SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

SW문제해결응용

<알림>

본 강의는 삼성 청년 SW아카데미의 컨텐츠로
보안서약서에 의거하여
강의 내용을 어떠한 사유로도 임의로 복사,
촬영, 녹음, 복제, 보관, 전송하거나
허가 받지 않은 저장매체를
이용한 보관, 제3자에게 누설, 공개,
또는 사용하는 등의 행위를 금합니다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Day1. DFS

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

그래프와 트리

그래프 (Graph)

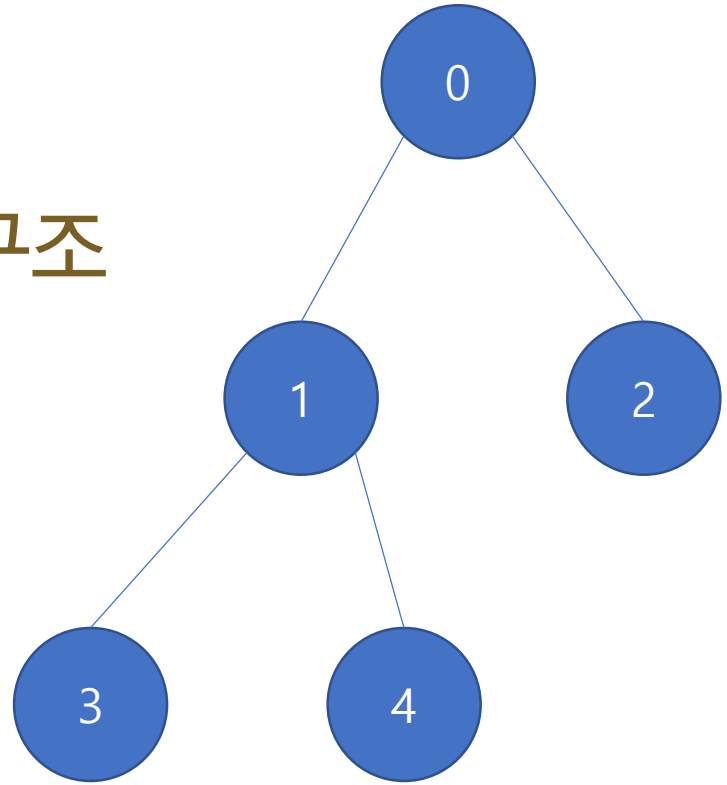
정점(node/vectex)와 간선(edge)로 구성된 자료구조이다.

- 네트워크 연결
- 사람과 사람 사이 관계
- 지도 내 최단경로

등등을 표현할 때 사용된다.

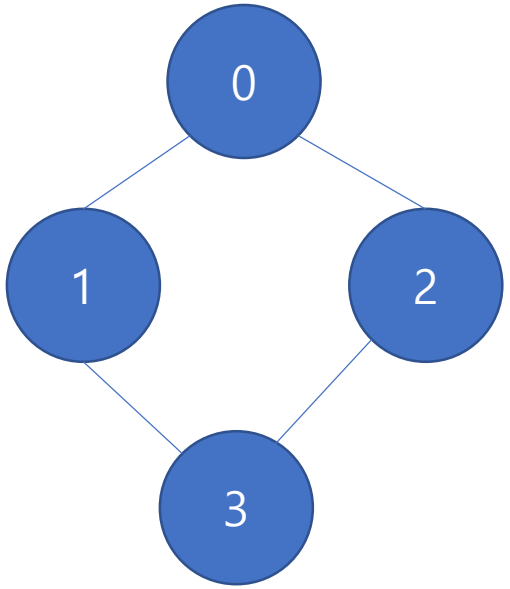
노드와 노드 사이의 상관 관계를 나타내는 자료 구조

- 실생활과 연계성이 높기 때문에 출제 빈도가 높다.

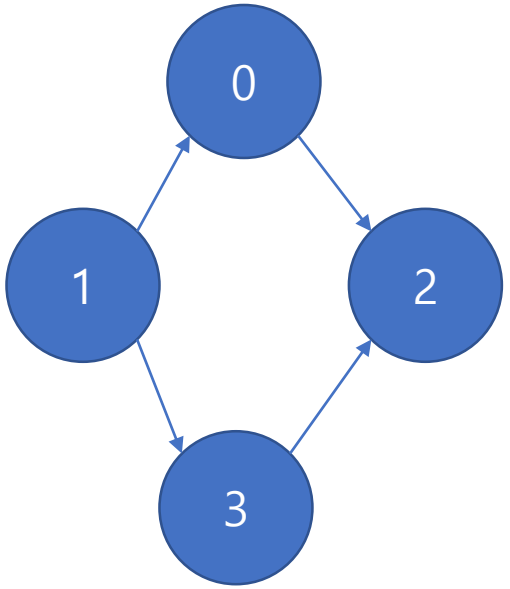


방향 여부 : 간선에 방향이 있다.

- **무향/양방향 or 유향/단방향**



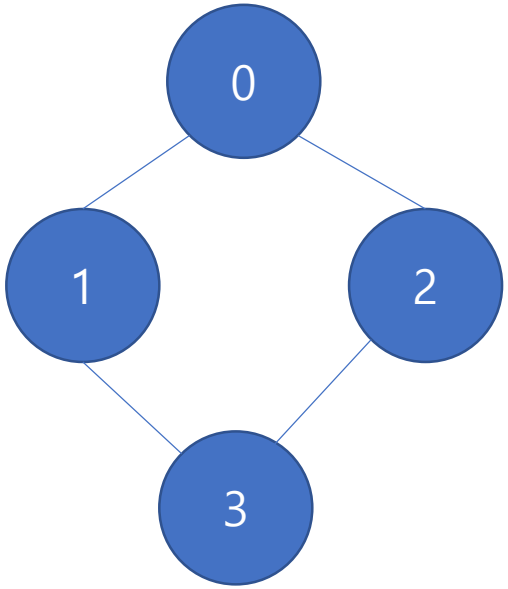
무향/양방향 그래프



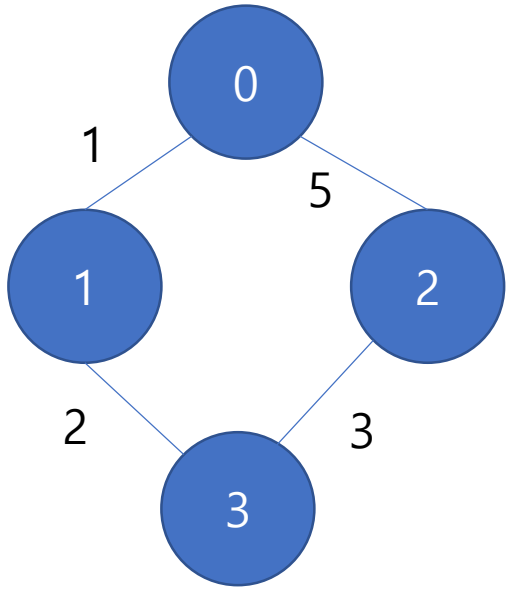
유향/단방향 그래프

가중치 : 간선에 가중치가 있다.

- **최단 경로 or 저비용 경로 찾기 문제 출제 (다익스트라 알고리즘 사용)**



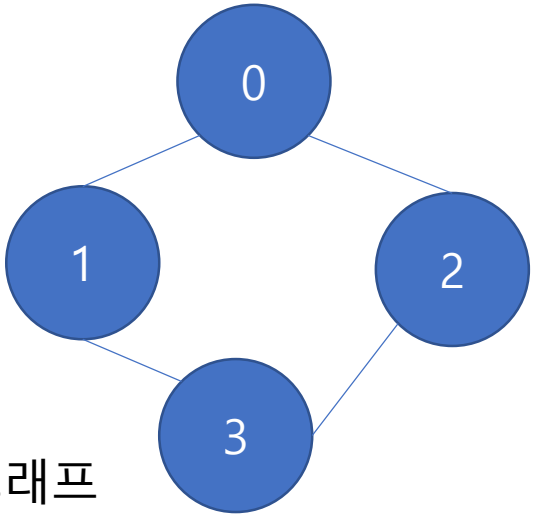
가중치 X
모든 간선 1



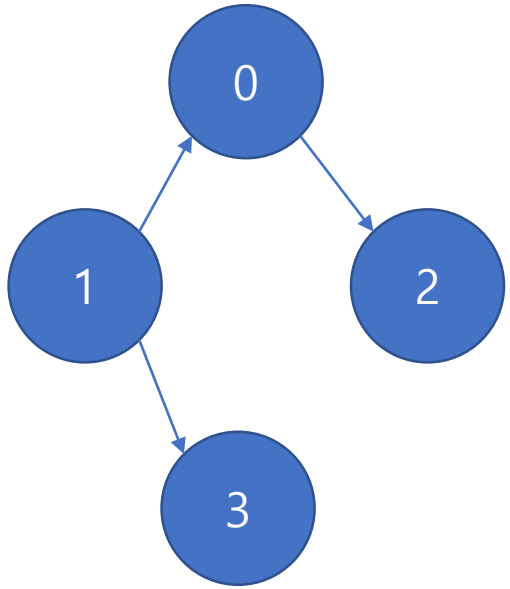
가중치 O
최단경로, 저비용

순환/비순환 그래프

- **순환 : 출발한 정점으로 돌아올 수 있는 경로가 있는 그래프**
  - 순환이 존재하면 위상정렬 불가능
  - DFS를 활용해서 순환 여부 확인 가능
- **비순환 : 어떤 정점에서 시작해도 순환이 없는 그래프**
  - 방향 그래프 중 비순환일 경우 DAG라 한다. (위상 정렬 가능)
  - 트리는 비순환 그래프의 한 종류



순환 그래프

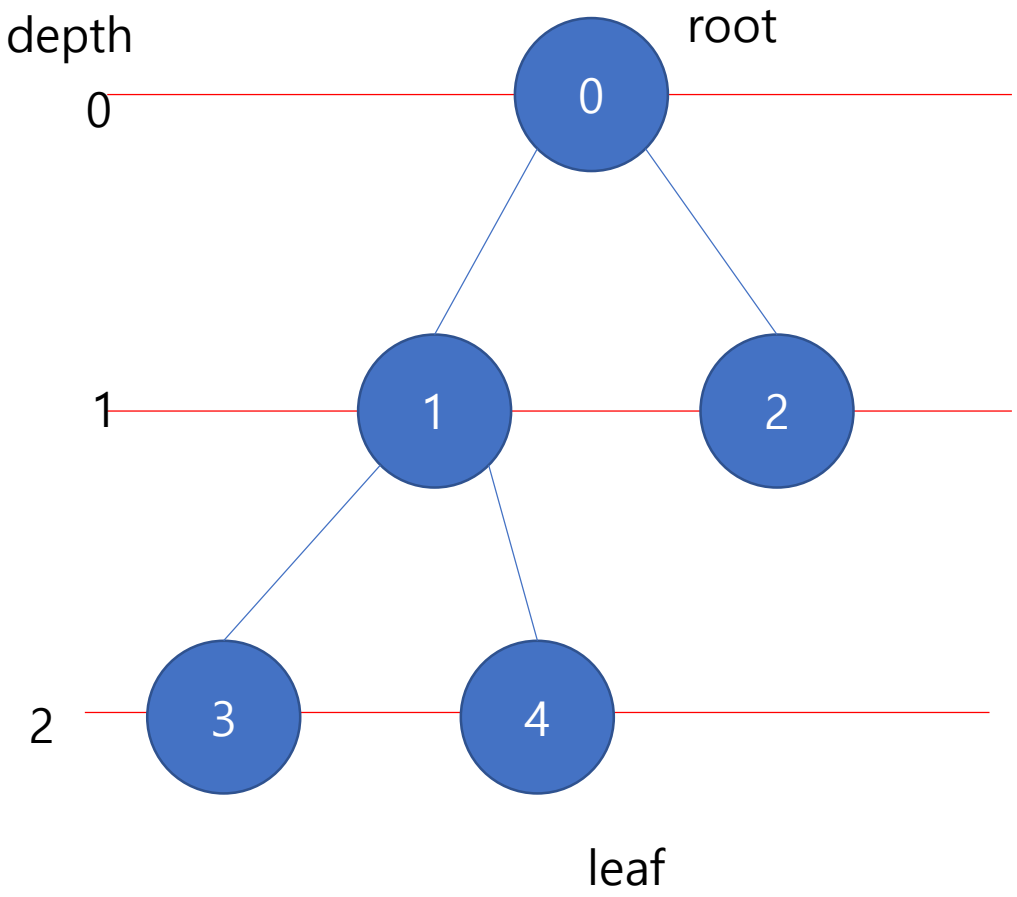


비순환 그래프

사이클이 없는 연결 그래프, 그래프의 한 종류

용어를 알고 있자.

- root : 최상위 노드 (부모가 없는 노드)
- leaf : 최하위 노드 (자식이 없는 노드)
- depth(깊이) : 루트에서 특정 노드 까지의 간선 개수
- level : 같은 깊이를 가진 노드들의 집합
- degree(차수) : 해당 노드에 연결된 모든 간선의 개수



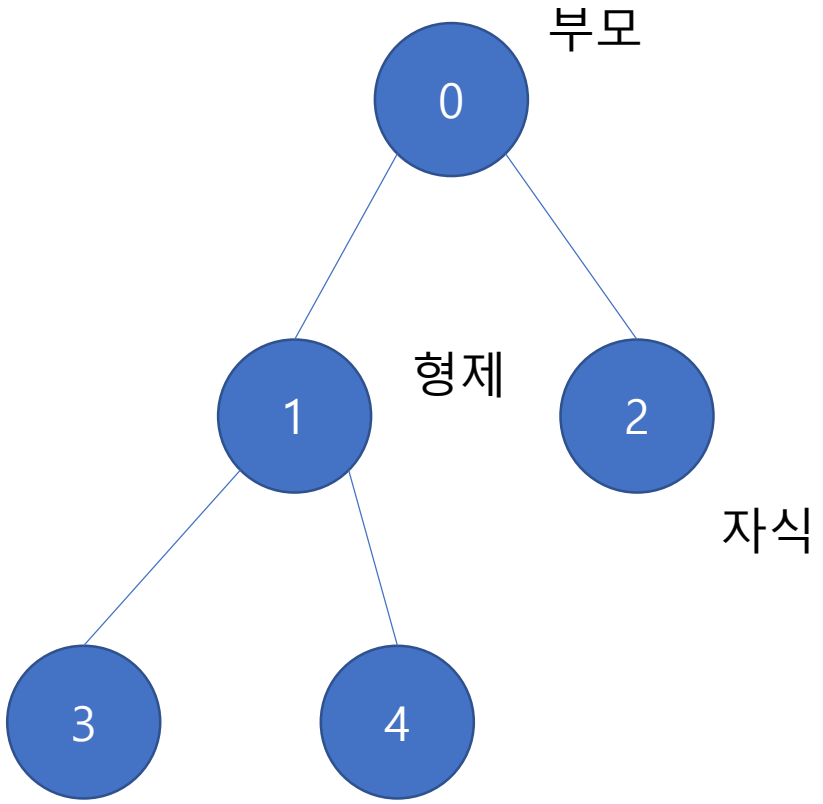
트리의 특징

특징

- 사이클이 없는 무향 그래프(비순환 그래프)
- 루트 노드가 존재(부모, 자식 관계)
- N개의 정점을 가진 트리는 항상 N-1 개의 간선을 가짐
- 한 정점에서 다른 정점까지 유일한 경로가 존재

부모 자식 관계가 있다.

- 0번 노드 (부모) → 1,2번 노드(자식)
- 1번 노드 ↔ 2번 노드 (형제 sibling)



트리의 종류

엄청 많다. 알고만 있자.

- **이진 트리 (Binary Tree)**
  - 각 노드가 최대 두 개의 자식(왼쪽, 오른쪽) 을 가짐
- **이진 탐색 트리 (Binary Search Tree , BST)**
  - 왼쪽 서브트리 < 루트 < 오른쪽 서브트리 (정렬)
- **힙 (Heap)**
  - 완전 이진 트리 + 부모-자식 간 정렬 조건
  - ex) 우선순위 큐
- **트라이 (Trie)**
  - 문자열 검색에 특화된 트리
  - 각 노드에 알파벳 저장
  - ex) 사전 검색, 자동 완성

트리의 순회 방법

모든 노드를 방문하는 방법에는 DFS, BFS 가 존재한다.

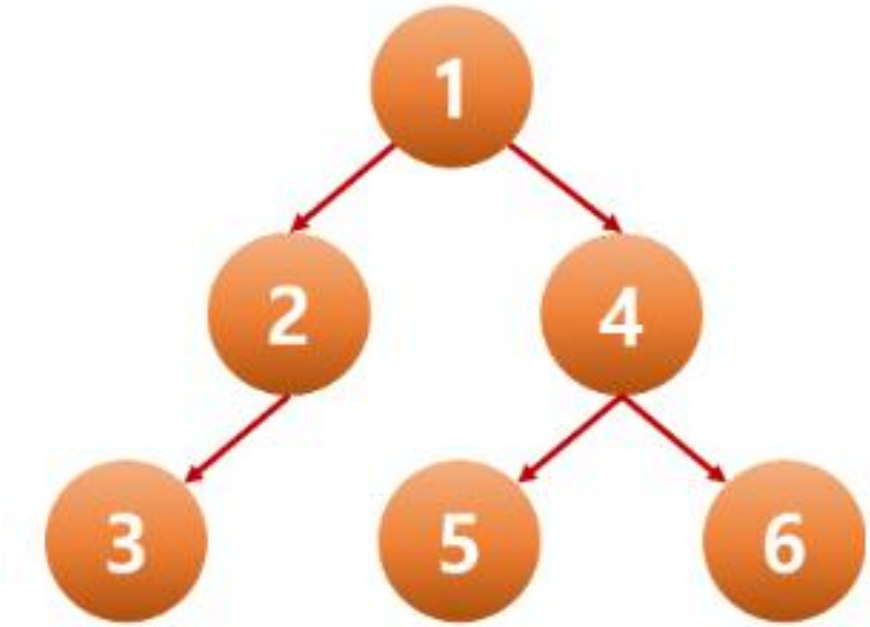
- 해당 알고리즘은 추후에 배운다.

그 중 트리는 순회 방법이 있는 데,

순회 3가지 방법을 알고 있어야 한다.

- 전위 순회(Preorder) : 루트 → 왼쪽 → 오른쪽
  - 1 - 2 - 3 - 4 - 5 - 6
- 중위 순회(Inorder) : 왼쪽 → 루트 → 오른쪽
  - 3 - 2 - 1 - 5- 4 - 6
- 후위 순회(Postorder) : 왼쪽 → 오른쪽 → 루트
  - 3 - 2 - 5 - 6- 4 - 1

왼쪽 → 오른쪽은 고정이다. 루트의 위치만 바뀐다.



인접행렬과 인접리스트

그래프 표현하기

그래프 자료구조를 표현하는 방법은 두가지이다.

1. 인접행렬
2. 인접리스트

하나하나 구현해본다.

인접 행렬

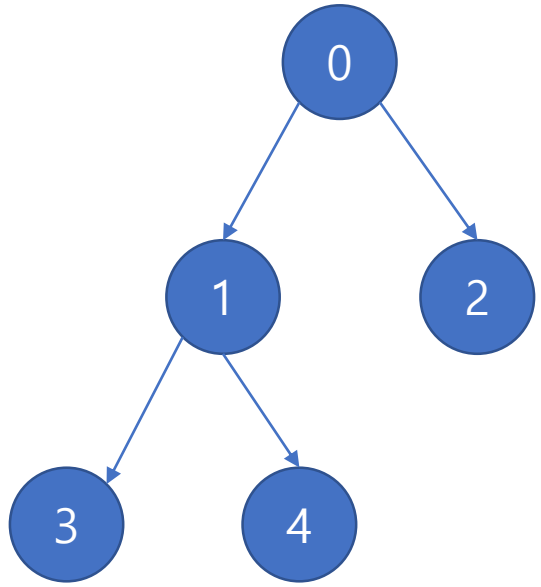
오른쪽과 같은 그래프가 있다.

노드와 노드의 관계를 표현한다.

- 0 → 1
- 0 → 2
- 1 → 3
- 1 → 4

이 관계를 2차원 배열로 표현하자.

- DAT 활용
  - index : 노드의 번호
  - value : 연결 여부 (0/1)



to

from

	0	1	2	3	4
0	0	1	1	0	0
1	0	0	0	1	1
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

인접행렬 코드로 구현하기

- **배열을 사용하기 때문에, 조회 속도가 빠르다.**
- **모든 노드 간의 연결 여부 파악이 쉽다.**
- **쉽게 가중치 처리가 가능하다.**
- **불필요한 메모리 낭비가 있을 수 있다.**
  - 저장할 데이터는 4, 사용한 메모리는 25byte
  - N이 1000 개 이하면, 인접 행렬도 좋다.

```
int arr[5][5];
int main() {
        //노드 개수 N, 간선 개수 M
        int N, M;
        cin >> N >> M;
        for (int i = 0; i < M; i++) {
                int from, to;
                cin >> from >> to;
                //from -> to 로 연결되어 있다.
                arr[from][to] = 1;
        }
```

https://gist.github.com/hoconoco/c113e7074db0fb9b0fd67674cc592fa8

인접 리스트는 vector<vector> 를 사용한다.

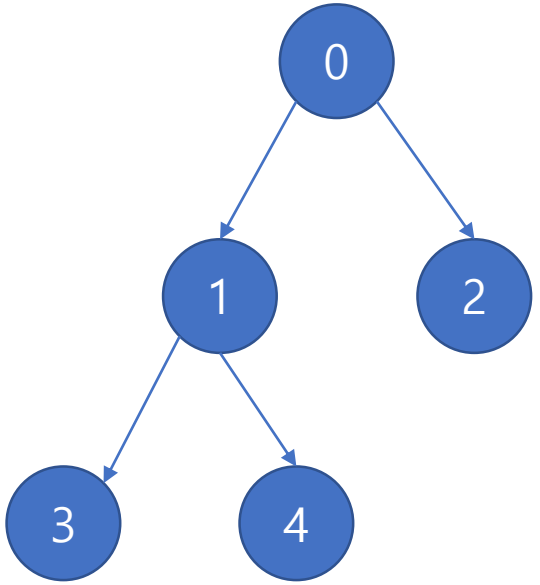
from → to 로 갈 수 있을 경우

index : from

value : to

형식으로 데이터를 추가한다.

- 0 → 1, 2
- 1 → 3, 4



- 링크드 리스트 or 다양한 형태로 표현이 가능하나,

요즘 코딩테스트 출제 유형은 vector< vector < >> 를 물어본다.

인접리스트 코드로 구현하기

- **vector 로 구현한다.**

```
vector<vector<int >> arr(5);
int main() {
        //노드 개수 N, 간선 개수 M
        int N, M;
        cin >> N >> M;
        for (int i = 0; i < M; i++) {
                int from, to;
                cin >> from >> to;
                //from -> to로 연결되어 있다.
                arr[from].push_back(to);
        }
```

https://gist.github.com/hoconoco/e7f4b0cd10a98185772de475a93e49dc

인접 행렬 vs 인접 리스트

인접 행렬 (2차원 배열을 사용하여 그래프의 간선 정보 저장)

- **장점**
  - 모든 노드 간의 연결여부 파악이 쉽다
  - 조회가 빠르다
  - 가중치 넣기가 쉽다
  - 노드를 작은 순/큰 순으로 접근하는 데 추가적인 연산 작업이 없어도 된다.
- **단점**
  - 메모리 낭비 (N 1000 이하 사용)

인접 리스트 (Vector를 사용하여 그래프의 간선 정보 저장)

- **장점**
  - 간성 정보만큼만 메모리 사용
  - 탐색 속도가 빠르다. (N번 반복하지 않고, .size() 만큼만 조회)
- **단점**
  - 간선 개수만큼 조회 시간이 걸린다.
  - 노드를 작은 순/큰 순으로 접근하는 데 추가적인 연산작업이 필요 (정렬)

SAMSUNG SW ACADEMY FOR YOUTH

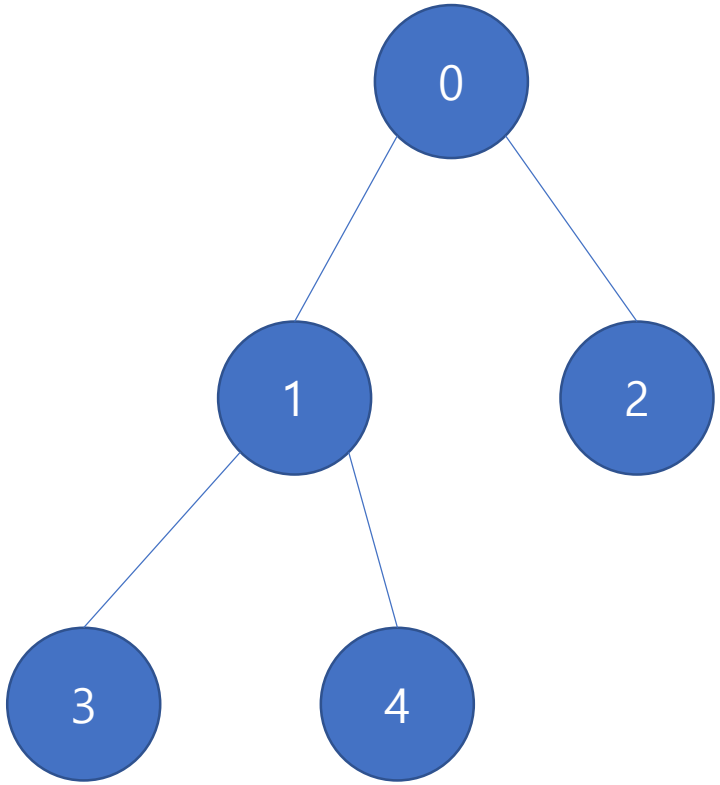
함께가요 미래로!
Enabling People

DFS 소개

그래프에 저장된 데이터를 탐색하고자 한다.

다양한 알고리즘이 사용된다.

- **DFS**
- **BFS**
- **다익스트라 등등**

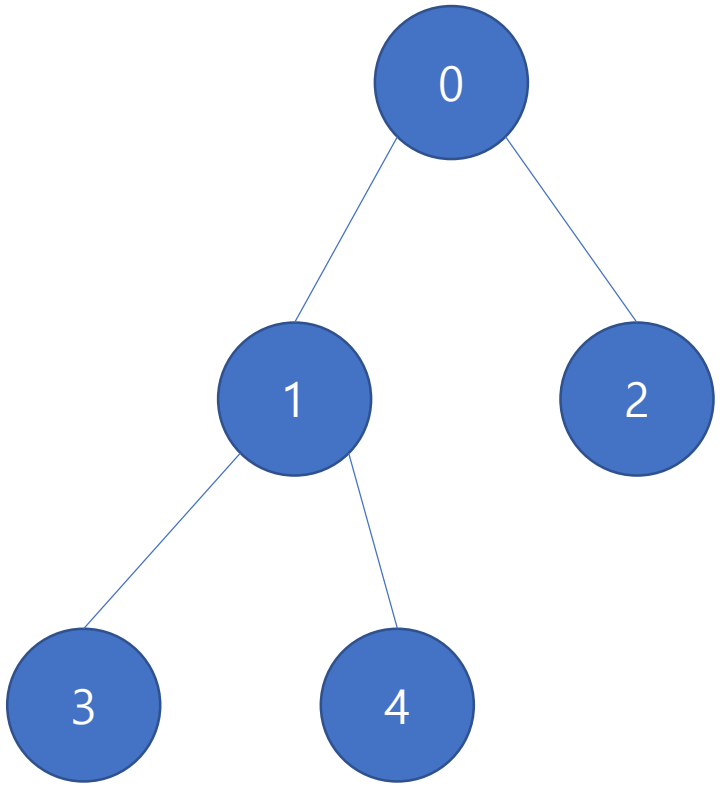


Depth First Search 깊이 우선 탐색

- 0 → 1 → 3 → 4 → 2

갈 수 있는 최대 깊이만큼 탐색한다.

- 재귀로 구현한다.
- 그래프에서 이뤄지는 백트래킹



재귀 함수와 DFS는 유사하다.

DFS는 다음 노드로 호출한다.

- now : 현재 노드
- next : 다음 방문할 노드

```
void func(int level) {

    func(level+1);

}
```

```
void dfs(int now) {

    dfs(next);

}
```

DFS 기본

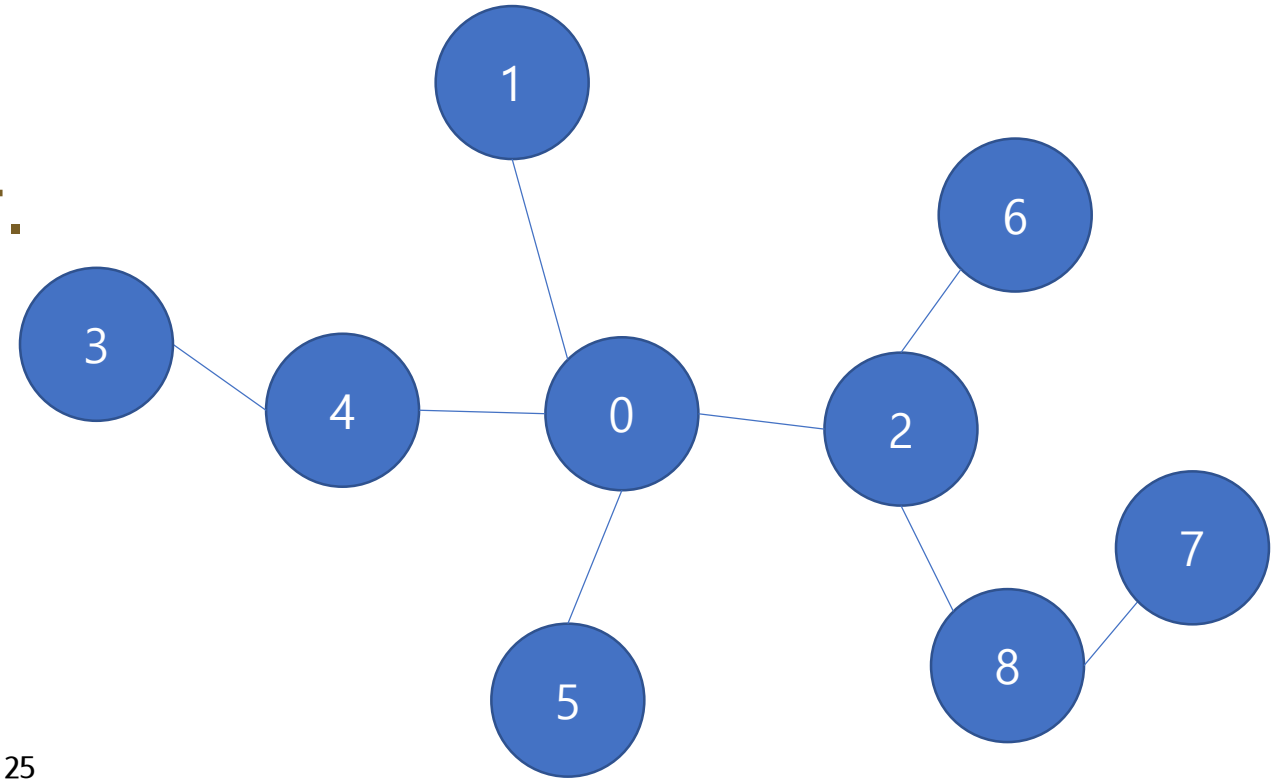
그래프를 탐색한다.

오른쪽과 같은 그래프가 있다.

- **노드 개수 : 9**
- **간선 개수 : 8**

DFS로 탐색한다면

0→1→2→6→8→7→4→3→5 이다.



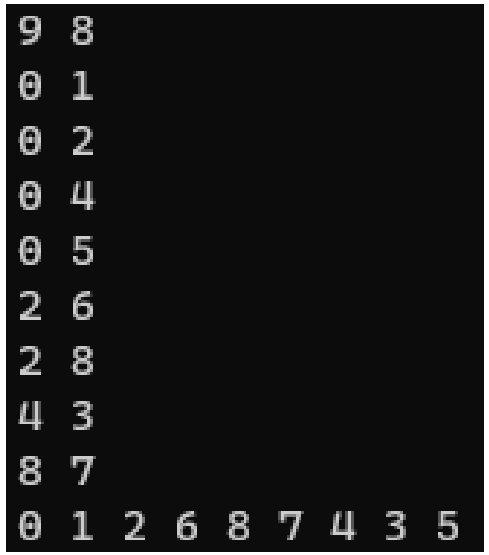
인접행렬로 그래프를 구현하고, DFS로 탐색한다.

```
9 8
0 1
0 2
0 4
0 5
2 6
2 8
4 3
8 7
0 1 2 6 8 7 4 3 5
```

```
//인접행렬로 구현
int arr[10][10];
void dfs(int now) {
        //now에서 갈 수 있는 next를 뽑는다.
        cout << now << " ";
        //갈수 있는 노드 탐색
        for (int next = 0; next < N; next++) {
                if (arr[now][next] == 1) {
                        //dfs 실행
                        dfs(next);
                }
        }
}
```

https://gist.github.com/hoconoco/449089c359d4b5d93f74cef12f11c36c

인접리스트로 그래프를 구현하고, DFS로 탐색한다.




```cpp
//인접리스트로 구현
vector<vector<int>> arr(10);

//now : 현재 진입한 노드
void dfs(int now) {

        cout << now << " ";
        //now에서 갈 수 있는 노드(next)로 dfs
        for (int i = 0; i < arr[now].size(); i++) {
                int next = arr[now][i];
                //dfs 실행
                dfs(next);
        }
}
```

https://gist.github.com/hoconoco/7d3efa042d8a4823bd45f5942256e27c

현재 코드는 단방향 그래프일 때 구현이다.

양방향으로 구현하자.

- **입력할 때, to→from 도 연결하면 된다.**

- **실행한다.**
  - 어마어마하다.
  - 제대로 된 탐색이 되지 않는다.
  - why? → 중복되기 때문

```
int main() {
    cin >> N >> M;
    for (int i = 0; i < M; i++) {
        int from, to;
        cin >> from >> to;
        //양방향 연결
        arr[from][to] = 1;
        arr[to][from] = 1;
    }
    //1번 노드로 dfs 시작
    dfs(1);
```

양방향 입력 시 중복되는 이유

인접행렬을 예시로 보자.

오른쪽과 같은 양방향 그래프로 배열에 값을 넣으면 표와 같다.

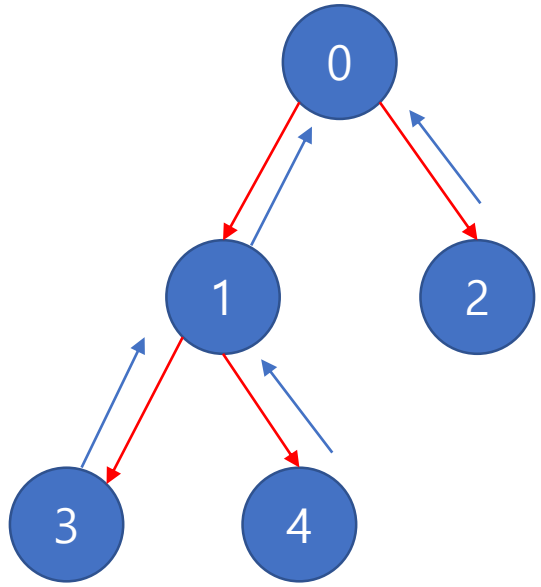
여기서 dfs를 수행할 경우 아래와 같이 동작한다.

1) 0 → 1 수행
2) 1 → 0 수행
3) 0 → 1 수행

반복된다.

디버깅해서 now, next 값을 조사식으로 확인해도 마찬가지!

→ 해결방법. 방문 여부를 체크해야 한다!



to

from	0	1	2	3	4
0	0	1	1	0	0
1	1	0	0	1	1
2	1	0	0	0	0
3	0	1	0	0	0
4	0	1	0	0	0

인접행렬 + dfs (양방향)

- **0번 노드로 시작하면서 방문 체크를 해야 한다!**

```
        //양방향 연결
        arr[from][to] = 1;
        arr[to][from] = 1;
}
//0번 노드로 dfs 시작
//0번 노드 방문 체크
visited[0] = 1;
dfs(0);
```

```
//방문여부 체크
int visited[10];
void dfs(int now) {
        //now에서 갈 수 있는 next를 뽑는다.
        cout << now << " ";
        //갈수 있는 노드 탐색
        for (int next = 0; next < N; next++) {
                if (arr[now][next] == 1) {
                        //이미 방문했다면 넘어가
                        if (visited[next] == 1) continue;
                        //방문 체크
                        visited[next] = 1;
                        //dfs 실행
                        dfs(next);
                        visited[next] = 0;
                }
        }
}
```

https://gist.github.com/hoconoco/6e3380644a83637bbb0f24612fe04e51

인접리스트 + dfs (양방향)

- 0번 노드로 시작하면서 방문 체크를 해야 한다!

```
    //양방향 연결
    arr[from].push_back(to);
    arr[to].push_back(from);
}
//0번 노드로 dfs 시작
//0번 노드 방문 체크
visited[0] = 1;
dfs(0);
```

```
//방문여부 체크
int visited[10];
void dfs(int now) {
    //now에서 갈 수 있는 next를 뽑는다.
    cout << now << " ";
    //갈수 있는 노드 탐색
    for (int i = 0; i < arr[now].size(); i++) {
        int next = arr[now][i];
        //이미 방문했다면 넘어가
        if (visited[next] == 1) continue;
        //방문 체크
        visited[next] = 1;
        //dfs 실행
        dfs(next);
        visited[next] = 0;
    }
}
```

https://gist.github.com/hoconoco/d587a4b8680ae301d2224d55b7f72b32

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

내일 방송에서 만나요!

삼성 청년 SW 아카데미