**그래프**

**기초 용어 정리**

**그래프는 정점과 간선으로 구성하는 자료구조**를 뜻한다. 상당히 많은 상황을 그래프로 표현할 수 있으며, 많은 알고리즘 문제 역시 그래프로 표현하여 문제를 해결할 수 있다.

- **정점(Vertex , Node)** : Vertex라고도 표현하고 Node라는 용어를 사용하기도 한다. 같은 것이라고 생각하면 되겠다. 정점은 위 그래프에서 동그라미 친 1,2,3,4,5,6 과 같은 각각의 지점을 의미한다.

- **간선(Edge, Link)**: 역시 Edge와 Link 두가지 용어를 사용한다. 그래프에서 각 정점들을 연결하는 선이다. 화살표 표시가 된 것도 있고 그냥 직선으로 표시된 것도 있는데, 둘 다 '간선' 이라고 사용한다.

- **가중치** : 간선 위에 표시된 숫자이다. 가중치가 있는 그래프가 있고 없는 그래프가 있는데, 가중치가 없다면 모두 동일한 가중치를 가진다고 보면 된다. 해당 간선을 타고 이동할 때 필요한 비용 등을 표현하는 것에 사용된다.

- **Directed Graph** : 말 그대로 방향성이 있는 그래프이다. 위 두개의 그래프 중 왼쪽 그래프가 Directed Graph이다. 간선의 방향에 따른 이동만이 가능한 경우에 사용한다.

- **undirected Graph** : 오른쪽 그래프 처럼 간선이 방향성이 없는 일반 실선으로 표시된 그래프이다. 방향성이 없다고 이동을 못하는 것이 아니라, 모든 간선들은 양방향 이동이 가능함을 의미한다.

도표, 라인, 텍스트, 원이(가) 표시된 사진

자동 생성된 설명

- **Self-loops** : 정점에서 자기 자신으로 돌아오는 간선을 뜻한다. 오른쪽 그래프에서 6번 정점에 연결된 가중치 2의 간선이 이에 해당한다.

- **Adjacency** : 방향성이 있는 directed graph에서의 간선으로 연결된 인접 정점을 뜻한다. 하지만 이는 쌍방향 모두에 적용되는 것이 아니다. 예를들어 위 Directed Graph에서 정점4가 정점1에 Adjacency하다고 한다면, 정점1은 정점4에 Adjacency하다고 표현할 수 없다.

- **Degree** : 각 정점에 연결되어 있는 Edge들의 수이다. 이 중 out-degree는 해당 정점에서 나가는 간선을, in-degree는 해당 정점으로 들어오는 간선이다. 따라서 특정 정점에서의 모든 out-degree와 in-degree를 더하면 그 정점의 Degree가 된다.

- **Path** : 정점 A에서 정점 B로 이동할 수 있는 경로를 의미한다. 그래프 내에서 정점 끼리의 path는 하나만 존재할 수 있는 것이 아니라, 여러 종류의 길이 존재할 수 있다. 또한 두 개의 정점이 이동할 수 없기도 한다. 여기서 정점A 에서 정점 B로 가는 Path가 존재한다면 정점 B는 정점 A로부터 reachable하다고 표현한다.

라인, 원, 텍스트, 도표이(가) 표시된 사진

자동 생성된 설명

- **Simple Path**: 경로 상의 모든 정점들이 중복되지 않는다면 이 경로를 simple path라고 한다. 예를 들어 위의 그래프에서 경로 <1,4,2,3,5> 는 simple path 라고 할 수 있지만 <1,4,2,1>은 simple path가 아니다.

- **Cycle** : 그래프가 path를 따라 동일한 정점으로 돌아올 수 있는 경우를 의미한다. 위 그래프의 왼쪽 컴포넌트는 <1,4,2> path가 cycle을 이룬다. 이 중, 돌아오는 경로 안에서 중복된 노드가 시작점(끝점) 이외에는 발생하지 않는 경우를 Simple Cycle이라고 표현한다.

-**An acyclic graph** : 그래프에 싸이클이 없는 경우 그 그래프를 acyclic graph라고 부른다.

- **Connected Graph** : 기본적으로 connected graph는 방향성이 없는 그래프이다. 모든 쌍의 정점들이 path로 연결이 가능한 그래프를 connected graph라고 칭한다.

- **Connected Component** : 위 그래프 전체도 하나의 그래프이다. 이 중 방향이없는 그래프의 정점 하위 집합을 최대로 연결한 것을 Connected Component라고 한다.

- **Forest**: 그래프 중 Acyclic하고 Undirected하면 Forest라고 부른다.

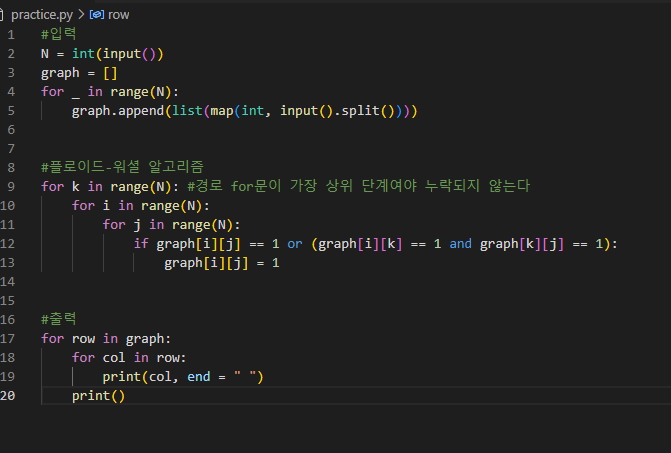
- **Tree**: Forest에서 하나 하나의 Component들은 Tree가 된다. 따라서 Connected, Acyclic, Undirected 모두를 만족하는 graph가 Tree라고 볼 수 있다.

- **Dag** : Tree가 방향성이 없고 싸이클이 존재하지 않는 그래프라면, Dag는 방향성이 있는 그래프 중 싸이클이 나오지 않는 그래프를 의미한다.

**그래프 기본 알고리즘**

그래프 형태의 자료구조로 표현한 문제를 해결하는 것에는 매우 다양한 알고리즘이 존재한다. 이 중 가장 대표적인 것이 그래프 탐색 기법 중 Breadth-first search(BFS)와 Depth-first search(DFS)이다. 그래프 내의 최단 경로를 구하는 것은 다익스트라, 벨만-포드, 플로이드 등의 알고리즘이 있다.

**백준 11403 경로찾기 문제**

****

* N은 그래프의 정점 수입니다.
* graph는 그래프의 인접 행렬입니다.
* input() 함수를 사용하여 표준 입력에서 정수 입력을 받습니다.
* list() 함수를 사용하여 정수 목록을 생성합니다
* for k in range(N)는 중간 정점 k를 순회하는 반복문입니다.
* for i in range(N)는 시작 정점 i를 순회하는 반복문입니다.
* for j in range(N)는 도착 정점 j를 순회하는 반복문입니다.
* graph[i][j] == 1 or (graph[i][k] == 1 and graph[k][j] == 1)는 i에서 j로 가는 경로가 있는지 여부를 확인하는 조건입니다.
* graph[i][j] = 1은 i에서 j로 가는 경로가 있으면 i와 j 사이의 간선을 1로 설정합니다
* for row in graph는 그래프의 각 행을 순회하는 반복문입니다.
* for col in row는 행의 각 열을 순회하는 반복문입니다.
* print() 함수를 사용하여 그래프를 표준 출력에 출력합니다.

위 코드는 Floyd-Warshall 알고리즘을 사용하여 그래프의 전이 폐쇄를 찾는 코드입니다. Floyd-Warshall 알고리즘은 그래프의 모든 쌍의 정점 사이의 최단 경로를 찾는 알고리즘입니다.

위 코드는 다음과 같은 순서로 동작합니다.

1. N과 graph를 초기화합니다.
2. Floyd-Warshall 알고리즘을 수행합니다.
3. 그래프의 전이 폐쇄를 출력합니다.

Floyd-Warshall 알고리즘 설명:

Floyd-Warshall 알고리즘은 다음과 같은 점화식을 사용하여 동작합니다.

dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])

위 점화식은 i에서 j로 가는 최단 경로의 길이는 i에서 k로 가는 최단 경로의 길이와 k에서 j로 가는 최단 경로의 길이의 합보다 작거나 같음을 의미합니다.

Floyd-Warshall 알고리즘은 다음과 같은 3중 반복문을 사용하여 위 점화식을 구현합니다.

for k in range(N):

for i in range(N):

for j in range(N):

if dist[i][j] > dist[i][k] + dist[k][j]:

dist[i][j] = dist[i][k] + dist[k][j]

이 코드는 k를 중간 정점으로 하여 i에서 j로 가는 최단 경로를 갱신합니다.

위 코드는 for k in range(N)가 가장 상위 단계에 있는 이유는 i와 j의 최단 경로를 찾을 때 k를 중간 정점으로 고려해야 하기 때문입니다. 만약 for i in range(N)가 가장 상위 단계에 있다면 k를 중간 정점으로 고려하지 못하여 최단 경로를 찾지 못할 수 있습니다