Chapter 01.
그래프 탐색
– BFS, DFS

Clip 01 | 그래프 이론
그래프 용어
그래프의 표현 방법 (인접행렬, 인접 리스트)

Clip 02 | [1260] DFS와 BFS
탐색의 기본 구현

Clip 03 | [11724] 연결 요소의 개수
연결 요소 (Connected Component)
집합의 개수 카운트

Clip 04 | [2606] 바이러스
문제를 그래프로 치환하는 방법

Clip 05 | [2573] 빙산
객체의 상태관리

Clip 06 | [1941] 소문난 칠공주
재귀를 통한 순열의 생성
탐색과 유효성 검증

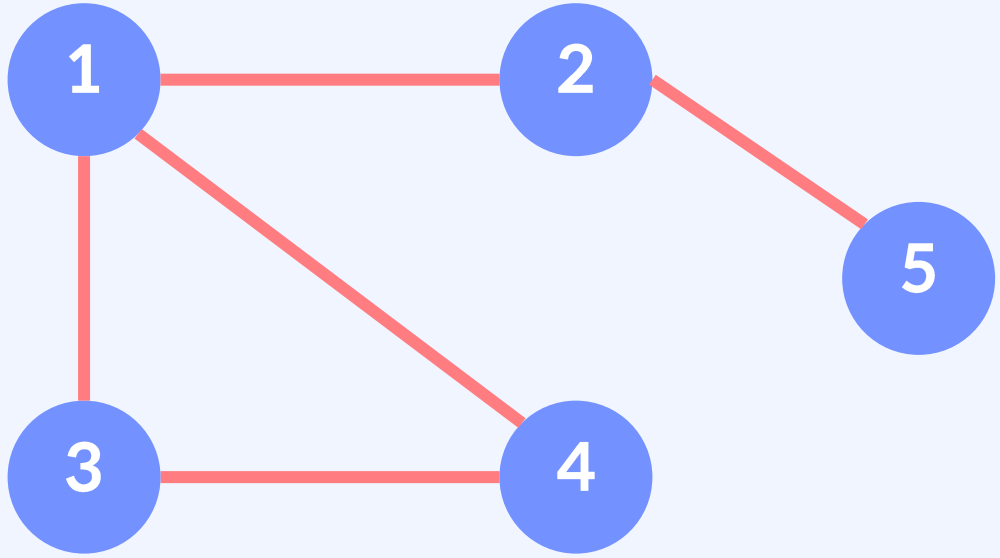
Ch01. 그래프 탐색 BFS, DFS

1 그래프 이론

그래프?

정의

- 정점과 간선의 집합으로 구성되는 자료구조



그래프?

용어

- 방향그래프
  - 간선에 방향이 있는 그래프
  - A→B 로 향하는 간선과, B→A로 향하는 간선이 서로 다를 수 있다

- 무방향그래프
  - 간선에 방향이 없는 그래프
    - A-B를 연결하는 간선이 동일한 간선이다

그래프?

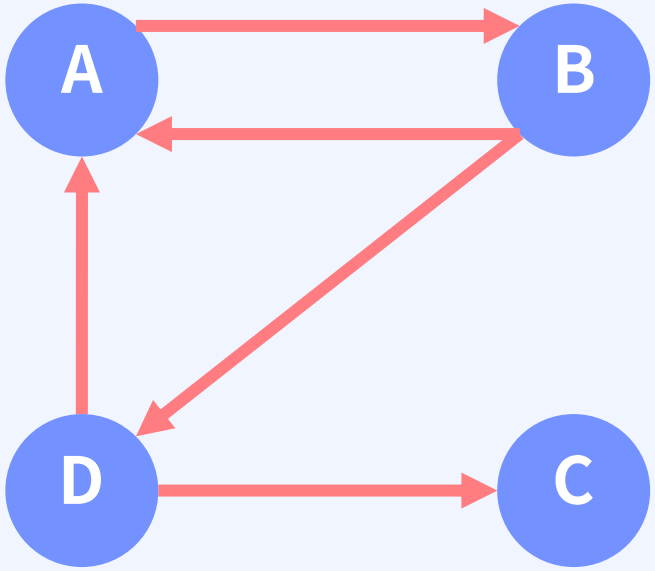
용어

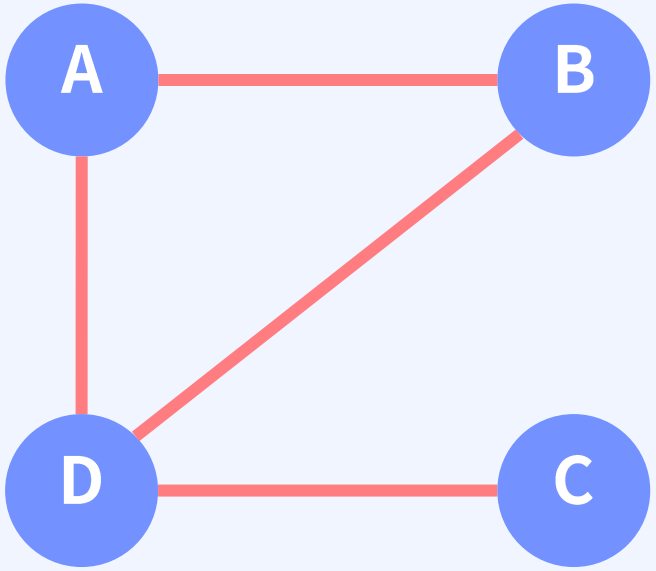
- 정점의 차수(Degree)
  - 정점에 연결된 간선의 수
  - 무방향 그래프:
    정점의 차수와 간선의 수가 같음
  - 방향그래프:
    진입차수(in-degree) 진출차수(out-degree) 로 나눠짐

그래프?

용어

- 정점의 차수(Degree)



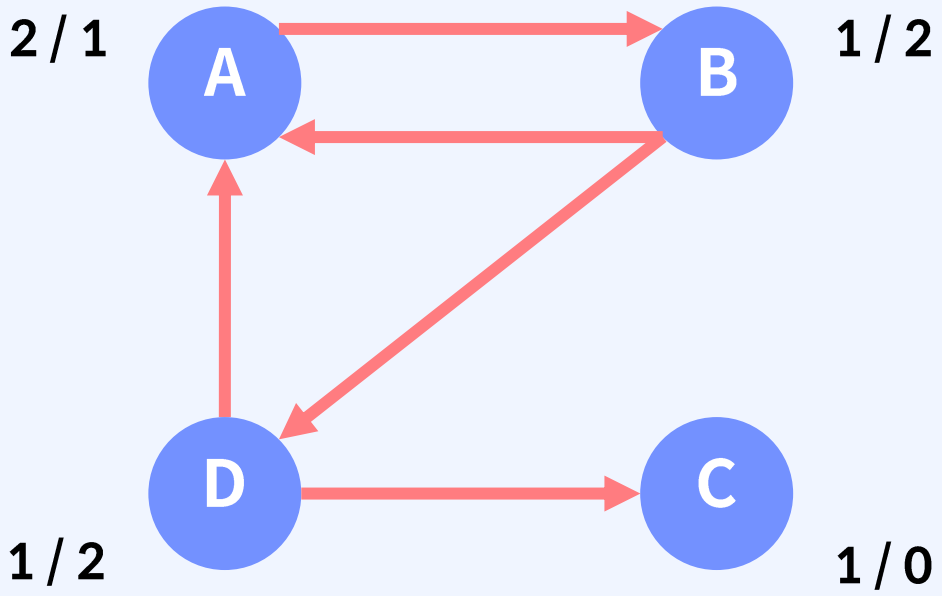


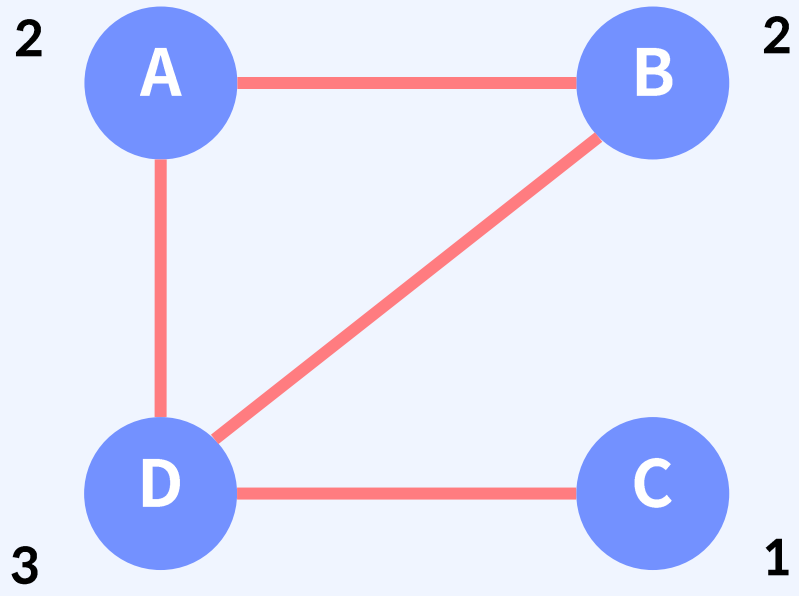
각 정점 별 차수를 계산해보자

그래프?

용어

- 정점의 차수(Degree)





각 정점 별 차수를 계산해보자

그래프?

용어

- 경로
  - 정점을 연결된 간선을 따라
    탐색하는 순서대로 나타낸 것

- 사이클
  - 간선의 경로 중 시작 정점과 끝 정점이
    동일한 경로
  - 사이클이 없는 그래프: 트리

그래프?

용어

- 가중치 그래프
  - 간선에 가중치 혹은 비용이 할당된 그래프
  - 연결된 정점들 간 탐색에 드는 비용이나, 연결강도 등을 표현함
  - 구현 시 객체(클래스)로 묶어서 표현하면 가독성 좋은 코드를 작성할 수 있다

그래프?

용어

- 가중치 그래프

```
class Node {
    int node;
    int cost;

    public Node(int node, int dist) {
        this.node = node;
        this.dist = dist;
    }
}
```

그래프에 주어지는 정보가
많아질수록

객체로 묶어서 사용하면
편리하다

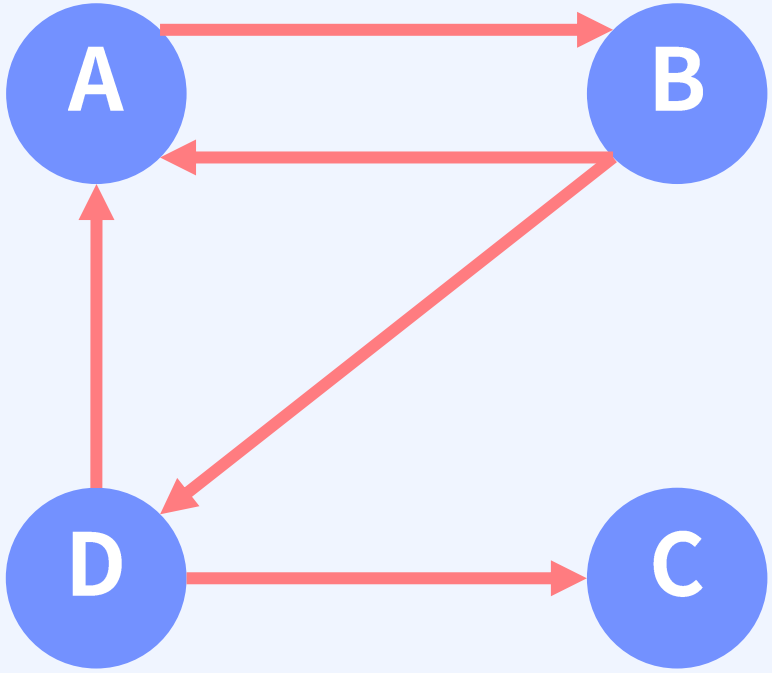
그래프 표현 방법

1.인접 행렬

- 일반적으로 2차원 배열을 이용해서 표현한다
- adj[행][열] = 연결여부(or 가중치)
- 공간 복잡도
  - V개의 정점이 있다면, V*V 만큼의 공간을 사용한다
- 시간 복잡도
  - 연결관계(가중치) 조회/저장: O(1)
  - 정점에 연결된 모든 간선 조회: O(V)

그래프 표현 방법

1.인접 행렬



	A	B	C	D
A	0	1	0	0
B	1	0	0	1
C	0	0	0	0
D	1	0	1	0

그래프의 정점간 연결 관계를 행렬로 표현

그래프 표현 방법

1. 인접 행렬

그래프의 입력 정보 예시

- 첫번째 줄: {정점의 수} {간선의 수}
- 두번째 줄 이후:
  - 연결된 간선의 정보

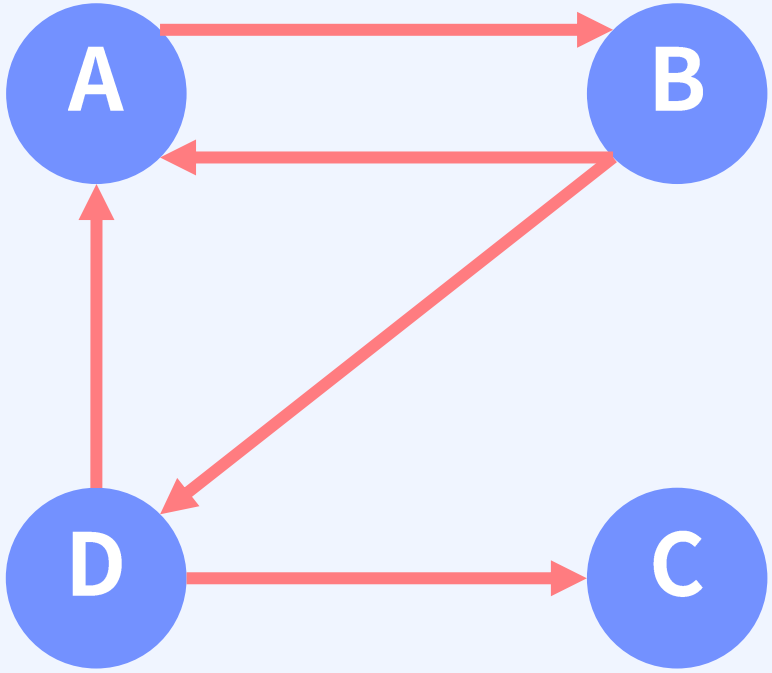
```
5 3
1 2
2 3
3 1
```

```java
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    // 정점의 개수와 간선의 개수 입력
    int v = sc.nextInt();
    int e = sc.nextInt();
    int[][] adj = new int[v+1][v+1];

    // 연결정보 입력
     for(int i = 0; i < e; i++) {
       int src = sc.nextInt();
       int dst = sc.nextInt();
       adj[src][dst] = 1;
    }
}
```

그래프 표현 방법

2. 인접 리스트



A	B	
B	A	D
C		
D	A	C

그래프의 정점간 연결 관계를 리스트로 표현

그래프 표현 방법

2. 인접 리스트

- 간선의 정보를 기반으로 저장하는 방법
- 알고리즘 문제 풀이에서는 구현의 편의를 위해 List<Node>[] graph 형태로 선언한다
  - 정점의 정보: 배열
  - 간선의 정보: 리스트
- graph[행] = new ArrayList<Node>()
  - 정점별로 간선의 정보를 저장하는 리스트를 만든다

그래프 표현 방법

2. 인접 리스트

- 공간 복잡도
  - V개의 정점, E개의 간선이 있다면
    V + E 만큼의 공간을 사용한다
- 시간 복잡도
  - 연결관계(가중치) 조회/저장: O(Outdegree(V))
  - 정점에 연결된 모든 간선 조회: O(Outdegree(V))

그래프 표현 방법

2. 인접 리스트

그래프의 입력 정보 예시

- 첫번째 줄: {정점의 수} {간선의 수}
- 두번째 줄 이후:
  - 연결된 간선의 정보

```
5 3
1 2
2 3
3 1
```

```java
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    // 정점의 개수와 간선의 개수 입력
    int v = sc.nextInt();
    int e = sc.nextInt();
    // 리스트 선언 및 인스턴스화
    List<Integer> graph[] = new List[v+1];
    for(int i = 1; i <= v; i++) {
        graph[i] = new ArrayList<>();
    }
    for(int i = 0; i < e; i++) {
        int src = sc.nextInt();
        int dst = sc.nextInt();
        graph[src].add(dst);
    }
}
```

그래프 표현 방법

3. 인접 행렬 vs 인접 리스트

정점의 수: V | 간선의 수: E

인접 행렬

- 공간 복잡도: V * V
- 시간 복잡도
  - 단일 정점 조회/저장: O(1)
  - 정점 모든 간선 조회: O(V)

인접 리스트

- 공간 복잡도: V + E
- 시간 복잡도
  - 단일 정점 조회/저장: O(Outdegree(V))
  - 정점 모든 간선 조회: O(Outdegree(V))

Ch01. 그래프 탐색 BFS, DFS

2 [1260] DFS와 BFS

BOJ1260: DFS와 BFS

문제 요약

- 시작 정점 (1<=V<=N)에서 DFS와 BFS 탐색을 통한 경로 출력
- 여러 개의 간선이 있을 경우 오름차순으로 방문
- 1 <= N <= 1,000 | 1 <= M <= 10,000

BOJ1260: DFS와 BFS

문제 분석

- N의 사이즈가 작아, 인접행렬로 구현해도 무방하다
- DFS는 Stack 혹은 재귀함수로 구현할 수 있다
- BFS는 Queue를 이용해 구현할 수 있다

BOJ1260: DFS와 BFS

예제 1

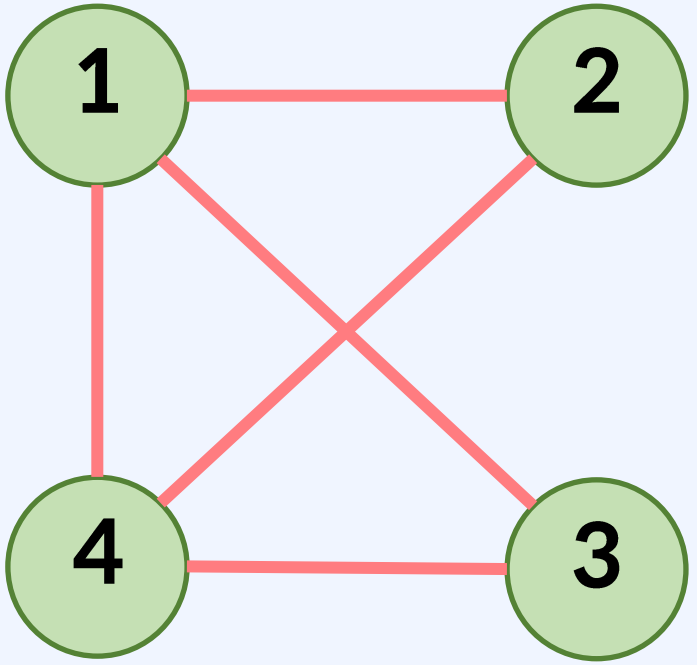
입력 데이터
4 5 1 1 2 1 3 1 4 2 4 3 4

출력 데이터
1 2 4 3 1 2 3 4

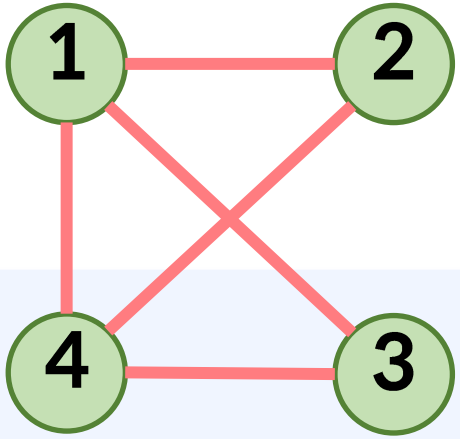
BOJ1260: DFS와 BFS

예제 1

입력 데이터
4 5 1 1 2 1 3 1 4 2 4 3 4



BOJ1260: DFS와 BFS



예제 1

입력 데이터
4 5 1 1 2 1 3 1 4 2 4 3 4

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	0	0	0	0

DFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점의 행으로 이동
4. (1) ~ (3) 의 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	0	0	0

DFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점의 행으로 이동
4. (1) ~ (3) 의 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	0	0	0

DFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점의 행으로 이동
4. (1) ~ (3) 의 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	0	0

DFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점의 행으로 이동
4. (1) ~ (3) 의 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	0	0

DFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점의 행으로 이동
4. (1) ~ (3) 의 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	0	0

DFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점의 행으로 이동
4. (1) ~ (3) 의 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	0	0

DFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점의 행으로 이동
4. (1) ~ (3) 의 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	0	1

DFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점의 행으로 이동
4. (1) ~ (3) 의 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	0	1

DFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점의 행으로 이동
4. (1) ~ (3) 의 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	0	1

DFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점의 행으로 이동
4. (1) ~ (3) 의 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	0	1

DFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점의 행으로 이동
4. (1) ~ (3) 의 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	1	1

DFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점의 행으로 이동
4. (1) ~ (3) 의 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	0	0	0

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	0	0	0

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	0	0

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	0	0

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복

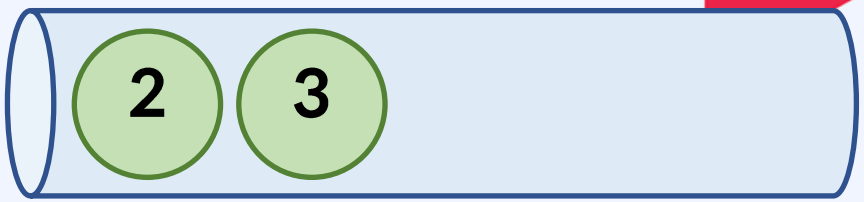
	1	2	3	4
1	0	1	1	1
2	1	0	0	1
3	1	0	0	1
4	1	1	1	0

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	1	0

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복



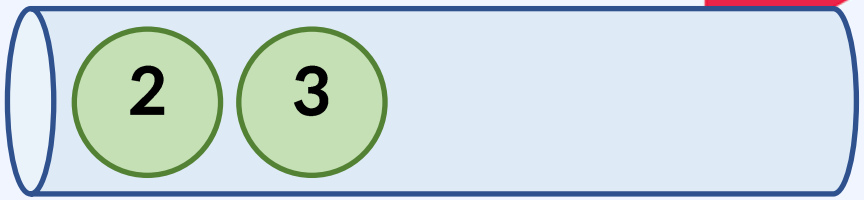
	1	2	3	4
1	0	1	1	1
2	1	0	0	1
3	1	0	0	1
4	1	1	1	0

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	1	0

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복



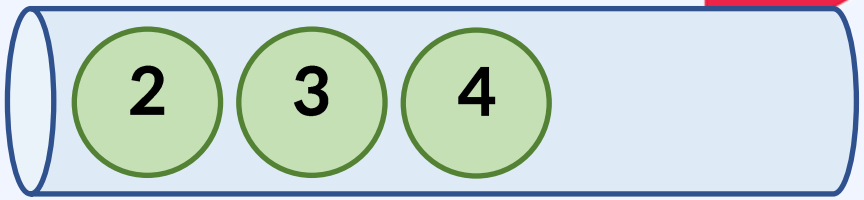
	1	2	3	4
1	0	1	1	1
2	1	0	0	1
3	1	0	0	1
4	1	1	1	0

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	1	1

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복



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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	1	1

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복

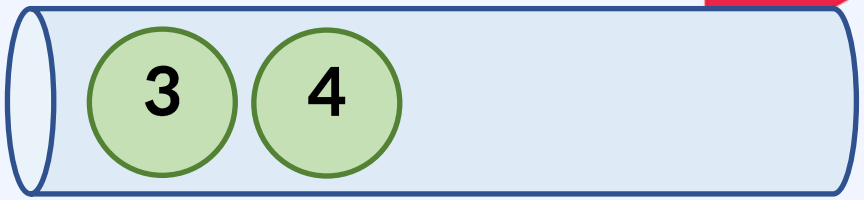
	1	2	3	4
1	0	1	1	1
2	1	0	0	1
3	1	0	0	1
4	1	1	1	0

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	1	1

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복



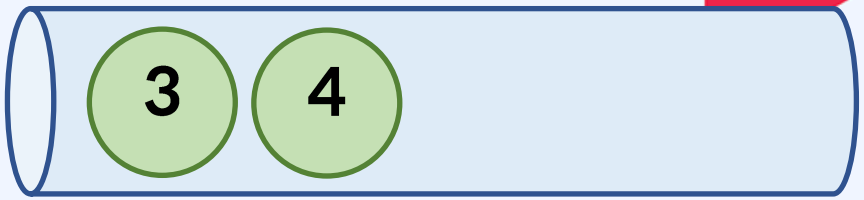
	1	2	3	4
1	0	1	1	1
2	1	0	0	1
3	1	0	0	1
4	1	1	1	0

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	1	1

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복



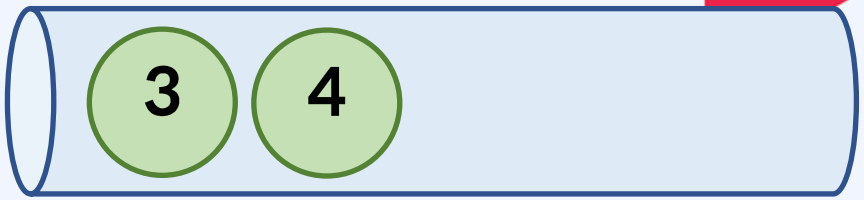
	1	2	3	4
1	0	1	1	1
2	1	0	0	1
3	1	0	0	1
4	1	1	1	0

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	1	1

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복



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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	1	1

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	1	1

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	1	1

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	1	1

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	1	1

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	1	1

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	1	1

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복

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

BOJ1260: DFS와 BFS

	1	2	3	4
visited[]	1	1	1	1

BFS

1. 선택한 정점의 행을 순회
2. 아직 방문하지 않은 정점 중에 연결된 정점을 선택
3. 선택된 정점을 큐에 넣고 방문 체크 표시
4. 순회가 끝나면 큐의 데이터를 뽑아 (1) ~ (3) 과정을 반복

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

BOJ1260: DFS와 BFS

정답 출력

DFS

처음 방문하는 정점이
재귀에 진입할 때 마다

BFS

처음 방문하는 정점이
큐에서 빠질 때 마다

Ch01. 그래프 탐색 BFS, DFS

3 [11724] 연결 요소의 개수

BOJ11724: 연결 요소의 개수

문제 요약

- 연결 요소 (Connected Component)의 개수를 구하기
- 서로 연결되지 않은 독립적인 그래프 집합이 몇 개인지 카운트
- A와 B가 연결된 정점?
  - A 에서 DFS / BFS 탐색 시 B에 도달
  - B 에서 BFS / DFS 탐색 시 A에 도달

BOJ11724: 연결 요소의 개수

문제 분석

- 가중치가 없는 양방향 그래프
- 모든 정점을 돌면서 방문한 적 없는 정점은 새로운 연결 요소에 포함되는 노드
  - DFS / BFS 를 진행하고 연결 요소의 개수를 증가시킨다.
- 이미 확인한 연결 요소에 포함된 정점은 visited가 true 이다
  - 무시해도 연결 요소의 개수에 영향을 주지 않는다

BOJ11724: 연결 요소의 개수

예제 1

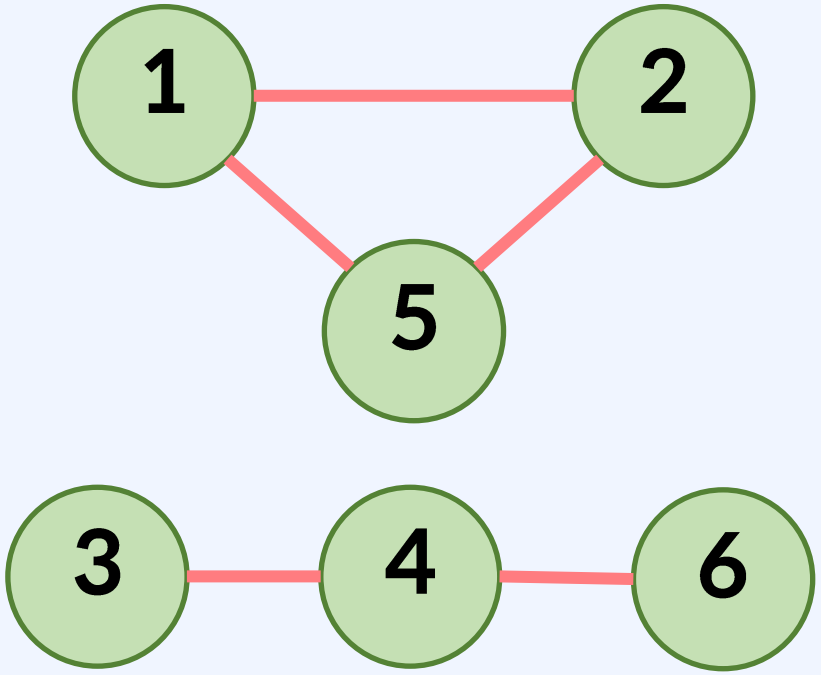
입력 데이터
6 5 1 2 2 5 5 1 3 4 4 6

출력 데이터
2

BOJ11724: 연결 요소의 개수

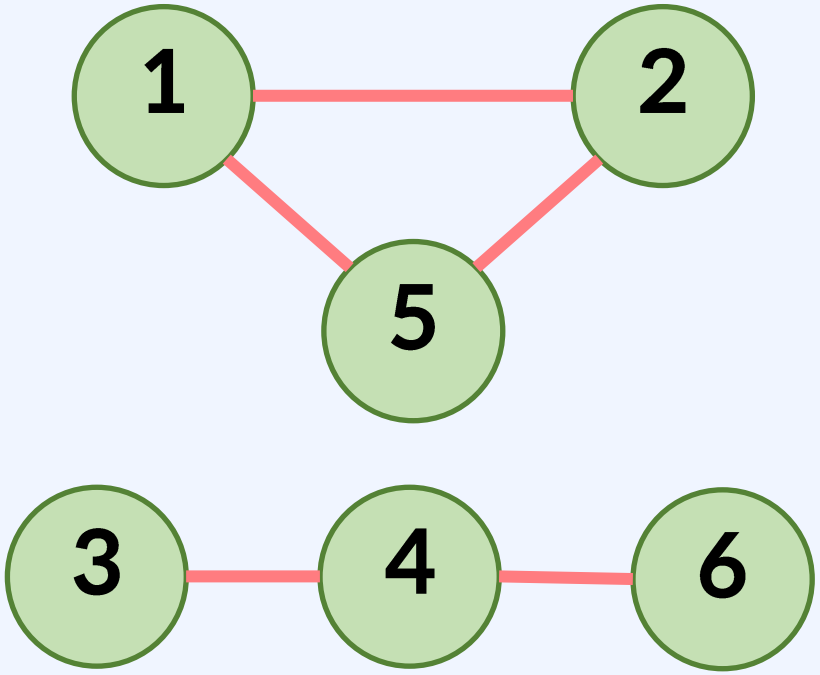
예제 1

입력 데이터
6 5 1 2 2 5 5 1 3 4 4 6



BOJ11724: 연결 요소의 개수

예제 1



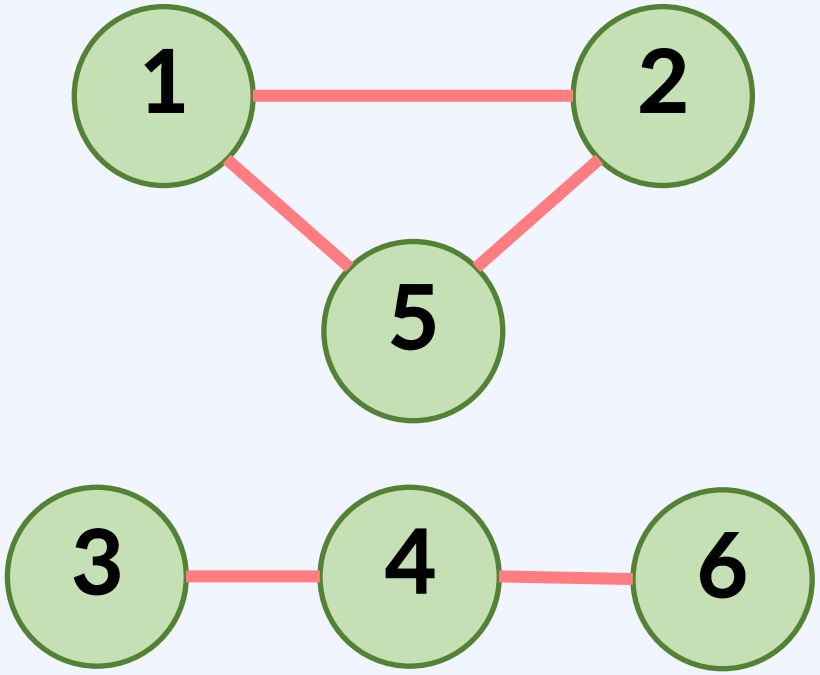
	1	2	3	4	5	6
visited[]	1	1	0	0	1	0

(1) 정점에서 DFS / BFS 수행

- 1 / 2 / 5 방문하고 종료

BOJ11724: 연결 요소의 개수

예제 1



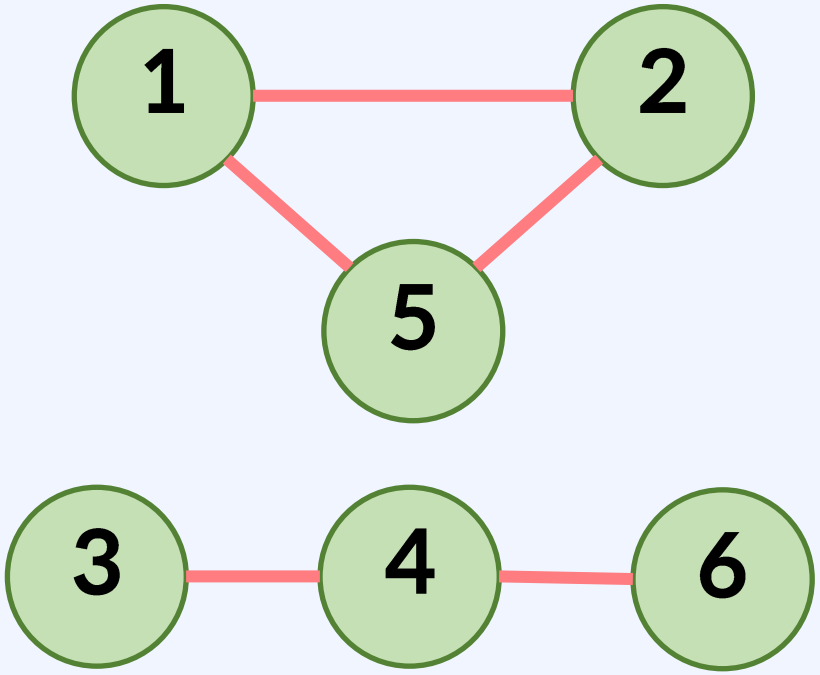
	1	2	3	4	5	6
visited[]	1	1	0	0	1	0

(2) 정점은 방문 이력이 있으므로 무시

(이미 연결요소 확인이 완료되었다)

BOJ11724: 연결 요소의 개수

예제 1



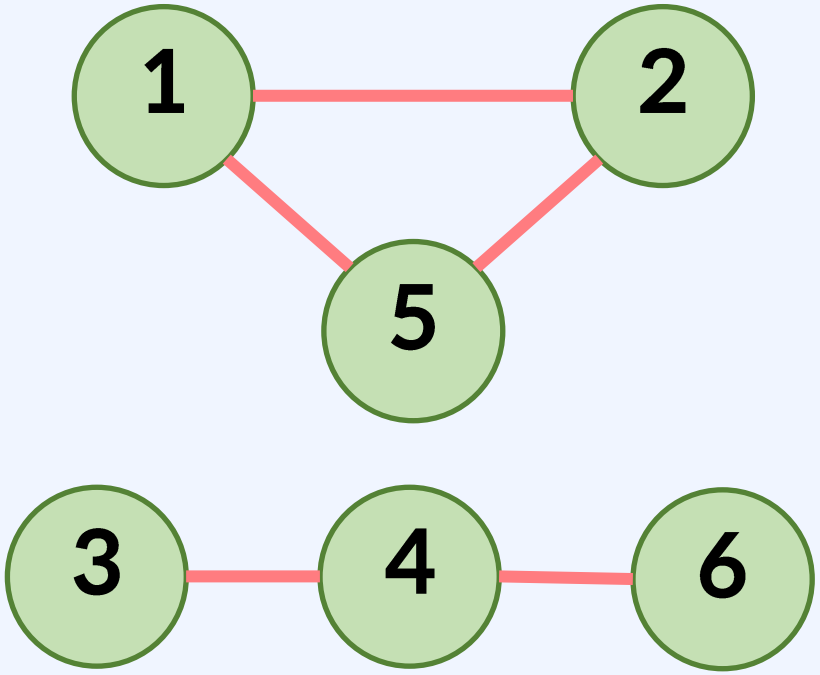
	1	2	3	4	5	6
visited[]	1	1	1	1	1	1

(3) 정점에서 DFS / BFS 수행

- 3 / 4 / 6 방문하고 종료

BOJ11724: 연결 요소의 개수

예제 1



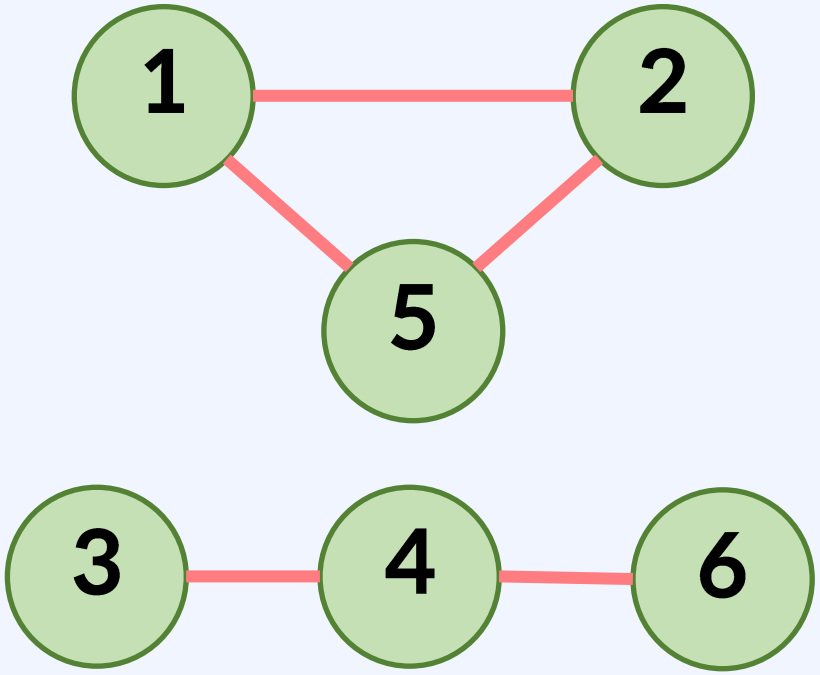
	1	2	3	4	5	6
visited[]	1	1	1	1	1	1

(4) 정점은 방문 이력이 있으므로 무시

(이미 연결요소 확인이 완료되었다)

BOJ11724: 연결 요소의 개수

예제 1



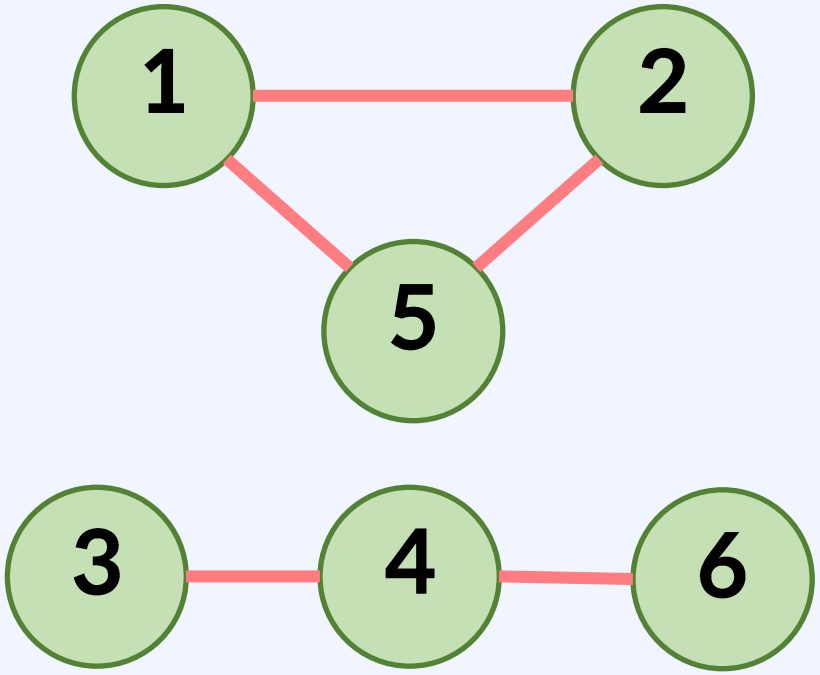
	1	2	3	4	5	6
visited[]	1	1	1	1	1	1

(5) 정점은 방문 이력이 있으므로 무시

(이미 연결요소 확인이 완료되었다)

BOJ11724: 연결 요소의 개수

예제 1



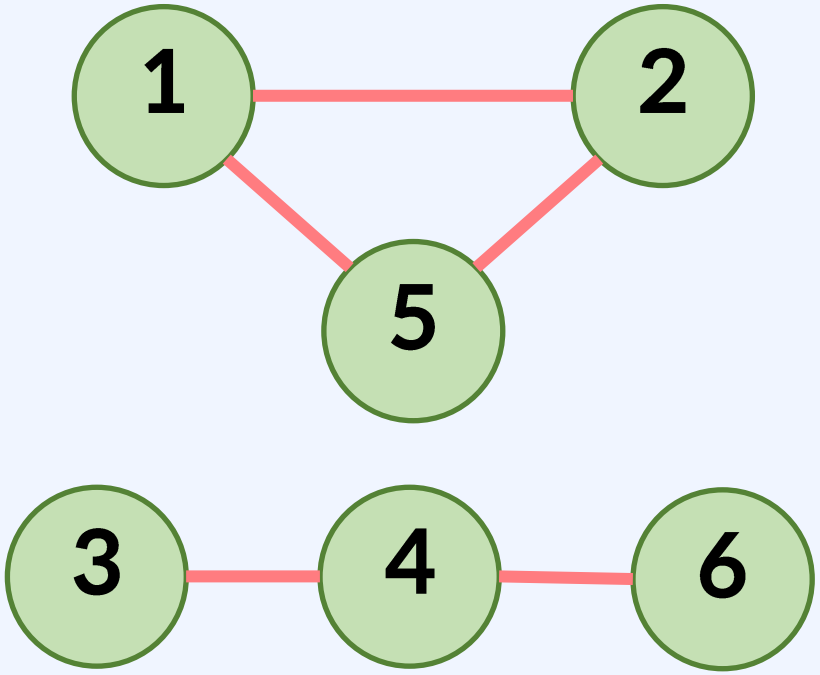
	1	2	3	4	5	6
visited[]	1	1	1	1	1	1

(6) 정점은 방문 이력이 있으므로 무시

(이미 연결요소 확인이 완료되었다)

BOJ11724: 연결 요소의 개수

예제 1



	1	2	3	4	5	6
visited[]	1	1	1	1	1	1

DFS / BFS 탐색은 총 2회 진행
→ 2 출력

BOJ11724: 연결 요소의 개수

예제 2

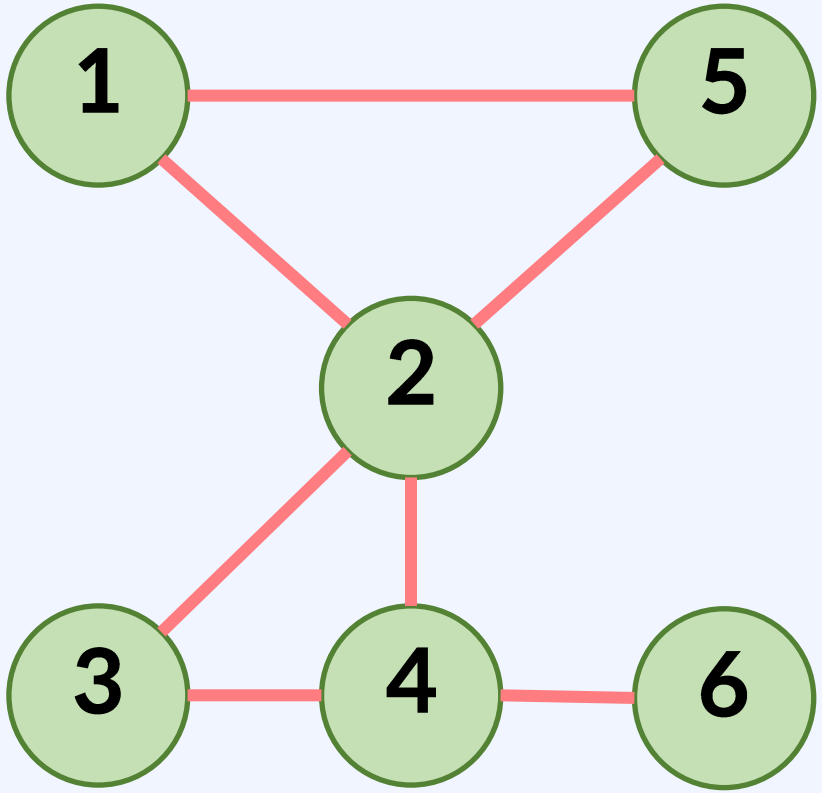
입력 데이터
6 8 1 2 2 5 5 1 3 4 4 6 5 4 2 4 2 3

출력 데이터
1

BOJ11724: 연결 요소의 개수

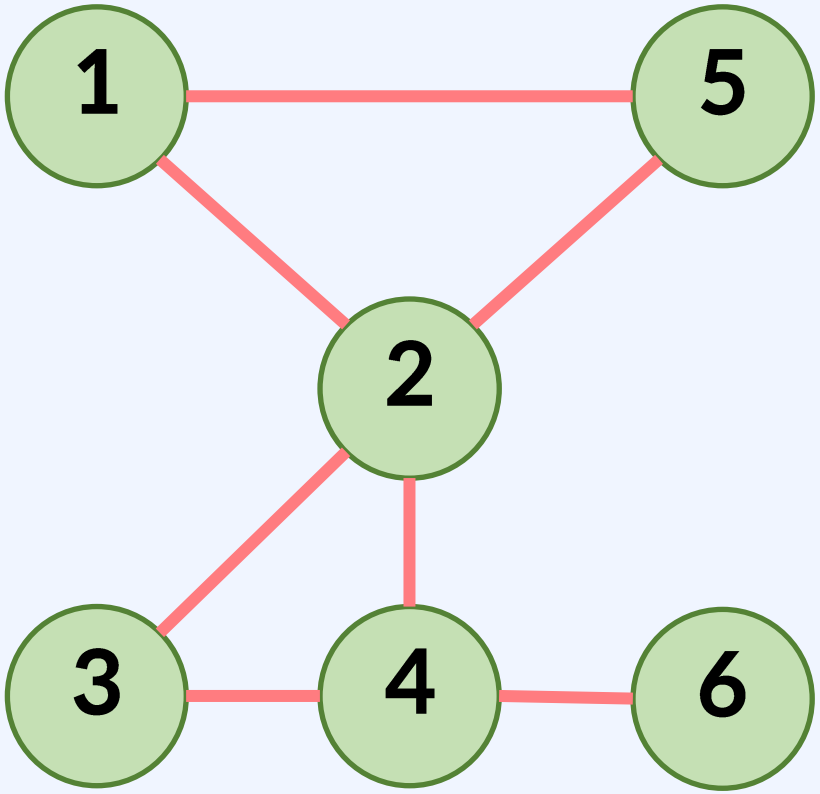
예제 2

입력 데이터
6 8 1 2 2 5 5 1 3 4 4 6 5 4 2 4 2 3



BOJ11724: 연결 요소의 개수

예제 2



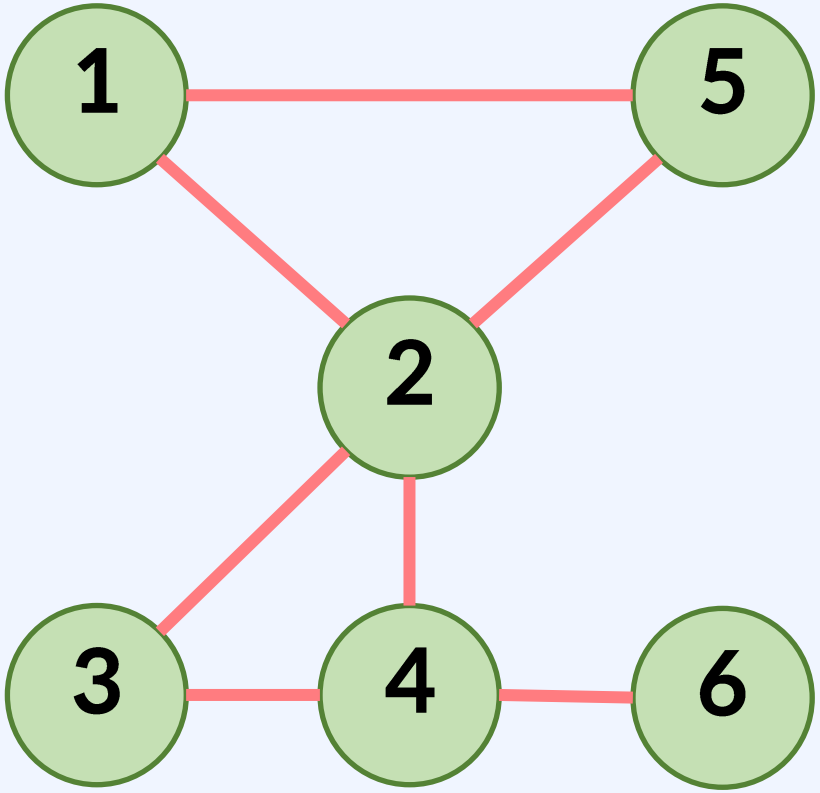
	1	2	3	4	5	6
visited[]	1	1	1	1	1	1

(1) 정점에서 DFS / BFS 수행

- 1 / 2 / 3 / 4 /6 /5 방문하고 종료

BOJ11724: 연결 요소의 개수

예제 2



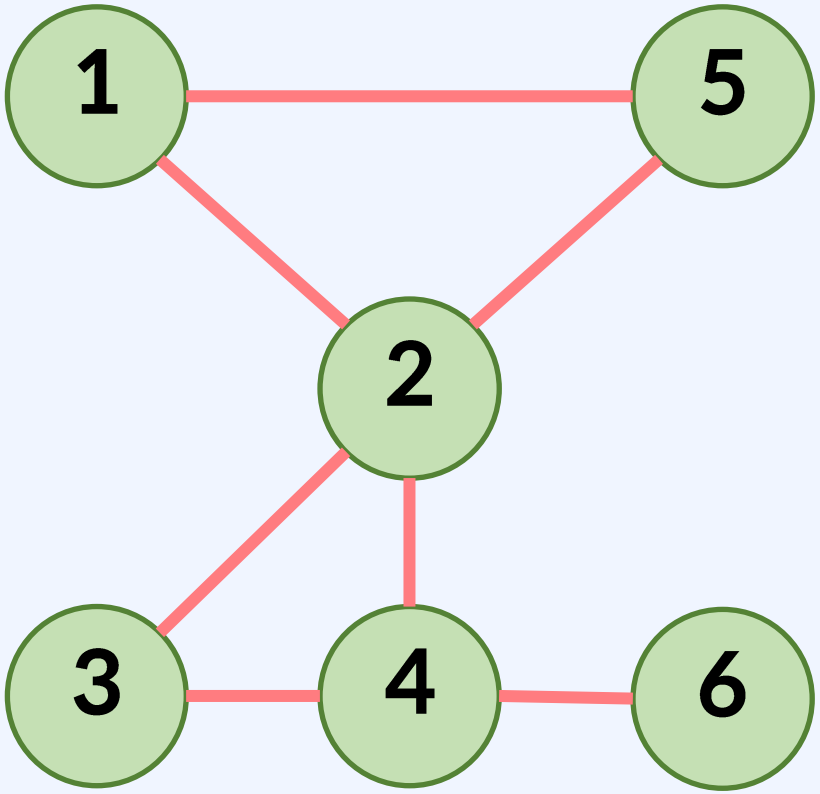
	1	2	3	4	5	6
visited[]	1	1	1	1	1	1

(2) ~ (6) 정점은 방문 이력이 있으므로 무시

(이미 연결요소 확인이 완료되었다)

BOJ11724: 연결 요소의 개수

예제 2

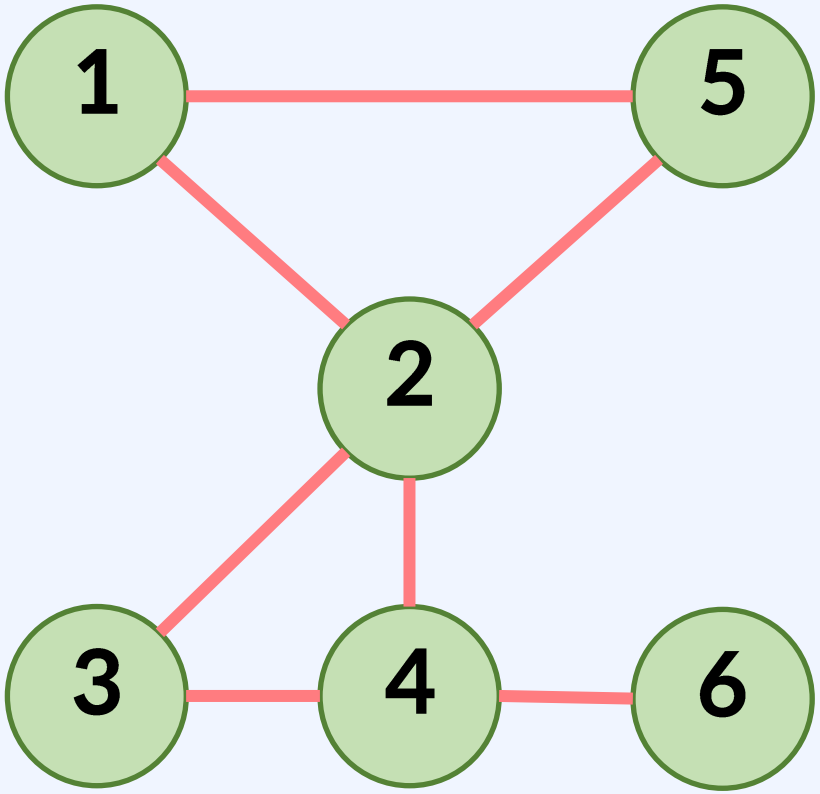


	1	2	3	4	5	6
visited[]	1	1	1	1	1	1

DFS / BFS 탐색은 총 1회 진행
→ 1 출력

BOJ11724: 연결 요소의 개수

예제 2



	1	2	3	4	5	6
visited[]	1	1	1	1	1	1

DFS / BFS 탐색은 총 1회 진행
→ 1 출력

Ch01. 그래프 탐색 BFS, DFS

4 [2606] 바이러스

BOJ2606: 바이러스

문제 요약

- {i}번 컴퓨터가 바이러스에 걸리면?
  - 네트워크로 연결되어 있는 모든 컴퓨터가 바이러스에 걸린다
- 컴퓨터의 수는 100 이하의 자연수
- 네트워크 경로는 방향성이 없음

BOJ2606: 바이러스

문제 분석

- 컴퓨터: 정점
- 네트워크 연결 정보: 간선

- 무방향 그래프

- 1번 컴퓨터가 바이러스에 걸렸을 때
  1번을 통해 바이러스에 걸리는 컴퓨터의 수를 출력
  → 1번 정점과 연결된 모든 정점의 수를 구하고 1을 뺀다 (자신)

BOJ2606: 바이러스

예제

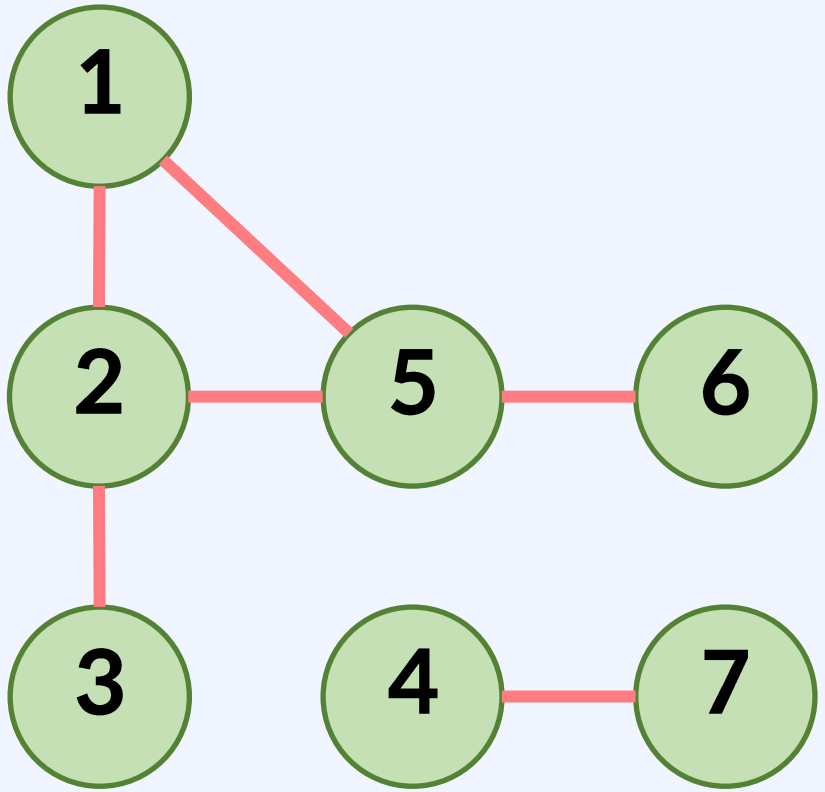
입력 데이터
7 6 1 2 2 3 1 5 5 2 5 6 4 7

출력 데이터
4

BOJ2606: 바이러스

예제

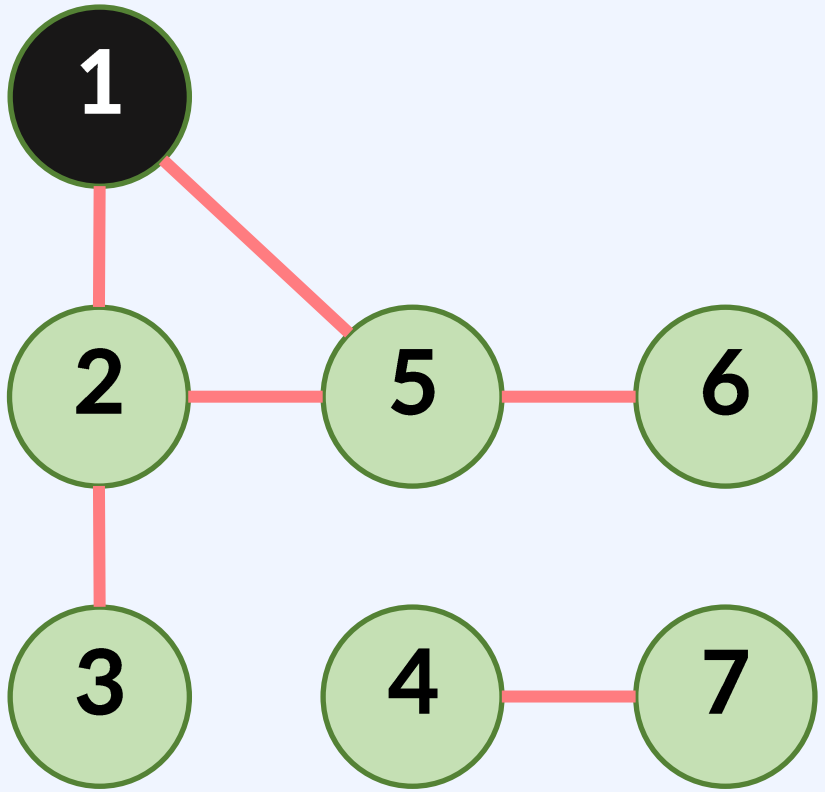
입력 데이터
7 6 1 2 2 3 1 5 5 2 5 6 4 7



BOJ2606: 바이러스

예제

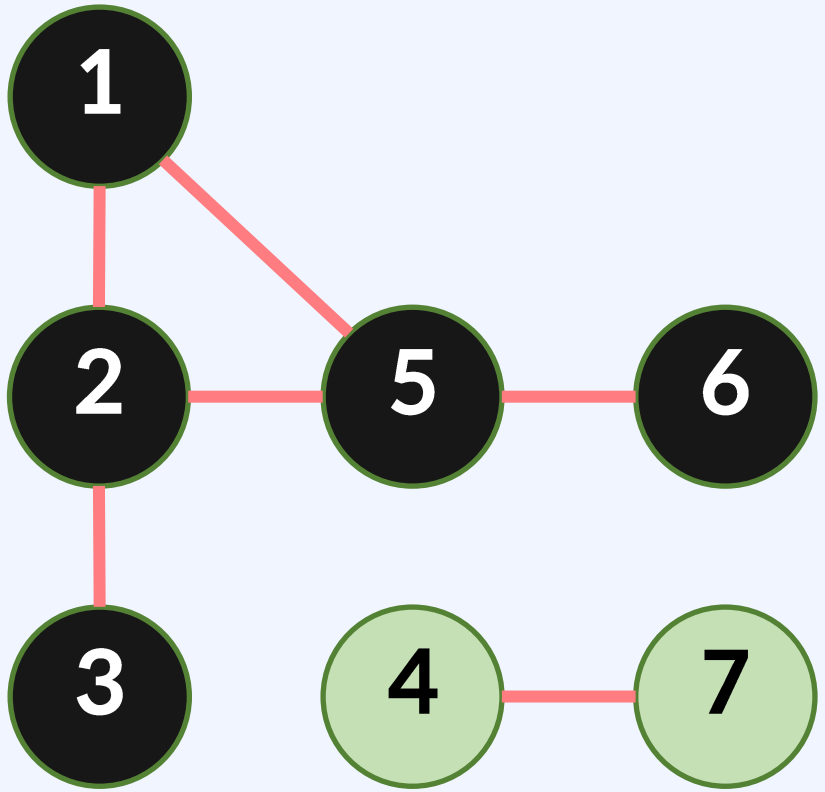
입력 데이터
7 6 1 2 2 3 1 5 5 2 5 6 4 7



BOJ2606: 바이러스

예제

입력 데이터
7 6 1 2 2 3 1 5 5 2 5 6 4 7



BOJ2606: 바이러스

그래프 탐색

- 그래프 연결관계를 표현 (인접행렬 가능)
- 1번 정점을 기준으로 DFS / BFS를 수행
- 중복 방문하지 않도록 visited[] 배열 관리
- 방문한 정점의 수에서 1번 정점인 1개를 빼고 출력

Ch01. 그래프 탐색 BFS, DFS

5 [2573] 빙산

BOJ2573: 빙산

문제 요약

- N * M 배열에 빙산의 높이가 입력
  - 3 <= N, M <= 300
- 빙산이 존재하는 칸은 10,000 개 이하

- 매년 바다(높이0)에 접한 면적이 녹으며 감소
  - 동/서/남/북 위치에 따라 1 ~ 4 감소

- 덩어리가 나누어지지 않고 한번에 녹으면 0 출력

BOJ2573: 빙산

예제

입력 데이터
5 7 0 0 0 0 0 0 0 0 2 4 5 3 0 0 0 3 0 2 5 2 0 0 7 6 2 4 0 0 0 0 0 0 0 0 0

출력 데이터
2

BOJ2573: 빙산

예제

0	0	0	0	0	0	0
0	2	4	5	3	0	0
0	3	0	2	5	2	0
0	7	6	2	4	0	0
0	0	0	0	0	0	0

BOJ2573: 빙산

예제

0	0	0	0	0	0	0
0	2-2	4-2	5-1	3-2	0	0
0	3-2	0	2-1	5	2-3	0
0	7-2	6-2	2-1	4-2	0	0
0	0	0	0	0	0	0

BOJ2573: 빙산

예제

0	0	0	0	0	0	0
0	0	2	4	1	0	0
0	1	0	1	5	0	0
0	5	4	1	2	0	0
0	0	0	0	0	0	0

BOJ2573: 빙산

예제

0	0	0	0	0	0	0
0	0	2-3	4-1	1-2	0	0
0	1-3	0	1-1	5-1	0	0
0	5-2	4-2	1-1	2-2	0	0
0	0	0	0	0	0	0

BOJ2573: 빙산

예제

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

BOJ2573: 빙산

문제 분석

- 탐색을 2종류로 분리
  - 1. 녹을 예정인 빙산을 탐색 (리스트 순회)
  - 2. 빙산이 떨어졌는지 확인하는 탐색 (DFS / BFS)

- 탐색과 동시에 빙산을 녹이면, 다음 결과에 영향이 감
  - 변경될 높이 정보를 기록했다가, 한번에 반영

- 탐색결과를 매번 순회하면
  n * m * {최대 빙산 사이즈=10} 만큼 반복된다
  - 빙산의 좌표를 따로 관리하면 최적화가 가능하다

BOJ2573: 빙산

빙산 관리

```
class Ice {
    int row;
    int col;
    int height;
    public Ice(int r, int c, int h) {
        this.row = r;
        this.col = c;
        this.height = h;
    }
}
```

- 빙산의 좌표와 높이를 기록
- 빙산 객체를 리스트로 관리
  List<Ice> iceList
- 빙산 객체를 순회하며
  상 / 하 / 좌 / 우가 바다인지 검사
  - 바다를 만나면 height를 감소

BOJ2573: 빙산

빙산 관리

- 관리하던 빙산 리스트를 순회하며
  - 높이가 0이하가 된 빙산이 발견되면?
    - 빙산 리스트에서 제거
    - 좌표 정보를 바다로 변경
  - 높이가 1 이상이라면?
    - 해당 좌표를 다음 DFS를 위해 visted[] 를 false 로 초기화

BOJ2573: 빙산

빙산 관리

- 첫번째 빙산과 연결된 다른 빙산의 수를 계산
  - 전체 빙산 리스트의 개수와 결과가 다르면? → 2개 이상의 조각

```
// iceList 첫번째 빙산이 몇개와 연결되어있는지 카운트
// 모든 빙산의 개수와, 첫번째 빙산과 연결된 빙산의 개수가
// 다르면 빙산이 분리되었다는 뜻
if(iceList.size() > 0 && dfs(iceList.get(0).row, iceList.get(0).col) != iceList.size()) {
    System.out.println(year);
    System.exit(0);
}
```

Ch01. 그래프 탐색 BFS, DFS

6 [1941] 소문난 칠공주

BOJ1941: 소문난 칠공주

문제 요약

- 5x5의 S와 Y로 구성된 문자열
- 7개의 연결된 칸을 선택했을 때
  S가 4개 이상 포함되는 조합의 개수 구하기

BOJ1941: 소문난 칠공주

문제 분석

- 25명의 학생을 7명을 뽑음 → ${}_{25}C_7$ == 480,700
  - 조합의 수가 많지 않다

- 따라서 일단 무작정 7명을 뽑고
  연결되어 있는지를 판단해도 충분하다
  (여기서 연결은 상하좌우로 붙어있다는 뜻이다)

- 포함된 S의 개수와 연결 여부를 확인하여 개수를 구함

BOJ1941: 소문난 칠공주

7명을 뽑는 방법?

1. 학생들에게 [0, 24] 범위로 번호를 배정

2. 배열에 뽑은 여부를 체크하며 순열 생성

BOJ1941: 소문난 칠공주

7공주 유효성 검증

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

가로: 두 숫자의 차이가 1

세로: 두 숫자의 차이가 5

단, 두 숫자중 큰 수를 5로 나눈 나머지가 0이면?
→ 경계에 걸친 수, 인접 X

BOJ1941: 소문난 칠공주

7공주 유효성 검증

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

가로: 두 숫자의 차이가 1

세로: 두 숫자의 차이가 5

단, 두 숫자중 큰 수를 5로 나눈 나머지가 0이면?
→ 경계에 걸친 수, 인접 X

BOJ1941: 소문난 칠공주

7공주 유효성 검증

가로: 두 숫자의 차이가 1

세로: 두 숫자의 차이가 5

단, 두 숫자 중 큰 수를 5로 나눈 나머지가 0이면?
→ 경계에 걸친 수, 인접 X

```
public static boolean isFriend(int a, int b) {
    int diff = Math.abs(a - b);
    int max = Math.max(a, b);

    if(diff == 1 && max % 5 != 0)
        return true;
    if(diff == 5)
        return true;

    return false;
}
```

BOJ1941: 소문난 칠공주

DFS (or BFS)

- 순열로 뽑은 학생들이 인접한 자리에 있는지 체크
- 직전에 만든 유효성 검증 함수를 이용해 탐색을 하며 대조한다
- 7명이 모두 인접하면 count에서 7이 반환된다

```
public static int dfs(int studentNum) {
    int count = 1;
    check[studentNum] = true;
    for (int i = 1; i < 7; i++) {
        int me = pick.get(studentNum);
        int you = pick.get(i);
        if (!check[i] && isFriend(me, you)) {
            count += dfs(i);
        }
    }
    return count;
}
```

BOJ1941: 소문난 칠공주

순열 생성

- 재귀함수를 이용한 순열 생성 (매개변수: 학생 번호)
  - 현재 학생을 고르지 않고 순열을 생성
  - 현재학생을 고르며 순열을 생성

```
int ret = 0; // 조합의 개수
// studentNum 번째 학생을 포함하지 않는 경우
ret += nextCombination(studentNum + 1);
// studentNum 번째 학생을 포함하는 경우
pick.add(studentNum);
ret += nextCombination(studentNum + 1);
pick.remove(pick.size() - 1);
return ret;
```

BOJ1941: 소문난 칠공주

뽑은 학생 검증

- 순열생성에서 뽑힌 학생은 pick[] 리스트에 담겨있다

- 리스트에 7명이 모이면
  1. 이다솜파가 4명 이상인지 검사
  2. DFS(BFS)를 통해 7명이 모두 인접한지 검사
  3. 위 조건을 하나라도 만족하지 않으면, 조합이 아님

BOJ1941: 소문난 칠공주

뽑은 학생 검증

1. 이다솜파가 4명 이상인지 검사

```
// 이다솜파 인원 체크
for (int i = 0; i < 7; i++) {
    if (students[pick.get(i)] == 1) cnt++;
}
// 이다솜파가 4명 미만이라면 조합으로 사용하지 않음
if (cnt < 4) return 0;
```

BOJ1941: 소문난 칠공주

뽑은 학생 검증

2. DFS(BFS)를 통해 7명이 모두 인접한지 검사

```
// DFS 탐색 전 초기화
for (int i = 0; i < 7; i++) {
    check[i] = false;
}
// 7명이 모두 인접해 있다면 조합으로 인정함
if(dfs(0) == 7) return 1;

// 25명의 학생을 다 순열 생성에 사용했는데, 7명이 모이지 않았다면 종료
if (studentNum >= 25) return 0;
```