3 주차

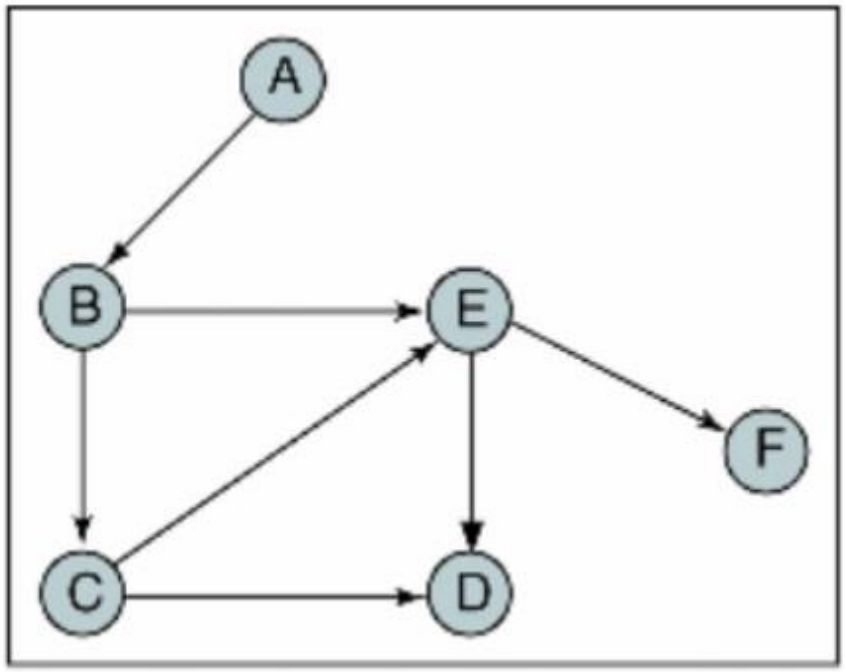
DFS & BFS

Graph

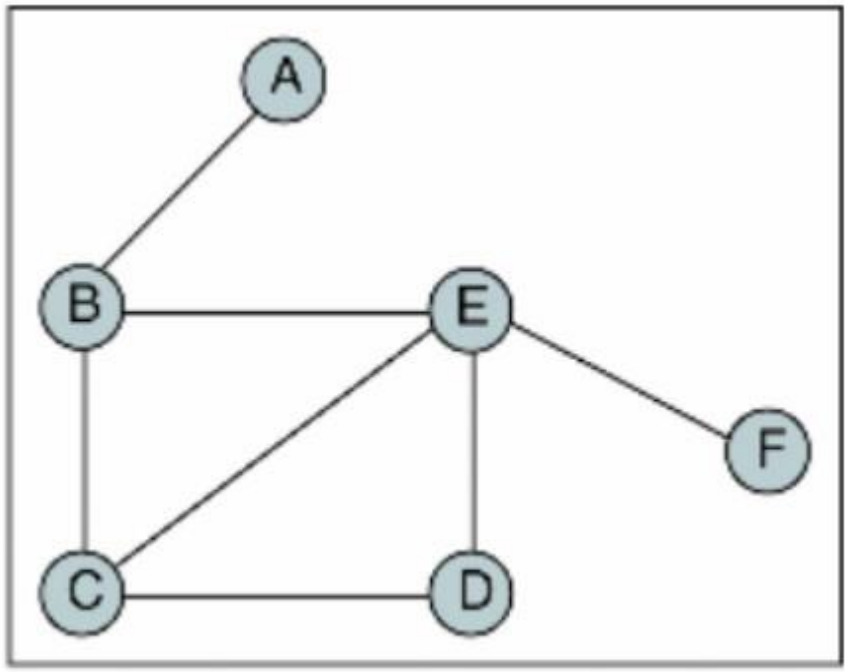
- 어떤 요소들 사이의 연결관계를 표현하기 위한 자료구조
- 노드와 노드를 연결하는 간선으로 이루어져있음
- 트리도 그래프의 일종

·지도, 도로, 강의 커리큘럼 등 실생활에서의 수많은 것들을 그래프로 모델링할 수 있음

방향 그래프, 무방향 그래프



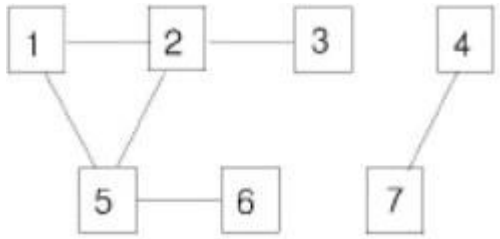
(a) Directed graph



(b) Undirected graph

연결 요소 – connected component

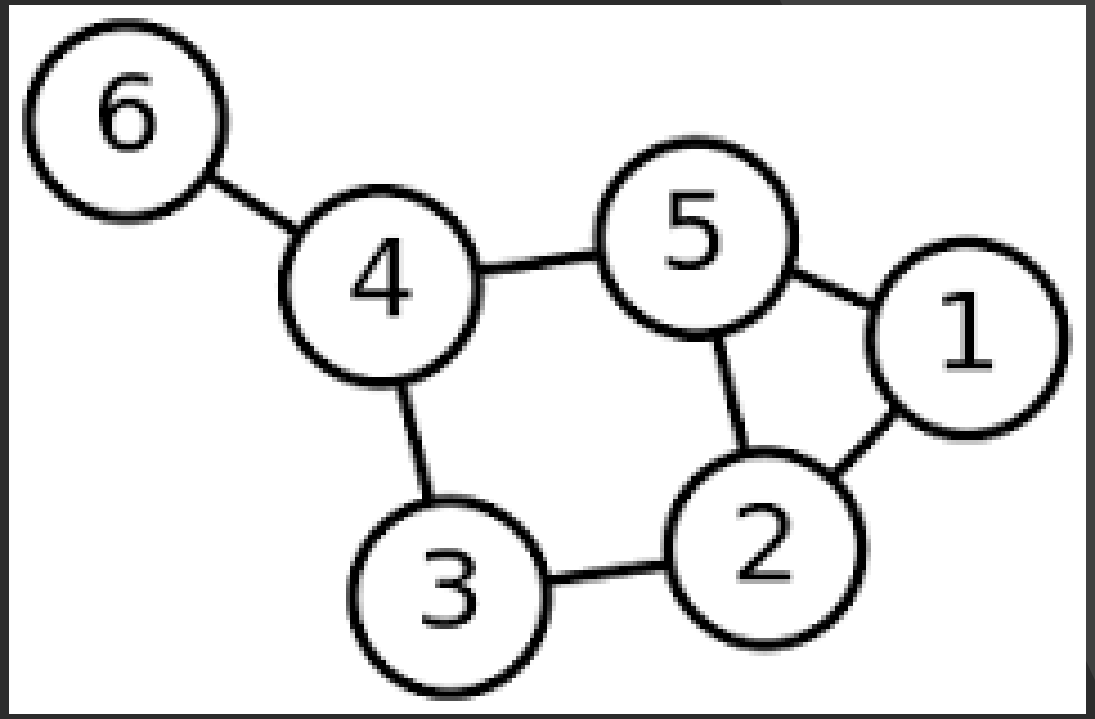
그래프는 모두 연결되어있지 않을 수도 있음!



각각의 그룹을 "연결 요소 " 라고 부름
위의 그림은 두 개의 연결요소로 이루어진 하나의 그래프

그래프

- 노드와 노드를 연결하는 간선을 하나로 모아놓은 자료구조
- 방향그래프와 무방향그래프로 나뉨
- 사이클 가능, 자체 간선도 가능
- 루트 노드, 부모-자식의 개념이 없음
- 순회 방법 : DFS, BFS
- 간선의 수가 제한되어있지 않음

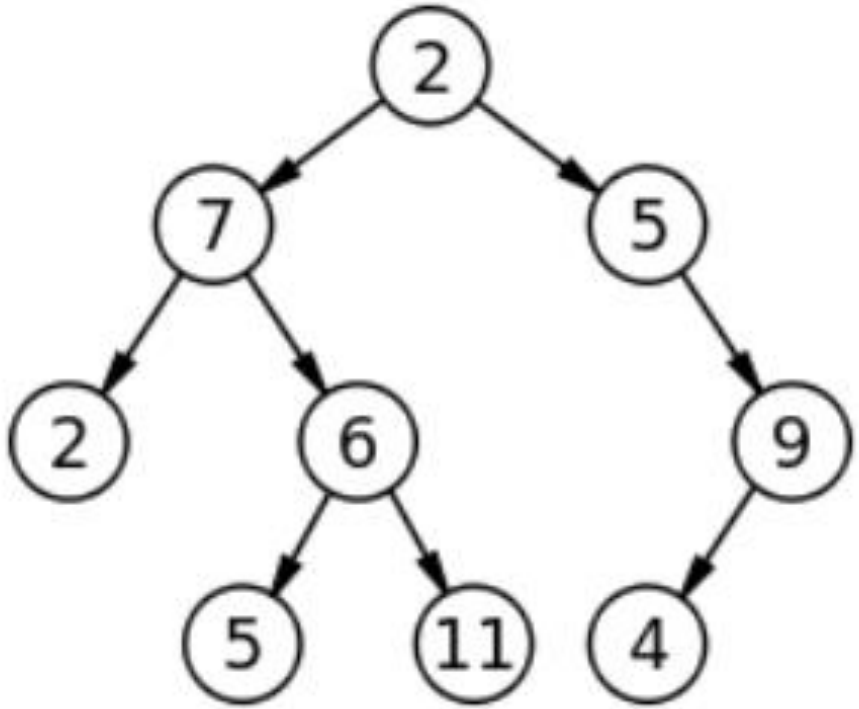


그래프 용어

- 정점, 간선
- 무향그래프/유향그래프
- 인접 정점
- 정점의 차수
- 사이클

트리

1. 연결 그래프이다. (컴포넌트가 하나이다.)
2. 방향을 무시하였을 때, 싸이클이 존재하지 않는다.
3. 트리의 간선 개수는 반드시 n – 1이다.
4. 계층관계 (부모-자식관계)
5. 순회 방법 : DFS, BFS, 전위순회, 중위순회, 후위순회



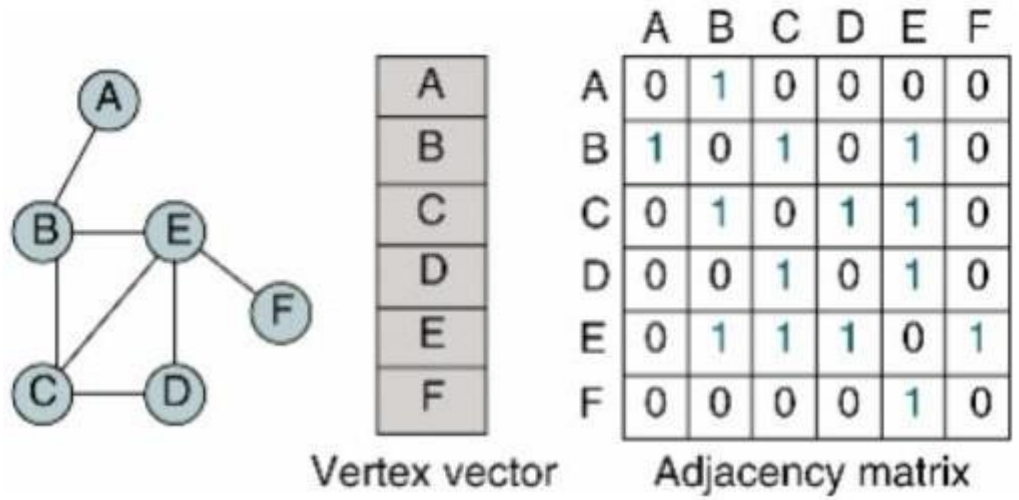
그래프의 표현

그래프를 코드로 표현하는 방법은 크게 두가지

1. 인접 행렬(이차원 배열)
2. 인접 리스트(이차원 벡터)

1. 인접 행렬(이차원 배열)

모든 두 정점간의 연결관계를 이차원 배열로표현



장점: 구현이 간단함

두 노드가 연결되어있는지 알고 싶을 때O(1)에 가능

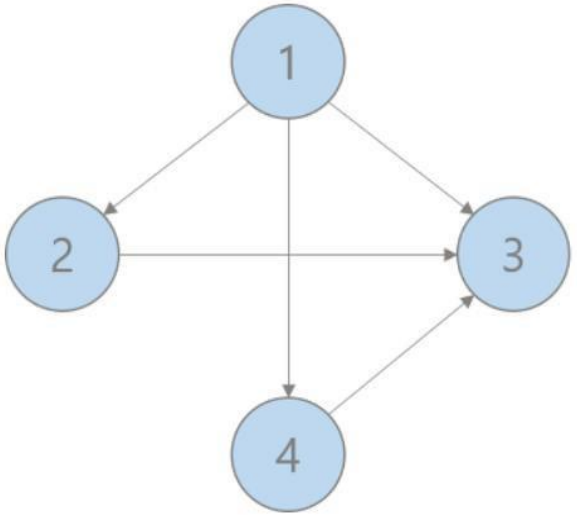
단점: 공간을 많이 차지

한 노드와 연결된 모든 노드를 알고 싶으면 O(V)

모든 노드의 연결 관계를 탐색하려면O(V^2)

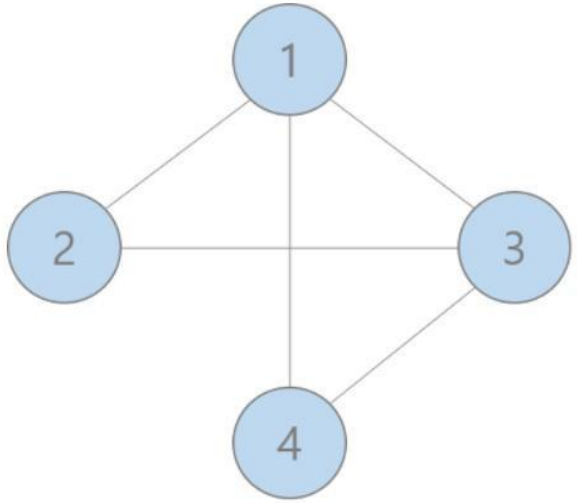
1. 인접 행렬(이차원 배열)

방향 그래프



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

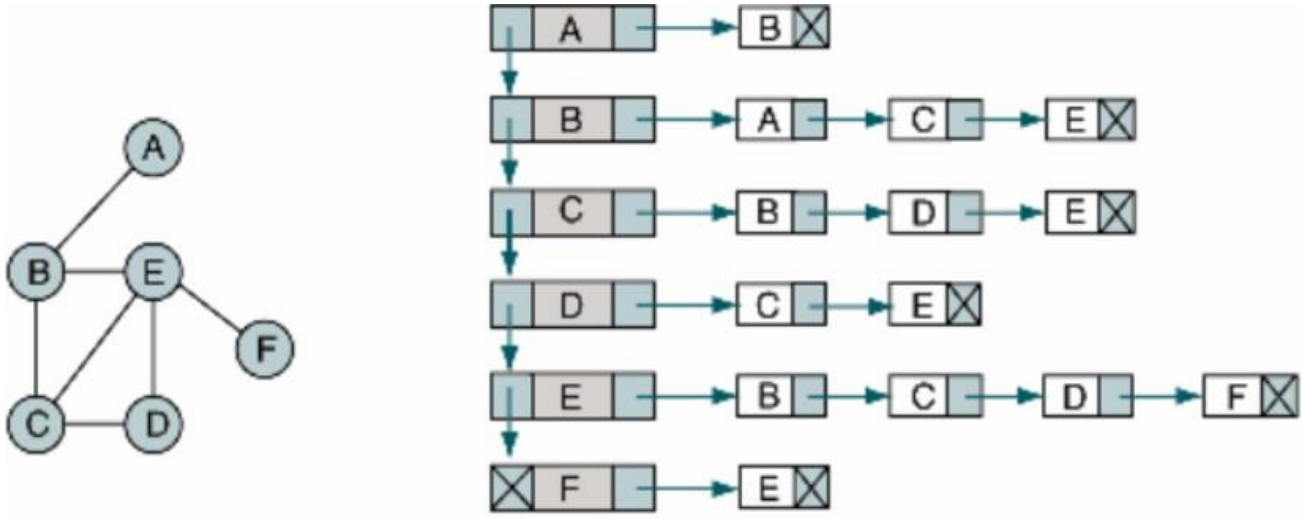
무방향 그래프



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

2. 인접 리스트(이차원 벡터)

실제로 연결된 노드들에 대한 정보만 저장

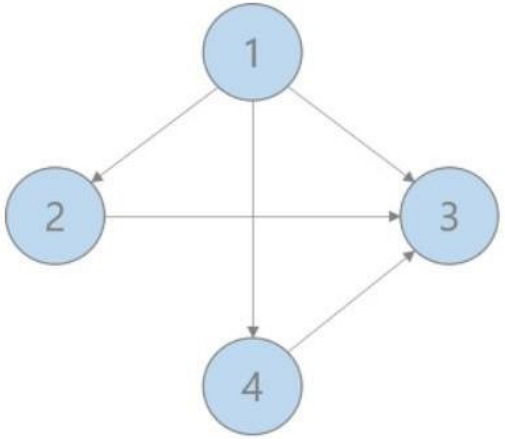


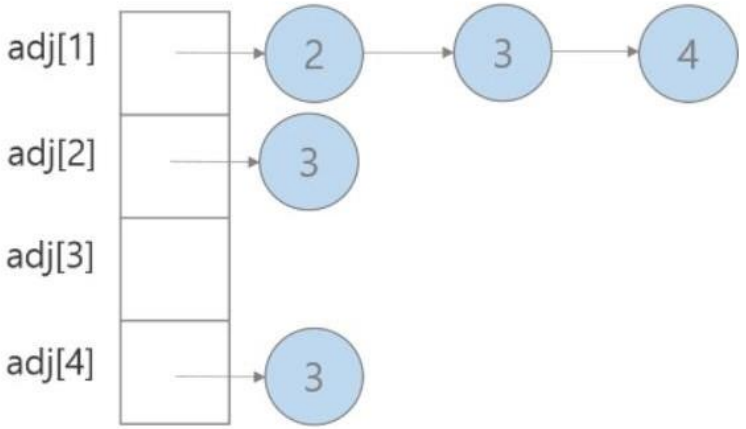
장점: 간선의 개수만큼만 메모리 차지
모든 노드의 연결 관계를 O(V)에 탐색 가능

단점: 어떤 두 노드가 연결되어있는지 찾으려면 O(V)

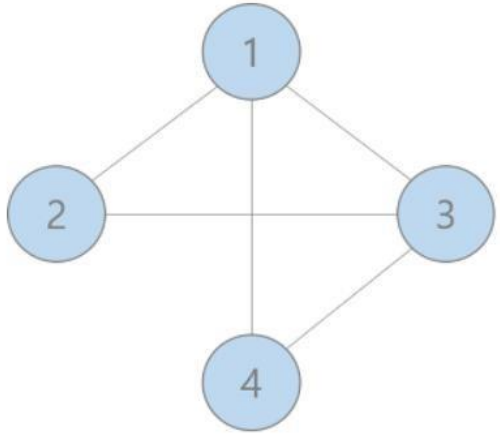
2. 인접 리스트(이차원 벡터)

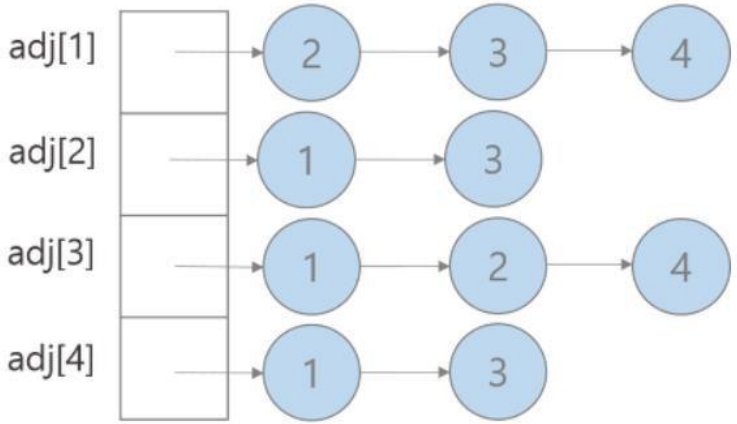
방향 그래프





무방향 그래프





그래프의 구현방법(1) - 인접리스트

- 가장 일반적인 방법
- vector, arraylist를 이용해서 구현
- 어떤 노드에 인접한 노드들을 쉽게 찾을 수 있다.
- 모든 간선의 수를 O(N+E)안에 알 수 있다.
- 간선의 존재 여부를 알기 위해선 정점의 차수만큼의 시간이 필요

```
vector<int> graph[50];

graph[2].push_back(3);
graph[3].push_back(2);
```

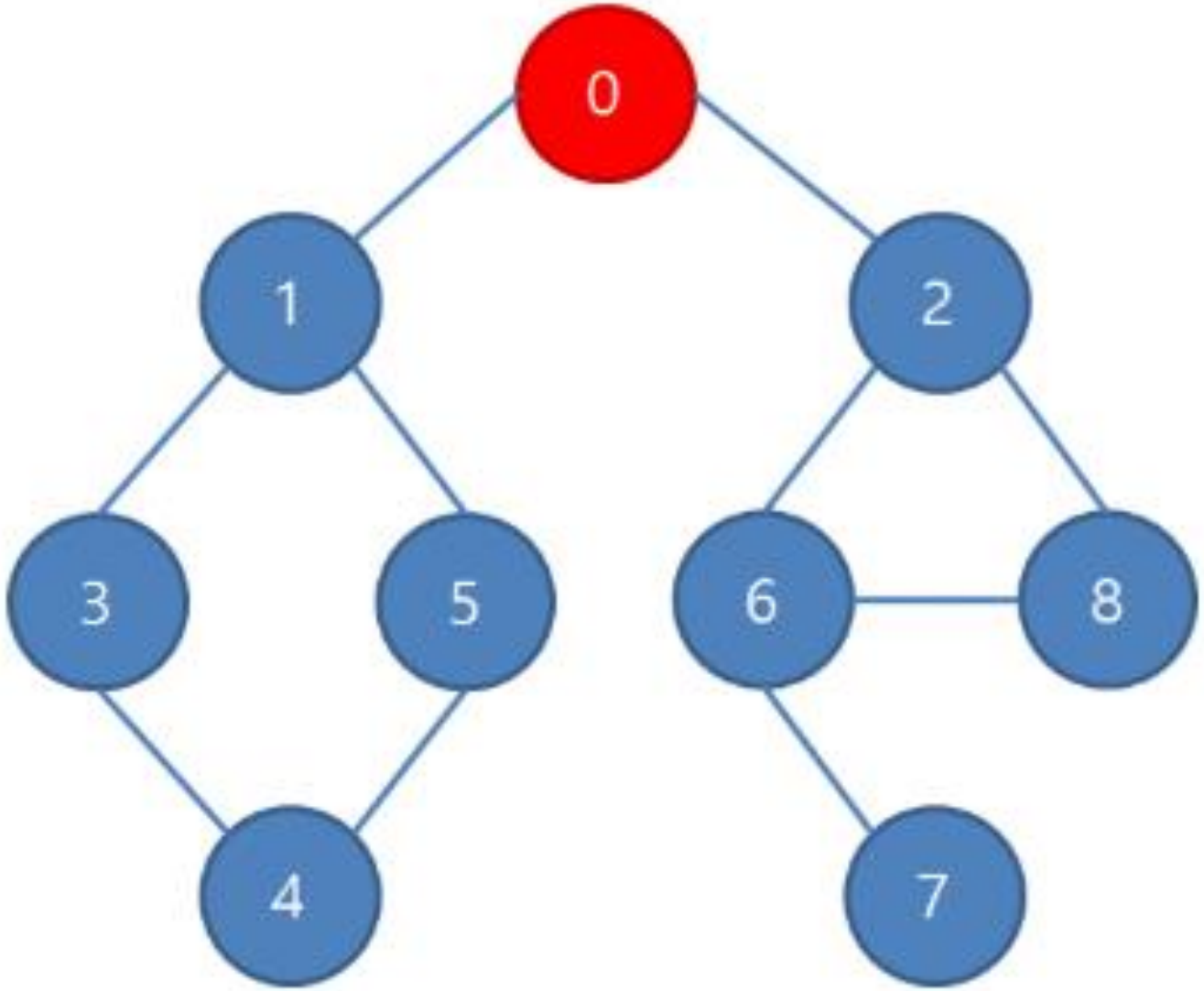
그래프의 구현방법(2) – 인접행렬

- 간선이 많은 밀집 그래프의 경우 사용
- 두 정점을 연결하는 간선의 여부를 O(1)만에 알 수 있다.
- 정점의 차수는 O(N)만에 알 수 있다.
- 어떤 노드에 인접한 노드를 찾기 위해선 모든 노드를 순회해야한다.
- 그래프에 존재하는 모든 간선의 수는 O(N^2)만에 알 수 있다.

```
bool graph[50][50];

graph[2][3] = true;
graph[3][2] = true;
```

그래프 구현!



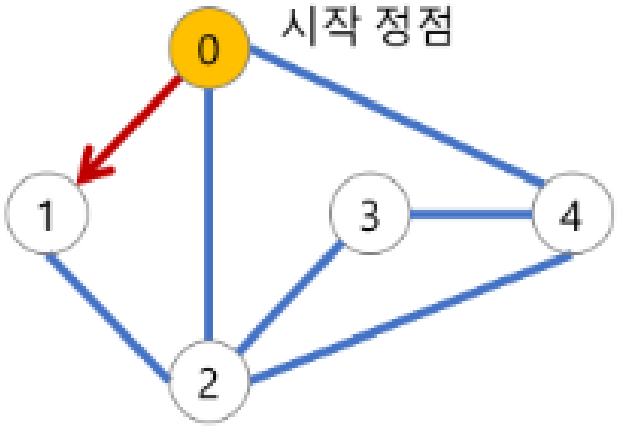
n = 8 (정점 개수)
m = 10 (간선 개수)
0 1
0 2
1 3
1 5
3 4
4 5
2 6
2 8
6 8
6 7

DFS(깊이 우선 탐색)

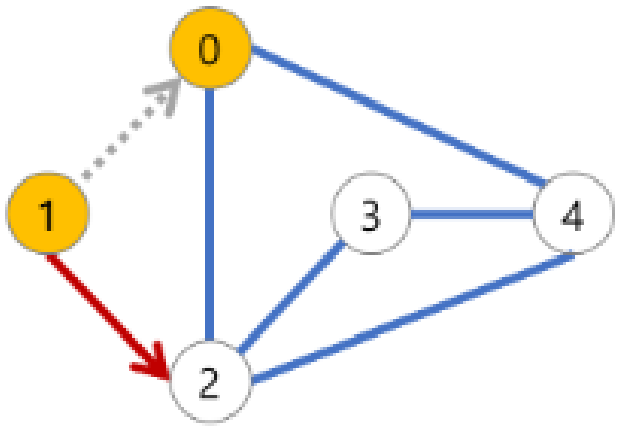
- 루트 노드(혹은 다른 임의의 노드)에서 시작해서 다음 분기(branch)로 넘어가기 전에 해당 분기를 완벽하게 탐색하는 방법
- 재귀를 이용해서 구현
- 모든 노드를 방문하고자 하는 경우 사용
- 시간 복잡도 : O(N+E)

DFS 동작

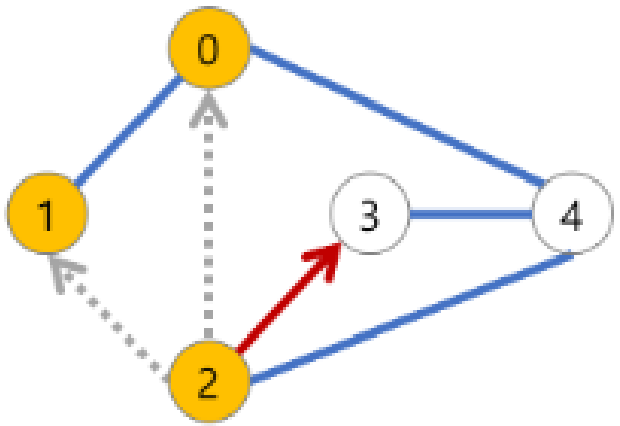
(1) 정점 1 방문



(2) 정점 2 방문

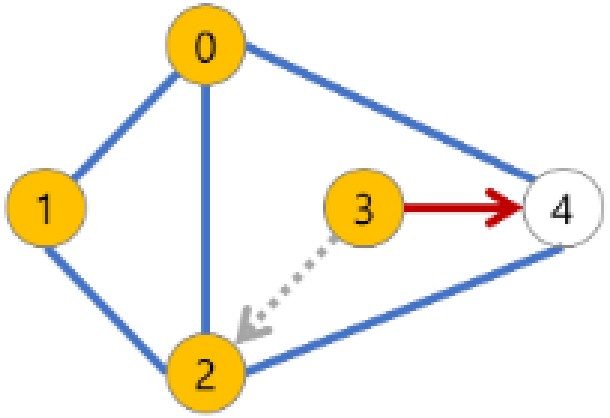


(3) 정점 3 방문

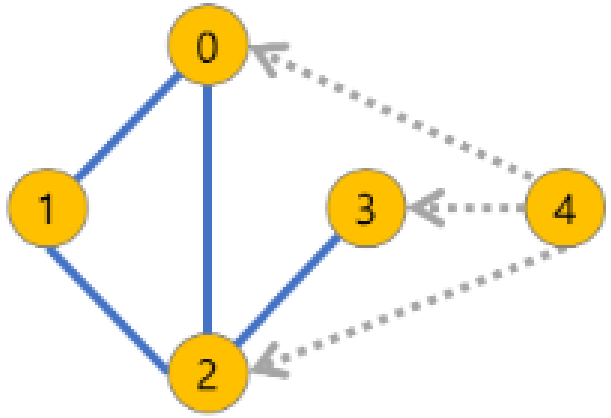


DFS 동작

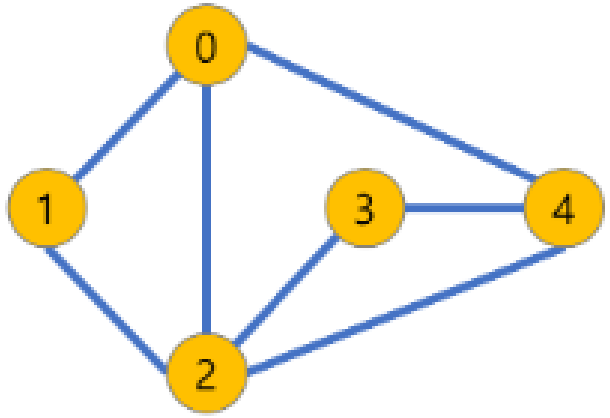
(4) 정점 4 방문



(5) 정점 3으로 backtracking
(다시 돌아와서 탐색하지 않은 정점이 있는지 확인)

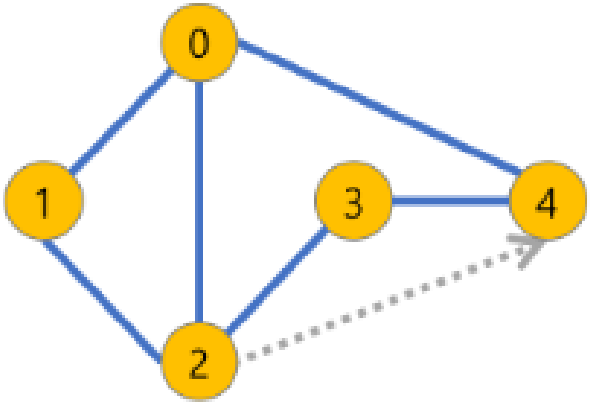


(6) 정점 2로 backtracking

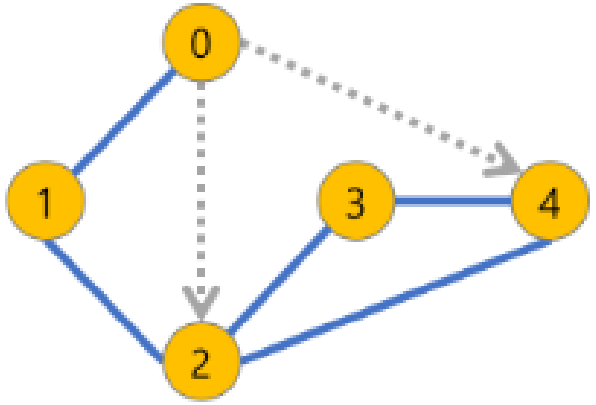


DFS 동작

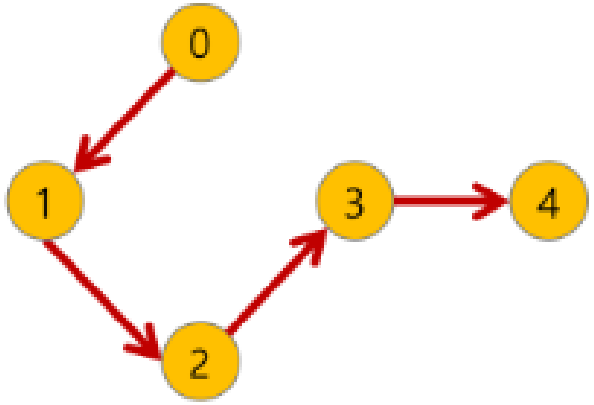
(7) 정점 1로 backtracking



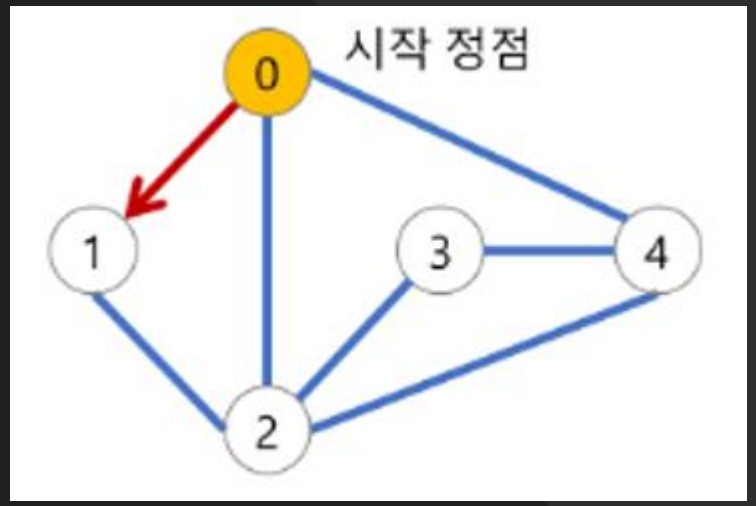
(8) 정점 0으로 backtracking (탐색 종료)



(9) 탐색 결과 (방문 순서: 0,1,2,3,4)



```
1    void dfs(int curr){
2        visited[curr] = true;
3        cout << "node " << curr << " visited" << endl;
4        for(int next: adj[curr])
5            if(!visited[next]) dfs(next);
6    }
```

adj[0] : 1 – 2 – 4
adj[1] : 0 - 2
adj[2] : 0 – 1 - 3
adj[3] : 2 - 4
adj[4] : 0 - 2

백트래킹

- 어떤 노드의 유망성 점검 후, 유망하지 않으면 그 노드의 부모노드로 되돌아 간 후(백트래킹) 다른 자손노드를 검색.

- 유망하다는 것이 무슨 의미인가? (답이 될 가능성이 없다.)

백트래킹 – 1987 알파벳

문제

세로 R칸, 가로 C칸으로 된 표 모양의 보드가 있다. 보드의 각 칸에는 대문자 알파벳이 하나씩 적혀 있고, 좌측 상단 칸 (1행 1열) 에는 말이 놓여 있다.

말은 상하좌우로 인접한 네 칸 중의 한 칸으로 이동할 수 있는데, 새로 이동한 칸에 적혀 있는 알파벳은 지금까지 지나온 모든 칸에 적혀 있는 알파벳과는 달라야 한다. 즉, 같은 알파벳이 적힌 칸을 두 번 지날 수 없다.

좌측 상단에서 시작해서, 말이 최대한 몇 칸을 지날 수 있는지를 구하는 프로그램을 작성하시오. 말이 지나는 칸은 좌측 상단의 칸도 포함된다.

입력

첫째 줄에 R과 C가 빈칸을 사이에 두고 주어진다. ($1 \le R,C \le 20$) 둘째 줄부터 R개의 줄에 걸쳐서 보드에 적혀 있는 C개의 대문자 알파벳들이 빈칸 없이 주어진다.

출력

첫째 줄에 말이 지날 수 있는 최대의 칸 수를 출력한다.

예제 입력 1 복사

```
2 4
CAAB
ADCB
```

예제 출력 1 복사

```
3
```

1987_알파벳

visit에 무엇을 저장할까?

```
3 5
A B A D E
C F E F A
D E F G H
```

1987_알파벳

3 5
A B A D E
C F E F A
D E F G H

```
void go(int cury, int curx, int cnt) {
    int cur_num = arr[cury][curx];
    visit[cur_num] = true;
    ans = max(ans, cnt);
    for (int i = 0; i < 4; i++) {
        int ny = cury + my[i];
        int nx = curx + mx[i];
        if (ny < 1 || nx < 1 || ny > r || nx > c)
            continue;
        int nxt_num = arr[ny][nx];
        if (!visit[nxt_num]) {
            go(ny, nx, cnt + 1);
        }
    }
    visit[cur_num] = false;
}
```

백트래킹 – 9663 N-Queen

문제

N-Queen 문제는 크기가 N × N인 체스판 위에 퀸 N개를 서로 공격할 수 없게 놓는 문제이다.

N이 주어졌을 때, 퀸을 놓는 방법의 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N이 주어진다. (1 ≤ N < 15)

출력

첫째 줄에 퀸 N개를 서로 공격할 수 없게 놓는 경우의 수를 출력한다.

예제 입력 1 복사

```
8
```

예제 출력 1 복사

```
92
```

9663_N-Queen