자료구조 (2)

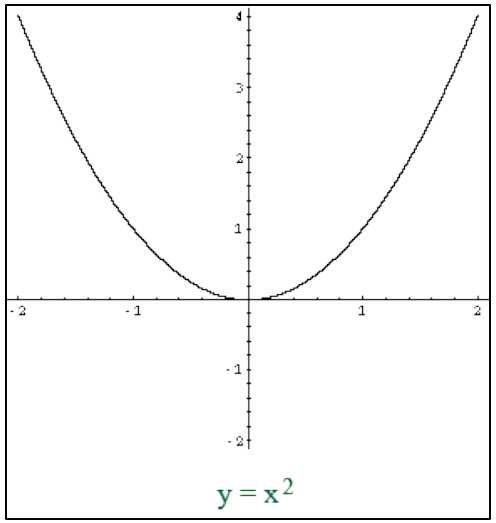
25.05.12 (월) 오후 5시 ~ 7시

목차

그래프

그래프 | 01 기본 용어 소개

- 수학에서 말하는 그래프는 보통 이런건데, 자료구조에서 그래프는 이것과 많이 다르다.



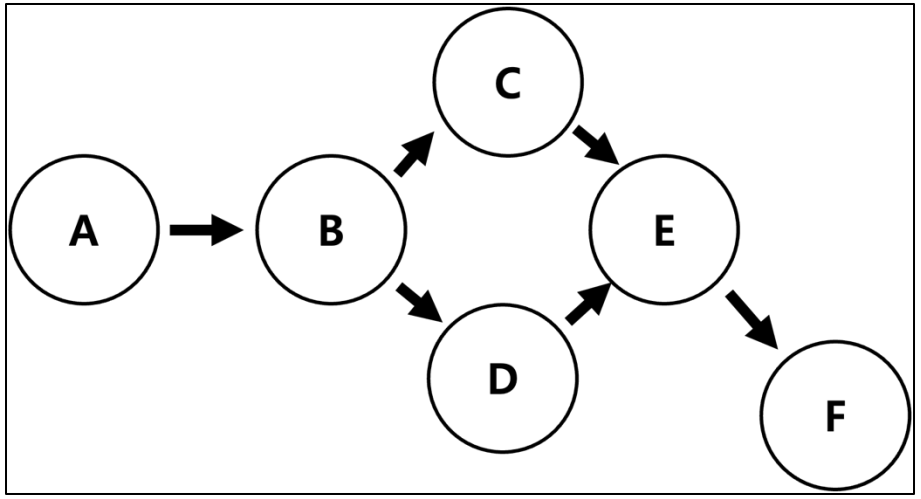
- 자료구조에서 **그래프는 정점(Vertex)와 간선(Edge)의 쌍으로 구성된 집합**이다. 수학적으로는 G = (V, E) 과 같이 정의되며, 그래프 G는 정점의 집합 V와 간선의 집합 E로 이루어진 순서쌍이라는 의미다.
  - **정점**은 그래프의 노드를 의미한다.

    예) 어떤 도시들, 사람들, 서버
  - **간선**은 정점들을 잇는 선을 의미한다.

    예) 도시간 도로, 사람 간 친구 관계, 컴퓨터 간 연결

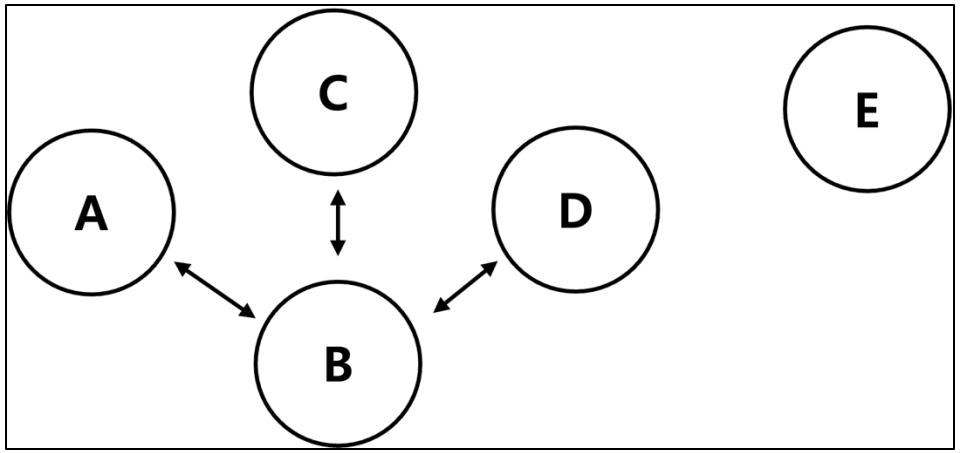
그래프 | 01 기본 용어 소개

- 그래프 G = (V, E)에 대해 아래와 같은 정보가 주어졌다면, 다음과 같은 그래프를 그릴 수 있다.
  - V = {A, B, C, D, E, F}
  - E = {{A, B}, {B, C}, {B, D}, {C, E}, {D, E}, {E, F}}



그래프 | 01 기본 용어 소개

- 그래프는 간선에 방향이 존재하는 **방향 그래프**(Drected Graph), 방향이 존재하지 않는 **무방향 그래프**(Undirected Graph)로 나눌 수 있다.
  - 이전 슬라이드의 그래프는 방향 그래프이다.
- 무방향 그래프 G = (V, E)에 대해 아래와 같은 정보가 주어졌다면, 다음과 같은 그래프를 그릴 수 있다.
  - V = {A, B, C, D, E}
  - E = {{A, B}, {B, C}, {B, D}}

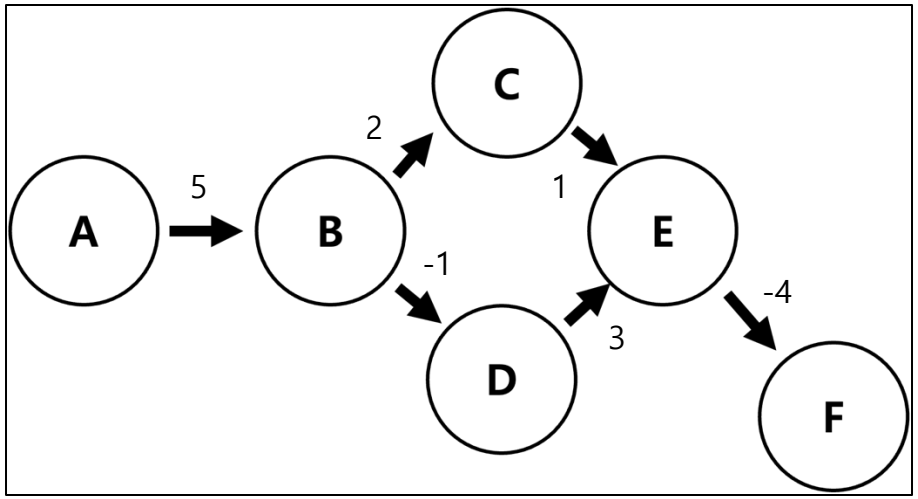


그래프 | 01 기본 용어 소개

- **경로**
  - 정점 v1에서 시작해서, v1과 연결된 간선 e1을 따라 v2로, v2와 연결된 간선 e2를 따라 v3로, ..., vk까지 도달하기 까지의 정점들의 나열을 경로라고 한다.
- **사이클**
  - 시작한 정점으로 되돌아오는 경로를 의미한다.
- **차수**
  - 정점에 연결되어 있는 간선의 개수를 말한다.
  - 방향이 있는 그래프에서, 한 정점의 차수는 In-degree + Out-degree 로 정의된다.
  - **진입 차수(In-degree)** : 정점 v로 들어오는 간선의 개수
  - **진출 차수(Out-degree)** : 정점 v에서 나가는 간선의 개수

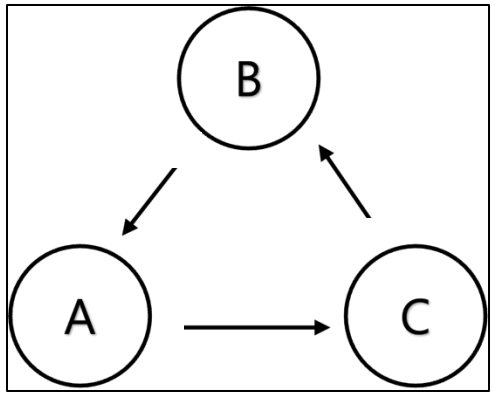
그래프 | 01 기본 용어 소개

- **Directed acyclic graph (DAG)**
  - 사이클이 없는(a – cyclic) 방향 그래프(directed graph)를 의미한다. 아래 그래프는 DAG이다.
  - DAG, In-degree는 이후에 스터디할 위상정렬에 나오는 개념이므로 꼭 알아둬야 한다.
- **Weighted Graph**
  - 그래프의 각 간선에 가중치(weight)가 부여된 그래프를 말한다.
  - 문제마다 다르지만 보통 '비용', '거리', '시간' 등을 나타낸다.

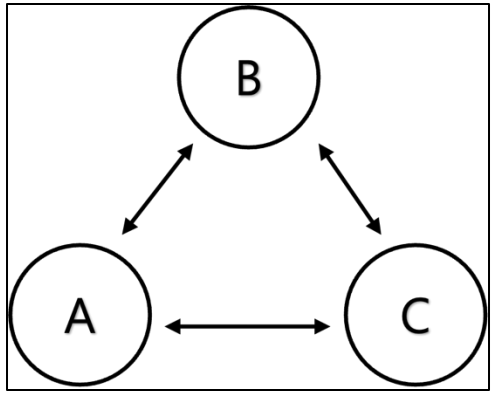


그래프 | 01 기본 용어 소개

- **Connected graph (연결 그래프)**
  - 모든 정점 쌍 (u, v)에 대해, u에서 v로 가는 경로가 존재하는 그래프
  - 오른쪽 그래프는 연결 그래프이다.



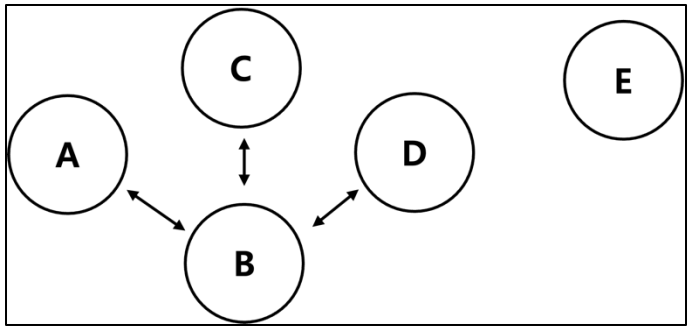
- **Complete graph (완전 그래프)**
  - 모든 정점 쌍 사이에 직접 연결된 간선이 존재하는 그래프
  - 오른쪽 그래프는 완전 그래프이다.



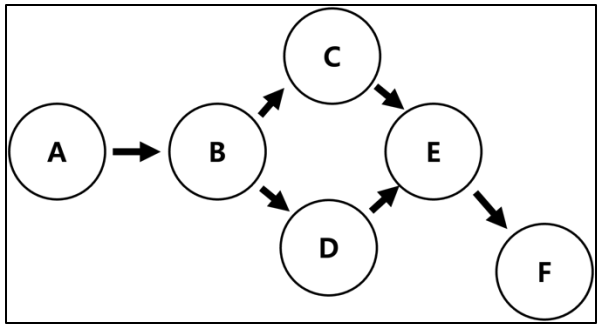
- **Spare graph (희소 그래프)**
  - 간선의 수가 정점의 수에 비해 상대적으로 적은 그래프를 말한다.
  - 상대적으로 중요도가 떨어지는 용어지만, 가끔씩 나온다.

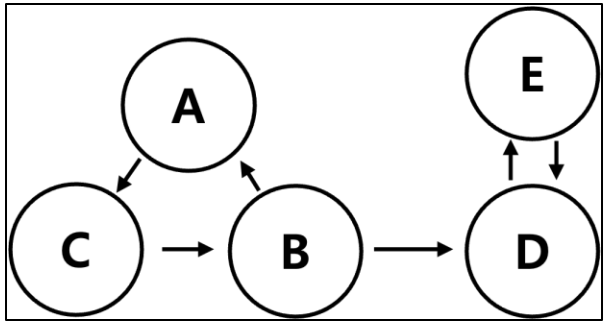
그래프 | 01 기본 용어 소개

- **Component(연결 요소)** *백준에서 플래티넘 5 이상 문제에서 나오는 용어이지만, 간단히 소개하겠다.
  - 서로 연결되어 있는 덩어리(묶음)를 의미한다.
  - 무방향 그래프에서의 컴포넌트(연결 요소)는 '간선 따라 전부 연결되어 있으면 한 묶음'이라고 생각하면 된다. 아래 그래프는 2개의 컴포넌트를 가진다.



  - 방향 그래프에서의 컴포넌트는 약한 연결 요소(WCC, Weakly Connected Component), 강한 연결 요소(Strongly Connected Component)로 나뉜다.
  - 약한 연결 요소는 '방향 무시하고 연결되면 한 묶음'이라고 생각하면 된다. 하단 왼쪽 그래프의 WCC는 1개, 하단 오른쪽 그래프의 SCC는 1개이다.
  - 강한 연결 요소는 '정점 쌍 간에 서로 왕복 가능해야 한 묶음'이라고 생각하면 된다. 하단 왼쪽 그래프의 SCC는 0개, 하단 오른쪽 그래프의 SCC는 2개이다.





그래프 | 02 그래프의 표현

- **그래프를 코드로 어떻게 표현할 수 있을까?**
- **보통 PS에서는 그래프를 두 가지 방식(인접 행렬, 인접 리스트)으로 표현한다.**
- **인접 행렬**
  - 정점이 n개일 때, n x n 크기의 2차원 배열을 사용하여
    - adj[i][j] = 1 이면, i → j 간선이 존재
    - adj[i][j] = 0 이면, i → j 간선이 없음.
  - 장점
    - 간선 존재 여부를 O(1)에 확인 가능
  - 단점
    - 메모리 사용량 많음, 희소 그래프에는 비효율적
  - 코드 예시

```cpp
const int N = 1000;
int adj[N][N];

void add_edge(int u, int v) {
    adj[u][v] = 1;
    adj[v][u] = 1; // 무방향 그래프일 경우
}
```

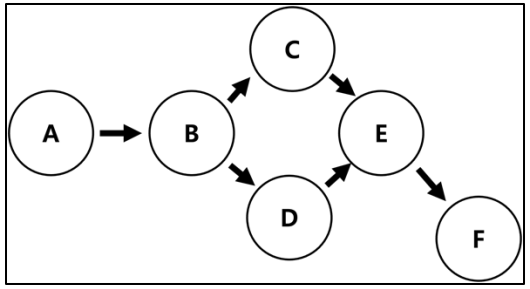
그래프 | 02 그래프의 표현

- **인접 리스트**
  - 각 정점마다 연결된 정점들의 리스트(vector)를 저장
    - graph[3] = {1, 4, 6} : 3번 정점에서 1번, 4번 정점으로 향하는 간선이 존재함.
  - 장점
    - 공간복잡도가 O(V + E)로, 메모리 효율적
    - 정점의 연결을 순회하기 쉽다.
  - 단점
    - 두 정점 간의 간선 존재 여부를 확인할 때 O(degree) 시간이 걸린다.
  - 코드 예시

```
1   const int N = 1000;
2   vector<int> graph[N];
3
4   void add_edge(int u, int v) {
5       graph[u].push_back(v);
6       graph[v].push_back(u); // 무방향 그래프일 경우
7   }
```

그래프 | 03 그래프의 탐색

- 그래프 탐색이란, 간선을 따라 그래프에 있는 정점들을 방문하는 것을 말한다. 그래프 탐색 방법 중 DFS, BFS가 있다.
- **깊이 우선 탐색 (DFS, Depth-Frist Search)**
  - 한 방향으로 쭉 들어갔다가, 더 이상 갈 곳이 없으면 되돌아와서 다른 방향을 탐색하는 방법이다.
  - 재귀 또는 스택으로 구현할 수 있다.
  - 아래 그래프를 DFS로 탐색하면, 방문 순서는 {A, B, C, E, F, D} 또는 {A, B, D, E, F, C}이다.

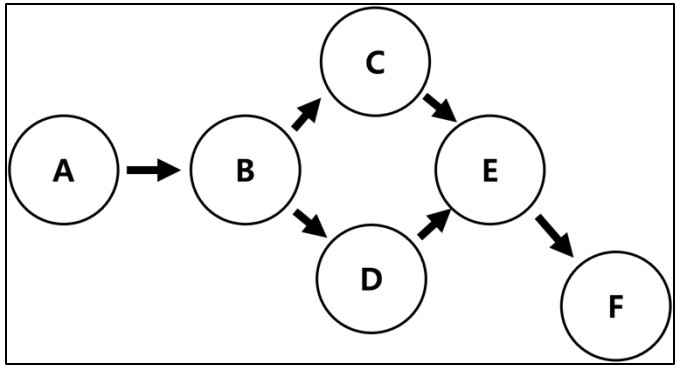


  - DFS 코드 예시

```
1   bool visited[N];
2   vector<int> graph[N];
3
4   void dfs(int v) {
5       visited[v] = true;
6       for (int u : graph[v]) {
7           if (!visited[u]) dfs(u);
8       }
9   }
```

그래프 | 03 그래프의 탐색

- **너비 우선 탐색 (BFS, Breadth-First Search)**
  - 현재 정점에서 가까운 노드부터 차례대로 방문하는 탐색 방식이다.
  - 큐를 사용하여 구현할 수 있다.
  - 아래 그래프를 BFS로 탐색하면, 방문 순서는 {A, B, C, D, E, F} 또는 {A, B, D, C, E ,F}이다.



- BFS 코드 예시 ---------- >>

```
1   bool visited[N];
2   vector<int> graph[N];
3
4   void bfs(int start) {
5       queue<int> q;
6       q.push(start);
7       visited[start] = true;
8
9       while (!q.empty()) {
10          int v = q.front(); q.pop();
11          for (int u : graph[v]) {
12              if (!visited[u]) {
13                  visited[u] = true;
14                  q.push(u);
15              }
16          }
17      }
18  }
```

그래프 | 04 예제 (1) : 백준 1260번 DFS와 BFS

- **문제 요약**
  - 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.

그래프 | 04 예제 (1) : 백준 1260번 DFS와 BFS

- **구현**

Line 4 ~ 6

graph[i]는 정점 i에 연결된 정점들의 리스트입니다.

visited[i]는 i번 정점이 방문되었는지를 나타냅니다. *다음 슬라이드 먼저 보기

Line 8 ~ 22 BFS

9, 11 : 큐를 선언하고, 시작 정점 v를 큐에 넣고 방문 표시를 한다.

15 : 현재 방문 중인 정점을 출력한다.

13 ~ 20 : 큐가 빌 때까지, 큐 앞에 있는 정점에 연결된 모든 정점을 큐에 넣고 방문 처리한다.

Line 24 ~ 32 DFS

25 : 현재 노드를 방문했다면, 종료한다.

26 : 현재 방문 중인 정점을 출력한다.

27 ~ 30 : 현재 방문 중인 정점에 연결된 정점 중 방문하지 않은 정점이 있다면 방문한다.

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

vector<vector<int>> graph;
int N, M, V;
bool visited[1010];

void bfs(){
  queue<int> q;

  q.push(V); visited[V] = true;

  while(!q.empty()){
    int cur = q.front(); q.pop();
    cout << cur << " ";
    for(auto x : graph[cur]){
      if(visited[x]) continue;
      q.push(x); visited[x] = true;
    }
  }
  return;
}

void dfs(int cur){
  if(visited[cur]) return;
  cout << cur << " "; visited[cur] = true;
  for(auto x : graph[cur]){
    if(visited[x]) continue;
    dfs(x);
  }
  return;
}
```

그래프 | 04 예제 (1) : 백준 1260번 DFS와 BFS

- **구현**

Line 39 ~ 45

인접 리스트 형태로 그래프를 입력받는다.

무방향 그래프이므로, u와 연결된 v / v와 연결된 u를 모두 기록해준다.

Line 47 ~ 49

문제에서 정점 번호가 작은 순서대로 방문하라고 했으므로, 해당 작업을 수행한다.

Line 51 ~ 52

방문배열을 초기화하고, V를 시작 정점으로 깊이우선탐색을 수행한다.

Line 54 ~ 55

방문 배열을 초기화하고, V를 시작 정점으로 너비우선탐색을 수행한다.

```
35  int main(){
36    ios::sync_with_stdio(false);
37    cin.tie(0); cout.tie(0);
38
39    cin >> N >> M >> V;
40    graph.resize(N+1);
41    while(M--){
42      int u, v; cin >> u >> v;
43      graph[u].push_back(v);
44      graph[v].push_back(u);
45    }
46
47    for(int i = 1; i <= N; i++){
48      sort(graph[i].begin(), graph[i].end());
49    }
50
51    fill(visited, visited + 1010, false);
52    dfs(V); cout << "\n";
53
54    fill(visited, visited + 1010, false);
55    bfs(); cout << "\n";
56
57    return 0;
58  }
```

그래프 | 04 예제 (2) : 백준 2606번 바이러스

- **문제 요약**
  - 한 컴퓨터가 웜 바이러스에 걸리면 그 컴퓨터와 네트워크 상에서 연결되어 있는 모든 컴퓨터는 웜 바이러스에 걸리게 된다.
  - 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.
- **입력**
  - 컴퓨터의 수, 네트워크 상에서 직접 연결되어 있는 컴퓨터 쌍의 수가 주어진다.
  - 그 쌍의 수만큼 한 줄에 한 쌍씩 네트워크 상에서 직접 연결되어 있는 컴퓨터의 번호 쌍이 주어진다.
- **출력**
  - 1번 컴퓨터를 통해 웜 바이러스에 걸리게 되는 컴퓨터의 수를 출력하는 프로그램을 작성하시오.

그래프 | 04 예제 (2) : 백준 2606번 바이러스

- **문제 풀이**

  1. 주어진 입력을 그래프로 표현하기
  2. 1번 컴퓨터와 연결된 모든 정점을 방문하기
  3. 1번 컴퓨터와 연결된 모든 정점의 개수 출력하기

그래프 | 04 예제 (2) : 백준 2606번 바이러스

- **구현**

Line 5 ~ 8

그래프, 방문 배열을 정의한다.

Line 10 ~ 18

정점 v에 대해 깊이우선탐색을 진행한다.

정점 v를 방문처리하고, v와 연결된 정점 u에 대해 방문하지 않은 정점이라면,

감염되었다는 처리를 하고 정점 u에 대한 깊이우선탐색을 진행한다.

Line 20 ~ 34

정점과 간선의 개수를 입력을 받는다.

간선에 연결된 정점 2개의 정보를 받고, 무방향 그래프이므로 u, v를 서로 연결해준다.

1번 정점과 연결된 정점의 수를 구하는게 문제이므로, 1번 정점에 대해 DFS를 수행한다.

```cpp
#include <iostream>
#include <vector>
using namespace std;

const int MAX = 101;
vector<int> graph[MAX];
bool visited[MAX];
int infected = 0;

void dfs(int v) {
    visited[v] = true;
    for (int u : graph[v]){
        if (!visited[u]) {
            infected++;
            dfs(u);
        }
    }
}

int main() {
    int n, m;
    cin >> n >> m;

    while (m--) {
        int u, v;
        cin >> u >> v;
        graph[u].push_back(v);
        graph[v].push_back(u);
    }

    dfs(1);
    cout << infected << '\n';
    return 0;
}
```

그래프 | 04 예제 (3) : 백준 11724번 연결 요소의 개수

- **문제 요약**
  - 방향 없는 그래프가 주어졌을 때, 연결 요소(Connected Component)의 개수를 구하는 프로그램을 작성하시오.
- **입력**
  - 첫째 줄에 정점의 개수 N과 간선의 개수 M이 주어진다. ($1 \le N \le 1,000$, $0 \le M \le N \times (N-1)/2$) 둘째 줄부터 M개의 줄에 간선의 양 끝점 u와 v가 주어진다. ($1 \le u, v \le N$, $u \ne v$) 같은 간선은 한 번만 주어진다.
- **출력**
  - 첫째 줄에 연결 요소의 개수를 출력한다.

그래프 | 04 예제 (3) : 백준 11724번 연결 요소의 개수

- **문제 풀이**

  다음과 같은 과정을 거치면 문제를 풀 수 있다.

  1. 주어진 입력으로 그래프를 만들기.

     연결 리스트로 만들지, 인접 행렬로 만들지는 본인의 선택이다.

     N이 최대 1,000이라서 인접 행렬로 만들어도 괜찮아 보이지만, 희소 그래프일 경우를 대비해 연결 리스트가 나아보인다.

  2. 그래프를 순회하기

     임의의 정점이 방문하지 않은 정점이라면, 연결 요소의 개수를 1 증가 시키고 BFS 또는 DFS를 수행한다.

     DFS, BFS 과정에서 그 정점과 연결된 모든 정점을 방문하게 된다.

그래프 | 04 예제 (3) : 백준 11724번 연결 요소의 개수

- **구현**

  Line 5. adj[i]는 정점 i와 연결된 정점 목록을 저장하는 그래프이다.

  Line 6. visited[i] : 정점 i를 방문했는지 여부를 저장한다. true면 방문했다는 의미다.

  Line 8 ~ 12

  DFS 방식으로 정점 v에서 시작해서 연결된 모든 정점을 방문한다.

  한 번의 DFS는 하나의 연결 요소를 전부 탐색하게 된다.

```
1   #include <iostream>
2   #include <vector>
3   using namespace std;
4
5   vector<int> adj[1001];
6   bool visited[1001];
7
8   void dfs(int v) {
9       visited[v] = true;
10      for (int u : adj[v])
11          if (!visited[u]) dfs(u);
12  }
```

그래프 | 04 예제 (3) : 백준 11724번 연결 요소의 개수

- **구현**

  Line 15. 정점의 개수 n, 간선의 개수 m을 입력받는다. 보통 v, e 변수로 받는다.

  Line 16. 간선 정보를 입력받는다. 무방향 그래프이므로 adj[u], adj[v] 모두에 서로를 추가한다.

  Line 22. 연결 요소의 개수를 세는 변수이다.

  Line 23 ~ 28

  방문하지 않은 정점을 시작점으로 DFS를 수행한다.

  이 DFS는 그 정점이 포함된 하나의 연결 컴포넌트 전체를 탐색한다.

  즉, 'DFS 호출 횟수 = 연결 요소 개수'이다.

```
14  int main() {
15      int n, m; cin >> n >> m;
16      while (m--) {
17          int u, v; cin >> u >> v;
18          adj[u].push_back(v);
19          adj[v].push_back(u);
20      }
21
22      int count = 0;
23      for (int i = 1; i <= n; ++i) {
24          if (!visited[i]) {
25              dfs(i);
26              count++;
27          }
28      }
29
30      cout << count;
31      return 0;
32  }
```

그래프 | 04 예제 (4) : 백준 2178번 미로 탐색

- **문제 요약**

N×M크기의 배열로 표현되는 미로가 있다.

1	0	1	1	1	1
1	0	1	0	1	0
1	0	1	0	1	1
1	1	1	0	1	1

미로에서 1은 이동할 수 있는 칸을 나타내고, 0은 이동할 수 없는 칸을 나타낸다. 이러한 미로가 주어졌을 때, (1, 1)에서 출발하여 (N, M)의 위치로 이동할 때 지나야 하는 최소의 칸 수를 구하는 프로그램을 작성하시오. 한 칸에서 다른 칸으로 이동할 때, 서로 인접한 칸으로만 이동할 수 있다.

위의 예에서는 15칸을 지나야 (N, M)의 위치로 이동할 수 있다. 칸을 셀 때에는 시작 위치와 도착 위치도 포함한다.

- **입력**
  - 첫째 줄에 두 정수 N, M($2 \le N, M \le 100$)이 주어진다. 다음 N개의 줄에는 M개의 정수로 미로가 주어진다. 각각의 수들은 붙어서 입력으로 주어진다.
- **출력**
  - 첫째 줄에 지나야 하는 최소의 칸 수를 출력한다. 항상 도착위치로 이동할 수 있는 경우만 입력으로 주어진다.

그래프 | 04 예제 (4) : 백준 2178번 미로 탐색

- **문제 풀이**
  - 이와 같은 문제를 **Flood Fill 알고리즘**이라고 하는데, 워낙 자주 나오는 컨셉이니 외우는게 좋고 PS를 하다보면 저절로 외워질 것이다.
  - Flood Fill은 어떤 기준 조건을 만족하는 인접한 모든 칸을 방문하는 알고리즘으로, DFS나 BFS를 통해 구현할 수 있다.

    물감을 종이에 뿌렸을 때, 주위로 번지는걸 생각하면 Flood Fill이라 불리는지 이해할 수 있다.
  - Flood Fill은 다음과 같이 구현된다.

```cpp
const int dx[4] = {1, -1, 0, 0}; // 하, 상, 우, 좌
const int dy[4] = {0, 0, 1, -1};

int board[100][100];
bool visited[100][100];
int n, m;

void dfs(int x, int y, int targetColor) {
    visited[x][y] = true;

    for (int dir = 0; dir < 4; dir++) {
        int nx = x + dx[dir];
        int ny = y + dy[dir];

        if (nx < 0 || ny < 0 || nx >= n || ny >= m) continue;
        if (visited[nx][ny]) continue;
        if (board[nx][ny] != targetColor) continue;

        dfs(nx, ny, targetColor);
    }
}
```

  - 이를 그대로 사용하면 위 문제를 풀 수 있다.

그래프 | 04 예제 (4) : 백준 2178번 미로 탐색

- **구현**

Line 6 ~ 10

6 : 미로의 세로(n, 행), 가로(m, 열) 크기를 저장하는 변수다.

7 : 미로를 입력 받을 배열(maze)와 거리를 저장할 배열(dist)을 선언한다.

8 : visited[i][j]가 참이면, 칸 (i, j)을 방문했다는 의미이다.

9, 10 : 변위를 나타낸다. 아래, 위, 오른쪽, 왼쪽을 의미한다.

Line 12 ~ 19

문제에서 주어진 입력을 받는다.

```
1   #include <iostream>
2   #include <queue>
3   #include <string>
4   using namespace std;
5
6   int n, m;
7   int maze[100][100], dist[100][100];
8   bool visited[100][100];
9   int dx[4] = {1, -1, 0, 0};
10  int dy[4] = {0, 0, 1, -1};
11
12  int main() {
13      cin >> n >> m;
14      for (int i = 0; i < n; i++) {
15          string s; cin >> s;
16          for (int j = 0; j < m; j++) {
17              maze[i][j] = s[j] - '0';
18          }
19      }
20
```

그래프 | 04 예제 (4) : 백준 2178번 미로 탐색

- **구현**

Line 21 ~ 24

21 : 미로의 좌표를 입력받을 수 있는 큐를 선언한다.

22 ~ 23 : (0, 0)를 큐에 넣고, 방문처리를 한다.

24 : (0, 0)은 이미 한 칸을 지난 상태이므로 1을 저장한다.

Line 26 ~ 39

BFS를 수행한다.

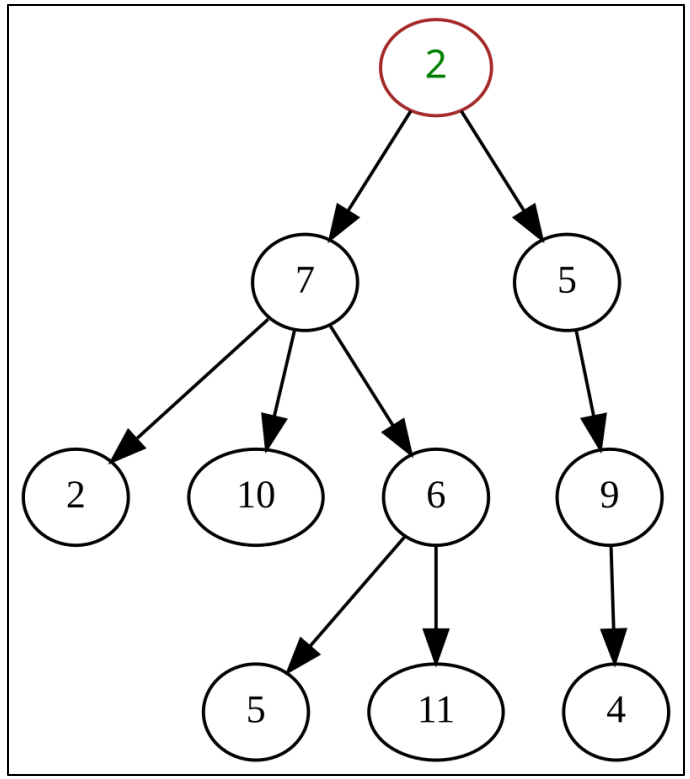
현재 큐 앞에 있는 좌표에 대해 다음을 수행한다.

- 오른쪽, 왼쪽, 위, 아래를 방문하려는 '시도'를 한다.
- 유효한 좌표이면서 방문하지 않은 좌표라면 방문한다.
- 방문처리를 하고 새로 방문하는 정점은 이전 정점보다 거리가 1 증가한다.
- 큐에 새로 방문하는 정점을 넣는다

```
21      queue<pair<int, int>> q;
22      q.push({0, 0});
23      visited[0][0] = true;
24      dist[0][0] = 1;
25
26      while (!q.empty()) {
27          auto [x, y] = q.front(); q.pop();
28          for (int dir = 0; dir < 4; dir++) {
29              int nx = x + dx[dir];
30              int ny = y + dy[dir];
31
32              if (nx < 0 || ny < 0 || nx >= n || ny >= m) continue;
33              if (!maze[nx][ny] || visited[nx][ny]) continue;
34
35              visited[nx][ny] = true;
36              dist[nx][ny] = dist[x][y] + 1;
37              q.push({nx, ny});
38          }
39      }
40
41      cout << dist[n - 1][m - 1] << '\n';
42  }
```

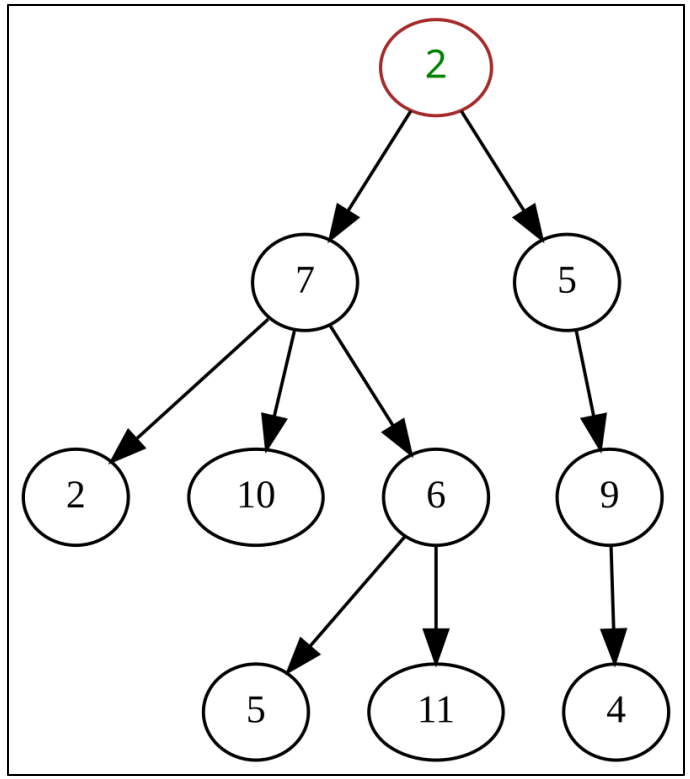
이진 트리

이진 트리 | 01 트리의 정의와 용어 정리



- **정의**
  - 트리는 **사이클이 없는 연결 그래프(무방향)**이다. 그래프의 특이한 형태라고 보면 된다.
  - **정점 n개, 간선 n – 1개를 가지며 모든 정점이 하나의 컴포넌트로 연결**되어 있어야 한다.
- **용어 정리**
  - 루트 (root) : 트리의 시작 정점. 보통 트리 탐색의 기준점. 트리 구조에서는 하나만 존재
  - 노드 (node) : 트리의 정점 (vertex). 루트, 자식, 리프 등 모두 노드라고 부른다.
  - 간선 (edge) : 노드 간의 연결 관계. 트리는 항상 간선이 n - 1개이다.
  - 부모 노드 (parent) : 어떤 노드를 기준으로 바로 위에 있는 노드를 말한다.
  - 자식 노드 (child) : 어떤 노드를 기준으로 바로 아래에 있는 노드를 말한다.
  - 형제 노드 (sibling) : 같은 부모를 가진 노드들을 의미한다.
  - 리프 노드 (leaf) : 자식 노드가 없는 노드를 의미한다.
  - 서브 트리 (sub-tree) : 어떤 노드를 루트로 하는 하위 트리를 말한다.

이진 트리 | 01 트리의 정의와 용어 정리



- **용어 정리**
  - 레벨 (level) : 루트로부터 얼마나 떨어져 있는지를 나타내는 깊이를 나타낸다.

    루트는 level 0 또는 1로 정한다.
  - 경로 (path) : 한 노드에서 다른 노드까지 이어지는 노드와 간선들의 나열을 의미한다.

    트리에서 위 경로는 유일(unique)하다.

예를 들어, 옆의 트리는 레벨 4인 트리이다. 2번 노드를 루트 노드로 가진다.

정점 5의 서브 트리는 정점 9, 4와 연결된 간선을 가지는 부분 트리이다.

정점 2, 10, 6은 형제 관계이다. 정점 7은 정점 2의 부모 노드이다.

이진 트리 | 02 이진 트리의 정의와 종류, 표현

- **이진 트리의 정의**

  이진 트리(Binary Tree)는 각 노드가 최대 두 개의 자식 노드를 가지는 트리를 말한다. 이 두 자식 노드는 흔히 왼쪽 자식(left child)과 오른쪽 자식(right child)으로 구분된다.

  * 알고리즘에서 다루는 트리는 보통 '이진 트리'를 의미한다.

- **이진 트리의 종류**

  1. Perfect binary tree : 모든 레벨의 노드가 가득 차있는 트리이다.
  2. Complete binary tree : 마지막 레벨 바로 전까지는 꽉 차있다. 마지막 레벨에서 왼쪽부터 차례대로 채워져 있는 트리이다.
  3. Skewed binary tree : 한 쪽으로 편향되게 채워져있는 트리이다.
  4. Binary Search Tree : 왼쪽 서브트리는 현재 노드보다 작은 값, 오른쪽은 큰 값을 가지는 트리이다.

     다음 시간에 BST에 대해 다룬다.
  5. Blanced binary Tree : 모든 노드의 왼쪽과 오른쪽 서브 트리의 높이가 1 이상 차이가 나지 않는 트리이다.

이진 트리 | 02 이진 트리의 정의와 종류, 표현

- **이진 트리의 표현**
  - 트리는 그래프의 일종이므로, 그래프와 동일하게 연결 리스트, 인접 행렬로 표현할 수 있다.
  - 이진 트리에서는 배열 방식으로 표현할 수도 있다.
  - N번 노드가 tree[i]라면, N번 노드의 왼쪽 자식 노드는 tree[i*2]이고 오른쪽 자식 노드는 tree[i*2 + 1]이다. 없으면 -1로 표시한다.

```
2    int tree[100];  // 트리를 1번 인덱스부터 저장
3
4    int root = 1; // 트리의 루트 노드 번호
5    tree[root]; // 트리의 루트 노드
6    tree[root * 2]; // 루트 노드의 왼쪽 자식
7    tree[root * 2 + 1]; // 루트 노드의 오른쪽 자식
```

이진 트리 | 03 이진 트리의 순회

- **이진 트리의 순회 : 트리의 모든 노드를 일정한 규칙에 따라 한 번씩 방문하는 방법**

전위 순회 : '루트 노드 → 왼쪽 노드 → 오른쪽 노드' 순으로 방문

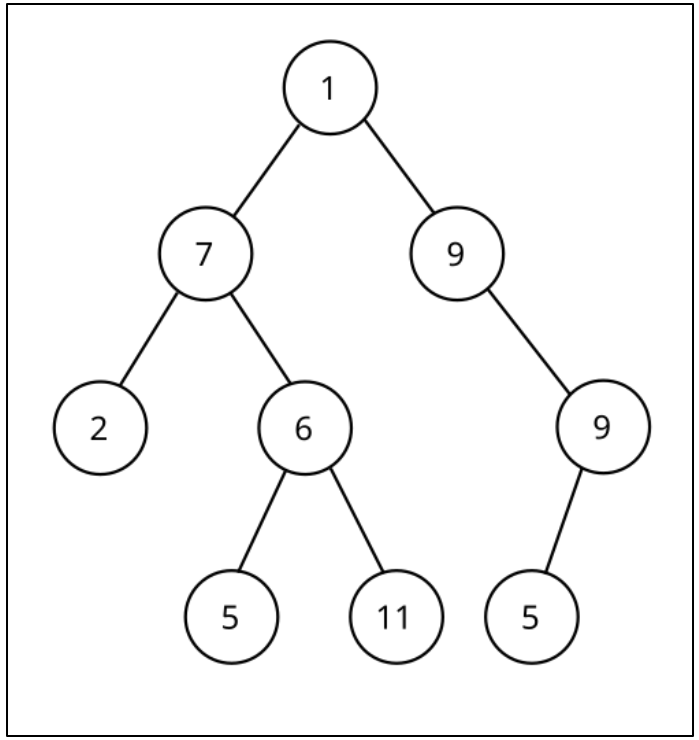
* 방문 순서 : '1 → {7을 루트로 하는 서브트리에 대한 전위 순회} → {9를 루트로 하는 서브트리에 대한 전위 순회}'

중위 순회 : '왼쪽 노드 → 루트 노드 → 오른쪽 노드' 순으로 방문

* 방문 순서 : '{7을 루트로 하는 서브트리에 대한 중위 순회} → 1 → {9를 루트로 하는 서브트리에 대한 중위 순회}'

후위 순회 : '왼쪽 노드 → 오른쪽 노드 → 루트 노드' 순으로 방문

* 방문 순서 : '{7을 루트로 하는 서브트리에 대한 후위 순회} → {9를 루트로 하는 서브트리에 대한 후위 순회} → 1'



이진 트리 | 04 예제 (1) : 백준 11725번 트리의 부모 찾기

- **문제 요약**
  - 루트 없는 트리가 주어진다. 이때, 트리의 루트를 1이라고 정했을 때, 각 노드의 부모를 구하는 프로그램을 작성하시오.
- **입력**
  - 첫째 줄에 노드의 개수 N ($2 \le N \le 100,000$)이 주어진다. 둘째 줄부터 N-1개의 줄에 트리 상에서 연결된 두 정점이 주어진다.
- **출력**
  - 첫째 줄부터 N-1개의 줄에 각 노드의 부모 노드 번호를 2번 노드부터 순서대로 출력한다.

이진 트리 | 04 예제 (1) : 백준 11725번 트리의 부모 찾기

- **문제 풀이**
  - 루트 노드가 1이라고 문제에서 정해줬다.
  - 이를 기준으로 DFS 또는 BFS를 수행하여, 특정 정점(R')에 방문하고, 연결된 정점들(lc, rc)의 부모 노드가 그 정점(R')임을 기록하면 된다.
  - 최악의 경우(모든 노드가 일직성 상에 배치)에는 DFS의 재귀 너무 깊어질 수 있으므로, BFS가 적당해 보인다.

이진 트리 | 04 예제 (1) : 백준 11725번 트리의 부모 찾기

- **구현**

Line 4 ~ 6

N: 노드 개수 / parent[i]: i번 노드의 부모를 저장

graph[i]: i번 노드에 연결된 노드 목록 (무방향 그래프)

Line 13 ~ 17

무방향 간선 u - v 입력 받고, 인접 리스트 양방향 연결함.

Line 19 ~ 28

루트 노드 1부터 BFS로 탐색을 함.

탐색하면서 각 노드의 부모를 기록함.

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

int N;
int parent[101010];
vector<int> graph[101010];

int main(){
  ios::sync_with_stdio(false);
  cin.tie(0); cout.tie(0);

  cin >> N;
  for(int i = 1; i < N; i++){ // [1, N) -> N - 1번 실행
    int u, v; cin >> u >> v;
    graph[u].push_back(v);
    graph[v].push_back(u);
  }

  queue<int> q;
  q.push(1);
  while(!q.empty()){
    auto cur = q.front(); q.pop();
    for(auto x : graph[cur]){
      if(parent[x]) continue;
      parent[x] = cur;
      q.push(x);
    }
  }

  for(int i = 2; i <= N; i++){
    cout << parent[i] << "\n";
  }
  return 0;
}
```

이진 트리 | 04 예제 (2) : 백준 1991번 트리 순회

- **문제 요약**
  - 이진 트리를 입력받아 전위 순회(preorder traversal), 중위 순회(inorder traversal), 후위 순회(postorder traversal)한 결과를 출력하는 프로그램을 작성하시오.
- **입력**
  - 첫째 줄에는 이진 트리의 노드의 개수 N($1 \le N \le 26$)이 주어진다.
  - 둘째 줄부터 N개의 줄에 걸쳐 각 노드와 그의 왼쪽 자식 노드, 오른쪽 자식 노드가 주어진다.
  - 노드의 이름은 A부터 차례대로 알파벳 대문자로 매겨지며, 항상 A가 루트 노드가 된다. 자식 노드가 없는 경우에는 .으로 표현한다.
- **출력**
  - 첫째 줄에 전위 순회, 둘째 줄에 중위 순회, 셋째 줄에 후위 순회한 결과를 출력한다. 각 줄에 N개의 알파벳을 공백 없이 출력하면 된다.

이진 트리 | 04 예제 (2) : 백준 1191번 트리 순회

- **문제 풀이**
  - 노드가 1개 있는 것도 트리의 정의를 만족하므로 트리이다.
  - 문제에서는 트리의 순회를 노드 관점에서 정의했는데, 다음과 같이 일반적인 정의가 가능하다.

    {루트 노드를 방문}

    {루트 노드의 오른쪽 자식을 루트 노드로 하는 부분 트리 방문}

    {루트 노드의 왼쪽 자식을 루트 노드로 하는 부분 트리 방문}
  - 위 3가지의 상대적인 순서에 따라(정확히는 루트 노드를 언제 방문하냐에 따라) 전위, 중위, 후위 순회가 결정된다.

이진 트리 | 04 예제 (2) : 백준 1191번 트리 순회

- **구현**

Line 4 ~ 7

lc[root] : root 노드의 왼쪽 자식 노드를 저장하는 배열이다.

lc[root] : root 노드의 왼쪽 자식 노드를 저장하는 배열이다.

```
1   #include <bits/stdc++.h>
2   using namespace std;
3
4   int N;
5   vector<char> graph[300];
6   char lc[300];
7   char rc[300];
```

Line 34 ~ 44

문제에 주어진 입력을 받는다.

Line 46 ~ 48

노드 A를 루트 노드로 하는 전위 순회, 중위 순회, 후위 순회를 진행한다.

```
29  int main(){
30    ios::sync_with_stdio(false);
31    cin.tie(0); cout.tie(0);
32
33    cin >> N;
34    while(N--){
35      char p, l, r; cin >> p >> l >> r;
36      if(l != '.'){
37        graph[p].push_back(l);
38        lc[p] = l;
39      }
40      if(r != '.'){
41        graph[p].push_back(r);
42        rc[p] = r;
43      }
44    }
45
46    preorder('A'); cout << "\n";
47    inorder('A'); cout << "\n";
48    postorder('A'); cout << "\n";
49    return 0;
50  }
```

위상 정렬

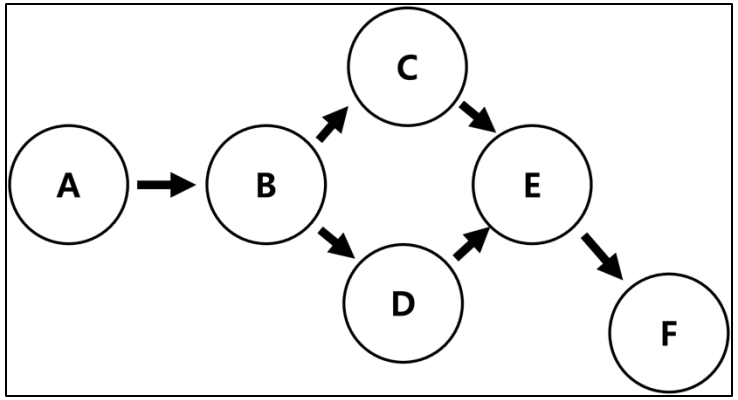
위상 정렬 | 01 개념

- **정의**
  - 방향 그래프(DAG)에서 모든 간선 (u → v)에 대해 정점 u가 v보다 앞에 오는 순서대로 정점을 나열하는 것

    아래 그래프에서 위상정렬을 수행하면 'A → B → {C, D} → {D, C} → E → F'가 된다.

- **특징**
  - 위상 정렬 결과는 하나가 아닐 수도 있다. 여러 개의 순서가 가능할 수 있다.

- **전제 조건**
  - 위상 정렬을 수행하고자 하는 그래프가 DAG(Directed Acyclic Graph)여야 한다. 즉, 간선에 방향이 있고 사이클이 없어야 한다.



위상 정렬 | 01 개념

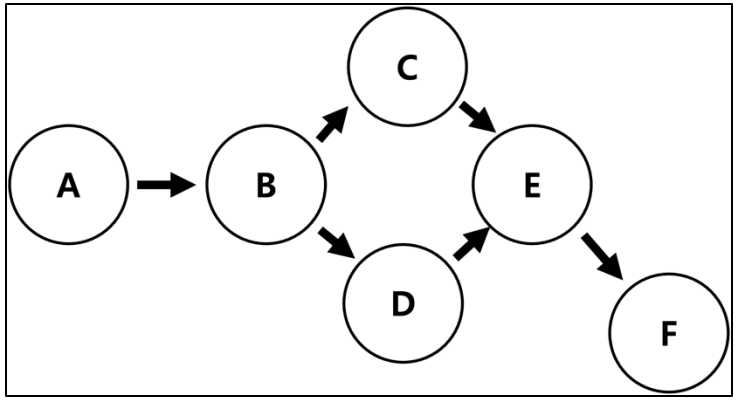
- **진입 차수(In-degree)의 활용**
  - 진입 차수 : 어떤 정점에 들어오는 간선의 개수를 진입 차수라고 한다.
  - 진입 차수를 활용하면, 위상 정렬을 수행할 수 있다.

1. 위상 정렬은 진입 차수가 0인 정점부터 시작한다. 진입 차수가 0이라는 건 선행 조건이 없는 작업이라는 뜻이다.

2. 진입 차수가 0인 노드를 큐에 넣고 하나씩 꺼내며 해당 노드가 향하는 정점의 진입 차수를 1씩 줄인다.

3. 진입 차수가 0이 된 정점은 새롭게 수행 가능한 작업이므로 큐에 넣는다.

위상 정렬을 진입 차수 기반으로 수행하는 알고리즘이 Kahn's Algorithm이라고 한다.



위상 정렬 | 02 Kahn's 알고리즘

- **핵심 아이디어**
  - 진입 차수가 0인 정점부터 차례대로 제거하면서 위상 정렬을 수행한다.
  - 제거된 정점이 가리키는 모든 정점의 진입 차수를 1씩 감소시키고, 그 중 진입 차수가 0이 된 정점들을 큐에 추가한다
- **알고리즘 순서 (Queue 이용)**
  1. 모든 정점의 진입 차수를 계산하고 진입 차수가 0인 정점들을 큐에 넣는다.
  2. 큐가 빌 때까지 다음을 반복한다.
     - 큐에서 정점 u를 꺼내고 결과 리스트에 추가한다.
     - u가 향하는 모든 정점 v에 대해:
     - v의 진입 차수를 1 감소
     - 만약 v의 진입 차수가 0이 되면 큐에 삽입
  3. 결과 리스트의 길이가 전체 정점 수와 같으면 → 위상 정렬 성공
  4. 아니라면 → 사이클 존재 → 위상 정렬 불가

위상 정렬 | 03 예제 : 백준 2252번 줄 세우기

- **문제 요약**
  - N명의 학생들을 키 순서대로 줄을 세우려고 한다. 각 학생의 키를 직접 재서 정렬하면 간단하겠지만, 마땅한 방법이 없어서 두 학생의 키를 비교하는 방법을 사용하기로 하였다. 그나마도 모든 학생들을 다 비교해 본 것이 아니고, 일부 학생들의 키만을 비교해 보았다.
  - 일부 학생들의 키를 비교한 결과가 주어졌을 때, 줄을 세우는 프로그램을 작성하시오.
- **입력**
  - 첫째 줄에 N(1 ≤ N ≤ 32,000), M(1 ≤ M ≤ 100,000)이 주어진다. M은 키를 비교한 횟수이다. 다음 M개의 줄에는 키를 비교한 두 학생의 번호 A, B가 주어진다. 이는 학생 A가 학생 B의 앞에 서야 한다는 의미이다.
  - 학생들의 번호는 1번부터 N번이다.
- **출력**
  - 첫째 줄에 학생들을 앞에서부터 줄을 세운 결과를 출력한다. 답이 여러 가지인 경우에는 아무거나 출력한다.

위상 정렬 | 03 예제 : 백준 2252번 줄 세우기

- **문제 풀이**
  - 이런 관계를 그래프로 만들면 당연히 DAG이다. A가 B보다 키가 크면서 B가 A보다 키가 클 수 없기 때문이다.
  - 따라서 위상 정렬이 가능하다.
  - 문제에서 키가 큰 순서대로 나열하라고 했으므로 간선의 방향은 A -> B이다.

위상 정렬 | 03 예제 : 백준 2252번 줄 세우기

- **구현**

  Line 5. in_degree[i] = {노드 i의 진입 차수}

  Line 17 ~ 21. 간선을 입력받고, 진입 차수를 계산해준다.

  Line 23 ~ 25. 진입 차수가 0인 정점을 큐에 넣는다.

  1. Line 28 ~ 37. 큐가 빌 때까지 다음을 반복한다 :
     - 큐에서 정점 cur를 꺼내고 출력한다.
     - cur가 향하는 모든 정점 v에 대해 다음을 수행한다 :
       - v의 진입 차수를 1 감소하고,
       - 만약 v의 진입 차수가 0이 되면 큐에 삽입한다.

```
1   #include <bits/stdc++.h>
2   using namespace std;
3
4   int N, M;
5   vector<int> in_degree;
6   vector<vector<int>> graph;
7
8
9   int main(){
10    ios::sync_with_stdio(false);
11    cout.tie(0); cin.tie(0);
12
13    cin >> N >> M;
14    graph.resize(N + 1);
15    in_degree.resize(N + 1);
16
17    while(M--){
18      int u, v; cin >> u >> v; // u -> v
19      graph[u].push_back(v);
20      in_degree[v]++;
21    }
22
23    queue<int> q;
24    for(int i = 1; i <= N; i++){
25      if(in_degree[i] == 0) q.push(i);
26    }
27
28    while(!q.empty()){
29      int cur = q.front();
30      q.pop();
31
32      cout << cur << " ";
33      for(auto x : graph[cur]){
34        in_degree[x]--;
35        if(!in_degree[x]) q.push(x);
36      }
37    }
38    return 0;
39  }
```

문제

- **기본**
  - 백준 6118번 숨바꼭질
  - 백준 2583번 영역 구하기
  - 백준 5014번 스타트링크
- **심화**
  - 백준 7569번 토마토
  - 백준 13549번 숨바꼭질 3
  - 백준 4179번 불!

수고하셨습니다!