그래프 정리

그래프

그래프가 뭐야?

- 정의상으론 '정점과 그 정점를 연결하는 간선을 하나로 모아놓은 자료 구조'라고 한다
  - 개인적으로 잘 와닿지 않는다. 딱딱하다.
- 내 식대로 이해하자면 '트리의 확장판'인 것 같다.
  - 트리는 계층 구조만을 표현할 수 있었다면
  - 그래프는 그런 계층 구조를 포함한 다양한 구조, 노드 간의 상호관계를 표현하는 자료구조인거지

종류

- 무방향 그래프 vs 방향 그래프
  - 방향이 있냐 없냐
- 가중치 그래프
  - 간선에 가중치를 두는가
- 단순 그래프 vs 다중 그래프
  - 두 정점 사이에 간선이 한개만 있냐 그 이상 있냐
- 이분 그래프
  - 정점 그룹 두개가 있고 서로 다른 그룹에 속한 정점끼리만 간선이 존재하는 그래프
- 사이클 없는 방향 그래프 (DAG, Directed Acyclic Graph)
  - 말 그대로 자기 자신에게 돌아오는 사이클이 없는 방향 그래프
  - 여러 작업의 상호 의존 관계를 표현하는데에 쓰임
- 연결 그래프
  - 어느 정점으로든 경로가 존재하는 그래프
  - 여기에 사이클도 없으면 트리임
- 완전 그래프
  - 모든 정점이 서로 연결되어 있는 그래프
- 희소 그래프 vs 밀집 그래프
  - 간선의 수가 아주 적냐 아주 많냐

용어

- 정점
- 간선
- 인접 정점
- 차수
- 경로
- 단순 경로: 같은 간선을 두 번 이상 지나지 않는 경로
- 부분 그래프: 그래프의 부분 집합 같은 개념

- 사이클: 자기 자신으로 돌아오는 단순 경로

구현 방법

인접 리스트

- 각 정점마다 어느 정점로 연결되는지 리스트를 두는 방식
- 자바로 치면
  - `ArrayList<Integer>[] nodes;`
  - `nodes[0].add(1);` ➡ 0번 정점은 1번으로의 간선을 가진다
- 자바스크립트로 치면
  - `let nodes = [[]];`
  - `nodes[0].push(1);` ➡ 0번 정점은 1번으로의 간선을 가진다

인접 행렬

- 2차원 boolean 배열로 연결되어 있는지를 표현하는 방식
- `boolean[][] matrix;`
  - matrix[0][1]이 true라면 ➡ 0번 정점은 1번으로의 간선을 가진다

뭐를 써야 할까?

- 메모리 효율의 문제
  - 인접 행렬은 '항상' (정점 개수의 제곱)만큼의 공간이 필요
  - 인접 리스트는 (정점 개수 + 간선 개수) 뿐
- 시간 효율의 문제
  - 인접 행렬은 두 정점을 잇는 간선이 있는지 바로 알 수 있음
  - 인접 리스트는 정점이 간선을 모두 순회하며 비교해야 알 수 있음
- 위의 사항을 고려하여 사용하자
  - ex) 밀집 그래프는 인접 행렬, 희소 그래프는 인접 리스트

깊이 우선 탐색(DFS)

- 최대한 깊이 들어가는 탐색법.
- 여태 트리에서 DFS 하던대로 하면 된다.
- 물론 그래프는 '사이클'이 존재하니 '루트에서 자식으로 들어갈 때 자식 ➡ 루트의 연결을 끊는 방법'은 통하지 않겠다.
  - visited 무조건 필요

너비 우선 탐색(BFS)

- 가까운 정점을 먼저 방문하는 탐색법.
- 큐를 사용해야 한다.

구현

0. 방문한 것을 재방문하지 않게 visited같은 변수를 둔다.
1. 큐에 시작 정점을 push 한다
2. 큐가 빌 때까지 아래를 반복한다
    1. 큐에서 정점을 꺼낸다
    2. 방문한다
    3. 인접 정점들을 큐에 넣는다

최단 경로 알고리즘

뭐를 써야 할까?

- 가중치 없으면 BFS
- 가중치 있으면 다익스트라
- 가중치가 음수도 될 수 있다면 벨만-포드
- 모든 정점 쌍에 대한 최단경로라면 플로이드-와샬

BFS

- 정점 A에서 BFS를 시작해서 B에 닿기까지의 깊이가 곧 최단거리

구현

0. distance[]를 둔다. distance[i]는 정점 A로부터 정점 i까지의 깊이를 담는다
    - 아직 방문하지 않았으면 -1로 두는 식으로 visited 역할도 함
1. 큐에 정점 A를 push 한다
2. 큐가 빌 때까지 아래를 반복한다
    1. 큐에서 정점을 꺼낸다
    2. 인접 정점 중 아직 방문 안한 정점이 있다면
        1. 현재 깊이 + 1을 그 정점의 깊이로 정해주고
        2. 큐에 해당 정점을 넣는다
- distance[i]는 정점 A-정점 i의 최단 거리
    - 원하면 B에 도달했을때 바로 return 등으로 나가도 됨

다익스트라

- 모든 정점들을 방문하며 시작 정점으로부터의 최단 거리를 구한다
- '바로 정점 Z를 갔을 때'와 '다른 우회경로로 Z를 갔을 때'를 비교해 더 적은 값을 구하는 게 핵심
- 여러 작은 경로의 최단거리가 모여서 큰 경로의 최단거리를 만든다는 점에서 다이나믹 프로그래밍을 활용한 알고리즘으로 볼 수 있다
- 우선순위 큐를 사용하면 더 시간 효율적으로 구현할 수 있다.

구현 (우선순위 큐)

0. distance[]를 둔다. distance[i]는 정점 A로부터 정점 i까지의 깊이를 담는다. 자기 자신까지의 거리는 0으로, 나머지 정점까지의 거리는 무한대로 초기화한다. 또, 기준을 가중치 값으로 하는 우선순위 큐를 만들어둔다. 한 아이템은 [도착 정점번호, 가중치]의 쌍이다. 자기 자신에게 도착하는 가중치를 0으로 집어넣어 둔다.
1. 우선순위 큐가 빌때까지 아래를 반복한다
    1. 큐에서 한 쌍을 꺼낸다. 각각 node, weight로 칭하겠다
    2. distance[꺼낸 정점 번호]보다 weight가 더 작을때만 아래를 수행한다
        1. 꺼낸 정점의 인접노드를 살펴보면서
        2. weight + 인접노드의 weight < distance[인접노드] 일 경우 우선순위 큐에 이 인접노드와 더 작아진 가중치 값을 넣는다.

플로이드-와샬
벨만-포드

최소 스패닝 트리(MST, Minimum Spanning Tree)

- 스패닝 트리란
    - 최소 연결 부분 그래프
        - 최소 연결: '트리' (최소로 연결하면 간선개수==정점개수-1이 되고 이건 트리 정의와 같음)
        - 부분 그래프: 그래프의 일부
    - 즉 그래프의 일부 간선만 취해서 만든 트리
- 최소 스패닝 트리란
    - 스패닝 트리 중 간선의 가중치 합이 최소가 되는 트리
- 보통 Kruskal 알고리즘으로 최소 스패닝 트리를 구한다.