# 위상 정렬 (Topological Sort)

---

### 정의

- G = (V,E)의 G가 간선(u,v)를 가질 때, u가 v보다 순서상으로 먼저 나타나도록 모든 정점을 선형으로 나열하는 것.
- 모든 방향 간선이 왼쪽에서 오른쪽으로 향할 수 있도록 수평선을 따르는 정점의 나열

### 특징

1. DFS를 사용한다.
2. 방향 비순환 그래프에서 사용이 가능하다(cycle이 있으면 불가능함)
3. 사건들 사이의 우선순위를 나타낸다.
4. 그래프에 따라서 위상 정렬의 결과가 `여러 개` 있을 수 있다.
5. `DAG(Directed acycle graph)` : 유향 비순환 그래프에서 가능하다.

### 시간 복잡도

- DFS를 사용하기 때문에 `O(V + E)` 의 시간복잡도를 가진다.

### 특정

- 방향 그래프 G가 비순환이면 G의 깊이 우선 검색은 역행 간선을 만들지 않고 그 역도 성립한다.
- Topological-sort(G)는 입력으로 주어진 방향 비순환 그래프 G의 위상 정렬 하나를 생성한다.

---

# Topological sort를 구하는 2가지 방법

위에서는 DFS를 사용해서 위상 정렬을 구한다고 했지만, 위상 정렬을 구할 수 있는 방법은 유일하지 않다.

크게 DFS와 BFS를 기반으로 하는 2가지 방식이 있다.

각각의 방식을 살펴본 후, **위상정렬의 유일성, 위상정렬이 가능한지(cycle detection)을 각각 어떤 방식으로 진행하는지 알아보겠다.

# 1. DFS

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <stack>
#define endl '\n'
using namespace std;

vector<bool> visited;
vector<vector<int>> adj;
stack<int> topoStack;
int V;

void init()
{
    V = 6;
    adj.resize(V);
    visited.resize(V + 1, false);

    adj[5].push_back(2);
    adj[5].push_back(0);
    adj[4].push_back(0);
    adj[4].push_back(1);
    adj[2].push_back(3);
    adj[3].push_back(1);
}

void dfs(const int& node)
{
    visited[node] = true;

    for (auto& ele : adj[node])
    {
```

```
            if (!visited[ele])
                dfs(ele);
        }
        topoStack.push(node);
    }

    void topologicalSort()
    {
        for (int i = 0; i < V; ++i)
        {
            if (!visited[i])
                dfs(i);
        }
        cout << "This is topological order" << endl;
        while (!topoStack.empty())
        {
            cout << topoStack.top() << " ";
            topoStack.pop();
        }
    }

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

            init();
        topologicalSort();

            return (0);
    }
```

**DFS 방식의 위상정렬의 타당성**

위상 정렬은 방향 비순환 그래프(DAG)에서 노드들을 한 방향으로 정렬하는 방식이다.

DFS는 한 노드에서 시작해서 그 노드에 연결된 다른 노드로 탐색을 진행한다. 탐색 과정에서 모든 자식 노드를 방문한 후, 해당 노드를 방문 완료로 표시한다. 이 방식은 자연스럽게 "종속된 노드를 먼저 방문"하게 되는 특징을 가지고 있다.

DFS를 마친 후, 스택에 쌓인 순서를 뒤집으면 **위상 정렬의 순서**가 된다.

- 노드가 스택에 쌓일 때, 해당 노드의 모든 자식이 먼저 stack에 쌓이게 된다. 따라서 스택의 역순으로 노드를 꺼내면, 각 노드가 그 노드로 향하는 간선의 출발점보다 먼저 나온다.
- DAG에서 사이클이 없으므로, DFS가 끝날 때까지 모든 노드가 정확히 한 번씩 방문되고, 스택에 쌓이게 된다.

**예시로 이해하기**

```
1 → 2 → 3
   ↘ 4
```

- 1번은 2번과 4번으로 갈 수 있다.
- 2번은 3번으로 갈 수 있다.

1. DFS(1)을 시작한다. -> DFS(1)은 2와 4로 갈 수 있지만 4로 먼저 간다고 가정한다
2. DFS(4)를 진행한다 -> 자식이 없으니 stack에 push한다. -> stack[4]
3. DFS(1)의 자식 노드인 DFS(2)를 실행한다.
4. DFS(2)에서 자식 노드인 DFS(3)을 실행한다.
5. DFS(3)에는 자식이 없으니 stack에 3이 들어간다 -> stack[4,3]
6. DFS(2)가 호출되면서 stack에 2가 들어간다 -> stack[4,3,2]
7. DFS(1)이 호출되면서 stack에 1이 들어간다 -> stack[4,3,2,1]
8. stack을 역순으로 출력하면 위상정렬의 순서가된다 -> [1,2,3,4]

- **위의 순서는 유일하지 않다**

# 2. Kahn's Algorithm (BFS기반)

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

int V;
bool is_double = false;
```

```
vector<vector<int>> adj;
vector<int> indegree;
vector<int> result;
void initCycleGraph() {
    // 사이클이 있는 그래프를 초기화합니다.
    V = 4;
    adj = vector<vector<int>>(V + 1);
    indegree = vector<int>(V + 1, 0);

    // 간선 추가 (1 -> 2 -> 3 -> 4 -> 1: 사이클 형성)
    adj[1].push_back(2);
    adj[2].push_back(3);
    adj[3].push_back(4);
    adj[4].push_back(1);

    // 진입 차수 업데이트
    indegree[2]++;
    indegree[3]++;
    indegree[4]++;
    indegree[1]++;
}

void initMultipleTopo() {
    // 여러 가지 위상 정렬 결과가 가능한 그래프를 초기화합니다.
    V = 6;
    adj = vector<vector<int>>(V + 1);
    indegree = vector<int>(V + 1, 0);

    // 간선 추가
    adj[1].push_back(3);
    adj[1].push_back(4);
    adj[2].push_back(3);
    adj[2].push_back(5);
    adj[4].push_back(6);
    adj[5].push_back(6);

    // 진입 차수 업데이트
    indegree[3]++;
    indegree[3]++;
    indegree[4]++;
    indegree[5]++;
```

```
    indegree[6]++;
    indegree[6]++;
}

void normalGraph() {
    // 사이클이 없는 DAG를 초기화합니다.
    V = 5;
    adj = vector<vector<int>>(V + 1);
    indegree = vector<int>(V + 1, 0);

    // 간선 추가
    adj[1].push_back(2);
    adj[2].push_back(3);
    adj[3].push_back(4);
    adj[4].push_back(5);

    // 진입 차수 업데이트
    indegree[2]++;
    indegree[3]++;
    indegree[4]++;
    indegree[5]++;
}

void topologicalSort()
{
    queue<int> q;

    for (size_t i = 1; i <= V; ++i)
    {
        if (!indegree[i])
            q.push(i);
    }

    while (!q.empty())
    {
        if (q.size() != 1)
            is_double = true;
        int node = q.front();
        q.pop();
        result.push_back(node);
```

```
        for (int neighbor : adj[node])
        {
            indegree[neighbor]--;
            if (indegree[neighbor] == 0)
                q.push(neighbor);
        }


    }if (result.size() == V)
        {
            for (auto& ele : result)
            {
                cout << ele << " ";
            }
            cout << endl;
        }
        else
            cout << "There is cycle : unvalid graph" << endl;
    if (is_double)
        cout << "There are multiple topological sorts" << endl;
    else
        cout << "There is single topological sort" << endl;

}

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

    //initCycleGraph();
    //normalGraph();
    //initMultipleTopo();
    topologicalSort();

        return (0);
}
```

*들어가기 앞서 : indegree와 outdegree에 대해*

**indegree(진입 차선)**
indegree란 특정 노드로 들어오는 간선의 수를 나타낸다. 즉, 해당 노드의 진입 차수는 다른 노드가 해당 노드로 향하는 간선의 충 개수라고 할 수 있다.

**outdegree(출 차선)**
outdegree란 특정 노드에서 나가는 간선의 수를 나타낸다. 즉, 해당 노드의 출차수는 특정 노드에서 다른 노드들로 향하는 간선의 총 개수를 의미한다.

```
1 → 2 → 3
↓
4
```

- 노드 1의 indegree (0), outdegree(2)
- 노드 2의 indegree(1), outdegree(1)
- 노드 3의 indegree(1), outdegree(0)
- 노드 4의 indegree(1), outdegree(0)

**Khan's Algorithm의 수도코드**

```
1. 들어오는 간선이 없는(indegree = 0) 정점을 queue에 넣는다.
2. 정점 개수만큼 이 행동을 반복한다 : 큐의 front 원소를 빼서 그 정점에서 나가는 간선을 모두 삭제한다. 이때 삭제하면서 indegree가 0이 된 새로운 정점이 생기면 그것들도 큐에 넣는다.
3. 큐에서 빼는 정점 순서가 위상 정렬의 결과다.
```

**Question**

**간선을 삭제한다는 것을 어떻게 구현할 수 있나요??**

- indegree배열을 만들어 준 뒤에, 해당 정점의 indegree값을 1빼주면 된다.

**Topological sort의 유일성 (Topological sort의 결과가 하나)라는 것은 어떻게 알 수 있나요??**

- topological sort를 진행하면서 queue의 size가 2 이상이 되는 경우가 1번이라도 발생하게 된다면 위상정렬의 유일성이 위배된다.

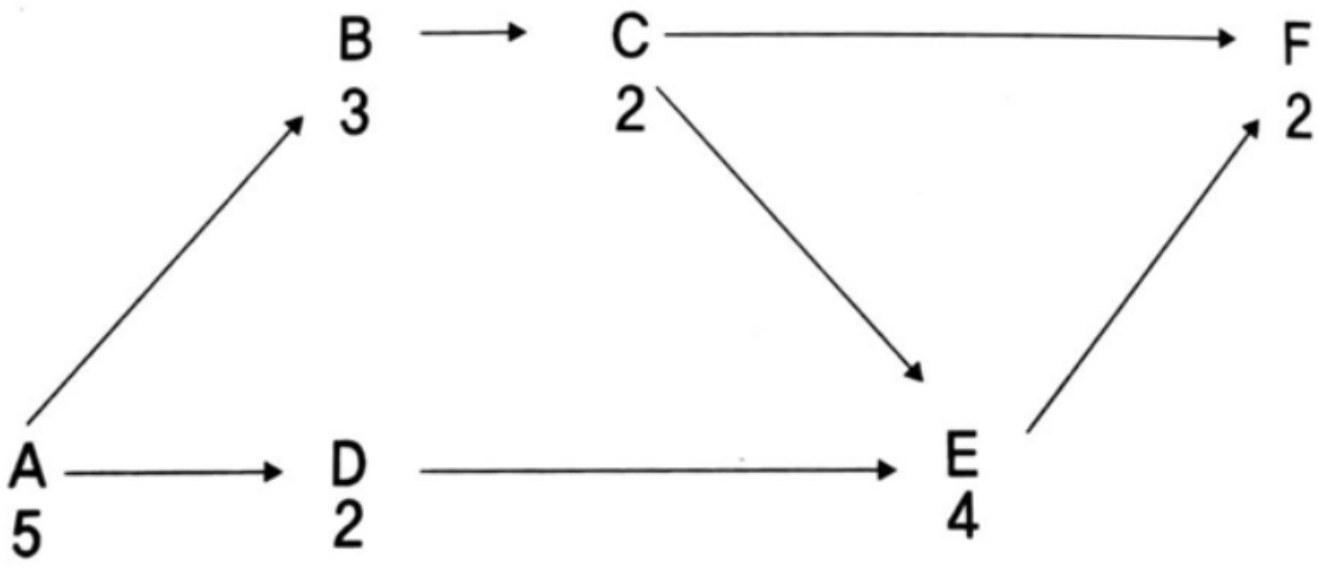
**Topological sort를 진행하면서 cycle이 발생함을 어떻게 알 수 있나요?**

- Topological sort의 결과가 있는 queue가 정점의 개수와 다르다면 cycle이 있다는 뜻이다.

⭐개인적으로는 kahn's algorithm이 구현이 더 쉬워서 선호함!!

---

# 예시로 보는 위상정렬 알고리즘




---

# 위상정렬과 DAG의 최단거리

위의 내용만을 가지고 백준이나, 프로그래밍 문제를 풀게 된다면 많이 당황스러울 것입니다.

왜냐하면 위상정렬의 정렬 순서를 바탕으로 문제를 해결해야하는 경우가 있기 때문입니다.

특히 위상 정렬의 결과를 바탕으로 최단거리를 구하는 방법이 있습니다. 이를 **방향 비순환 그래프(DAG)의 최단거리 구하기** 라고 지칭합니다.

**⭐방향 비순환 그래프, 일의 순서가 정해짐이 판명된다면, Topological sort의 순서대로 최단 거리를 구하자**

즉 위상정렬의 순서대로 최단거리를 update를 하게 된다면, 일반적인 최단거리 알고리즘의 시간복잡도보다 더 빠른 시간 안에 최단거리를 얻을 수 있습니다.

**PS 시간복잡도 분석**

1. 일반적인 최단거리 알고리즘 : `(V+E)logV`
2. 위상정렬을 사용한 DAG 최단거리 : `(V+E)`