2024 겨울학기 동국대학교SW역량강화캠프

9일차. 그래프 1

● 그래프 저장 방법 (2차원 배열)

▶ edge[x][y] = 0 이면 x와 y가 연결되어 있음을 뜻하고,
▶ edge[x][y] = 1 이면 x와 y가 연결되지 않았음을 뜻함.

장점:
- 직관적임.
- 두 정점 사이에 간선이 존재하는지를 O(1)만에 알 수 있음.

단점:
- 항상 N^2 크기의 배열이 필요함.
- 하나의 정점에 연결된 정점을 탐색하는데 항상 O(N)이 걸림.

● 그래프 저장 방법 (인접 리스트)

▶ 그래프 탐색 문제에서 “x와 연결된 정점 " 을 빠르게 찾을 수 있는 그래프 저장 형태

▶ x번 List에 x와 연결된 정점들을 저장하는 형태

```java
ArrayList<ArrayList<Integer>> edge = new ArrayList<>();
for(int i=0; i<=N; i++) edge.add(new ArrayList<Integer>());
while(M-- > 0){
    st = new StringTokenizer(br.readLine());
    int u = Integer.parseInt(st.nextToken());
    int v = Integer.parseInt(st.nextToken());

    edge.get(u).add(v);
    edge.get(v).add(u);
}
```

```python
edge = [[] for _ in range(N+1)]
for i in range(M):
    u, v = map(int, input().split())
    edge[u].append(v)
    edge[v].append(u)
```

● DFS와 BFS (4321)

문제

그래프를 DFS로 탐색한 결과와 BFS로 탐색한 결과를 출력하는 프로그램을 작성하시오. 단, 방문할 수 있는 정점이 여러 개인 경우에는 정점 번호가 작은 것을 먼저 방문하고, 더 이상 방문할 수 있는 점이 없는 경우 종료한다. 정점 번호는 1번부터 N번까지이다.

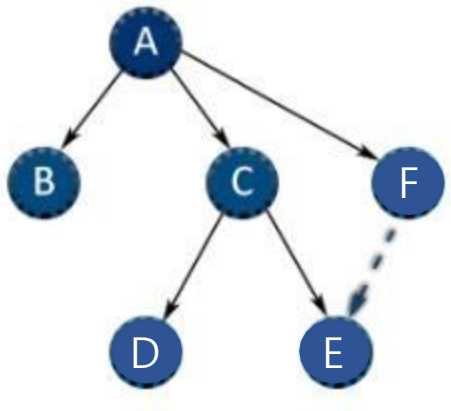
입력

첫째 줄에 정점의 개수 N($1 \le N \le 1,000$), 간선의 개수 M($1 \le M \le 10,000$), 탐색을 시작할 정점의 번호 V가 주어진다. 다음 M개의 줄에는 간선이 연결하는 두 정점의 번호가 주어진다. 어떤 두 정점 사이에 여러 개의 간선이 있을 수 있다. 입력으로 주어지는 간선은 양방향이다.

출력

첫째 줄에 DFS를 수행한 결과를, 그 다음 줄에는 BFS를 수행한 결과를 출력한다. V부터 방문된 점을 순서대로 출력하면 된다.

▶ DFS 정점의 탐색 순서를 출력

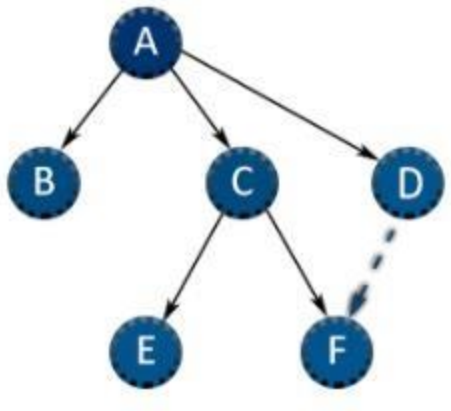


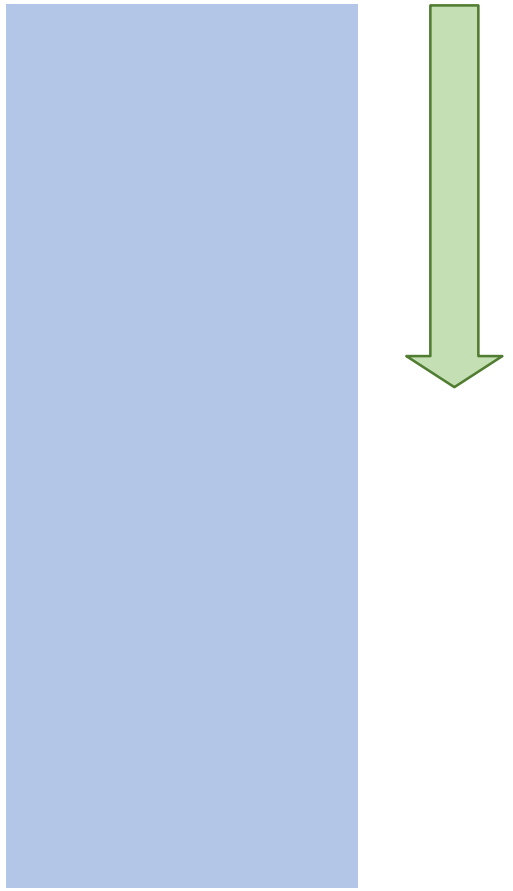
핵심 코드

```java
static void dfs(int v){
    if(chk[v]) return;
    chk[v] = true;
    System.out.print(v + " ");

    for(int u: edge.get(v)){
        dfs(u);
    }
}
```

```python
def dfs(v):
    global edge, chk

    if(chk[v]): return
    chk[v] = True

    print(v, end=' ')

    for u in edge[v]:
        dfs(u)
```

▶ BFS 정점의 탐색 순서를 출력





핵심 코드

```java
que.offer(V);
while(!que.isEmpty()){
    int v = que.poll();

    if(chk[v]) continue;
    chk[v] = true;
    System.out.print(v + " ");

    for(int u: edge.get(v)) que.offer(u);
}
```

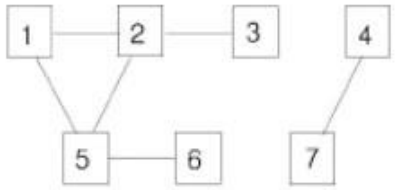
```python
# bfs
chk = [False] * (N+1)
que = deque();
que.append(V)
while que:
    v = que.popleft()

    if(chk[v]): continue
    chk[v] = True
    print(v, end=" ")

    for u in edge[v]:
        que.append(u)
```

● 바이러스 (877)

문제

신종 바이러스인 웜 바이러스는 네트워크를 통해 전파된다. 한 컴퓨터가 웜 바이러스에 걸리면 그 컴퓨터와 네트워크 상에서 연결되어 있는 모든 컴퓨터는 웜 바이러스에 걸리게 된다.

예를 들어 7대의 컴퓨터가 <그림 1>과 같이 네트워크 상에서 연결되어 있다고 하자. 1번 컴퓨터가 웜 바이러스에 걸리면 웜 바이러스는 2번과 5번 컴퓨터를 거쳐 3번과 6번 컴퓨터까지 전파되어 2, 3, 5, 6 네 대의 컴퓨터는 웜 바이러스에 걸리게 된다. 하지만 4번과 7번 컴퓨터는 1번 컴퓨터와 네트워크상에서 연결되어 있지 않기 때문에 영향을 받지 않는다.



< 그림 1 >

어느 날 1번 컴퓨터가 웜 바이러스에 걸렸다. 컴퓨터의 수와 네트워크 상에서 서로 연결되어 있는 정보가 주어질 때, 1번 컴퓨터를 통해 웜 바이러스에 걸리게 되는 컴퓨터의 수를 출력하는 프로그램을 작성하시오.

문제 해설

▶ 재귀함수를 이용한 풀이

▶ dfs 함수를 dfs(x) : x에서 갈 수 있는 모든 정점의 visited를 true로 바꾸는 함수

▶ dfs를 통해서 모든 갈 수 있는 정점들을 탐색

핵심 코드

```
int N = Integer.parseInt(br.readLine());
int M = Integer.parseInt(br.readLine());

edge = new ArrayList<>();
for(int i=0; i<=N; i++) edge.add(new ArrayList<>());

while(M-- > 0){
    st = new StringTokenizer(br.readLine());
    int u = Integer.parseInt(st.nextToken());
    int v = Integer.parseInt(st.nextToken());
    edge.get(u).add(v);
    edge.get(v).add(u);
}

chk = new boolean[N+1];
System.out.println(dfs(1) - 1);
```

```
static int dfs(int v){
    if(chk[v]) return 0;
    chk[v] = true;

    int sum=1;
    for(int u: edge.get(v)) sum += dfs(u);

    return sum;
}
```

● 연구활동 가는 길(3003)

문제

정올이는 GSHS에서 연구활동 때문에 교수님을 뵈러 A대학교를 가려고 한다.
출발점과 도착점을 포함하여 중간 지역은 n개이고, 한 지역에서 다른 지역으로 가는 방법이 총 m 개이다.
각 방법으로 지역 사이를 이동하는데는 어느 정도 비용이 필요하고, 한 지역에서 다른 지역으로 가는 어떠한 방법이 존재하면 같은 방법과 비용을 통해 역방향으로 갈 수 있다.
GSHS는 지역 1이고 A대학교는 지역 n이라고 할 때 대학까지 최소 비용을 구하여라.

n은 10 이하, m은 30 이하, 그리고 한 지역에서 다른 지역으로 가는 데에 필요한 비용은 200 이하 자연수이며
정점a->정점b로의 간선이 여러 개 있을 수 있으며, 자기 자신으로 가는 간선을 가질 수도 있다.

문제 해설

▶ DFS 완전탐색을 이용한 풀이

▶ dfs 함수를 dfs(x, dist) : 정점 x에 거리 dist로 도달한 상태, 인접한 정점들을 탐색하는 함수

▶ dfs를 통해서 모든 갈 수 있는 정점들을 탐색

핵심 코드

```
class Edge{
    private int to, dis;

    public Edge(int to, int dis){
        this.to = to;
        this.dis = dis;
    }

    public int getTo(){ return this.to; }
    public int getDis(){ return this.dis; }
}
```

```
static void dfs(int v, int d){
    if(minDis[v] != 0 && minDis[v] < d) return;
    minDis[v] = d;

    for(Edge u: edge.get(v)) dfs(u.getTo(), d + u.getDis());
}
```

```
edge = new ArrayList<>();
for(int i=0; i<=n; i++) edge.add(new ArrayList<>());
while(m-- > 0){
    st = new StringTokenizer(br.readLine());
    int u = Integer.parseInt(st.nextToken());
    int v = Integer.parseInt(st.nextToken());
    int d = Integer.parseInt(st.nextToken());

    edge.get(u).add(new Edge(v, d));
    edge.get(v).add(new Edge(u, d));
}

minDis = new int[n+1];
dfs(1, 1);

System.out.print(minDis[n]-1);
```