알고리즘2 (2024-2)

7. 그래프 자료구조의 활용

국립금오공과대학교 컴퓨터공학과

김 경 수

학습 목표

- 그래프 자료구조를 활용하는 문제의 특성을 이해하고, 그래프 자료구조를 활용하는 문제에서 적용할 수 있는 알고리즘의 종류를 이해하고 설명할 수 있다.
- 그래프 알고리즘이 적용되는 대표적인 문제들을 살펴보고 해당 문제들을 주어진 시간 내에 해결할 수 있다.
- 각 문제에서 그래프 알고리즘이 어떻게 적용되는지 이해하고 이를 논리적으로 설명할 수 있다.

문제 1. 배달

문제 1. 배달 (1/4)

제한시간: 70분

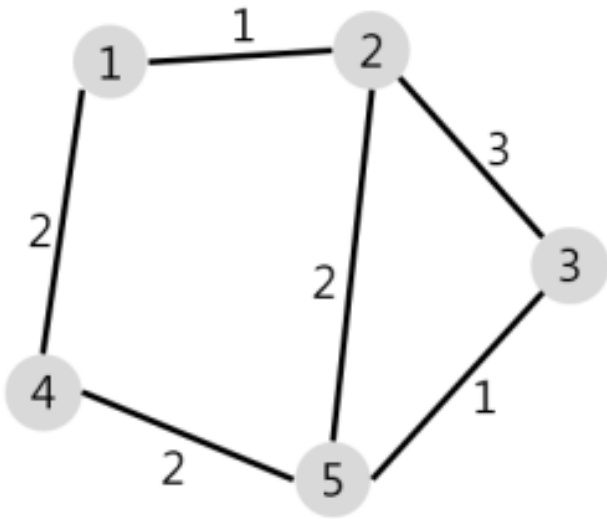
- N개의 마을로 이루어진 나라가 있습니다. 이 나라의 각 마을에는 1부터 N까지의 번호가 각각 하나씩 부여되어 있습니다.
- 각 마을은 양방향으로 통행할 수 있는 도로로 연결되어 있는데, 서로 다른 마을 간에 이동할 때는 이 도로를 지나야 합니다. 도로를 지날 때 걸리는 시간은 도로별로 다릅니다.
- 현재 1번 마을에 있는 음식점에서 각 마을로 음식 배달을 하려고 합니다.
- 각 마을로부터 음식 주문을 받으려고 하는데, N개의 마을 중에서 K 시간 이하로 배달이 가능한 마을에서만 주문을 받으려고 합니다.

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/12978

문제 1. 배달 (2/4)

제한시간: 70분

- 오른쪽 그림은 N = 5, K = 3인 경우의 예시입니다.
- 위 그림에서 1번 마을에 있는 음식점은 [1, 2, 4, 5] 번 마을까지는 3 이하의 시간에 배달할 수 있습니다. 그러나 3번 마을까지는 3시간 이내로 배달할 수 있는 경로가 없으므로 3번 마을에서는 주문을 받지 않습니다. 따라서 1번 마을에 있는 음식점이 배달 주문을 받을 수 있는 마을은 4개가 됩니다.
- 마을의 개수 N, 각 마을을 연결하는 도로의 정보 road, 음식 배달이 가능한 시간 K가 매개변수로 주어질 때, 음식 주문을 받을 수 있는 마을의 개수를 return 하도록 **solution1** 함수를 완성해주세요.



★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/12978

문제 1. 배달 (3/4)

제한시간: 70분

- **제한사항**
  - 마을의 개수 N은 1 이상 50 이하의 자연수입니다.
  - road의 길이(도로 정보의 개수)는 1 이상 2,000 이하입니다.
  - road의 각 원소는 마을을 연결하고 있는 각 도로의 정보를 나타냅니다.
  - road는 길이가 3인 배열이며, 순서대로 (a, b, c)를 나타냅니다.
    - a, b(1 ≤ a, b ≤ N, a != b)는 도로가 연결하는 두 마을의 번호이며, c(1 ≤ c ≤ 10,000, c는 자연수)는 도로를 지나는데 걸리는 시간입니다.
    - 두 마을 a, b를 연결하는 도로는 여러 개가 있을 수 있습니다.
    - 한 도로의 정보가 여러 번 중복해서 주어지지 않습니다.
  - K는 음식 배달이 가능한 시간을 나타내며, 1 이상 500,000 이하입니다.
  - 임의의 두 마을간에 항상 이동 가능한 경로가 존재합니다.
  - 1번 마을에 있는 음식점이 K 이하의 시간에 배달이 가능한 마을의 개수를 return 하면 됩니다.

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/12978

문제 1. 배달 (4/4)

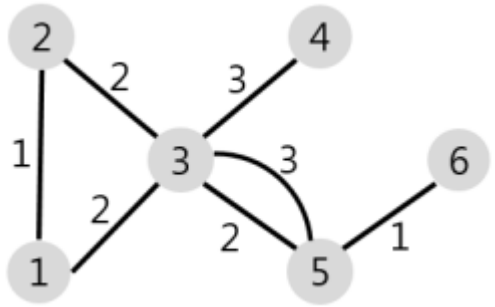
제한시간: 70분

- **입출력 예**

N	road	K	result
5	[[1,2,1],[2,3,3],[5,2,2],[1,4,2],[5,3,1],[5,4,2]]	3	4
6	[[1,2,1],[1,3,2],[2,3,2],[3,4,3],[3,5,2],[3,5,3],[5,6,1]]	4	4

➢ **입출력 예 #1:** 문제의 예시와 같습니다.

➢ **입출력 예 #2:** 주어진 마을과 도로의 모양은 아래 그림과 같습니다.



▪ 1번 마을에서 배달에 4시간 이하가 걸리는 마을은 [1, 2, 3, 5] 4개이므로 4를 return 합니다.

★ **문제 출처:** https://school.programmers.co.kr/learn/courses/30/lessons/12978

문제 1. 문제 분석

- **주어진 문제의 목표**
  - ➢ 음식 배달이 가능한 시간이 K로 주어질 때, 출발 마을을 기준으로 음식 주문을 받을 수 있는 마을의 개수를 계산하여 반환한다.
  - ➢ 이를 위해서는, 출발 마을에서 각 마을 까지의 최단 소요 시간을 계산하는 과정이 필수적으로 요구된다.

- **문제의 특성**
  - ✓ N개의 마을이 존재함 → **노드(node)**로 표현 가능
  - ✓ 각 마을은 도로로 연결되어 있음 → **간선(edge)**으로 표현 가능
  - ✓ 각 도로에는 음식 배달 시간이 있음 → 간선의 **가중치(weight)**로 표현 가능

그래프 자료구조를 활용할 수 있으며, 이 경우 그래프 자료구조의 대표적인 알고리즘들을 우선적으로 떠올려야 한다.

문제 1. 문제 해결 전략 수립

- **문제의 목표를 해결하기 위한 방법**
  - ① 출발 마을에서 각 마을 까지의 최단 소요 시간을 계산한다.
  - ② 각 마을 까지의 소요 시간이 K 이내라면, 해당 마을은 배달 가능 마을로 카운트한다.
  - ③ 모든 마을에 대해서 소요 시간을 계산한 후, 최종적인 배달 가능 마을의 개수를 반환한다.
- **최단 거리 알고리즘의 종류**
  - ① 단일 정점에서 나머지 모든 정점 까지의 최단 거리
    - ➢ **다익스트라 알고리즘(Dijkstra algorithm)**: 단, 음의 가중치는 허용하지 않음.
    - ➢ **벨만-포드 알고리즘(Bellman-Ford algorithm)**: 음의 가중치도 허용함.
  - ② 모든 정점 간의 최단 거리
    - ➢ **플로이드-와셜 알고리즘(Floyd-Warshall algorithm)**

본 문제에서는 다익스트라 알고리즘을 활용하는 것이 적합하다.

문제 1. 구현 및 검증

```
import heapq

def solution1(N, road, K):
    # 1. 각 노드에 연결된 간선들을 저장할 리스트
    graph = [[] for _ in range(N+1)]

    # 2. 출발점에서 각 노드까지의 최단 소요시간을 저장할 리스트
    distances = [float("inf")] * (N+1)
    distances[1] = 0

    # 3. 그래프 구성
    for a, b, cost in road:
        graph[a].append((b, cost))
        graph[b].append((a, cost))
```

⇦ 간선들을 가중치의 오름차순으로 정렬해야 하므로, heap을 사용한다.

⇦ 문제의 조건에서 양방향 통행이 가능하다고 했으므로, 그래프 구축 시 양방향 모두를 인접 리스트에 추가한다.

문제 1. 구현 및 검증

```
# 4. 다익스트라 알고리즘 시작
heap = []
heapq.heappush(heap, (0,1))  # 5. 출발점을 heap에 추가 (최단 소요시간, 노드번호)
while heap:
    dist, node = heapq.heappop(heap)

    # 6. 인접한 노드들의 최단 소요시간을 갱신하고 heap에 추가
    for next_node, next_dist in graph[node]:
        cost = dist + next_dist
        if cost < distances[next_node]:
            distances[next_node] = cost
            heapq.heappush(heap, (cost, next_node))

# 7. distances list에서 k 이하인 값의 개수를 구하여 반환
return sum(1 for dist in distances if dist <=K)
```

문제 1. 구현 및 검증

테스트 케이스 #1

```
N = 5
road = [[1,2,1],[2,3,3],[5,2,2],[1,4,2],[5,3,1],[5,4,2]]
K = 3
print( solution1(N, road, K) )
```

테스트 케이스 #2

```
N = 6
road = [[1,2,1],[1,3,2],[2,3,2],[3,4,3],[3,5,2],[3,5,3],[5,6,1]]
K = 4
print( solution1(N, road, K) )
```

문제 2. 전력망을 둘로 나누기

문제 2. 전력망을 둘로 나누기 (1/5)

제한시간: 50분

- n개의 송전탑이 전선을 통해 하나의 트리 형태로 연결되어 있습니다. 당신은 이 전선들 중 하나를 끊어서 현재의 전력망 네트워크를 2개로 분할하려고 합니다.
- 이때, 두 전력망이 갖게 되는 송전탑의 개수를 최대한 비슷하게 맞추고자 합니다.
- 송전탑의 개수 n, 그리고 전선 정보 wires가 매개변수로 주어집니다. 전선들 중 하나를 끊어서 송전탑 개수가 가능한 비슷하도록 두 전력망으로 나누었을 때, 두 전력망이 가지고 있는 송전탑 개수의 차이(절대값)를 return 하도록 **solution2** 함수를 완성해주세요.

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/86971

문제 2. 전력망을 둘로 나누기 (2/5)

제한시간: 50분

- **제한사항**
  - n은 2 이상 100 이하인 자연수입니다.
  - wires는 길이가 n-1인 정수형 2차원 배열입니다.
    - wires의 각 원소는 [v1, v2] 2개의 자연수로 이루어져 있으며, 이는 전력망의 v1번 송전탑과 v2번 송전탑이 전선으로 연결되어 있다는 것을 의미합니다.
    - 1 ≤ v1 < v2 ≤ n 입니다.
    - 전력망 네트워크가 하나의 트리 형태가 아닌 경우는 입력으로 주어지지 않습니다.

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/86971

문제 2. 전력망을 둘로 나누기 (3/5)

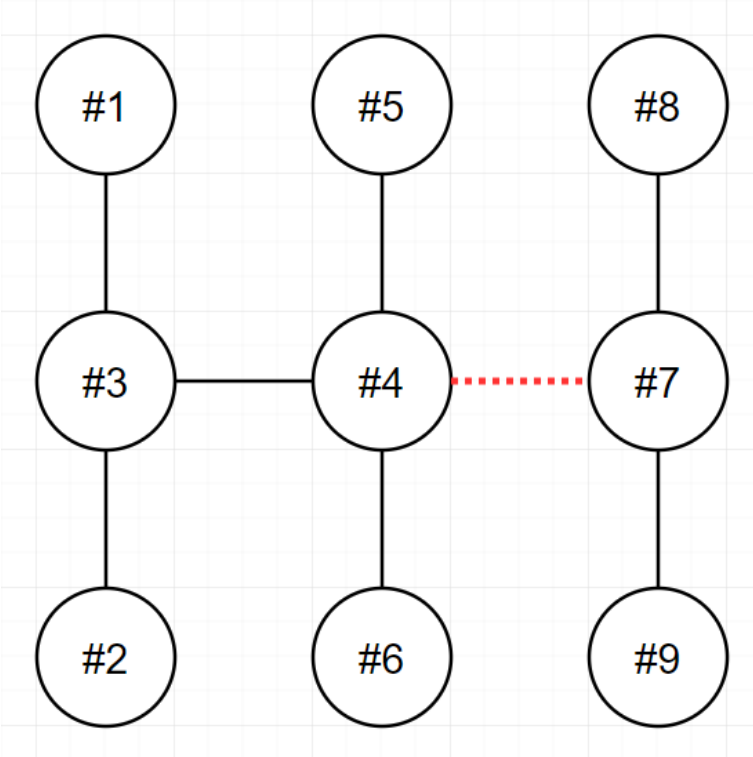
제한시간: 50분

- **입출력 예**

n	wires	result
9	[[1,3],[2,3],[3,4],[4,5],[4,6],[4,7],[7,8],[7,9]]	3

➢ **입출력 예 #1:** 오른쪽 그림은 주어진 입력을 해결하는 방법 중 하나를 나타낸 것입니다.

- 4번과 7번을 연결하는 전선을 끊으면 두 전력망은 각 6개와 3개의 송전탑을 가지며, 이보다 더 비슷한 개수로 전력망을 나눌 수 없습니다.
- 또 다른 방법으로는 3번과 4번을 연결하는 전선을 끊어도 최선의 정답을 도출할 수 있습니다.



★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/86971

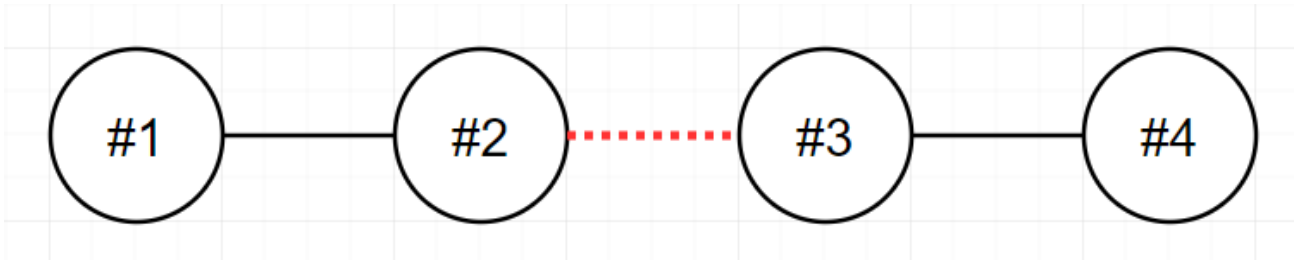
문제 2. 전력망을 둘로 나누기 (4/5)

제한시간: 50분

- **입출력 예**

n	wires	result
4	[[1,2],[2,3],[3,4]]	0

➢ **입출력 예 #2:** 아래 그림은 주어진 입력을 해결하는 방법을 나타낸 것입니다.



- 2번과 3번을 연결하는 전선을 끊으면 두 전력망이 모두 2개의 송전탑을 가지게 되며, 이 방법이 최선입니다.

★ **문제 출처:** https://school.programmers.co.kr/learn/courses/30/lessons/86971

문제 2. 전력망을 둘로 나누기 (5/5)

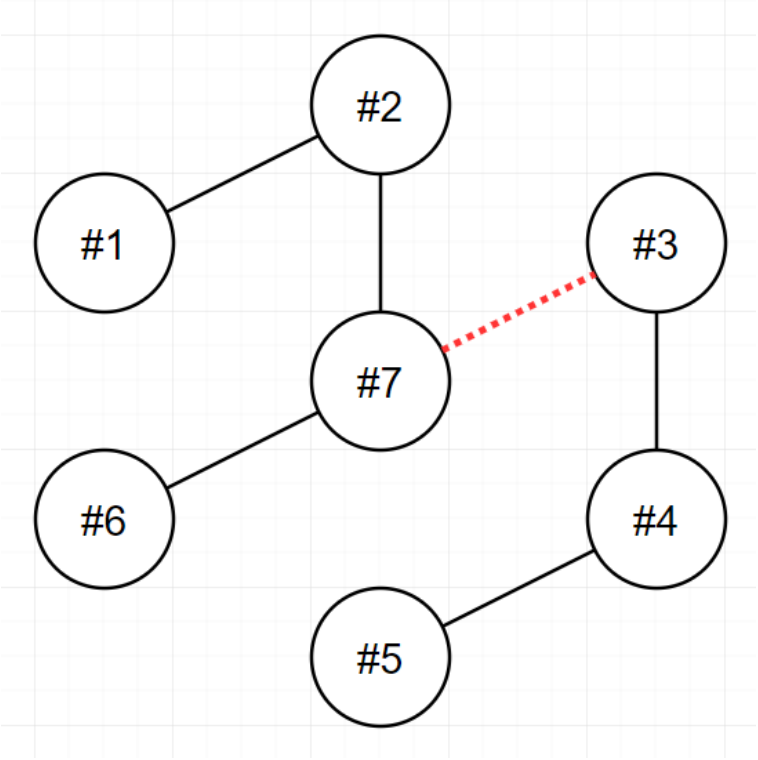
제한시간: 50분

- **입출력 예**

n	wires	result
7	[[1,2],[2,7],[3,7],[3,4],[4,5],[6,7]]	1

➢ **입출력 예 #3:** 오른쪽 그림은 주어진 입력을 해결하는 방법을 나타낸 것입니다.

- 3번과 7번을 연결하는 전선을 끊으면 두 전력망이 각각 4개와 3개의 송전탑을 가지게 되며, 이 방법이 최선입니다.



★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/86971

문제 2. 문제 분석

- **주어진 문제의 목표**
  - 전선들 중 하나를 끊어서 송전탑 개수가 가능한 비슷하도록 두 전력망으로 나눈다.
  - 이때, 두 전력망이 가지고 있는 송전탑 개수의 차이값을 반환한다.
- **문제의 특성**
  - 주어진 전력망의 구조는 하나의 트리로 표현된다.
  - 이때, 전선을 하나 끊으면 전력망은 두 개로 나누어진다.
    - 즉, 전선을 하나 끊었을 때 전력망이 세 개 이상으로 나누어지는 경우는 없음.
  - 분할된 전력망에서 송전탑 개수를 구한 다음 차이를 구하는 것 보다는 한 쪽의 송전탑 개수를 구한 후 전체 개수에서 빼는 것이 구현하는데 더 효과적임.
    - 왜냐하면, 전력망은 최대 두 개로만 나누어지므로...

문제 2. 문제 해결 전략 수립

- **각 전력망을 두 개로 나누고 두 전력망에서 송전탑 개수를 구하는 방법**
  - 트리 탐색 알고리즘을 이용하여 송전탑 개수를 카운트할 수 있다.
  - 트리 탐색 알고리즘의 종류: BFS (너비 우선 탐색), DFS (깊이 우선 탐색)
  - 여기서는 DFS를 사용하는 것이 바람직하다.
    - ✓ 왜냐하면, 문제에서 최적이나 최소(= BFS를 주로 사용)를 요구하지 않음.

문제 2. 문제 해결 과정 기술

- **문제 해결 방법**

① 전선을 하나씩 제거한다.

② 전선을 제거한 후 생성된 두 전력망의 송전탑 개수를 깊이 우선 탐색 알고리즘을 이용하여 카운트한다.

③ 두 전력망에서 발견된 송전탑 개수의 차이 값을 계산한다.

④ 상기 과정 ① - ③을 모든 전선을 끊을 때 까지 반복적으로 수행한다.

문제 2. 구현 및 검증

```
def solution2(n, wires):
    # 그래프 생성
    graph = [[] for _ in range(n+1)]
    for a, b in wires:
        graph[a].append(b)
        graph[b].append(a)

    # 송전탑 개수 카운팅을 위한 DFS 함수
    def dfs(node, parent):
        cnt = 1
        for child in graph[node]: # 현재 노드의 자식 노드에 방문
            if child != parent:   # 부모 노드가 아닌 경우에만 탐색
                cnt += dfs(child, node)

        return cnt
```

⇦ 송전탑 번호가 1부터 시작하므로 리스트 길이는 n+1

⇦ 현재 노드를 포함한 자식 노드들의 개수를 카운트하기 위해서 DFS 함수를 사용한다.

문제 2. 구현 및 검증

```
min_diff = float("inf")
for a, b in wires:
    # 간선 제거
    graph[a].remove(b)
    graph[b].remove(a)

    # 두 전력망의 송전탑 개수 계산
    cnt_a = dfs(a, b)
    cnt_b = n - cnt_a

    # 송전탑 개수 차이값의 최소값 계산
    min_diff = min(min_diff, abs(cnt_a - cnt_b))
```

⇦ 두 전력망으로 나누기 위해 임의의 간선을 제거

⇦ 간선을 제거한 후 분리된 두 전력망에서 송전탑의 개수 (= 노드의 개수)를 DFS 알고리즘을 이용하여 카운트

⇧ 두 전력망에 존재하는 송전탑 개수의 최소 차이 (difference)를 구하는 문제이므로 위와 같이 구현함.

문제 2. 구현 및 검증

```
        # 간선 복원
        graph[a].append(b)
        graph[b].append(a)

    return min_diff
```

⇦ 다른 전선을 끊었을 때 두 전력망의 송전탑 개수를 카운트하기 위해서 방금 끊었던 연결을 원상태로 복원해야 함.

문제 2. 구현 및 검증

테스트 케이스 #1

```
wires = [[1,3],[2,3],[3,4],[4,5],[4,6],[4,7],[7,8],[7,9]]
n = 9
print( solution2(n, wires) )
```

테스트 케이스 #2

```
wires = [[1,2],[2,3],[3,4]]
n = 4
print( solution2(n, wires) )
```

테스트 케이스 #3

```
wires = [[1,2],[2,7],[3,7],[3,4],[4,5],[6,7]]
n = 7
print( solution2(n, wires) )
```

문제 3. 네트워크

문제 3. 네트워크 (1/2)

제한시간: 50분

- 네트워크란 컴퓨터 상호 간에 정보를 교환할 수 있도록 연결된 형태를 의미한다.
- 예를 들어, 컴퓨터 A와 컴퓨터 B가 직접적으로 연결되어 있고, 컴퓨터 B와 컴퓨터 C가 직접적으로 연결되어 있을 때 컴퓨터 A와 컴퓨터 C도 간접적으로 연결되어 정보를 교환할 수 있다. 따라서 컴퓨터 A, B, C는 모두 같은 네트워크 상에 있다고 할 수 있다.
- 컴퓨터의 개수 n, 연결에 대한 정보가 담긴 2차원 배열 computers가 매개변수로 주어질 때, 네트워크의 개수를 return 하도록 **solution3** 함수를 작성하시오.
- **제한사항**
  - 컴퓨터의 개수 n은 1 이상 200 이하인 자연수이다.
  - 각 컴퓨터는 0부터 n-1인 정수로 표현한다.
  - i번 컴퓨터와 j번 컴퓨터가 연결되어 있으면 computers[i][j]를 1로 표현한다.
  - computer[i][i]는 항상 1이다.

★ **문제 출처:** https://school.programmers.co.kr/learn/courses/30/lessons/43162

문제 3. 네트워크 (2/2)

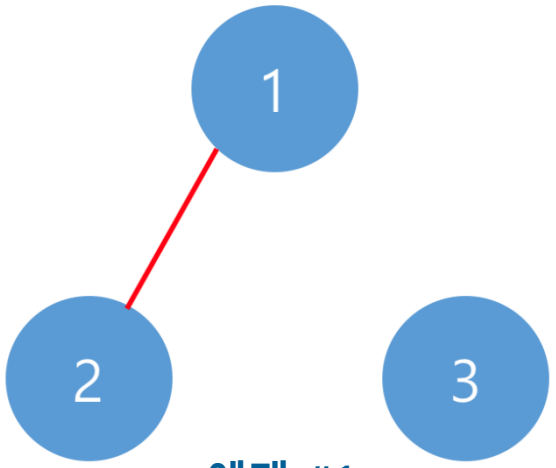
제한시간: 50분

- 입출력 예제

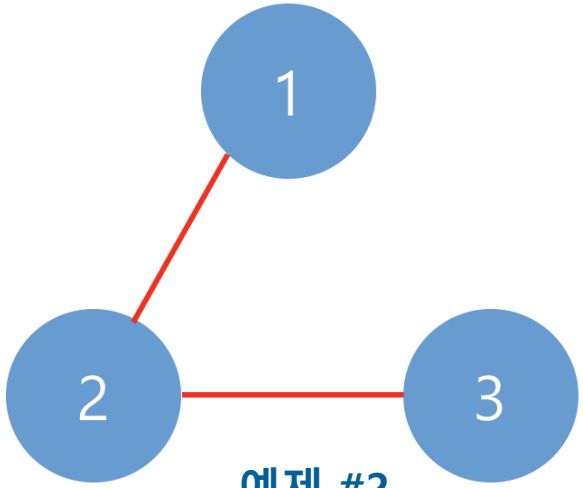
n	computers	return
3	[[1, 1, 0], [1, 1, 0], [0, 0, 1]]	2
3	[[1, 1, 0], [1, 1, 1], [0, 1, 1]]	1

➢ **예제 #1.** 아래 왼쪽 그림과 같이 2개의 네트워크가 존재함.

➢ **예제 #2.** 아래 오른쪽 그림과 같이 1개의 네트워크가 존재함.



예제 #1



예제 #2

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/43162

문제 3. 문제 분석

- **주어진 문제의 목표**
  - 컴퓨터의 개수 n, 연결에 대한 정보가 담긴 2차원 배열 computers가 매개변수로 주어질 때, **네트워크의 개수를 카운트**하여 반환한다.

- **문제의 특성**
  - 하나의 노드에서 출발하여 접근 가능한 모든 노드들은 단일 네트워크에 포함된다.
  - 즉, 특정 노드에서 출발하여 더 이상의 탐색이 진행되지 않을 때까지 탐색을 수행한다.
  - 깊이 우선 탐색을 수행한 횟수를 카운트한다.

문제 3. 문제 해결 전략 수립

- 입력 배열 "computers"를 기준으로 그래프를 구현한다.
- 번호가 낮은 노드부터 시작해서 연결된 모든 노드들에 대한 탐색을 진행한다.
  - ➢ 이를 통해, 출발 노드에서 도달 가능한 모든 노드들을 순차적으로 방문하고, 이를 하나의 네트워크로 정의한다.
  - ➢ 그래프 내 모든 노드를 탐색하는 문제는 "깊이 우선 탐색'을 활용하는 것이 일반적으로 유리함.

    (Why?) 깊이 우선 탐색이 구현하기도 상대적으로 더 쉽고 메모리도 더 적게 사용함.
- 상기 깊이 우선 탐색을 진행한 후 아직 방문하지 않은 노드가 존재하면, 이 중에서 번호가 가장 작은 노드를 "출발 노드"로 지정하고 깊이 우선 탐색을 다시 수행한다.
- 그래프 내 모든 노드들을 방문 완료한 후에는 깊이 우선 탐색을 몇 번 수행했는지 반환한다.

문제 3. 문제 해결 전략 수립

- **문제 해결 예제**

① "computers"를 그래프로 변환

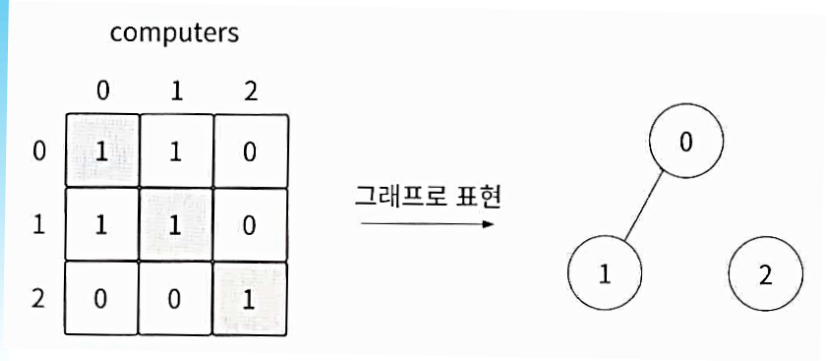
② 노드 0의 방문 여부를 "true"로 변환하고 DFS를 진행, answer += 1.

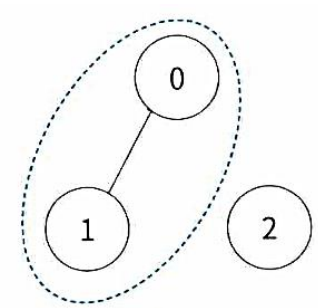
③ 노드 1의 방문 여부를 "true"로 변환하고, 1과 연결된 노드가 없으므로 DFS 종료.

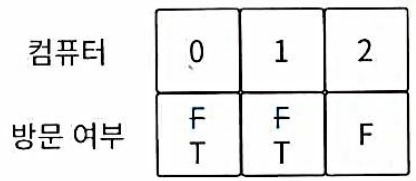
④ 노드 2는 아직 방문하지 않았으므로 "true"로 변환하고 DFS 진행, answer +=1

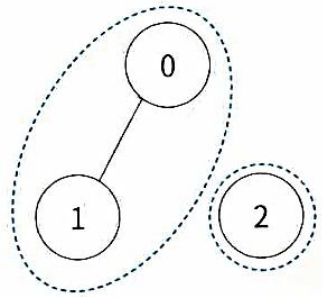
⑤ 노드 2와 연결된 노드가 더 이상 없으므로 DFS 종료

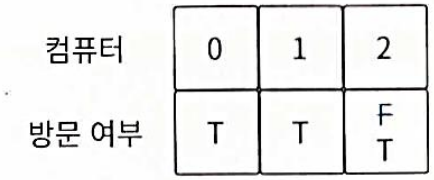
⑥ Return answer.











문제 3. 문제 해결 방법 기술

① 입력 “computers”를 그래프로 표현한다.

② 모든 노드들의 방문 여부를 “false”로 초기화한다.

③ 노드 1 ~ n 각각에 대하여 아래 작업을 반복한다.

- 만약 해당 노드를 방문했다면 continue.
- 해당 노드의 방문 여부를 “true”로 변경한다.
- 해당 노드를 시작으로 DFS를 호출하고 DFS 호출 횟수를 1 증가시킨다.

④ DFS를 호출한 총 횟수를 반환한다.

문제 3. 구현 및 검증

```
def dfs_nw(computers, visited, node):
    visited[node] = True  # 현재 노드의 방문 상태를 True로 변경

    # 현재 노드 node와 연결되어 있고, 아직 방문하지 않은 노드인 모든 노드에 대해서
    for idx, connected in enumerate(computers[node]):
        if connected and not visited[idx]:
            dfs_nw(computers, visited, idx)  # 해당 노드를 방문

def solution3(n, computers):
    answer = 0
    visited = [False] * n  # 각 노드 별 방문 여부를 저장하는 리스트
    for i in range(n):
        # 아직 방문하지 않은 노드라면, 해당 노드를 시작으로 DFS를 진행
        if not visited[i]:
            dfs_nw(computers, visited, i)
            answer += 1

    return answer
```

문제 3. 구현 및 검증

테스트 케이스 #1

```
n = 3
computers = [[1, 1, 0], [1, 1, 0], [0, 0, 1]]
print(solution3(n, computers))
```

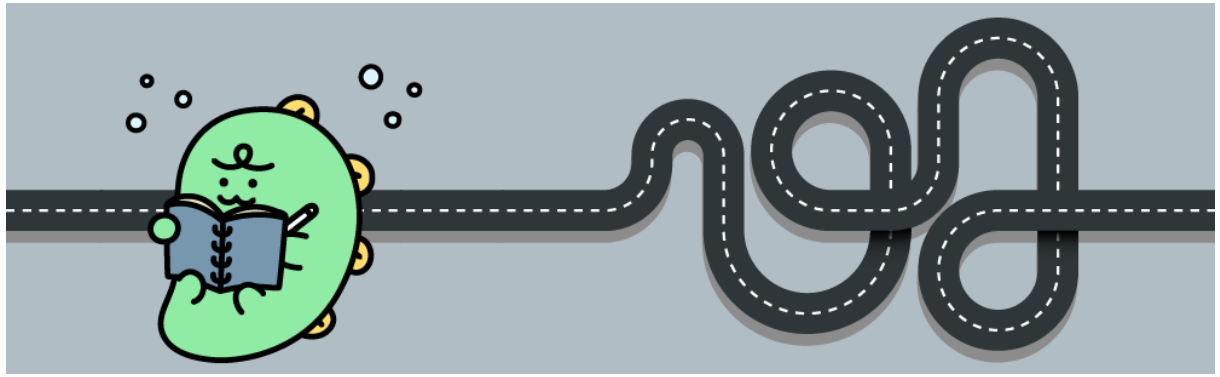
테스트 케이스 #2

```
n = 3
computers = [[1, 1, 0], [1, 1, 1], [0, 1, 1]]
print(solution3(n, computers))
```

문제 4. 경주로 건설

문제 4. 경주로 건설 (1/8)

제한시간: 90분



- 건설회사의 설계사인 죠르디는 고객사로부터 자동차 경주로 건설에 필요한 견적을 의뢰받았습니다.
- 제공된 경주로 설계 도면에 따르면 경주로 부지는 N x N 크기의 정사각형 격자 형태이며 각 격자는 1 x 1 크기입니다.
- 설계 도면에는 각 격자의 칸은 0 또는 1 로 채워져 있으며, 0은 칸이 비어 있음을 1은 해당 칸이 벽으로 채워져 있음을 나타냅니다.

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/67259

문제 4. 경주로 건설 (2/8)

제한시간: 90분

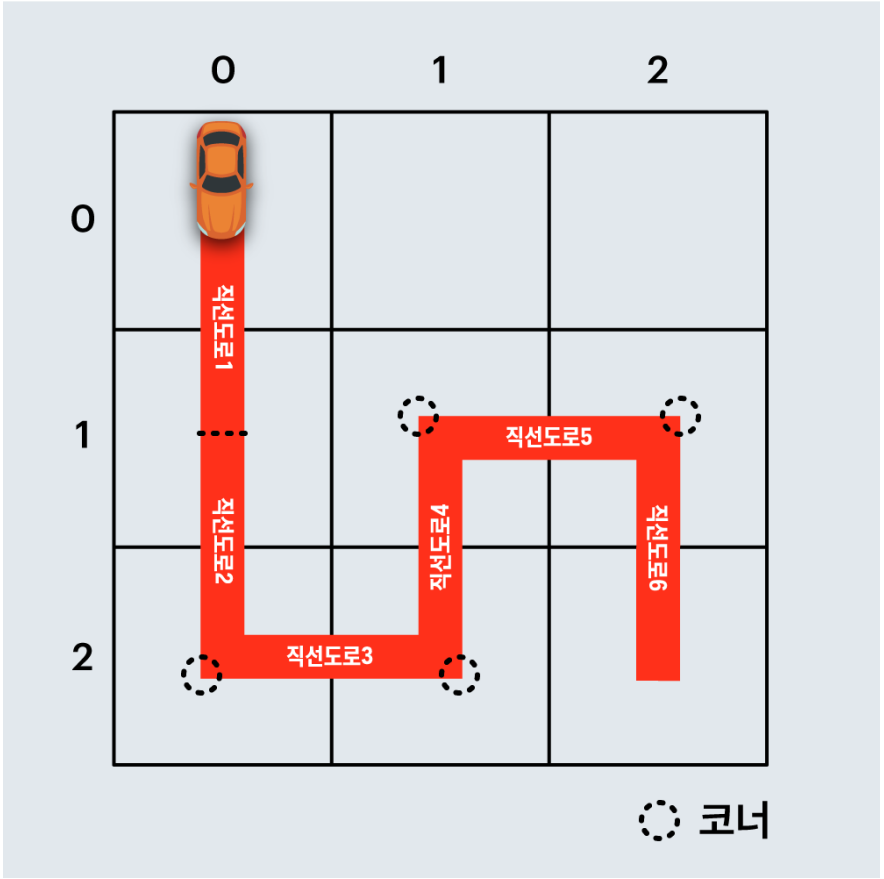
- 경주로의 출발점은 (0, 0) 칸(좌측 상단)이며, 도착점은 (N-1, N-1) 칸(우측 하단)입니다.
- 죠르디는 출발점인 (0, 0) 칸에서 출발한 자동차가 도착점인 (N-1, N-1) 칸까지 무사히 도달할 수 있게 중간에 끊기지 않도록 경주로를 건설해야 합니다.
- 경주로는 상, 하, 좌, 우로 인접한 두 빈 칸을 연결하여 건설할 수 있으며, 벽이 있는 칸에는 경주로를 건설할 수 없습니다.
- 이때, 인접한 두 빈 칸을 상하 또는 좌우로 연결한 경주로를 직선 도로 라고 합니다.
- 또한 두 직선 도로가 서로 직각으로 만나는 지점을 "코너"라고 부릅니다.
- 건설 비용을 계산해 보니 직선 도로 하나를 만들 때는 100원이 소요되며, 코너를 하나 만들 때는 500원이 추가로 듭니다.

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/67259

문제 4. 경주로 건설 (3/8)

제한시간: 90분

- 죠르디는 견적서 작성을 위해 경주로를 건설하는 데 필요한 최소 비용을 계산해야 합니다.
- 예를 들어, 아래 그림은 직선 도로 6개와 코너 4개로 구성된 임의의 경주로 예시이며, 건설 비용은 6 x 100 + 4 x 500 = 2600원 입니다.



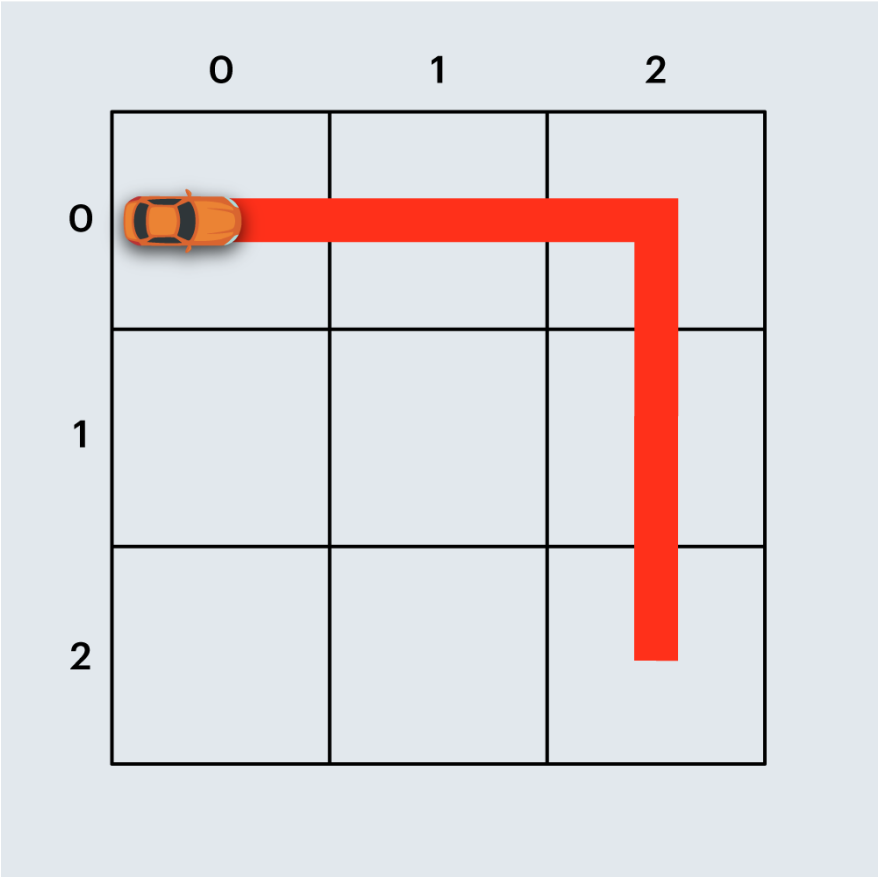
★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/67259

문제 4. 경주로 건설 (4/8)

제한시간: 90분

- 또 다른 예로, 아래 그림은 직선 도로 4개와 코너 1개로 구성된 경주로이며, 건설 비용은 4 x 100 + 1 x 500 = 900원 입니다.

- 도면의 상태(0은 비어 있음, 1은 벽)을 나타내는 2차원 배열 board가 매개변수로 주어질 때, 경주로를 건설하는데 필요한 최소 비용을 return 하도록 **solution4** 함수를 완성해주세요.



★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/67259

문제 4. 경주로 건설 (5/8)

제한시간: 90분

- **제한사항**
  - board는 2차원 정사각 배열로 배열의 크기는 3 이상 25 이하입니다.
  - board 배열의 각 원소의 값은 0 또는 1 입니다.
  - 도면의 가장 왼쪽 상단 좌표는 (0, 0)이며, 가장 우측 하단 좌표는 (N-1, N-1) 입니다.
  - 원소의 값 0은 칸이 비어 있어 도로 연결이 가능함을 1은 칸이 벽으로 채워져 있어 도로 연결이 불가능함을 나타냅니다.
  - board는 항상 출발점에서 도착점까지 경주로를 건설할 수 있는 형태로 주어집니다.
  - 출발점과 도착점 칸의 원소의 값은 항상 0으로 주어집니다.

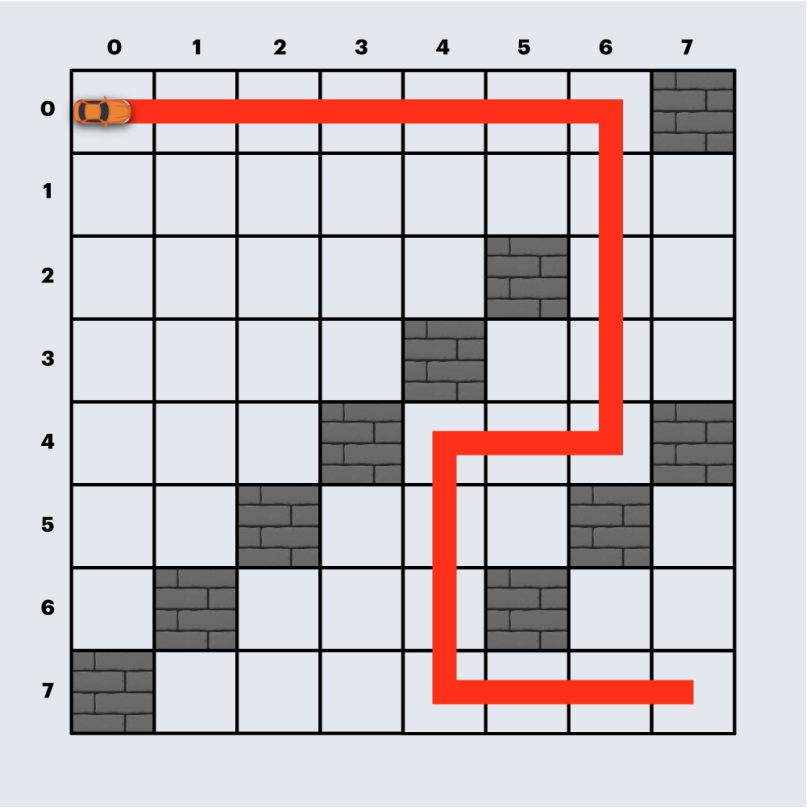
★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/67259

문제 4. 경주로 건설 (6/8)

제한시간: 90분

- 입출력 예제

board	result
[[0,0,0],[0,0,0],[0,0,0]]	900
[[0,0,0,0,0,0,0,1],[0,0,0,0,0,0,0,0],[0,0,0,0,0,1,0,0],[0,0,0,0,1,0,0,0],[0,0,0,1,0,0,0,1],[0,0,1,0,0,0,1,0],[0,1,0,0,0,1,0,0],[1,0,0,0,0,0,0,0]]	**3800**
[[0,0,1,0],[0,0,0,0],[0,1,0,1],[1,0,0,0]]	2100
[[0,0,0,0,0,0],[0,1,1,1,1,0],[0,0,1,0,0,0],[1,0,0,1,0,1],[0,1,0,0,0,1],[0,0,0,0,0,0]]	3200



- **입출력 예 #1:** 본문의 예시와 같습니다.
- **입출력 예 #2:** 오른쪽 그림과 같이 경주로를 건설하면 직선 도로 18개, 코너 4개로 총 3800원이 듭니다.

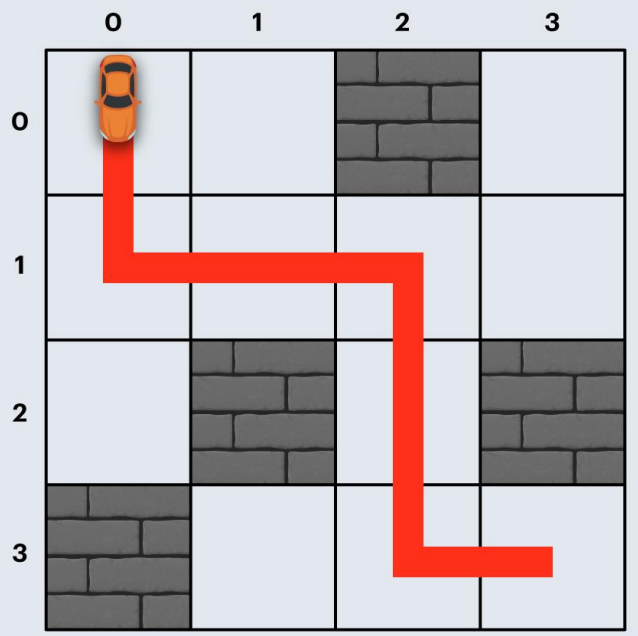
★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/67259

문제 4. 경주로 건설 (7/8)

제한시간: 90분

- **입출력 예제**

board	result
[[0,0,0],[0,0,0],[0,0,0]]	900
[[0,0,0,0,0,0,0,1],[0,0,0,0,0,0,0,0],[0,0,0,0,0,1,0,0],[0,0,0,0,1,0,0,0],[0,0,0,1,0,0,0,1],[0,0,1,0,0,0,1,0],[0,1,0,0,0,1,0,0],[1,0,0,0,0,0,0,0]]	3800
[[0,0,1,0],[0,0,0,0],[0,1,0,1],[1,0,0,0]]	**2100**
[[0,0,0,0,0,0],[0,1,1,1,1,0],[0,0,1,0,0,0],[1,0,0,1,0,1],[0,1,0,0,0,1],[0,0,0,0,0,0]]	3200



- **입출력 예 #3:** 오른쪽 그림과 경주로를 건설하면 직선 도로 6개, 코너 3개로 총 2100원이 듭니다.

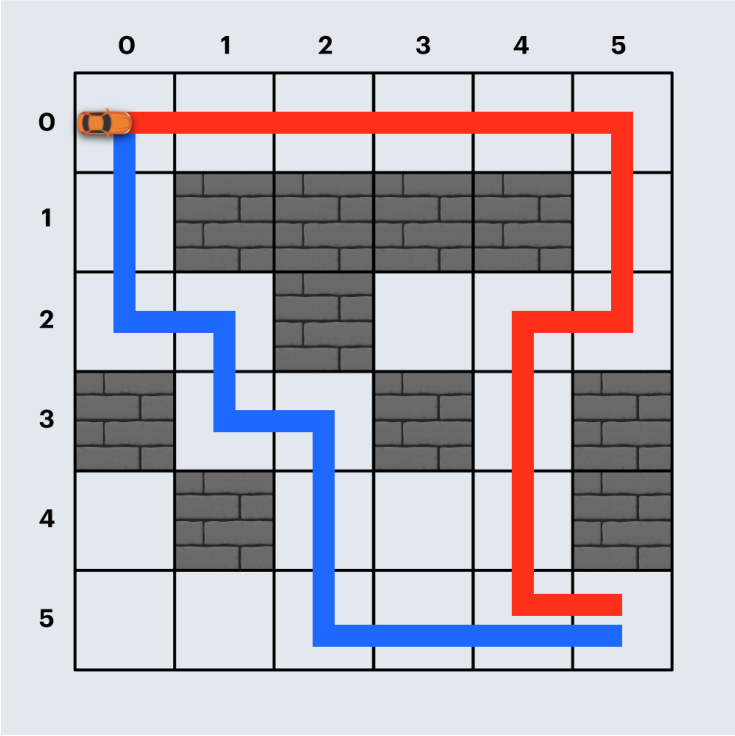
★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/67259

문제 4. 경주로 건설 (8/8)

제한시간: 90분

- **입출력 예제**

board	result
[[0,0,0],[0,0,0],[0,0,0]]	900
[[0,0,0,0,0,0,0,1],[0,0,0,0,0,0,0,0],[0,0,0,0,0,1,0,0],[0,0,0,0,1,0,0,0],[0,0,0,1,0,0,0,1],[0,0,1,0,0,0,1,0],[0,1,0,0,0,1,0,0],[1,0,0,0,0,0,0,0]]	3800
[[0,0,1,0],[0,0,0,0],[0,1,0,1],[1,0,0,0]]	2100
[[0,0,0,0,0,0],[0,1,1,1,1,0],[0,0,1,0,0,0],[1,0,0,1,0,1],[0,1,0,0,0,1],[0,0,0,0,0,0]]	**3200**



- **입출력 예 #4:** 붉은색 경로와 같이 경주로를 건설하면 직선 도로 12개, 코너 4개로 총 3200원이 듭니다. 만약, 파란색 경로와 같이 경주로를 건설한다면 직선 도로 10개, 코너 5개로 총 3500원이 들며, 더 많은 비용이 듭니다.

★ 문제 출처: https://school.programmers.co.kr/learn/courses/30/lessons/67259

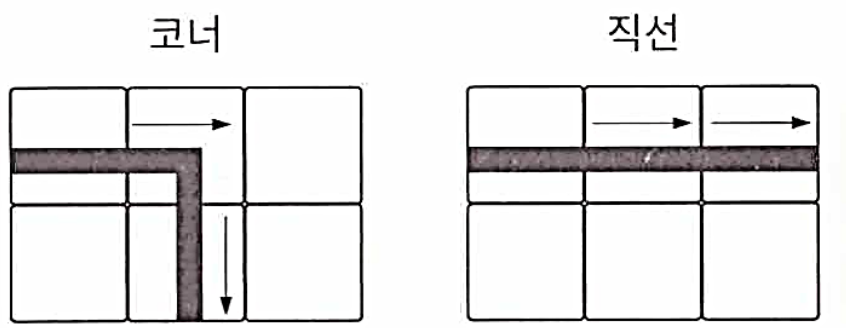
문제 4. 문제 분석

- **주어진 문제의 목표**
  - N × N 크기의 좌표가 주어지고, 출발점 위치는 (0, 0), 도착점 위치는 (N-1, N-1)이다.
  - 이때, 출발 지점에서 도착 지점까지 이어지는 경주로를 건설하는 데 필요한 최소 비용을 계산해서 반환하는 것이 본 문제의 목표이다.
- **문제의 특성**
  - 문제에서 "시작점"과 "끝점", 특히 "최소 비용"이 등장함.
    - 너비 우선 탐색을 사용하는 것이 바람직함.
    - (Why?) 너비 우선 탐색의 경우 깊이 우선 탐색과 달리 항상 최단 경로(= 최소 비용)을 보장함.
  - 좌표에서 이동 가능한 방향은 정해져 있다.
    - 이동 가능 방향: 상, 하, 좌, 우

문제 4. 문제 해결 전략 수립

- **문제 해결에서 집중해야 할 부분**
  - 직선 도로를 하나를 건설하는 데 소요되는 비용은 100원
  - 코너 하나를 건설하는 건설하는 데 소요되는 비용은 500원

- **너비 우선 탐색 시 고려해야 할 사항**
  - 큐에 추가해야 하는 정보가 무엇인가?
    - 현재까지의 도로 건설 비용
    - 현재 도로의 좌표
    - 도로의 직선 / 코너 여부 → 도로의 방향
      - ❖ "코너"란 현재 도로의 방향과 새로 건설하는 도로의 방향이 다른 경우임.



문제 4. 문제 해결 전략 수립

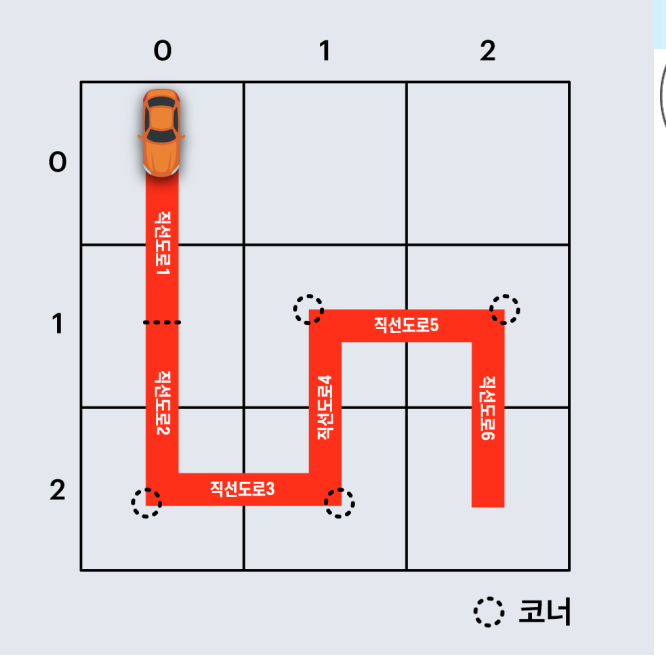
- **너비 우선 탐색을 수행**
  - ➢ 너비 우선 탐색을 통해 도착점에 도달하면 해당 경로의 비용과 지금까지 계산한 비용을 비교하여 더 적은 값으로 갱신함으로써 최소 건설 비용을 탐색할 수 있다.

- **탐색 관련 문제에서 자주 사용되는 기능**
  - ➢ 아래 기능들은 탐색 관련 문제에서 공통적으로 사용되는 기능들이므로 활용 사례와 구현 방법을 정확하게 숙지할 필요가 있다.
    - ✓ **좌표를 벗어난 경우를 체크하는 기능**
    - ✓ **탐색 가능한 범위인가를 체크하는 기능**
    - ✓ **탐색 비용을 계산하는 기능**

문제 4. 문제 해결 전략 수립

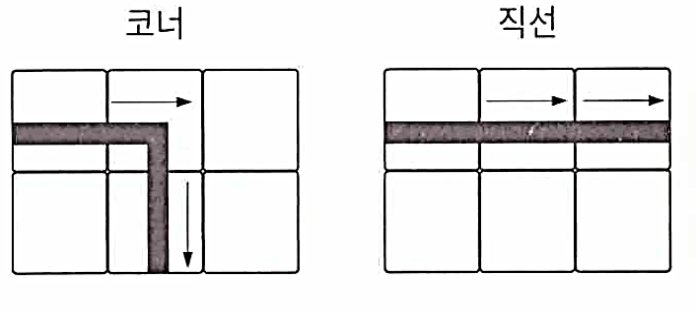
- **도로 건설 비용을 계산하는 방법**
  - 해당 칸의 경로가 직선인 경우: 기존 비용에 100원을 더함.
  - 해당 칸의 경로가 코너인 경우: 기존 비용에 600원을 더함.

    (Why?) 기존 방향 또는 현재 방향은 직선 코스이므로 100원
    나머지 방향은 다른 방향이므로, 코너가 되어 500원



- **직선/코너 여부를 판별하는 방법**
  - "이전의 진행 방향"과 "현재 진행 방향" 모두를 알아야 함.
  - 이를 위해 오른쪽 그림과 같이 오프셋 정보를 저장하여 탐색 시 활용할 수 있음.
  - 이전 방향과 현재 방향이 같으면 "**직선**", 다르면 "**코너**"
  - 이전 방향과 현재 방향이 서로 반대면 → 탐색 X

    (Why?) 문제에서 반대 방향으로 탐색은 안 한다고 했으므로...



directions

인덱스	0		1		2		3	
오프셋 좌표	0	-1	-1	0	0	1	1	0

문제 4. 구현 및 검증

```
def solution4(board):
    # 주어진 좌표가 보드의 범위 내에 있는지 확인
    def is_valid(x, y):
        return 0 <= x < N and 0 <= y < N

    # 주어진 좌표가 차단되었거나 이동할 수 없는지 확인
    def is_blocked(x, y):
        return (x, y) == 0 or not is_valid(x, y) or board[x][y] == 1

    # 이전 방향과 현재 방향에 따라 비용 계산
    def calculate_cost(direction, prev_direction, cost):
        if prev_direction == -1 or (prev_direction - direction) % 2 == 0:
            return cost + 100
        else:
            return cost + 600
```

문제 4. 구현 및 검증

```
# 주어진 좌표와 방향이 아직 방문하지 않았거나 새 비용이 더 작은 경우
def is_should_update(x, y, direction, new_cost):
    return visited[x][y][direction] == 0 or visited[x][y][direction] > new_cost

queue = [(0, 0, -1, 0)]
N = len(board)
directions = [(0, -1), (-1, 0), (0, 1), (1, 0)]
visited = [[[0 for _ in range(4)] for _ in range(N)] for _ in range(N)]
answer = float("inf")

# 큐가 빌 때까지 반복
while queue:
    x, y, prev_direction, cost = queue.pop(0)
    # 이동 가능한 모든 방향에 대해 반복
    for direction, (dx, dy) in enumerate(directions):
        new_x, new_y = x + dx, y + dy
```

문제 4. 구현 및 검증

```
            # 이동할 수 없는 좌표는 건너뛰기
            if is_blocked(new_x, new_y):
                continue

            new_cost = calculate_cost(direction, prev_direction, cost)

            # 도착지에 도달한 경우 최소 비용 업데이트
            if (new_x, new_y) == (N-1, N-1):
                answer = min(answer, new_cost)
            # 좌표와 방향이 아직 방문하지 않았거나 새 비용이 더 작은 경우 큐에 추가
            elif is_should_update(new_x, new_y, direction, new_cost):
                queue.append((new_x, new_y, direction, new_cost))
                visited[new_x][new_y][direction] = new_cost

    return answer
```

문제 4. 구현 및 검증

테스트 케이스 #1

```
board= [[0,0,0],[0,0,0],[0,0,0]]
print( solution4(board) )
```

테스트 케이스 #2

```
board= [   [0,0,0,0,0,0,0,1],[0,0,0,0,0,0,0,0],[0,0,0,0,0,1,0,0],
           [0,0,0,0,1,0,0,0],[0,0,0,1,0,0,0,1],[0,0,1,0,0,0,1,0],
           [0,1,0,0,0,1,0,0],[1,0,0,0,0,0,0,0]
       ]
print( solution4(board) )
```

문제 4. 구현 및 검증

테스트 케이스 #3

```
board= [[0,0,1,0],[0,0,0,0],[0,1,0,1],[1,0,0,0]]
print( solution4(board) )
```

테스트 케이스 #4

```
board= [  [0,0,0,0,0,0],[0,1,1,1,1,0],[0,0,1,0,0,0],
          [1,0,0,1,0,1],[0,1,0,0,0,1],[0,0,0,0,0,0]
       ]
print( solution4(board) )
```

Q & A