

#2. 그래프 탐색과 위상 정렬

나정휘

<https://justicehui.github.io/>

그래프의 탐색

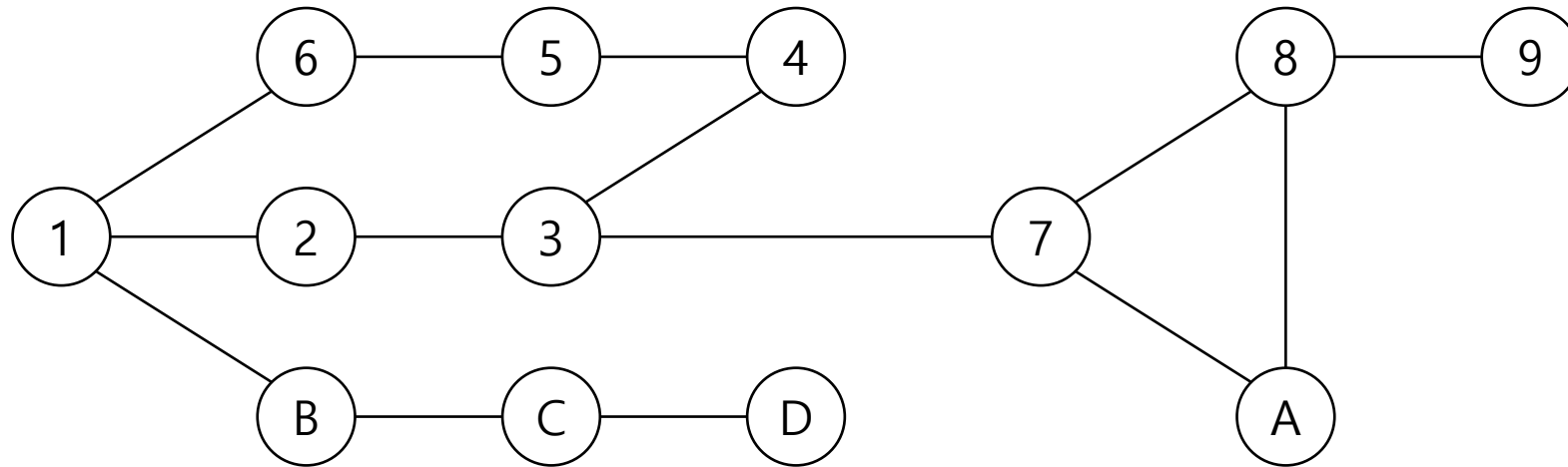
그래프의 탐색

- 그래프의 정점을 적당한 순서로 한 번씩 모두 보는 방법이라고 생각하면 편함
- 깊이 우선 탐색
- 너비 우선 탐색

그래프의 탐색

깊이 우선 탐색 (Depth-First Search, DFS)

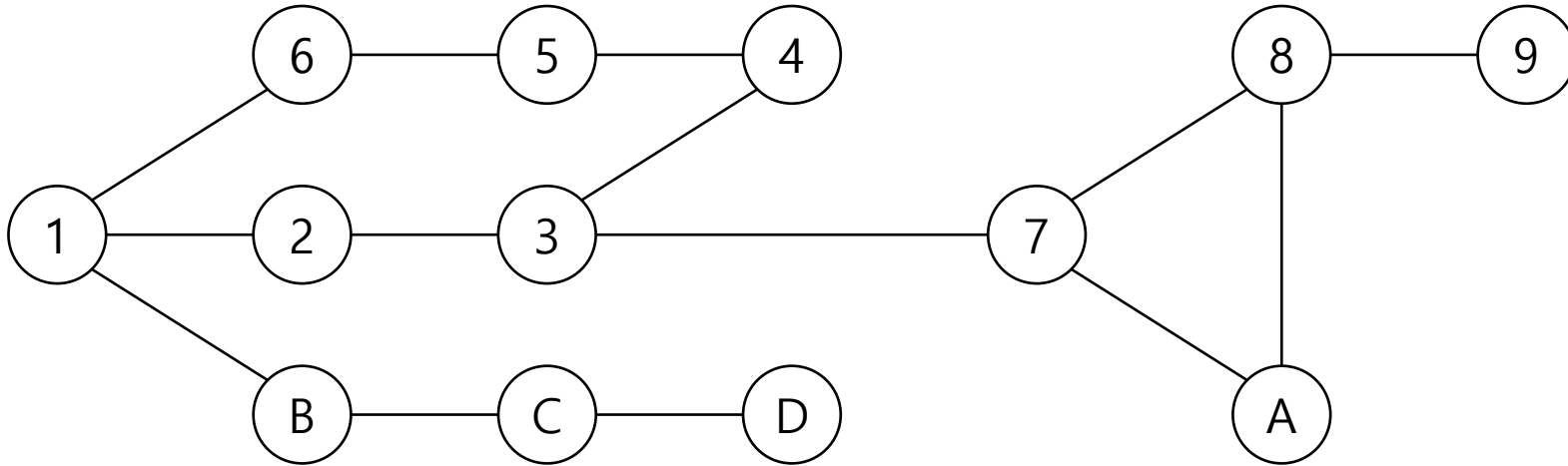
- 한 정점에서 시작
- 현재 정점과 인접한 정점 중 아직 방문하지 않은 정점을 하나 선택해 방문
- 만약 인접한 정점을 모두 방문한 상태라면 이전 정점으로 복귀



그래프의 탐색

깊이 우선 탐색 (Depth-First Search, DFS)

- 만약 인접한 정점을 모두 방문한 상태라면 이전 정점으로 돌아감
 - 스택이나 재귀를 이용해서 구현할 수 있음
 - PS할 때는 보통 재귀를 사용해서 구현함
 - 스택을 사용하는 구현은 함수 호출 스택을 직접 구현하면 됨



```
#include <bits/stdc++.h>
using namespace std;

int N, M, S, G[1010][1010], C[1010];

void DFS(int v){
    cout << v << " ";
    C[v] = 1;
    for(int i=1; i<=N; i++){
        if(G[v][i] && !C[i]) DFS(i);
    }
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M >> S;
    for(int i=0; i<M; i++){
        int u, v; cin >> u >> v;
        G[u][v] = G[v][u] = 1;
    }
    DFS(S);
}
```

그래프의 탐색

깊이 우선 탐색 (Depth-First Search, DFS)

- 인접 리스트 / 간선 리스트를 사용한 구현

```
#include <bits/stdc++.h>
using namespace std;

int N, M, S, C[1010];
vector<int> G[1010];

void DFS(int v){
    cout << v << " ";
    C[v] = 1;
    for(auto i : G[v]) if(!C[i]) DFS(i);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M >> S;
    for(int i=0; i<M; i++){
        int u, v; cin >> u >> v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    DFS(S);
}
```

```
#include <bits/stdc++.h>
using namespace std;

int N, M, S, C[1010];
int Head[1010], Next[20202], To[20202], E;

void AddEdge(int s, int e){
    int id = ++E;
    To[id] = e;
    Next[id] = Head[s];
    Head[s] = id;
}

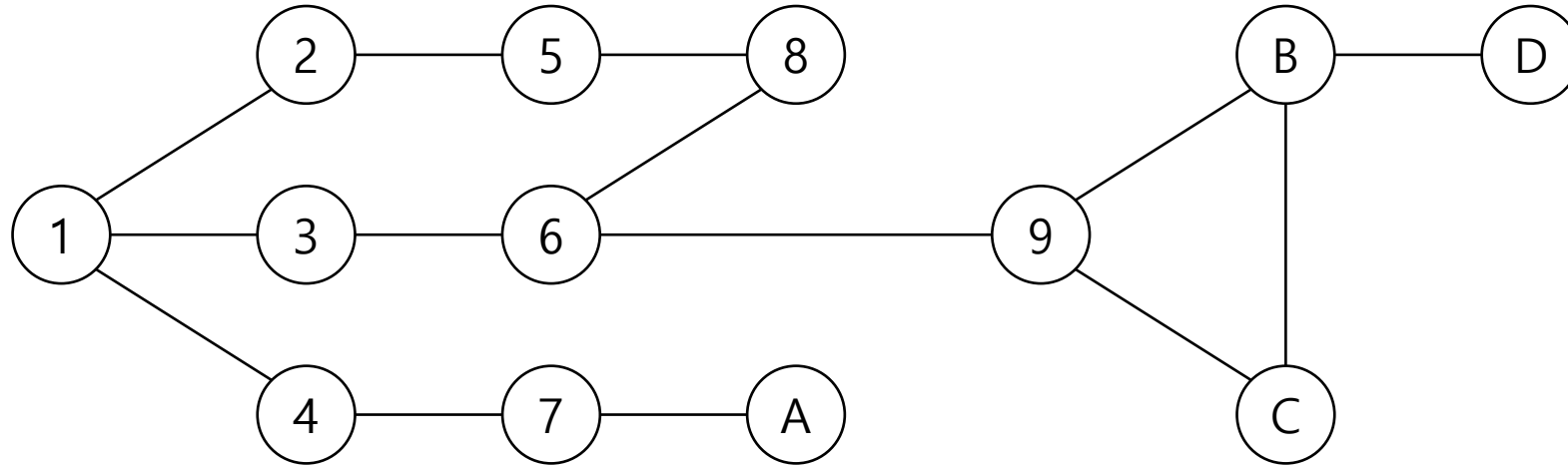
void DFS(int v){
    cout << v << " ";
    C[v] = 1;
    for(int i=Head[v]; i; i=Next[i]) if(!C[To[i]]) DFS(To[i]);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M >> S;
    vector<pair<int,int>> V;
    for(int i=0; i<M; i++){
        int u, v; cin >> u >> v;
        AddEdge(u, v);
    }
    DFS(S);
}
```

그래프의 탐색

너비 우선 탐색 (Breadth-First Search, BFS)

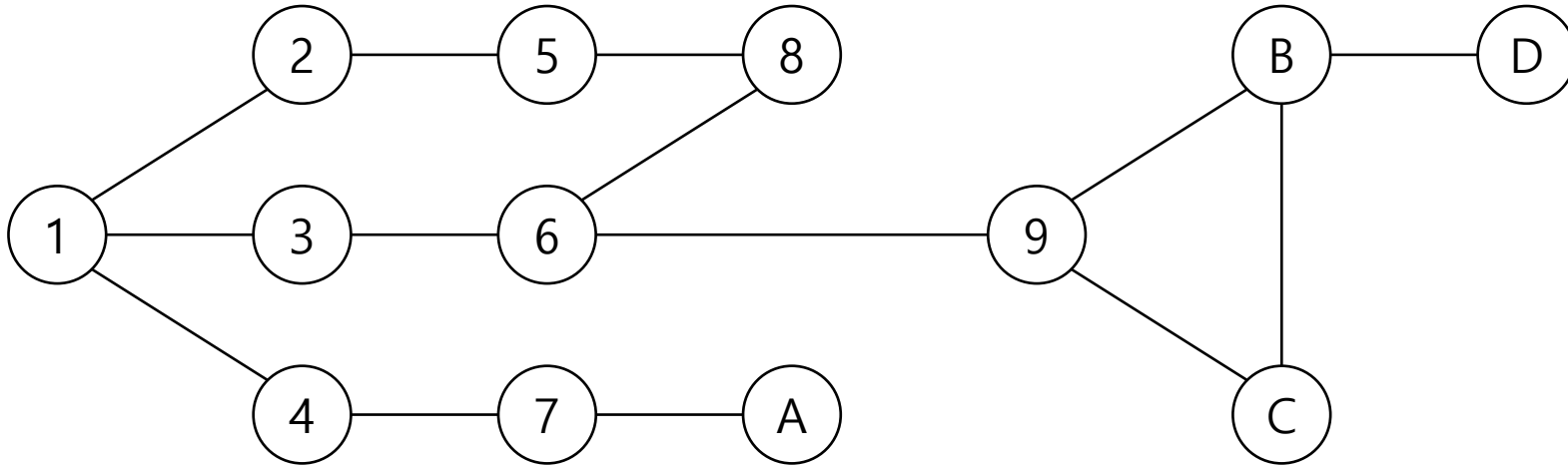
- 한 정점에서 시작
- 현재 정점과 인접한 정점 중 아직 방문하지 않은 정점을 모두 방문



그래프의 탐색

너비 우선 탐색 (Breadth-First Search, BFS)

- 현재 정점과 인접한 정점 중 아직 방문하지 않은 정점을 모두 방문
 - 정점들의 방문 "대기열"을 만들어야 하므로 큐를 사용해서 구현
 - 시작 정점과 거리가 가까운 정점부터 방문하게 됨
 - 큐에 넣는 시점에 방문 체크를 해야 함
 - 큐에서 빼는 시점에 체크하면 안 됨



```
void BFS(int v){
    queue<int> Q;
    Q.push(v); C[v] = 1;
    while(!Q.empty()){
        v = Q.front(); Q.pop();
        cout << v << " ";
        for(int i=1; i<=N; i++){
            if(G[v][i] && !C[i]) Q.push(i), C[i] = 1;
        }
    }
}
```

그래프의 탐색

너비 우선 탐색 (Breadth-First Search, BFS)

- 인접 리스트 / 간선 리스트를 사용한 구현

```
int N, M, S, C[1010];
vector<int> G[1010];

void BFS(int v){
    queue<int> Q;
    Q.push(v); C[v] = 1;
    while(!Q.empty()){
        v = Q.front(); Q.pop();
        cout << v << " ";
        for(auto i : G[v]) if(!C[i]) Q.push(i), C[i] = 1;
    }
}
```

```
int Head[1010], Next[2020], To[2020], E;

void BFS(int v){
    queue<int> Q;
    Q.push(v); C[v] = 1;
    while(!Q.empty()){
        v = Q.front(); Q.pop();
        cout << v << " ";
        for(int i=Head[v]; i; i=Next[i]) if(!C[To[i]])
            Q.push(To[i]), C[To[i]] = 1;
    }
}
```

그래프의 탐색

깊이 우선 탐색 vs 너비 우선 탐색

- 구현 방법
 - DFS: 재귀 or 스택
 - BFS: 큐
- 방문 순서
 - BFS: 거리가 가까운 정점부터 방문
 - DFS: 좋은 성질을 갖고 있지만 어려워서 지금은 생략
- 시간 복잡도
 - 인접 행렬 사용하면 $O(|V|^2)$
 - 인접 리스트 / 간선 리스트 사용하면 $O(|V| + |E|)$
- 두 알고리즘 모두 연결되어 있는 정점들만 방문함
 - 연결 그래프가 아니면 컴포넌트마다 따로 탐색해야 함
 - BOJ 11724. 연결 요소의 개수

그래프의 탐색

BOJ 2178 미로 탐색

- $N * M$ 크기의 격자에 장애물이 있음
- $(1, 1)$ 에서 (N, M) 으로 이동할 때 지나야 하는 칸의 최소 개수를 구하는 문제
- 이동할 수 있는 칸을 정점으로 생각하고 인접한 칸들을 간선으로 연결하면
- 한 점에서 다른 점으로 가는 최단 거리 + 1을 구하는 문제
- BFS는 최단 거리를 구하는 알고리즘이므로 BFS를 사용하면 문제를 해결할 수 있음
- 인접한 4방향 탐색
 - $(i+1, j)$ $(i-1, j)$ $(i, j+1)$ $(i, j-1)$
 - `int di[4] = {1, -1, 0, 0}, dj[4] = {0, 0, 1, -1};` 선언하고
 - 반복문 돌면서 $i + di[k], j + dj[k]$ 를 보는 방식으로 구현

그래프의 탐색

```

#include <bits/stdc++.h>
using namespace std;
constexpr int di[] = {1, -1, 0, 0};
constexpr int dj[] = {0, 0, 1, -1};

int N, M, A[111][111], D[111][111];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=N; i++){
        for(int j=1; j<=M; j++){
            char c; cin >> c;
            A[i][j] = c - '0';
        }
    }
    for(int i=1; i<=N; i++) for(int j=1; j<=M; j++) D[i][j] = -1;

    queue<pair<int,int>> Q;
    Q.push({1, 1}); D[1][1] = 1; // Q.emplace(1, 1);
    while(!Q.empty()){
        // auto [i,j] = Q.front(); Q.pop();
        int i = Q.front().first;
        int j = Q.front().second;
        Q.pop();
        for(int k=0; k<4; k++){
            int r = i + di[k], c = j + dj[k];
            if(r < 1 || c < 1 || r > N || c > M) continue;
            if(A[r][c] == 1 && D[r][c] == -1) D[r][c] = D[i][j] + 1, Q.push({r, c});
        }
    }
    cout << D[N][M];
}
```

위상 정렬

위상 정렬을 구하는 방법

- BFS를 응용하는 방법
- DFS를 응용하는 방법

위상 정렬의 성질

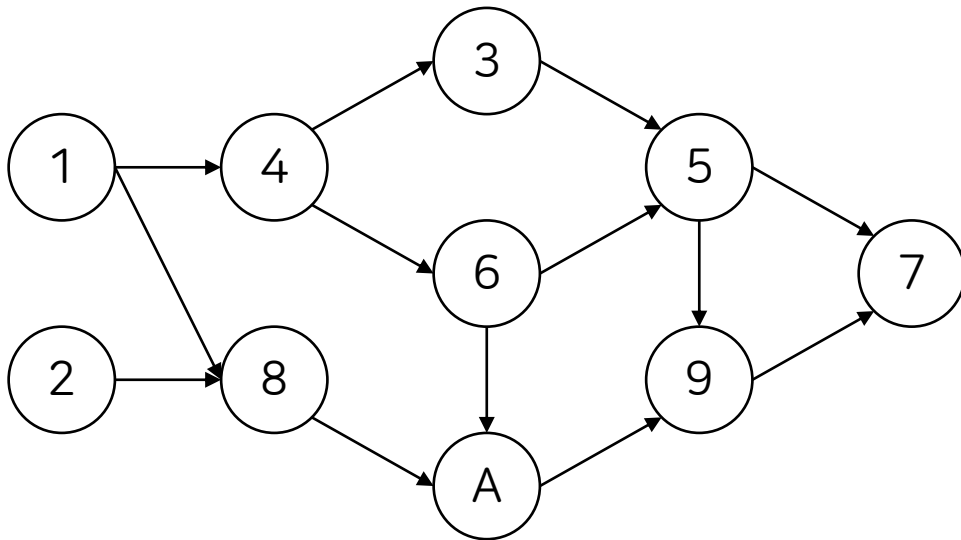
- 가장 앞에 오는 정점은 항상 in-degree가 0임
 - 만약 0이 아니라면 그 정점보다 다른 정점이 먼저 나와야 함
- in-degree가 0이면서 맨 앞이 아닌 정점은 앞으로 옮겨도 됨

위상 정렬

BFS와 유사한 방식

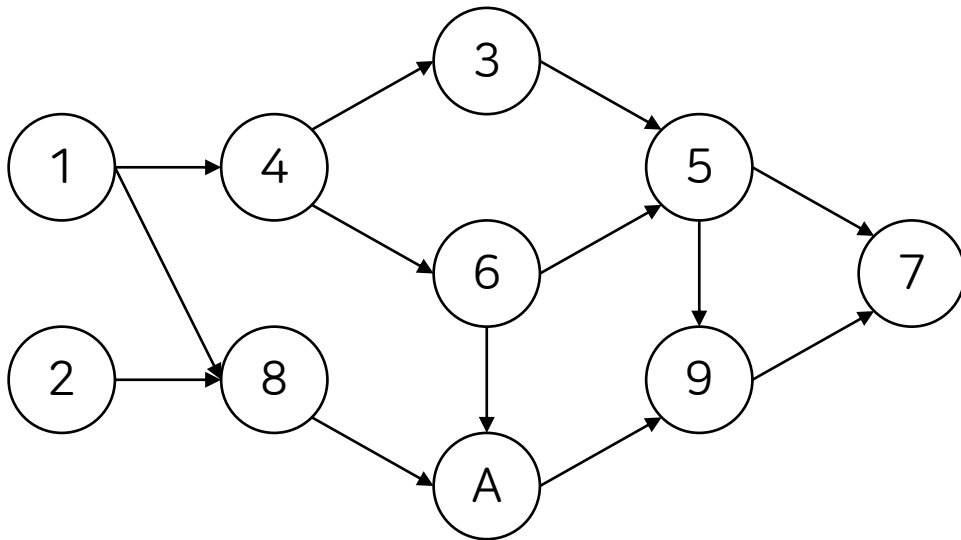
- in-degree가 0인 정점을 큐에 삽입
- 큐가 비어 있지 않으면 아래 과정을 반복
 - 큐에서 정점 v 를 제거
 - v 를 정답에 추가
 - v 에서 나가는 간선을 모두 제거, 해당 정점의 in-degree 1 감소
 - in-degree가 0이 된 정점을 큐에 삽입
- 사이클이 존재하면 모든 정점을 방문하기 전에 종료됨

위상 정렬



- $Q = \{1, 2\}$
- $V = 1$ $Q = \{2, 4\}$
- $V = 2$ $Q = \{4, 8\}$
- $V = 4$ $Q = \{8, 3, 6\}$
- $V = 8$ $Q = \{3, 6\}$
- $V = 3$ $Q = \{6\}$
- $V = 6$ $Q = \{A, 5\}$
- $V = A$ $Q = \{5\}$
- $V = 5$ $Q = \{9\}$
- $V = 9$ $Q = \{7\}$
- $V = 7$ $Q = \{\}$
- 1 2 4 8 3 6 A 5 9 7

위상 정렬



- $Q = \{2, 1\}$
- $V = 2$ $Q = \{1\}$
- $V = 1$ $Q = \{8, 4\}$
- $V = 8$ $Q = \{4\}$
- $V = 4$ $Q = \{6, 3\}$
- $V = 6$ $Q = \{3, A\}$
- $V = 3$ $Q = \{A, 5\}$
- $V = A$ $Q = \{5, 9\}$
- $V = 5$ $Q = \{9\}$
- $V = 9$ $Q = \{7\}$
- $V = 7$ $Q = \{\}$
- 2 1 8 4 6 3 A 5 9 7

위상 정렬

BFS와 유사한 방식

- 시간 복잡도: $O(|V| + |E|)$
- 큐에 있는 원소들의 출력 순서는 상관 없음
 - 큐에 들어간 순간 언제든지 위상 정렬 결과에 추가할 수 있음
 - 편의상 앞에 있는 원소부터 사용하는 것
 - BOJ 1766 문제집: 큐에서 가장 작은 원소부터 꺼내야 함

```
#include <bits/stdc++.h>
using namespace std;

int N, M, In[32323];
vector<int> G[32323];

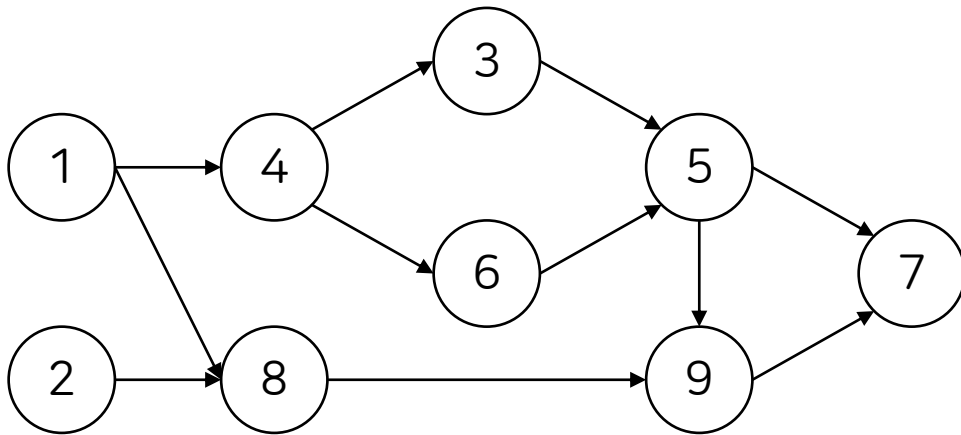
int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=M; i++){
        int s, e; cin >> s >> e;
        G[s].push_back(e);
        In[e] += 1;
    }
    queue<int> Q;
    for(int i=1; i<=N; i++) if(!In[i]) Q.push(i);
    while(!Q.empty()){
        int v = Q.front(); Q.pop();
        cout << v << " ";
        for(auto i : G[v]) if(--In[i]) Q.push(i);
    }
}
```

위상 정렬

깊이 우선 탐색

- DFS를 하면서, 정점을 빠져나오는 순서대로 기록
- 그 순서의 역순은 위상 정렬임
- 간선 $A \rightarrow B$ 가 있을 때
- A를 B보다 먼저 방문하면 항상 B를 먼저 탈출함
 - B를 탈출한 다음에 A를 탈출해야 함
- B를 A보다 먼저 방문하면 항상 B를 먼저 탈출함
 - B를 탈출한 다음에 A를 방문할 수 있음
 - B를 탈출하기 전에 A를 방문하면 사이클

위상 정렬

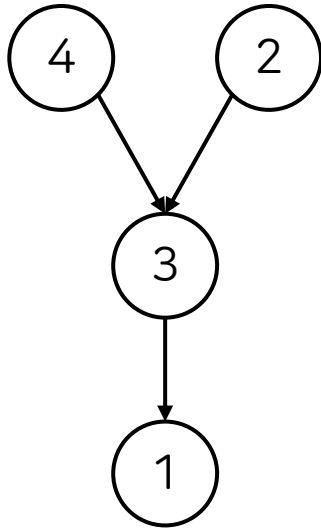
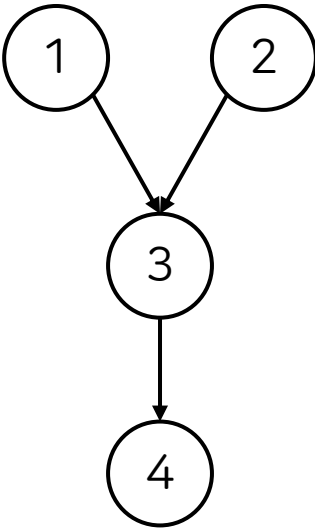


- 1 진입
 - 4 진입
 - 6 진입
 - 5 진입
 - 7 진입
 - 7 **탈출**(5)
 - 9 진입
 - 9 **탈출**(5)
 - 5 **탈출**(6)
 - 7 9 5 6 3 4 8 1 2
 - 2 1 8 4 3 6 5 9 7
- 6 **탈출**(4)
 - 3 진입
 - 3 **탈출**(4)
 - 4 **탈출**(1)
 - 8 진입
 - 8 **탈출**(1)
 - 1 **탈출**
 - 2 진입
 - 2 **탈출**

위상 정렬

깊이 우선 탐색

- 시간 복잡도: $O(|V| + |E|)$
- 방문하지 않은 모든 정점에 대해 DFS를 호출해야 함
 - 아래 그래프에서 DFS(1)만 호출하면 모든 정점을 볼 수 없음



```
#include <bits/stdc++.h>
using namespace std;

int N, M, C[32323];
vector<int> G[32323], V;

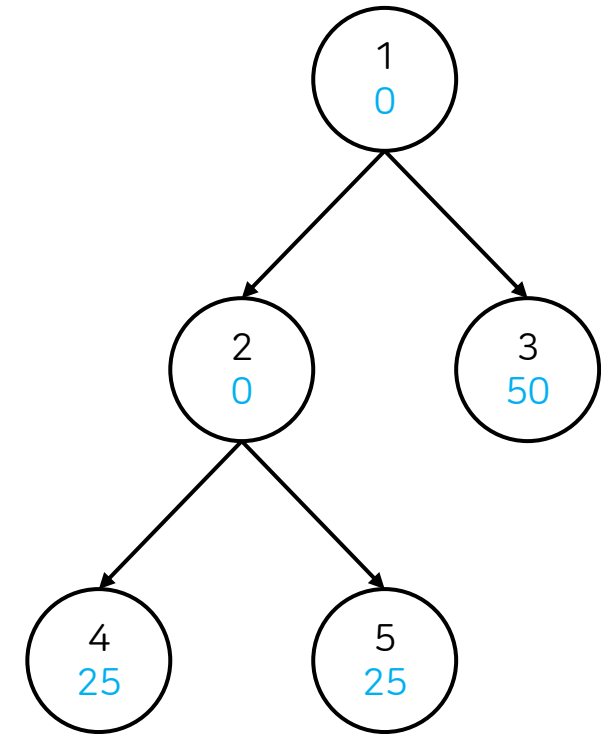
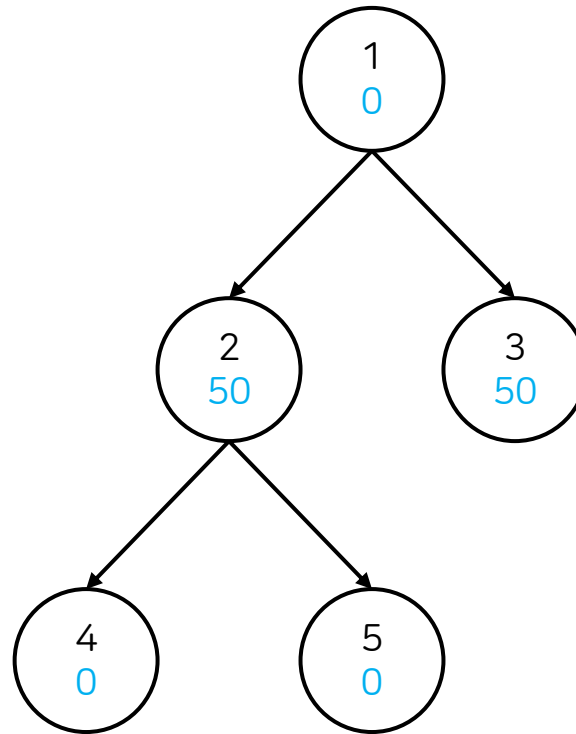
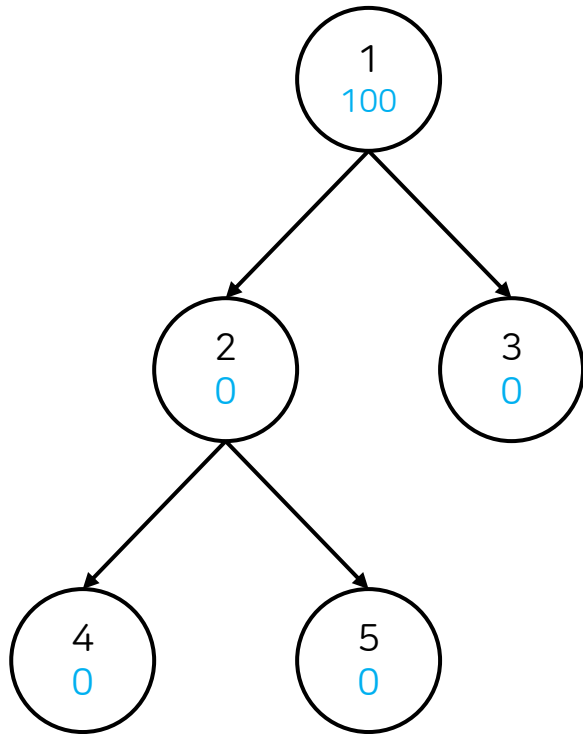
void DFS(int v){
    C[v] = 1;
    for(auto i : G[v]) if(!C[i]) DFS(i);
    V.push_back(v);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=M; i++){
        int s, e; cin >> s >> e;
        G[s].push_back(e);
    }
    for(int i=1; i<=N; i++) if(!C[i]) DFS(i);
    reverse(V.begin(), V.end());
    for(auto i : V) cout << i << " ";
}
```

위상 정렬

BOJ 28360 양동이 게임

- 양동이 DAG 형태로 연결되어 있음 (모든 간선은 번호가 작은 정점에서 큰 정점으로 가는 방향)
- 양동이의 out-degree가 0 초과면 양동이에 담긴 물이 연결되어 있는 양동으로 균등하게 이동함
- 양동이에 담긴 물의 양의 최댓값을 구하는 문제



위상 정렬

BOJ 28360 양동이 게임

- 양동이 DAG 형태로 연결되어 있음 (모든 간선은 번호가 작은 정점에서 큰 정점으로 가는 방향)
- 양동이의 out-degree가 0 초과면 양동이에 담긴 물이 연결되어 있는 양동으로 균등하게 이동함
- 양동이에 담긴 물의 양의 최댓값을 구하는 문제
- 번호가 작은 정점에서 큰 정점으로 가는 간선만 존재하므로 그래프에 사이클 없음
- 정점 번호 자체가 위상 정렬 순서임
- 번호가 큰 정점에서 작은 정점으로 물이 흘러가지 않으므로
- 번호가 작은 정점부터 차례대로 물을 흘려보내면 됨
- out-degree가 0일 때 0으로 나누지 않도록 조심

위상 정렬

BOJ 28360 양동이 게임



```
#include <bits/stdc++.h>
using namespace std;

int N, M;
double D[55];
vector<int> G[55];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M; D[1] = 100;
    for(int i=1,s,e; i<=M; i++) cin >> s >> e, G[s].push_back(e);
    for(int i=1; i<=N; i++){
        if(G[i].empty()) continue;
        double nxt = D[i] / G[i].size();
        for(auto j : G[i]) D[j] += nxt;
        D[i] = 0;
    }
    cout << fixed << setprecision(20) << *max_element(D+1, D+N+1);
}
```

위상 정렬

BOJ 28360 양동이 게임

- 작은 정점 $\rightarrow$ 큰 정점 조건이 없으면 직접 위상 정렬 구해야 함
 - 처음에 Q.push(1); 만 하면 안 됨
 - 1이 아닌 in-degree = 0 정점 존재할 수 있음



```
#include <bits/stdc++.h>
using namespace std;

int N, M, In[55];
double D[55];
vector<int> G[55];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M; D[1] = 100;
    for(int i=1,s,e; i<=M; i++) cin >> s >> e, G[s].push_back(e), In[e]++;

    queue<int> Q;
    for(int i=1; i<=N; i++) if(!In[i]) Q.push(i);
    while(!Q.empty()){
        int v = Q.front(); Q.pop();
        for(auto i : G[v]){
            D[i] += D[v] / G[v].size();
            if(--In[i]) Q.push(i);
        }
        if(!G[v].empty()) D[v] = 0;
    }
    cout << fixed << setprecision(20) << *max_element(D+1, D+N+1);
}
```