓ **스택 활용**: 데이터의 처리 순서가 역순(LIFO)인 경우
  - ✓ **큐 활용**: 데이터의 처리 순서가 앞에서부터 순차적(FIFO)인 경우
- 본 문제의 특성
  - 전형적인 **순차 로직**을 갖는 문제
  - 따라서, 배열 인덱스를 이용한 탐색 또는 큐를 활용하여 문제를 해결하는 것이 가능.

문제 3. 문제 해결 전략 수립

- **기본적인 문제 해결 아이디어**
  - ➢ 리스트 “goal”의 문장이 완성될 때 까지 아래 단계를 반복 수행한다.
    - ① 리스트 “cards1”의 맨 앞 카드에 적힌 단어와 “goal”의 맨 앞의 단어를 비교한다.
      - 두 단어가 같다면, “cards1”과 “goal” 리스트 맨 앞의 카드를 함께 삭제한다.
      - 두 단어가 다르다면,
    - ② 리스트 “cards2”의 맨 앞 카드에 적힌 단어와 “goal”의 맨 앞의 카드를 비교한다.
      - 두 단어가 같다면, “cards2”와 “goal” 리스트 맨 앞의 카드를 함께 삭제한다.
      - 두 단어가 다르다면,
    - ③ 앞의 단계 ① 또는 ②에서 리스트 “goal”의 맨 앞에 위치한 카드를 삭제하지 못하였다면 “No” 반환하고 종료한다.
  - ➢ 리스트 “goal”의 모든 단어가 삭제되었다면 “Yes”를 반환하고 종료한다.

문제 3. 문제 해결 전략 수립

- **문제 해결을 위한 방법**
  - **방법 1:** 입력으로 주어진 “cards1”과 “cards2” 리스트를 앞에서부터 순차적으로 순회하면서 “goal” 내 단어들과 순서대로 일대일 매칭을 수행하여 문장을 구성해 나가는 방법
  - **방법 2:** “cards1”과 “cards2”, “goal” 리스트를 deque으로 생각하여, “cards1”과 “cards2”, “goal” deque의 맨 앞의 요소들을 pop하고 이들을 일대일 비교하면서 문장을 구성해 나가는 방법
  - 어떤 방법이 더욱 효과적인가?
    - 파이썬을 이용한다면 위 두 가지 방법 모두 손쉽게 구현할 수 있다.
    - 단, C, C++을 사용하는 경우 첫 번째 방법이 구현하기에 더욱 수월하다.

문제 3. 문제 해결 방법 기술

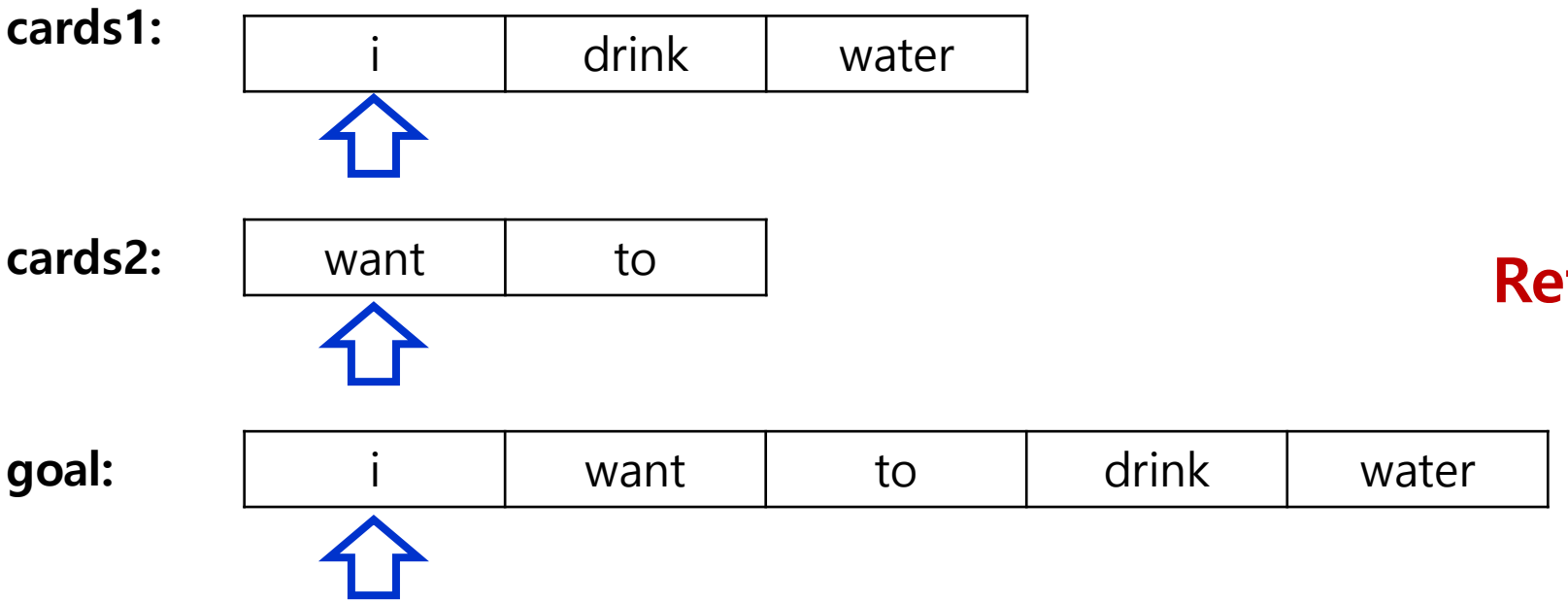
- **방법 1을 이용한 문제 해결 방법 기술**

```
// idx, jdx, kdx: 리스트 “cards1”, “cards2”, “goal”의 각 요소를 가리키는 인덱스
idx ← 1; jdx ← 1; kdx ← 1;
N ← length(goal);  N1 ← length(cards1);  N2 ← length(cards2);
while  kdx > N  do
    if  idx <=N1  AND  cards1[idx] == goal[kdx]  then  idx++; kdx++;
    else if  jdx <=N2  AND  cards2[jdx] == goal[kdx]  then  jdx++; kdx++;
    else  break;
if kdx > length(goal)  then  return True;
else return False;
```

문제 3. 문제 해결 방법 기술

- **방법 1을 이용한 문제 해결 방법 예제 (1)**

cards1	cards2	goal	result
["i", "drink", "water"]	["want", "to"]	["i", "want", "to", "drink", "water"]	"Yes"

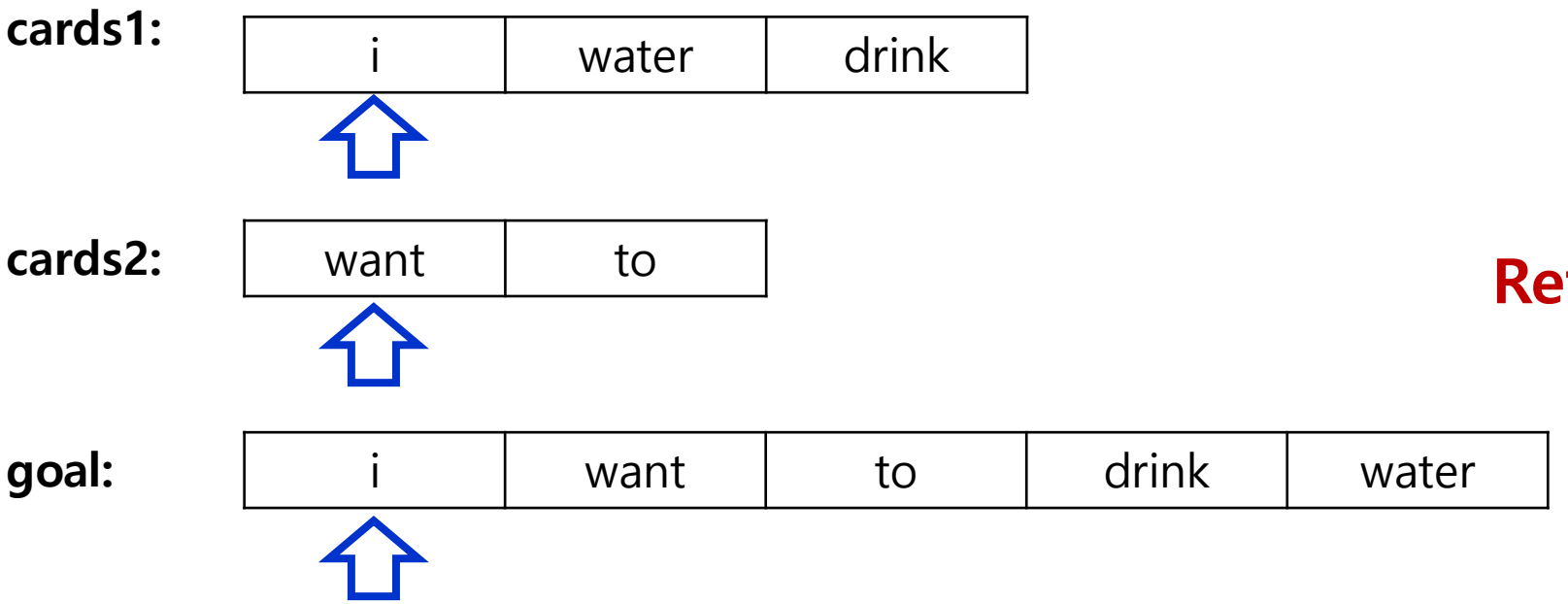


Return: "Yes"

문제 3. 문제 해결 방법 기술

- 방법 1을 이용한 문제 해결 방법 예제 (2)

cards1	cards2	goal	result
["i", "water", "drink"]	["want", "to"]	["i", "want", "to", "drink", "water"]	"No"



Return: "No"

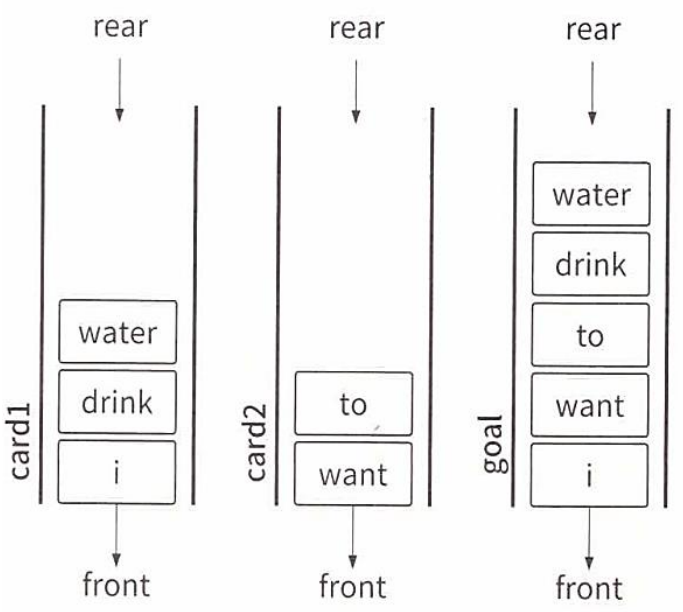
문제 3. 문제 해결 방법 기술

- **방법 2를 이용한 문제 해결 방법 기술**

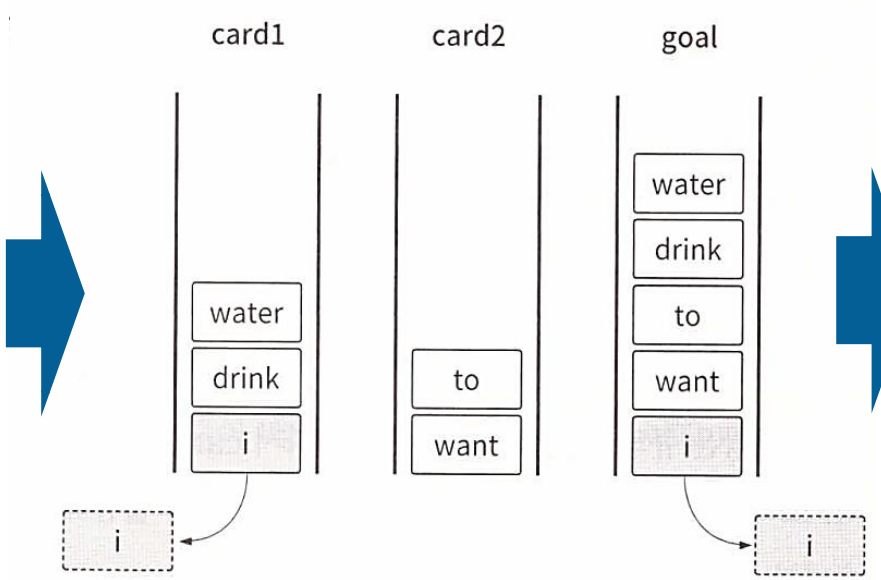
```
// cards1, cards2, goal을 deque으로 전환
cards1 ← deque(cards1);  cards2 ← deque(cards2);  goal ← deque(goal);
while  goal.isEmpty() == False  do
    if  cards1.isEmpty() == False  AND  cards1.left() == goal.left()  then
            cards1.popLeft();  goal.popLeft();
    else if cards2.isEmpty() == False  AND cards2.left() == goal.left()  then
            cards2.popLeft();  goal.popLeft();
    else  break;
if  goal.isEmpty() == True  then  return True;
else return False;
```

문제 3. 문제 해결 방법 기술

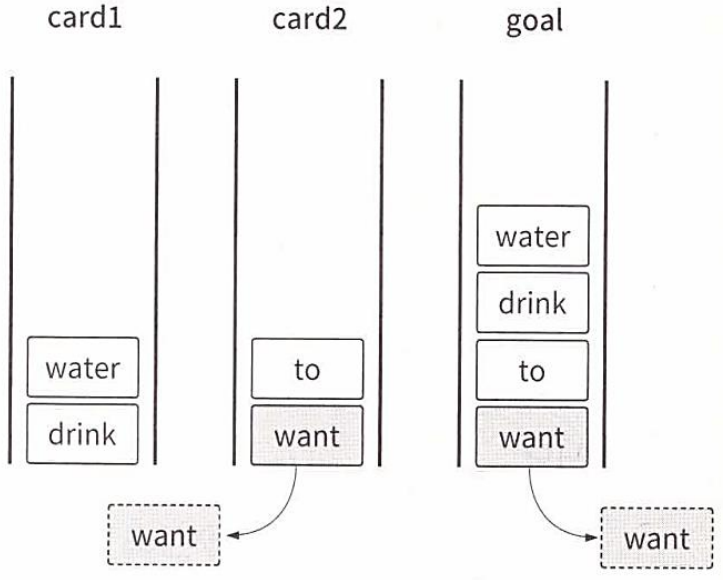
- 방법 2를 이용한 문제 해결 방법 예제



- 초기 상태



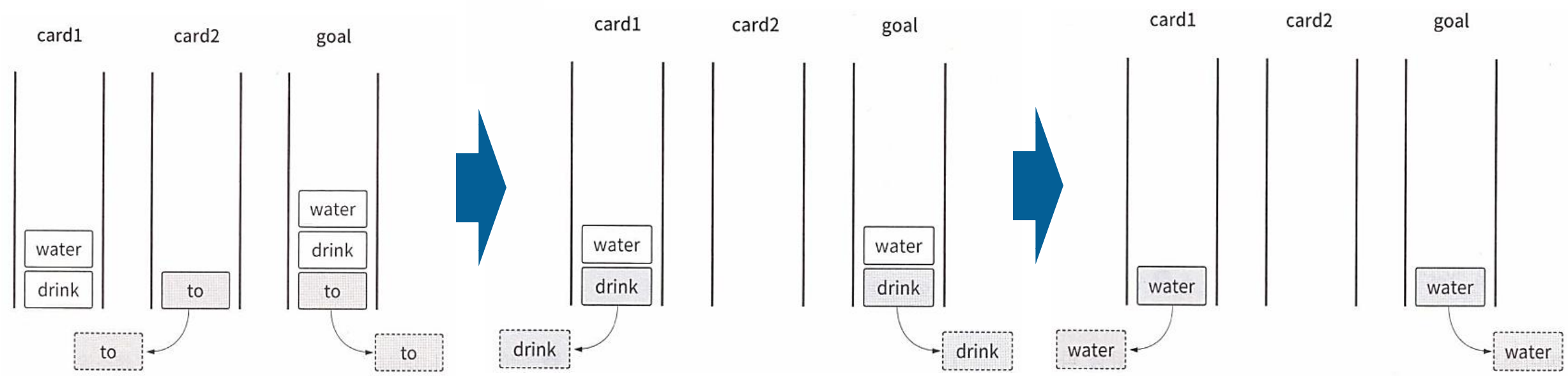
- card1의 front, goal의 front가 같으므로 둘 다 pop을 수행함.



- 같은 방법으로 card1, card2, goal의 Front를 비교함.
- 여기서는 card2와 goal이 같으므로 둘 다 pop을 수행함.

문제 3. 문제 해결 방법 기술

- **방법 2를 이용한 문제 해결 방법 예제**



- card2와 goal의 front가 같으므로 둘다 pop을 수행함.
- card2가 비었으므로, card1과 goal의 front만 비교함.
- 여기서는 두 리스트의 front가 같으므로 둘 다 pop을 수행함.
- card1과 goal의 front를 비교함.
- 두 리스트의 front가 같으므로 둘 다 pop을 수행함.

문제 3. 구현 및 검증

- 방법 1을 구현한 결과

```
def solution3(cards1, cards2, goal):
    idx = jdx = kdx = 0
    N1 = len(cards1)
    N2 = len(cards2)
    N = len(goal)

    while kdx < N:
        if idx < N1 and cards1[idx] == goal[kdx]:
            idx = idx + 1
            kdx = kdx + 1
        elif jdx < N2 and cards2[jdx] == goal[kdx]:
            jdx = jdx + 1
            kdx = kdx + 1
        else:
            return "No“

    return "Yes" if kdx >= N else "No"
```

문제 3. 구현 및 검증

- 방법 2를 구현한 결과

```
from collections import deque

def solution3(cards1, cards2, goal):
    cards1 = deque(cards1)
    cards2 = deque(cards2)
    goal = deque(goal)

    while goal:
        if cards1 and cards1[0] == goal[0]:
            cards1.popleft()
            goal.popleft()
        elif cards2 and cards2[0] == goal[0]:
            cards2.popleft()
            goal.popleft()
        else:
            break

    return "Yes" if not goal else "No"
```

문제 3. 구현 및 검증

- 테스트 케이스

```
# Test case 1
cards1 = ["i", "drink", "water"]
cards2 = ["want", "to"]
goal = ["i", "want", "to", "drink", "water"]

print(solution3(cards1, cards2, goal))
```

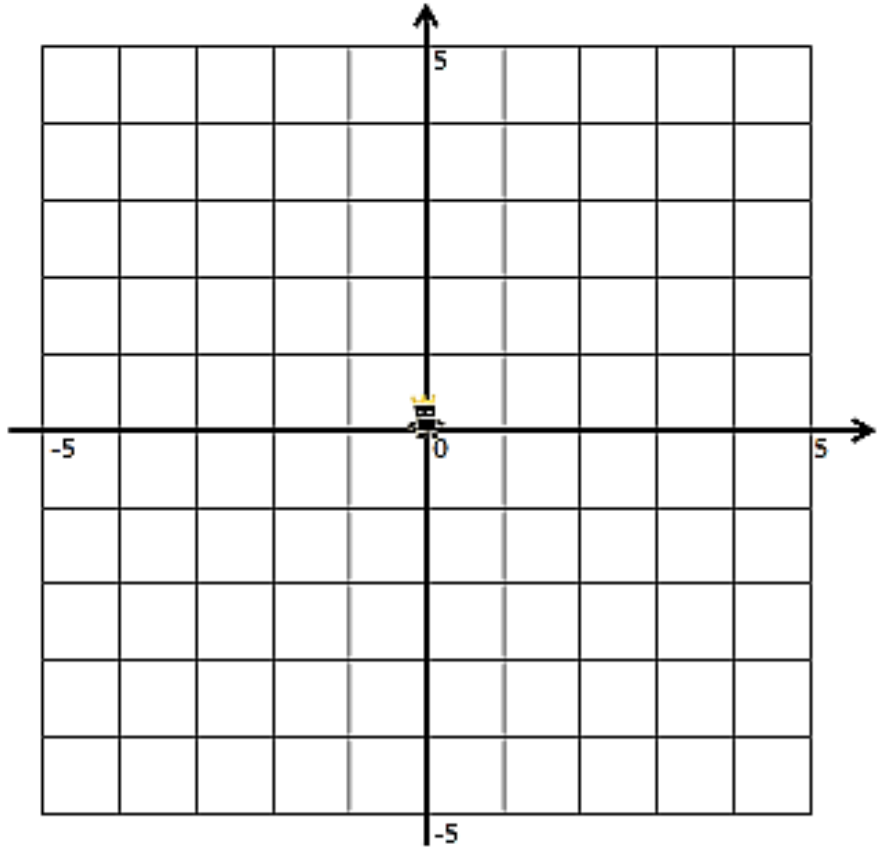
```
# Test case 2
cards1 = ["i", "drink", "water"]
cards2 = ["want", "to"]
goal = ["i", "want", "to", "drink", "water"]

print(solution3(cards1, cards2, goal))
```

문제 4. 방문 길이

문제 4. 방문 길이 (1/4)

제한시간: 40분

- 게임 캐릭터를 4가지 명령어를 통해 움직이려 합니다.
- 명령어는 다음과 같습니다.
  - ✓ U: 위쪽으로 한 칸 가기
  - ✓ D: 아래쪽으로 한 칸 가기
  - ✓ R: 오른쪽으로 한 칸 가기
  - ✓ L: 왼쪽으로 한 칸 가기
- 캐릭터는 좌표평면의 (0, 0) 위치에서 시작합니다.
- 좌표평면의 경계는 왼쪽 위(-5, 5), 왼쪽 아래(-5, -5), 오른쪽 위(5, 5), 오른쪽 아래(5, -5)로 이루어져 있습니다.



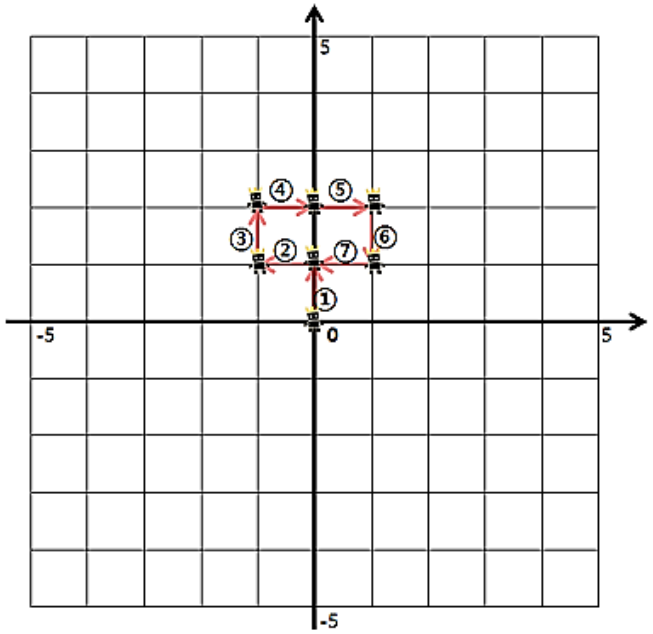
★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/49994

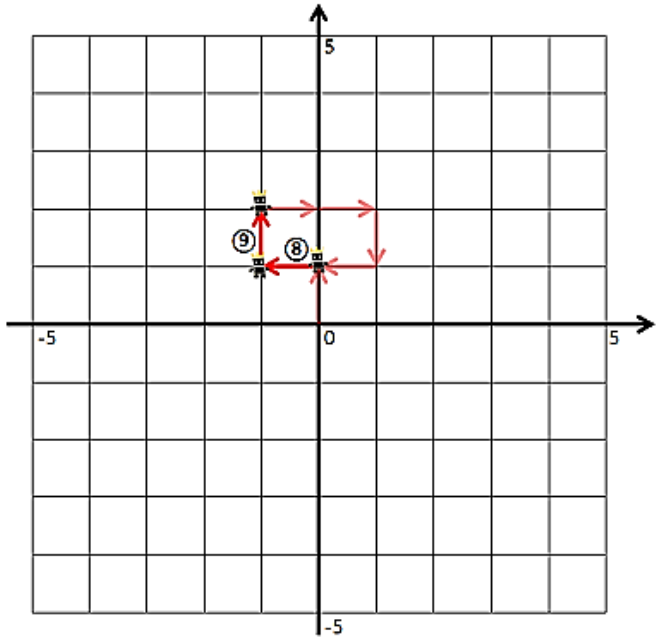
제한시간: 40분

- 예를 들어, "**ULURRDLLU**"로 명령했다면

①	②	③	④	⑤	⑥	⑦	⑧	⑨
U	L	U	R	R	D	L	L	U

- 아래 왼쪽 그림과 같이 1번 명령어부터 7번 명령어까지 움직입니다.
- 아래 오른쪽 그림과 같이 8번 명령어부터 9번 명령어까지 움직입니다.





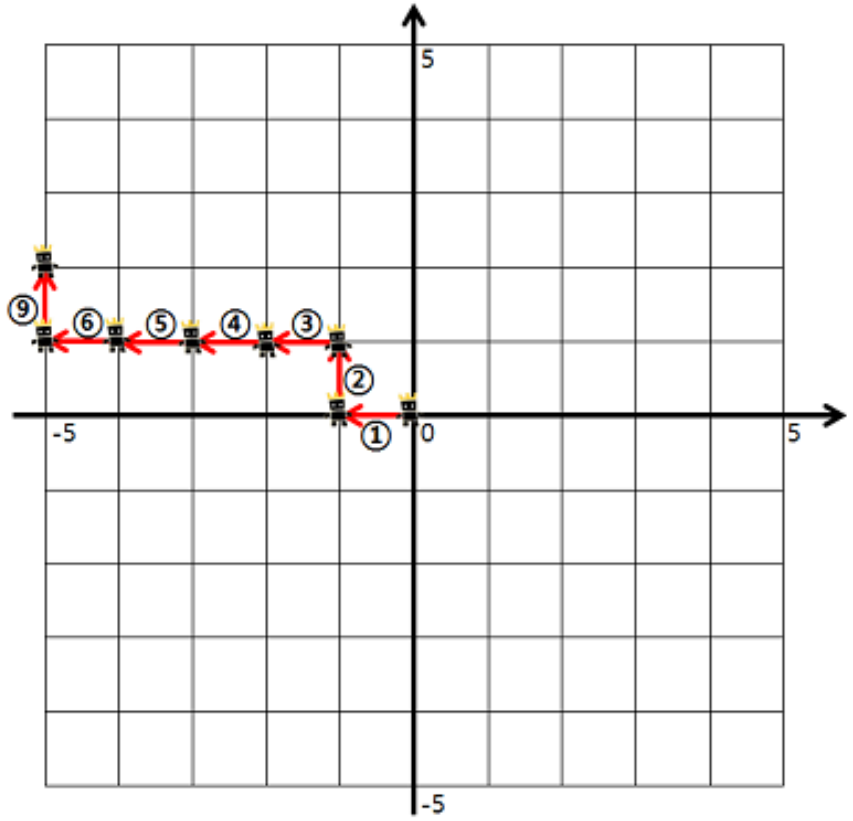
- 이때, 우리는 게임 캐릭터가 지나간 길 중 **캐릭터가 처음 걸어본 길의 길이**를 구하려고 합니다.
- 예를 들어, 위의 예시에서 게임 캐릭터가 움직인 길이는 9이지만, 캐릭터가 처음 걸어본 길의 길이는 7이 됩니다. (8, 9번 명령어에서 움직인 길은 2, 3번 명령어에서 이미 거쳐 간 길입니다)
- 단, 좌표평면의 경계를 넘어가는 명령어는 무시합니다.

문제 4. 방문 길이 (3/4)

제한시간: 40분

- 예를 들어, "**LULLLLLLU**"로 명령했다면,

①	②	③	④	⑤	⑥	⑦	⑧	⑨
L	U	L	L	L	L	L	L	U



- 1번 명령어부터 6번 명령어대로 움직인 후, 7, 8번 명령어는 무시하고, 다시 9번 명령어대로 움직입니다.
- 이때 캐릭터가 처음 걸어본 길의 길이는 7이 됩니다.

- 명령어가 매개변수 dirs로 주어질 때, 게임 캐릭터가 처음 걸어본 길의 길이를 구하여 return 하는 **solution4** 함수를 완성해 주세요.

★ **문제 출처:** https://school.programmers.co.kr/learn/courses/30/lessons/49994

문제 4. 방문 길이 (4/4)

제한시간: 40분

- **제한사항**
  - dirs는 string형으로 주어지며, 'U', 'D', 'R', 'L' 이외에 문자는 주어지지 않습니다.
  - dirs의 길이는 500 이하의 자연수입니다.

- **입출력 예제**
  - 입출력 예 #1: 문제의 예시와 같습니다.
  - 입출력 예 #2: 문제의 예시와 같습니다.

dirs	answer
"ULURRDLLU"	7
"LULLLLLLU"	7

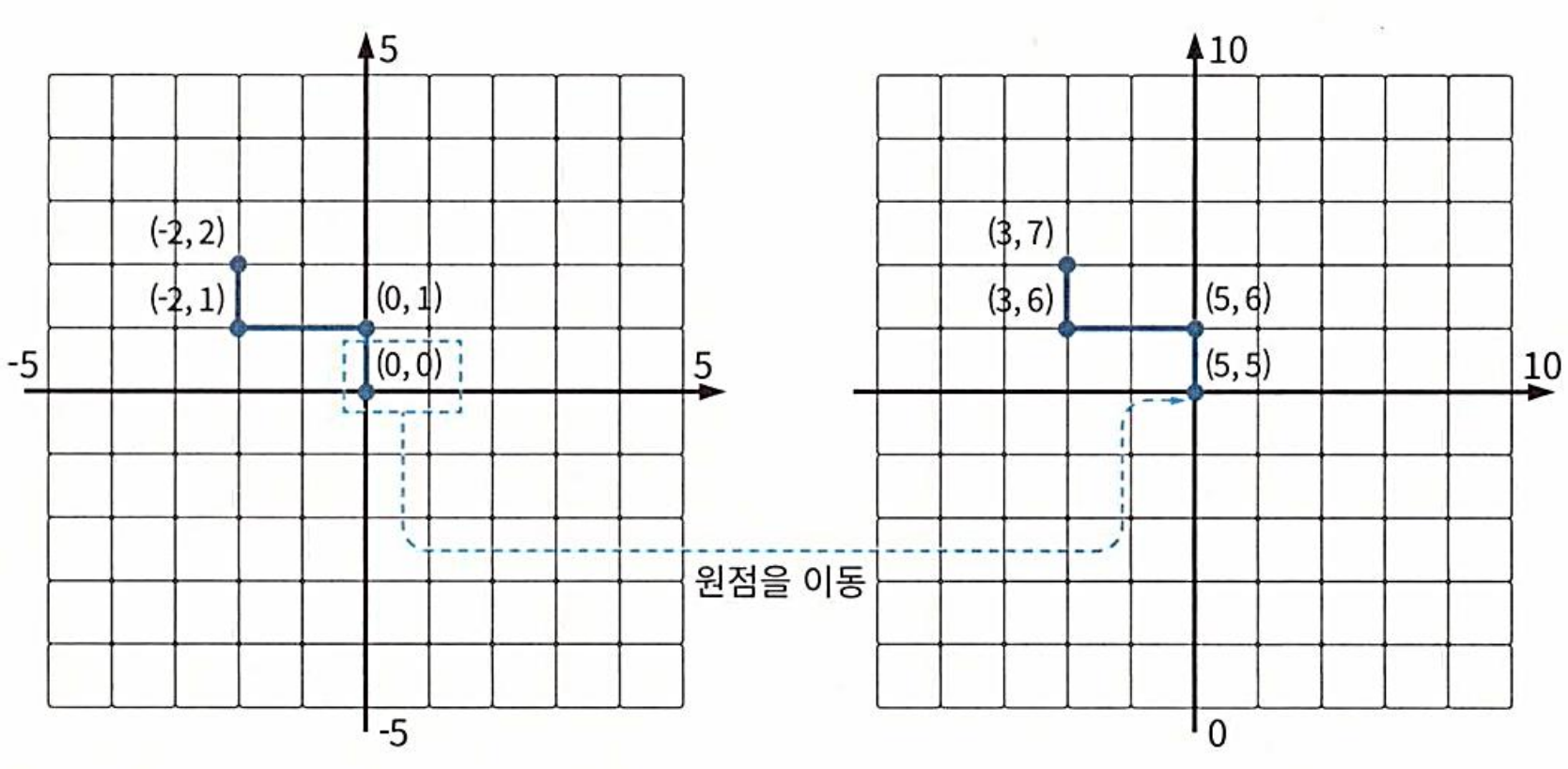
★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/49994

문제 4. 문제 분석

- **문제의 목표**
  - ➢ 로봇이 이동할 때, 중복 경로의 길이를 제외한 움직인 경로의 **길이**를 반환한다.
- **유의 사항 및 해결 방안**
  - ① 중복 경로는 포함하지 않는다.
    - 경로 저장 시 집합(set)을 활용하여 중복된 경로를 제거할 수 있다.
  - ② 음수 좌표를 포함하고 있다.
    - 원점을 기준으로 x축과 y축 좌표를 평행 이동하여 모든 좌표를 양수로 변환한다.
    - 본 문제의 목표는 움직인 경로의 전체 길이를 반환하는 것이므로 이렇게 구현해도 무방하다.

문제 4. 문제 분석

- 좌표의 평행 이동



문제 4. 문제 해결 전략 수립

① 입력 스트링 “dirs”로부터 이동 명령어를 읽는다.

② 현재 위치 (x, y)를 기준으로 다음 이동 좌표 (nx, ny)를 이동 명령어에 따라 구한다.

③ 다음 이동 좌표 (nx, ny) 가 유효한 범위 내에 포함되는지 체크한다.

- 유효하지 않으면 단계 ①로 이동해서 다음 이동 명령을 읽는다.

④ 현재 좌표 (x, y)에서 (nx, ny)까지의 이동 경로를 집합에 추가한다.

⑤ (nx, ny)에서 현재 좌표 (x, y)까지의 이동 경로를 집합에 추가한다.

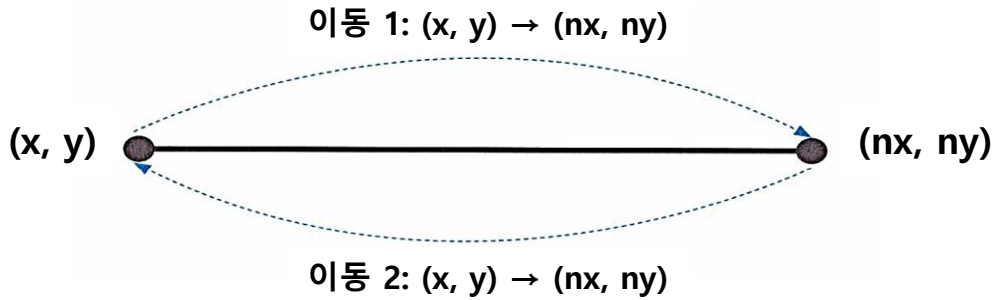
⑥ 현재 좌표 (x, y)에서 (nx, ny)로 이동한다. (즉, 현재 좌표 (x, y)를 (nx, ny)로 업데이트)

⑦ “dirs”에 포함된 모든 명령을 처리하였다면 종료, 그렇지 않다면 단계 ①로 이동한다.

⑧ 집합에 포함된 모든 이동 경로의 개수를 2로 나눈 값을 최종 결과로 반환한다.

문제 4. 문제 해결 전략 수립

- **앞서 단계 ④-⑤에서 경로를 두 개 추가하는 이유**
  - ✓ 이동 1: (x, y) → (nx, ny)
  - ✓ 이동 2: (nx, ny) → (x, y)
    - ➢ 아래 그림과 같이 두 이동 명령은 방향만 다를 뿐 길(path) 자체는 동일하다.



    - ➢ 이때, 이전에 "(x, y) → (nx, ny)" 이동한 적이 있으면, 추후 "(nx, ny) → (x, y)" 이동하는 명령이 나타나면, 이는 "(x, y) → (nx, ny)" 와 동일한 경로(path)이므로 사실상 같은 것으로 인식해야 한다. (∵ 본 문제에서는 "이동 방향"은 고려하지 않으므로)
    - ➢ 따라서, 상기의 "이동 1"과 "이동 2"를 모두 집합(set)에 추가해야 한다.

문제 4. 구현 및 검증

```
# 좌표 이동 시 유효 범위를 체크하는 함수
def is_valid_move(nx, ny):
    return 0 <= nx < 11 and 0 <= ny < 11

# 이동 명령에 따라 현재 좌표(x, y)에서 새로운 위치로 이동하고
# 이동한 위치의 좌표 (nx, ny)를 반환
def update_location(x, y, dir):
    if dir == 'U':  # Up
        nx, ny = x, y + 1
    elif dir == 'D':  # Down
        nx, ny = x, y - 1
    elif dir == 'L':  # Left
        nx, ny = x - 1, y
    elif dir == 'R':  # Right
        nx, ny = x + 1, y

    return nx, ny
```

문제 4. 구현 및 검증

```
# solution 함수
def solution4(dirs):
    x, y = 5, 5  # 음수 좌표를 양수로 변환하기 위한 평행이동
    ans = set()
    for dir in dirs:
        nx, ny = update_location(x, y, dir)  # 이동 명령에 따라 좌표 이동
        if not is_valid_move(nx, ny):  # 이동한 위치 좌표의 범위 체크
            continue

        # 현재 위치(x, y)에서 새로운 위치(nx, ny)로의 경로를 저장
        ans.add((x, y, nx, ny))
        ans.add((nx, ny, x, y))

        # 새로운 위치(좌표)로 이동
        x, y = nx, ny

    return len(ans)/2
```

문제 4. 구현 및 검증

- 테스트 케이스

```
# Test case 1
dirs = "ULURRDLLU"
print( solution4(dirs) )
```

```
# Test case 2
dirs = "LULLLLLLU"
print( solution4(dirs) )
```

Q & A