그래프 : 노드와 엣지와 같은 성분들의 집합

🡪 그래프를 그리는 두 가지 방법

1. Adjacency matrix : 행렬을 이용해 그린다. dense graph에서 많이 사용

2. Adjacency list : Linked list를 이용해 그린다. sparse graph에서 많이 사용

🡪 신장나무(spanning tree)를 그리는 2가지 탐색 방법

1. DFS : stack을 이용해 구현할 수 있다. 한 우물만 파는 방식
2. BFS : queue를 이용해 구현할 수 있다. 두루두루 탐색하는 방식

🡪 가중치 그래프의 최소비용 신장나무를 그리는 2가지 탐색 방법

1. PFS : heap을 이용해 구현할 수 있다. node 위주의 가중치 탐색
2. Kruskal Algorithm : heap을 이용해 구현할 수 있다. edge 위주의 가중치 탐색

🡪 가중치 그래프의 최소비용 경로를 찾는 2가지 방법

1. Priority-base Algorithm : 앞서 사용한 PFS에서 원하는 노드까지 가는데 걸리는 비용들의 누적합을 통해 원하는 노드까지의 최소비용 경로, 최소비용을 찾을 수 있다.
2. Dijkstra Algorithm : 특정 노드를 기준으로 모든 노드들까지의 최단경로를 확인할 수 있는 알고리즘이다.

🡪 Directed 그래프의 4가지 고려사항

1. Reachability : DFS 함수의 변형을 통해 그래프속 노드에서 다른 특정 노드로 갈 수 있는지 조사할 수 있다.
2. Shortest path : Floyd Algorithm 을 이용해 노드간의 경로의 최소비용을 구할 수 있다.

노드 사이의 간선과, 기존의 경로들을 비교하여 최소 비용을 업데이트 하는 방식이다.

1. Topological sort :

Togological sorting의 경우 선행작업이 모두 수행되면 해당하는 작업을 수행하는 것으로 , 선행차수는 함수내에서 노드의 연결성분들을 조사하여 구한다.

Reverse Topological sorting은 후속작업이 모두 완료되면 해당하는 작업을 수행

1. Strongly Connected : AP\_recur 함수의 변형을 통해 해당 노드간 상호 연결이 되어있는지 확인할 수 있다.

🡪 Network

1. Critical activity : Topological sorting, Reverse Topological sorting을 이용해 전체 작업일정에 영향을 주는 critical activity, critical path를 찾을 수 있다.
2. Ford-Fulkerson Algorithm : capacity 그래프를 통해 flow, residual 그래프를 만들 수 있다.
3. Graph Matching : Biparite graph를 이용해 구성원들간의 그룹을 구성할 수 있다.