graphDFS - Concept

DFS란?

- 그래프의 모든 정점을 발견하는 가장 단수하고 고전적인 방법
- 간선과 노드가 있을 때, 가장 깊은 곳까지 내려간 이후에 순차적으로 아직 돌지 않은 곳을 순회한다.
- 예시

```
// 그래프의 인접 리스트 표현
vector<vector<int> > adj;
// 각 정점 방문 여부 확인
vector<bool> visited;
// 깊이 우선 탐색 구현
void dfs(int here) {
        cout << "DFS vistis" << here << endl;
        visited[here] = true;
        // 모든 인접 정점 순회
        for (int i = 0; i < adj[here].size(); i++) {
                int there = adj[here][i];
                // 아직 방문한 적 없으면 방문
                if (!visited[there])
                        dfs(there);
        }
        // 더이상 방문할 정점이 없으니, 재귀 호출 종료하고 이전 정점으로 복귀
}

// 모든 정점 방문
void dfsAll() {
        //visited를 모두 false로 표기화
        visited = vector<bool>(adj.size(), false);
        // 모든 정점을 순회하면서, 아직 방문한 적 없으면 방문
```

```
        for (int i = 0; i < adj.size(); i++)
                if (!visited[i])
                        dfs(i);
}
```

- 깊이 우선 탐색의 시간 복잡도
  - 리스트 -> O(|V| + |E|) : V(정점의 개수), E(간선 개수)
  - 행렬 -> O(V^2)

위상 정렬

- 의존성이 있는 작업(방향이 있는 그래프)들이 주어질 때, 이들을 어떤 순서로 수행해야 하는 지 알려준다
- 각 작업을 정점으로 표현하고, 작업 간의 관계를 간선으로 표현한 방향 그래프를 의존성 그래프라 한다
- 의존성 그래프의 특징
  - 비순환 구조
- 위상 정렬을 구현하는 방법
  - 간선이 하나도 없는 정점들을 찾아서 정렬 결과의 뒤에 붙임
  - 그래프에서 해당 정점들을 찾아서 지움
  - 위 과정들을 지속적으로 반복한다
- dfs로 구현하는 방법
  - dfs가 종료할 때마다 현재 정점의 번호를 기록. 그렇게 dfs를 완전히 끝내고 기록된 순서를 뒤집으면 위상정렬 결과를 얻을 수 있다

오일러 서킷

- 그래프의 모든 간선을 정확히 한 번 지나서 시작점으로 돌아오는 경로를 찾는다
  - 한붓 그리기 문제도 이 유형 중 하나
- 오일러 서킷 존재 여부
  - 오일러 서킷이 존재할 수 없는 경우를 따진다
    - 그래프의 간선들이 두 개 이상의 컴포넌트에 의해 나누어져 있는 경우
  - 오일러 서킷 경로를 그려주는 시작점(u)과 끝점(v)을 잡아서 생각해보자

    - u != v인 경우, v는 홀수 개의 간선과 인접해 있음. v를 지나가기 위해서는 들어가는 간선 1, 나가는 간선 1개가 필요하다. 이는 v로 들어와서 나가려면 적어도 짝수의 간선이 필요하며 이게 불가능한 건 v와 접한 간선이 홀수 개인 것이다
    - u == v인 경우, v는 짝수 개의 간선과 인접해 있다
    - 이처럼 각 정점에 인접한 간선의 수가 오일러 서킷의 존재 여부와 밀접하다
- DFS를 이용한 오일러 서킷 찾기

```
// 그래프의 인접 행렬 표현, adj[i][j] = i와 j사이의 간선의 수
vector<vector<int> > adj;
// 무향 그래프의 인접 행렬 adj가 주어질 때 오일러 서킷을 계산한다
// 결과로 얻어지는 circuit을 뒤집으면 오일러 서킷이 된다
void getEulerCircuit(int here, vector<int>& circuit) {
        for (int there = 0; there < adj[here].size(); there++)
                while (adj[here][there] > 0) {
                        adj[here][there]--;
                        adj[there][here]--;
                        getEulerCircuit(there, circuit);
                }
        circuit.push_back(here);
}
```

- 오일러 트레일 : 모든 간선을 정확히 한 번 씩 지나지만, 시작점과 끝점이 다른 경로
    - 오일러 서킷 찾는 알고리즘에서 마지막으로 똑같은 지점으로 돌아가는 부분만 안 하면 오일러 트레일