Chapter. 02

알고리즘

그래프와 탐색
Graph & Search

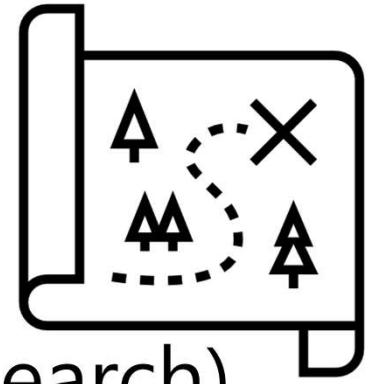
FAST CAMPUS
ONLINE
알고리즘 공채 대비반I

강사. 류호석

Life Changing Education

Chapter. 02

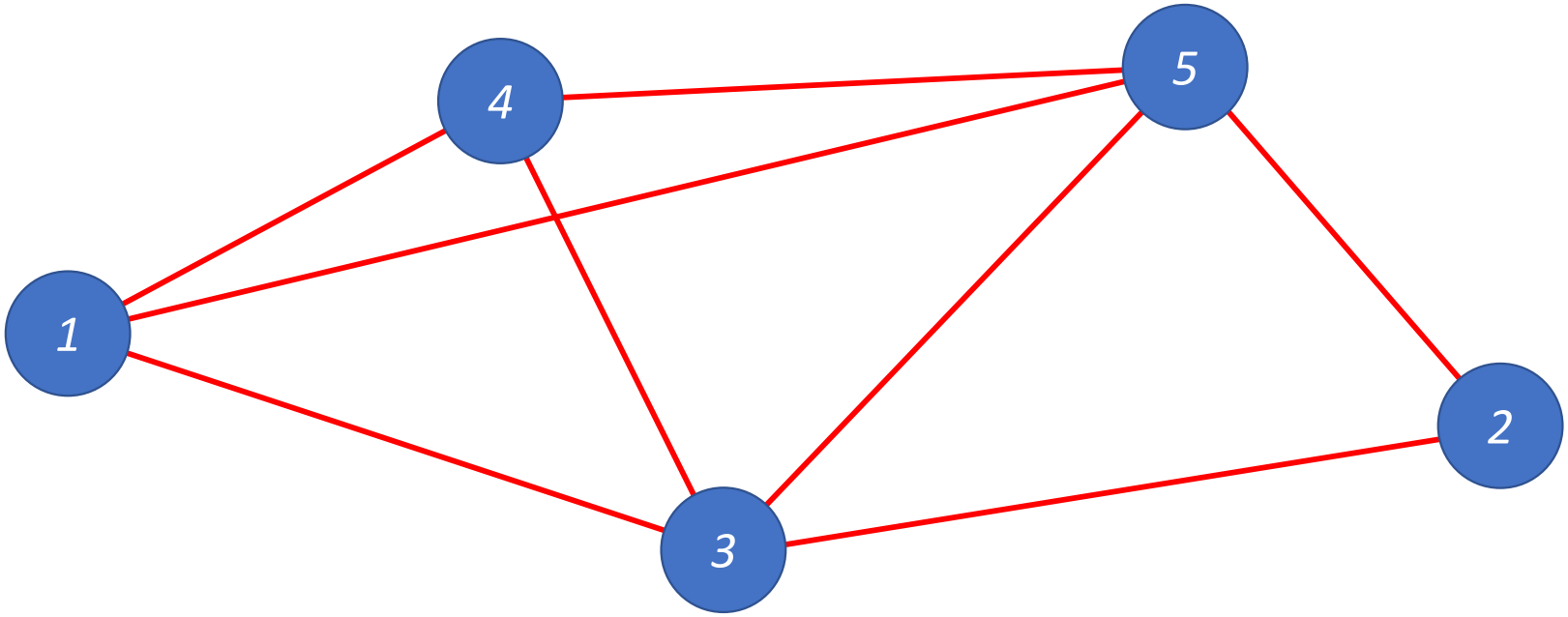
알고리즘
그래프와 탐색(Graph & Search)



FAST CAMPUS
ONLINE
류호석 강사.

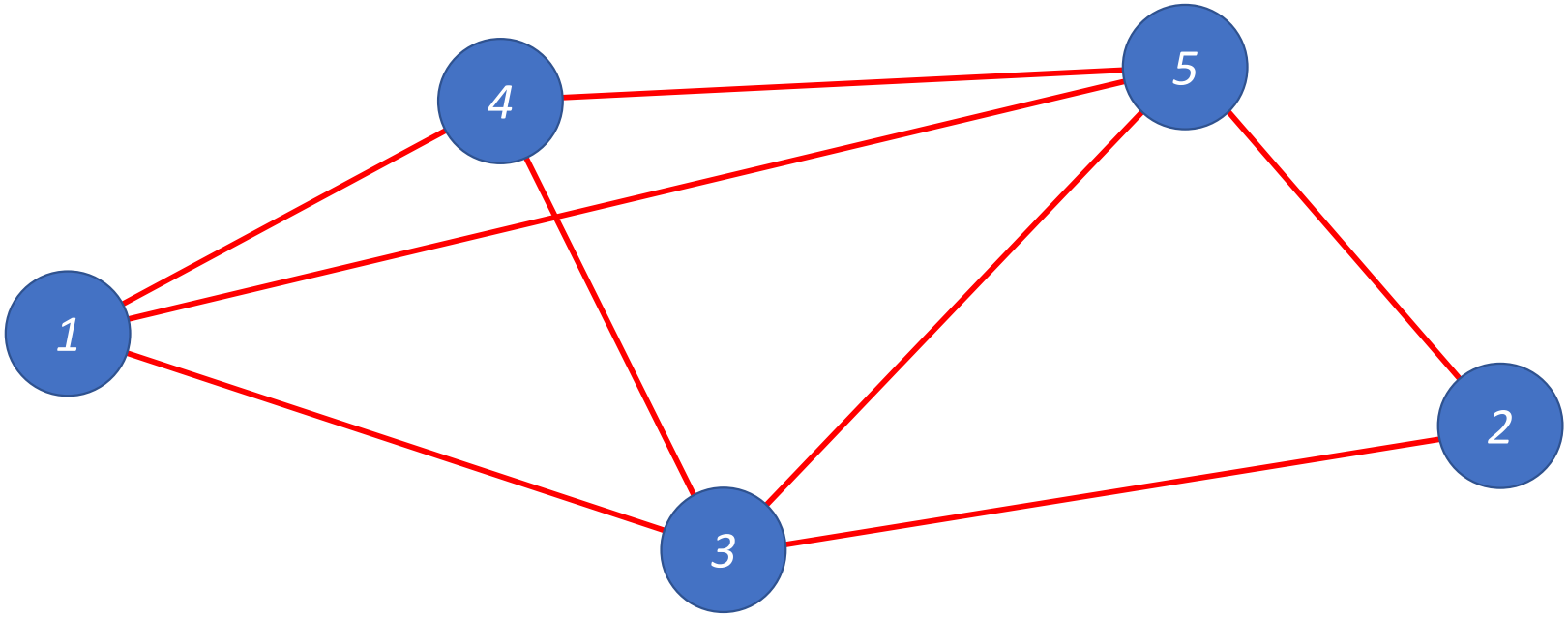
I 그래프(Graph)란?

자료 구조로써 Graph = **정점(Vertex)** + **간선(Edge)**



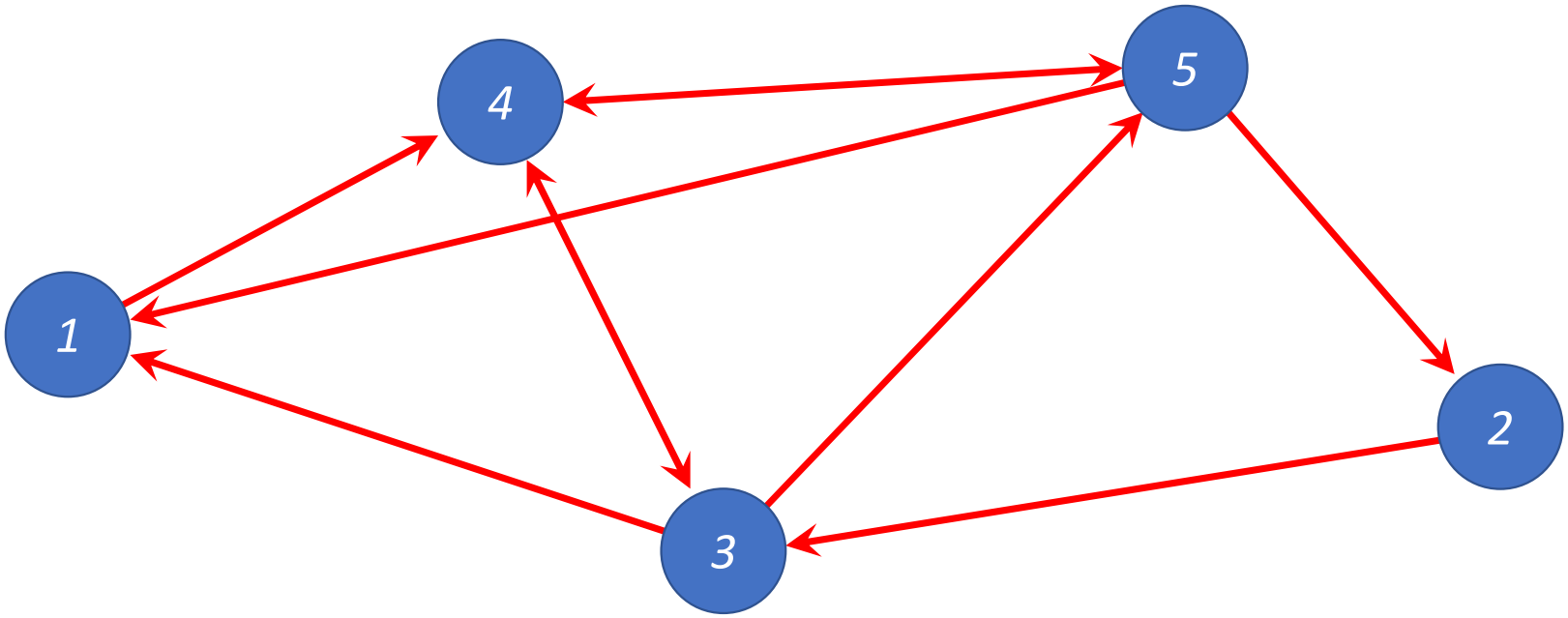
I 그래프(Graph)란?

간선(Edge) ➔ 무방향 / 방향



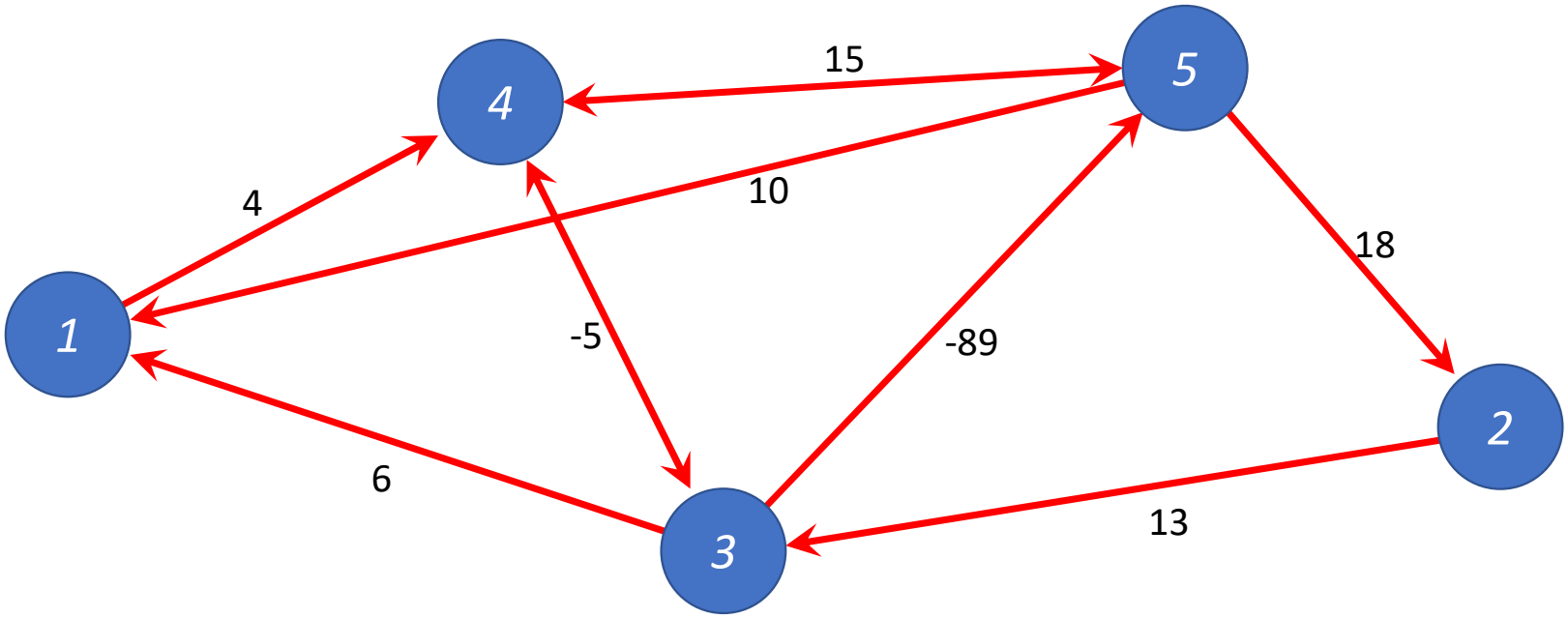
I 그래프(Graph)란?

간선(Edge) ➔ 무방향 / 방향



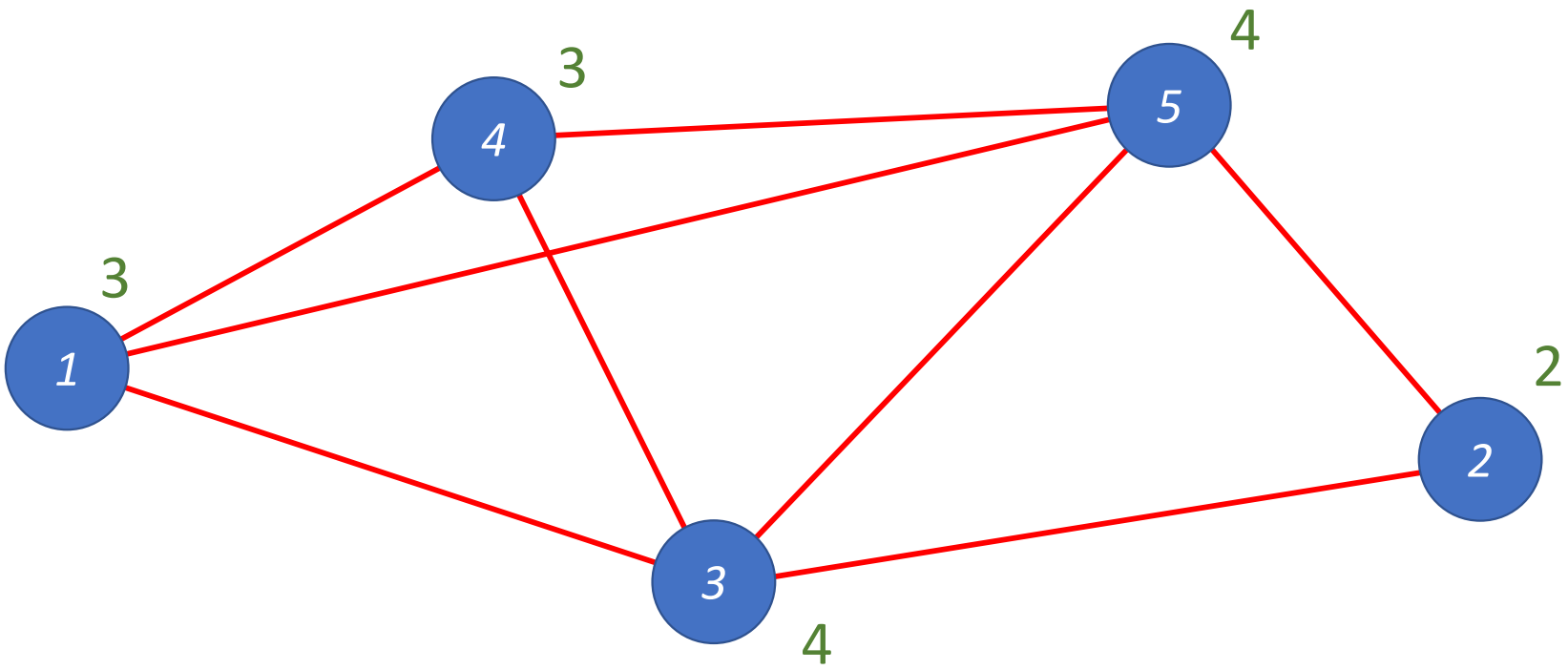
I 그래프(Graph)란?

간선(Edge) ➔ (무방향 / 방향) + 가중치



I 정점의 차수(Degree)와 성질

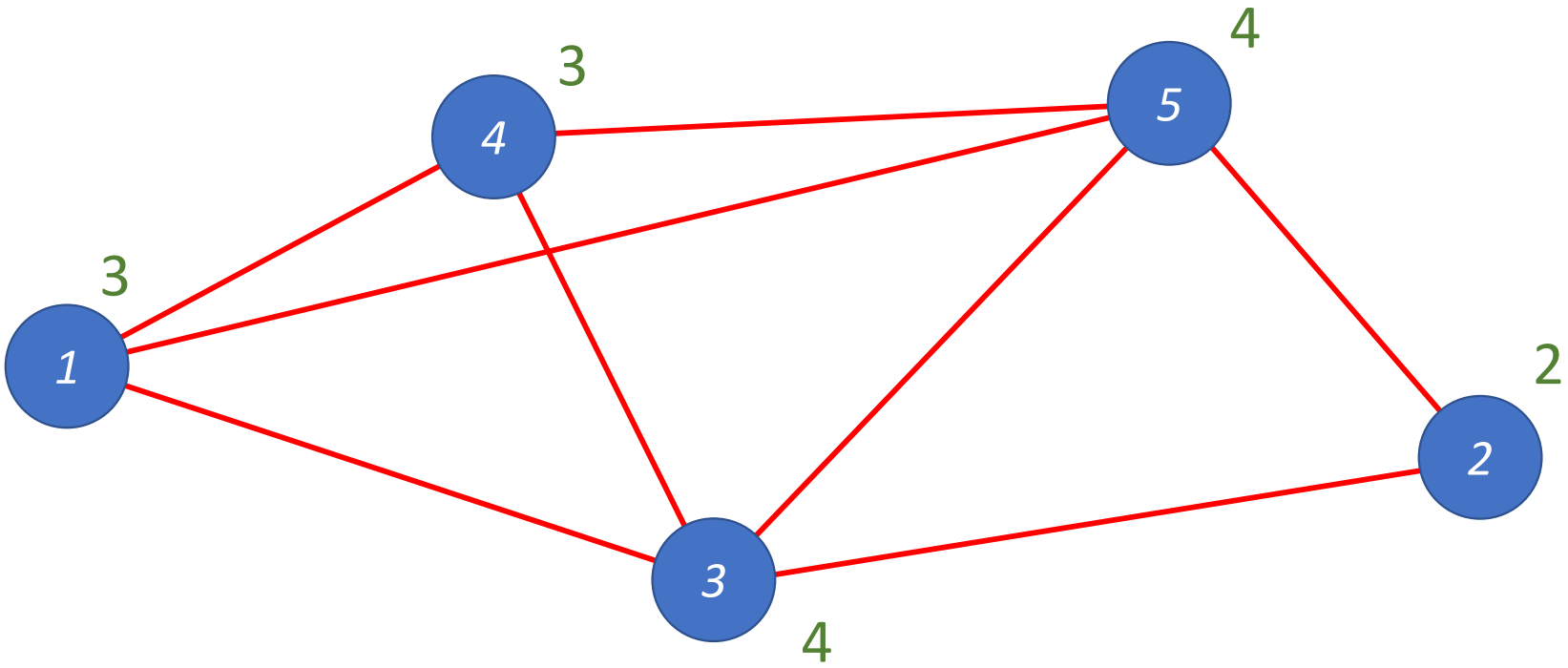
$deg(x)$:= 정점 x의 차수(degree), 정점 x에 연결된 간선의 수



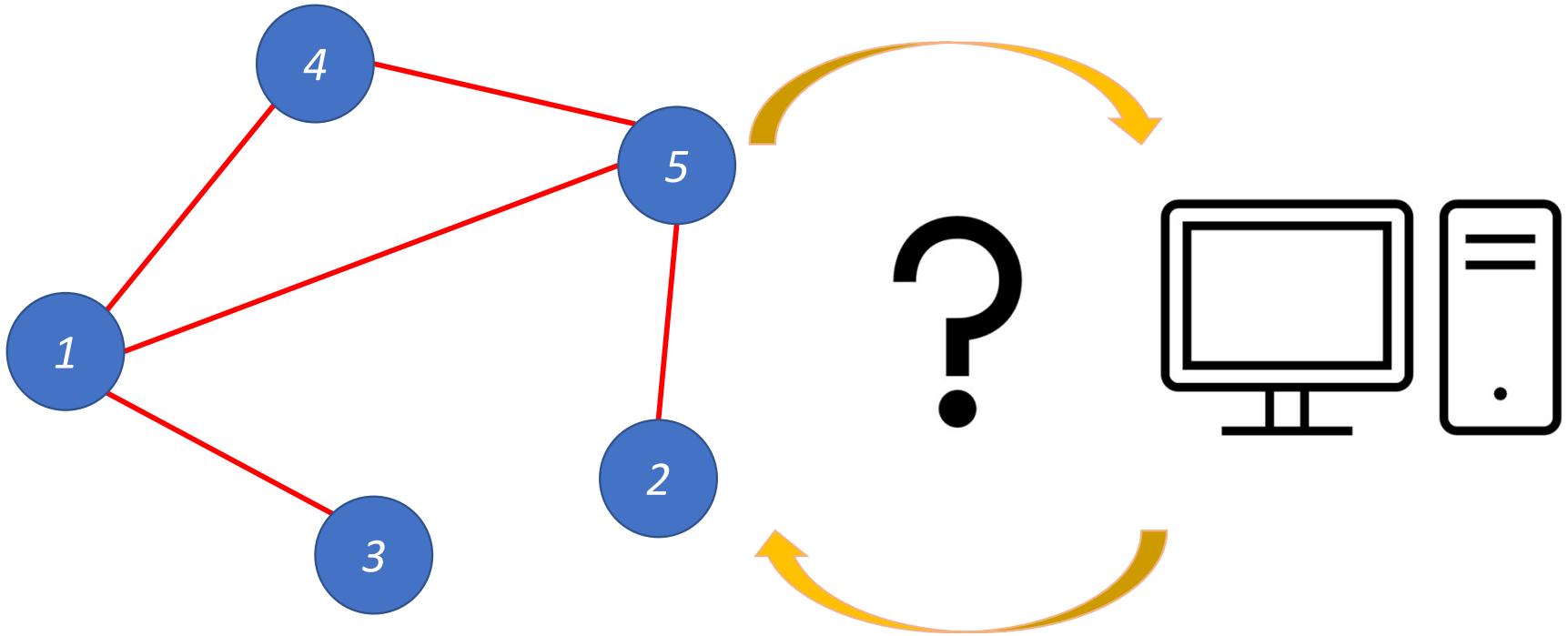
I 정점의 차수(Degree)와 성질

$\sum deg(x)$ = 모든 정점의 차수의 합 = 간선의 개수의 2배!

$$3 + 3 + 4 + 4 + 2 = 16 = 8 * 2$$



I 그래프(Graph)를 저장하는 방법



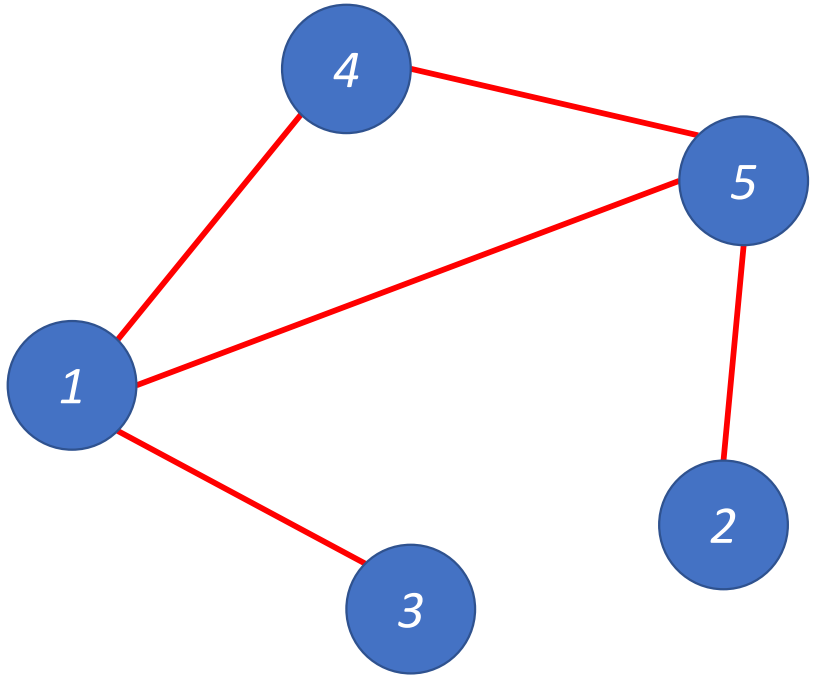
I 그래프(Graph)를 저장하는 방법

그래프를 저장하는 대표적인 두 가지 방법

1. 인접 행렬 (Adjacency Matrix)
2. 인접 리스트 (Adjacency List)

I 그래프(Graph)를 저장하는 방법

1. 인접 행렬 (Adjacency Matrix)



adj	*1*	*2*	*3*	*4*	*5*
1	0	0	1	1	1
2	0	0	0	0	1
3	1	0	0	0	0
4	1	0	0	0	1
5	1	1	0	1	0

adj[*A*, *B*] = 1 ➔ *A* 에서 *B* 로 향하는 간선이 있다.

I 그래프(Graph)를 저장하는 방법

1. 인접 행렬 (Adjacency Matrix)

- `int[][] adj = int new[V][V];`
- $O(V^2)$ 만큼의 공간 필요
- A에서 B로 이동 가능? 가중치 얼마?
  - $O(1)$
- 정점 A에서 갈 수 있는 정점들은?
  - $O(V)$

adj	*1*	*2*	*3*	*4*	*5*
1	0	0	1	1	1
2	0	0	0	0	1
3	1	0	0	0	0
4	1	0	0	0	1
5	1	1	0	1	0

I 그래프(Graph)를 저장하는 방법

1. 인접 행렬 (Adjacency Matrix)

- $O(V\text{^}2)$ 만큼의 공간 필요

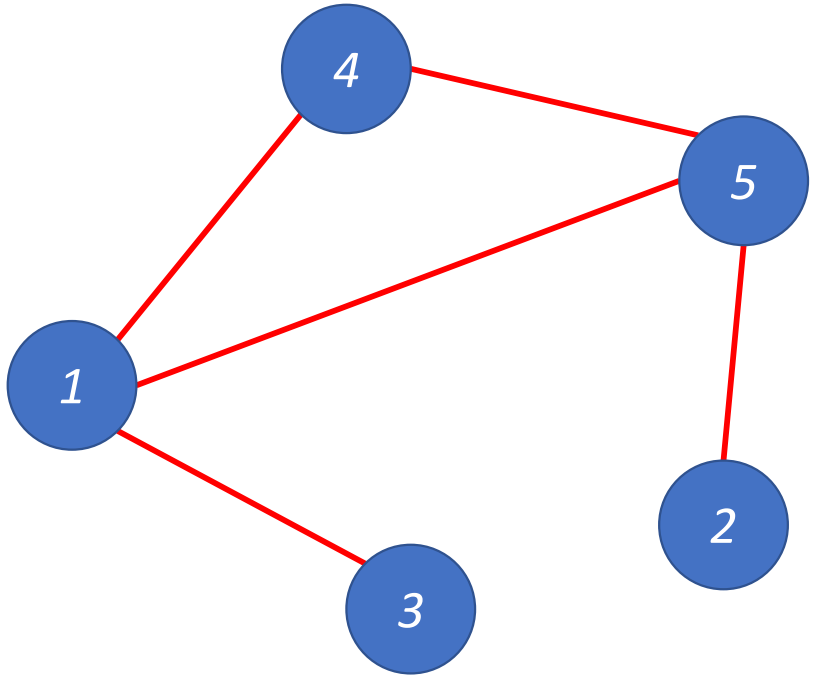
V=10만, E=50만

➔ $V^2 = 100$억 $= 10$ G!!!!!!!

adj	*1*	*2*	*3*	*4*	*5*
1	0	0	1	1	1
2	0	0	0	0	1
3	1	0	0	0	0
4	1	0	0	0	1
5	1	1	0	1	0

I 그래프(Graph)를 저장하는 방법

2. 인접 리스트(Adjacency List)



adj			
1	5	3	4
2	5		
3	1		
4	1	5	
5	2	1	4

adj[A] = $\{B_1, B_2, B_3\}$ ➔ *A* 에서 B_1, B_2, B_3 로 향하는 간선이 있다.

I 그래프(Graph)를 저장하는 방법

2. 인접 리스트(Adjacency List)

- `ArrayList<ArrayList<Integer>> adj;`
- $O(E)$ 만큼의 공간 필요
- A에서 B로 이동 가능? 가중치 얼마?
  - $O(\min(\deg(A), \deg(B)))$
- 정점 A에서 갈 수 있는 정점들은?
  - $O(\deg(A))$

adj			
1	5	3	4
2	5		
3	1		
4	1	5	
5	2	1	4

I 그래프(Graph)를 저장하는 방법

2. 인접 리스트(Adjacency List)

- $O(E)$ 만큼의 공간 필요

V=10만, E=50만

➔ $5 * 10^5 = 500\text{K}$

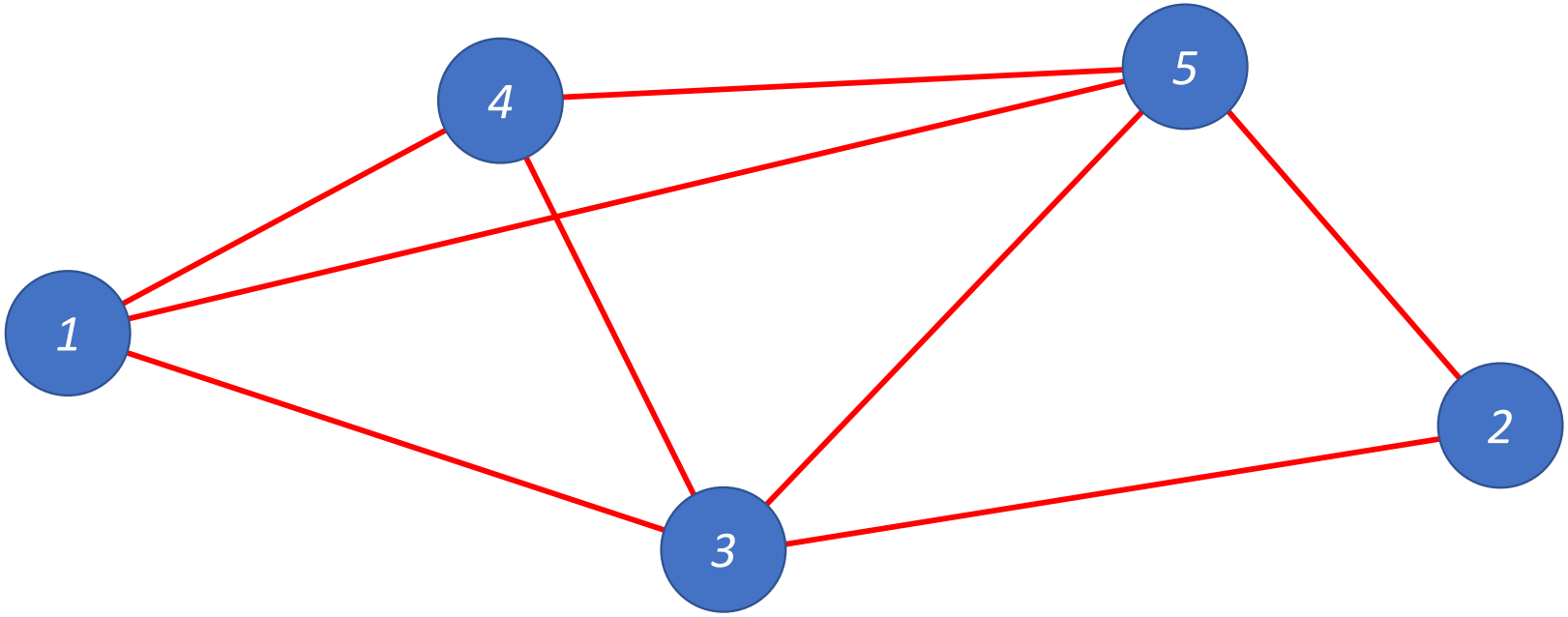
adj			
1	5	3	4
2	5		
3	1		
4	1	5	
5	2	1	4

I 그래프(Graph)를 저장하는 방법 - 요약

	인접 행렬	인접 리스트
*A*와 *B*를 잇는 간선 존재 여부 확인	$O(1)$	$O(\min(\deg(A), \deg(B))$
*A*와 연결된 모든 정점 확인	$O(\|V\|)$	$O(\deg(A))$
공간 복잡도	$O(\|V\|^2)$	$O(\|E\|)$

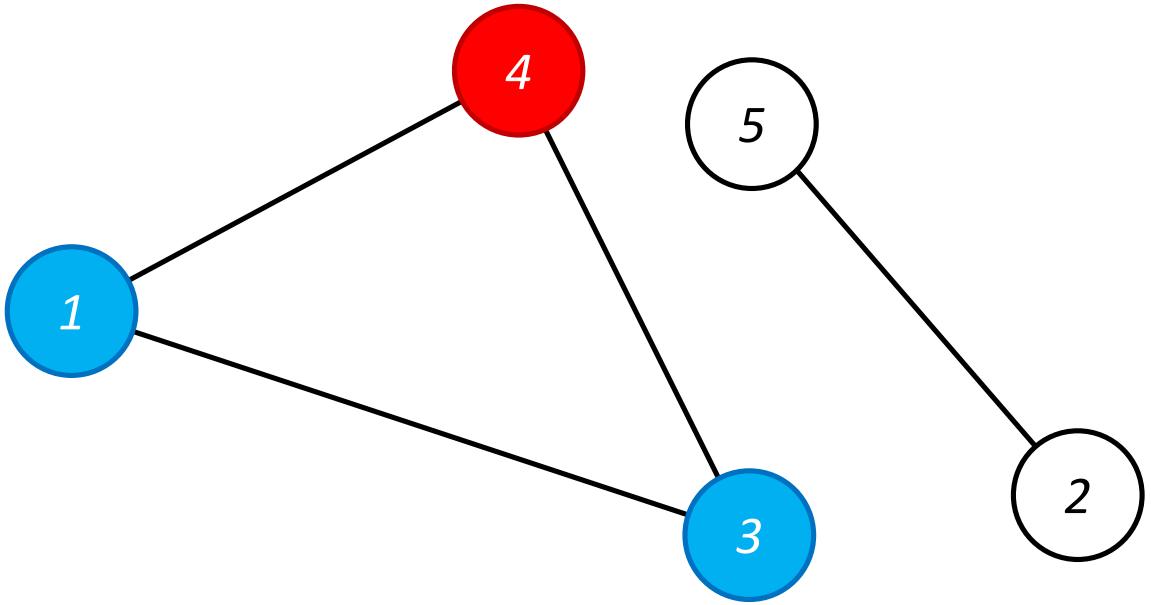
I 그래프(Graph) 문제의 핵심!

- **정점(Vertex)** & **간선(Edge)** 에 대한 정확한 정의
- 간선 저장 방식을 확인하기!



I 그래프(Graph)에서의 탐색(Search)이란?

탐색(Search) = **시작점**에서 간선을 0개 이상 사용해서 갈 수 있는 정점들은 무엇인가?



adj		
1	3	4
2	5	
3	1	4
4	1	3
5	2	

I 그래프(Graph)에서의 탐색(Search)이란?

탐색(Search) = **시작점**에서 간선을 0개 이상 사용해서 갈 수 있는 정점들은 무엇인가?

adj		
1	3	4
2	5	
3	1	4
4	1	3
5	2	

1	*2*	*3*	*4*	*5*
?	?	?	?	?

I 그래프(Graph)에서의 탐색(Search)이란?

탐색(Search) = **시작점**에서 간선을 0개 이상 사용해서 갈 수 있는 정점들은 무엇인가?

adj		
1	3	4
2	5	
3	1	4
4	1	3
5	2	

1	*2*	*3*	*4*	*5*
1	0	1	1	0

I 그래프(Graph)에서의 탐색(Search)이란?

탐색(Search) = **시작점**에서 간선을 0개 이상 사용해서 갈 수 있는 정점들은 무엇인가?

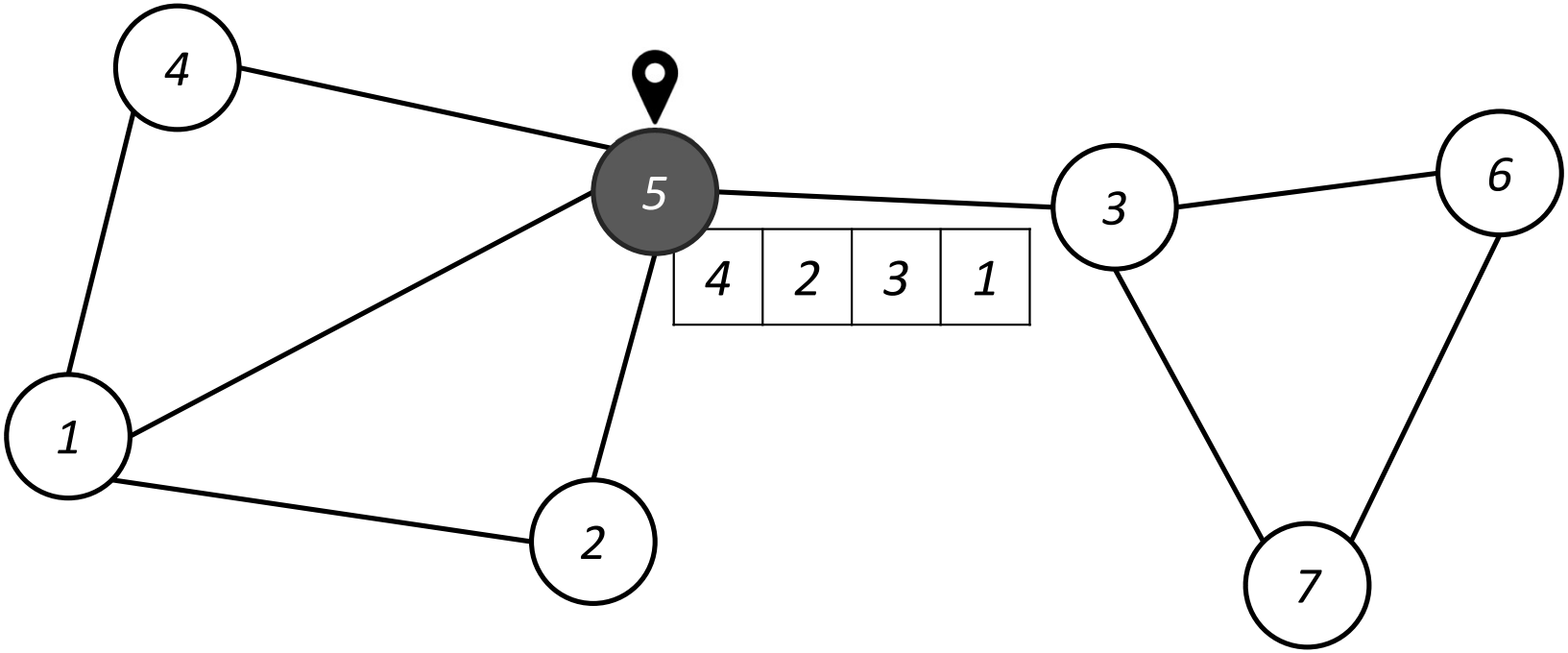
1. 깊이 우선 탐색(Depth First Search)

2. 너비 우선 탐색(Breadth First Search)

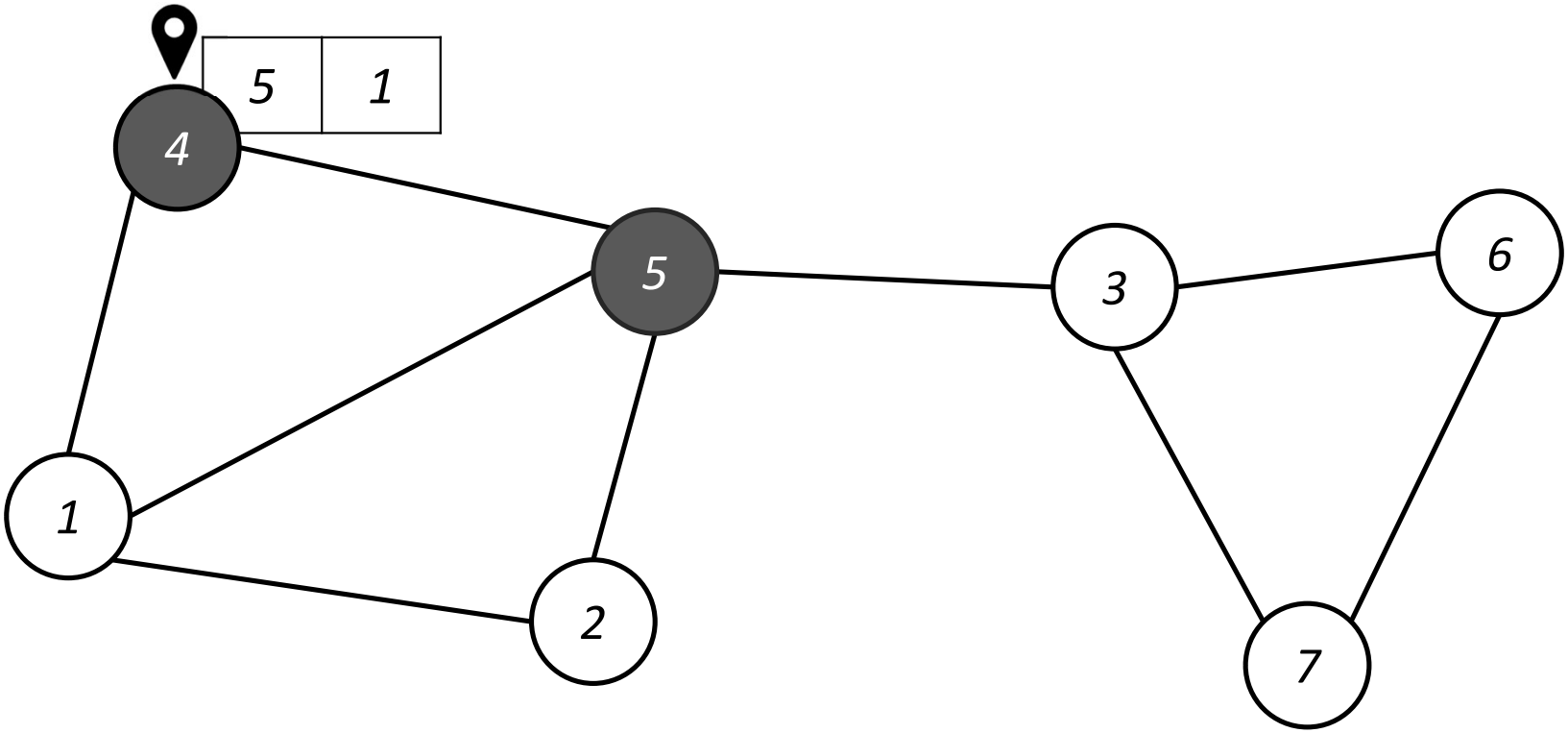
I 깊이 우선 탐색(*DFS, Depth First Search*)

```
// x 를 갈 수 있다는 걸 알고 방문한 상태
static void dfs(int x){
    // x 를 방문했다.
    visit[x] = true;

    // x 에서 갈 수 있는 곳들을 모두 방문한다.
    for (int y: x 에서 갈 수 있는 점들){
        if (visit[y])  // y 를 이미 갈 수 있다는 사실을 안다면, 굳이 갈 필요 없다.
            continue;

        // y에서 갈 수 있는 곳들도 확인 해보자
        dfs(y);
    }
}

main(){
    dfs(5);
}
```

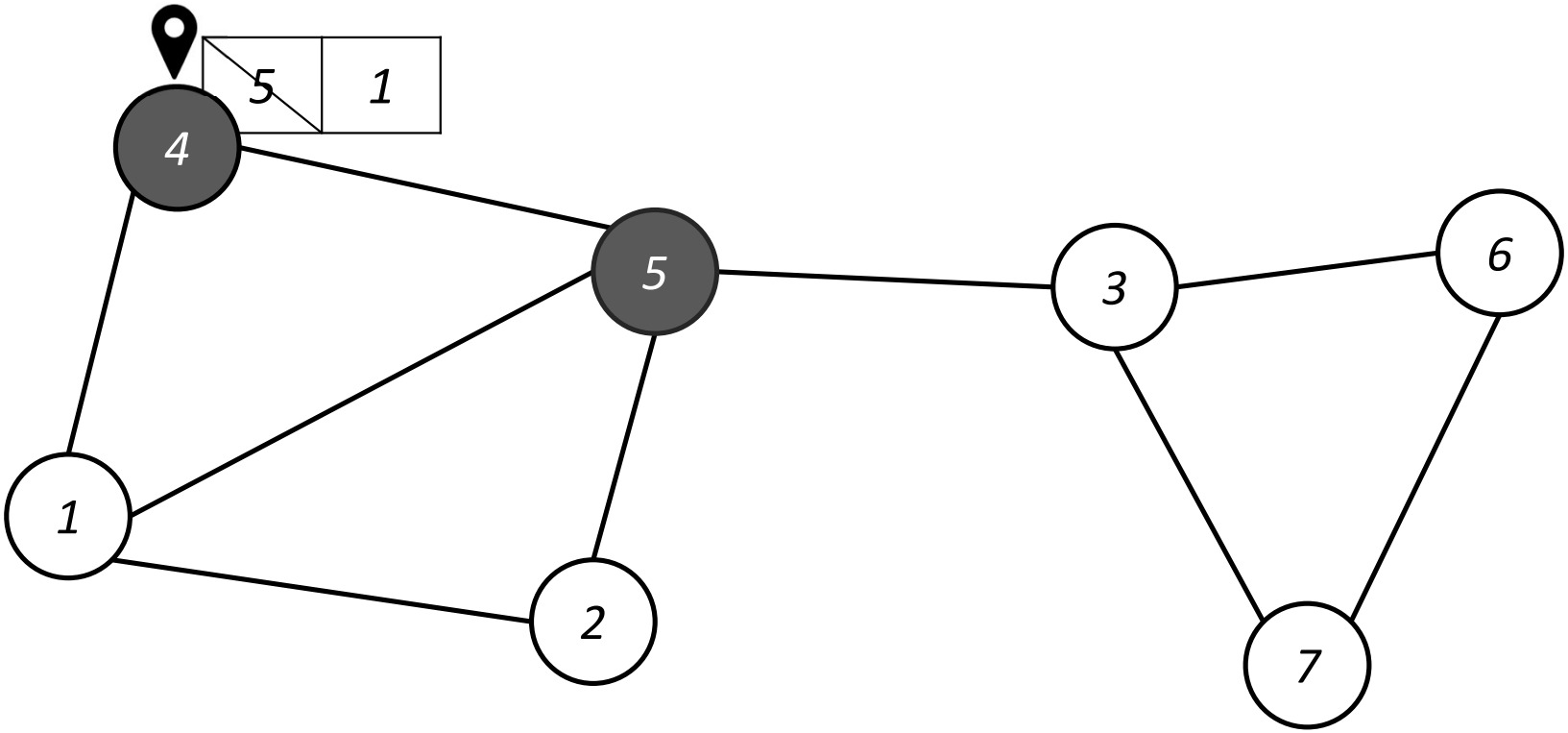
I 깊이 우선 탐색(*DFS, Depth First Search*)



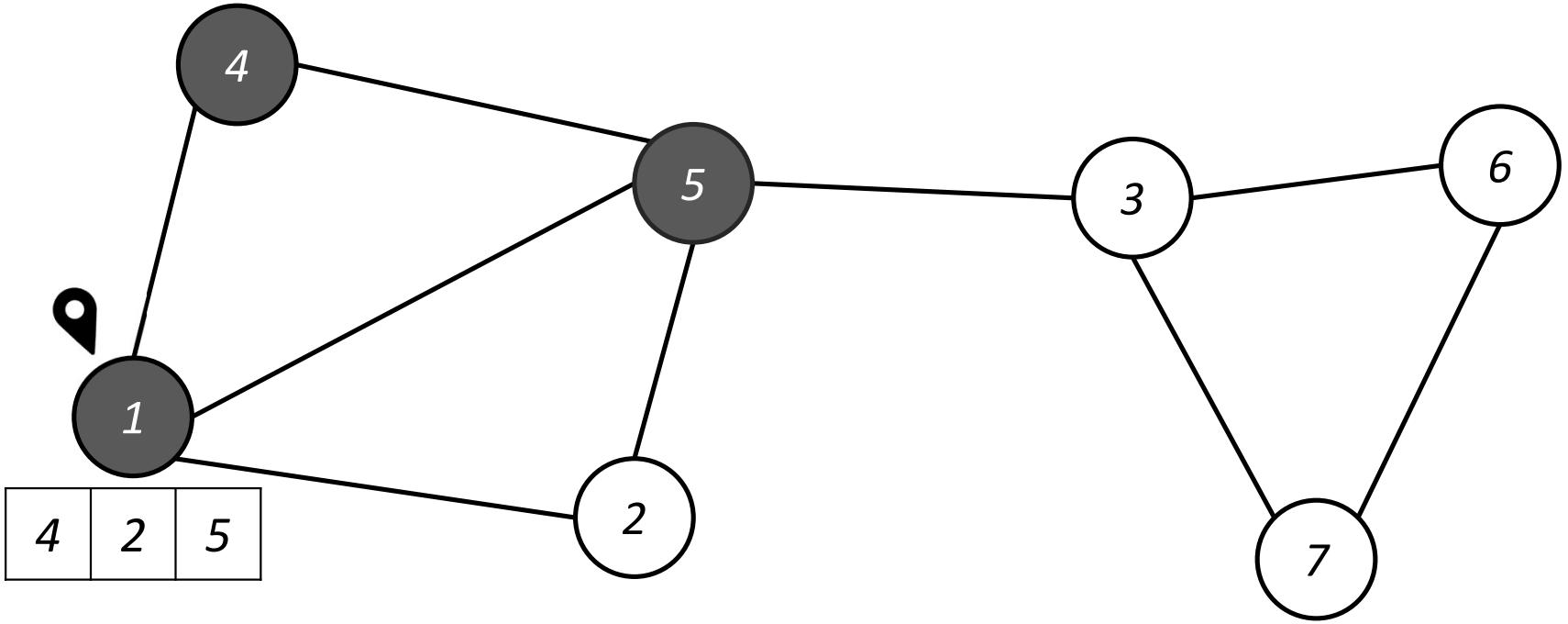
I 깊이 우선 탐색(*DFS, Depth First Search*)



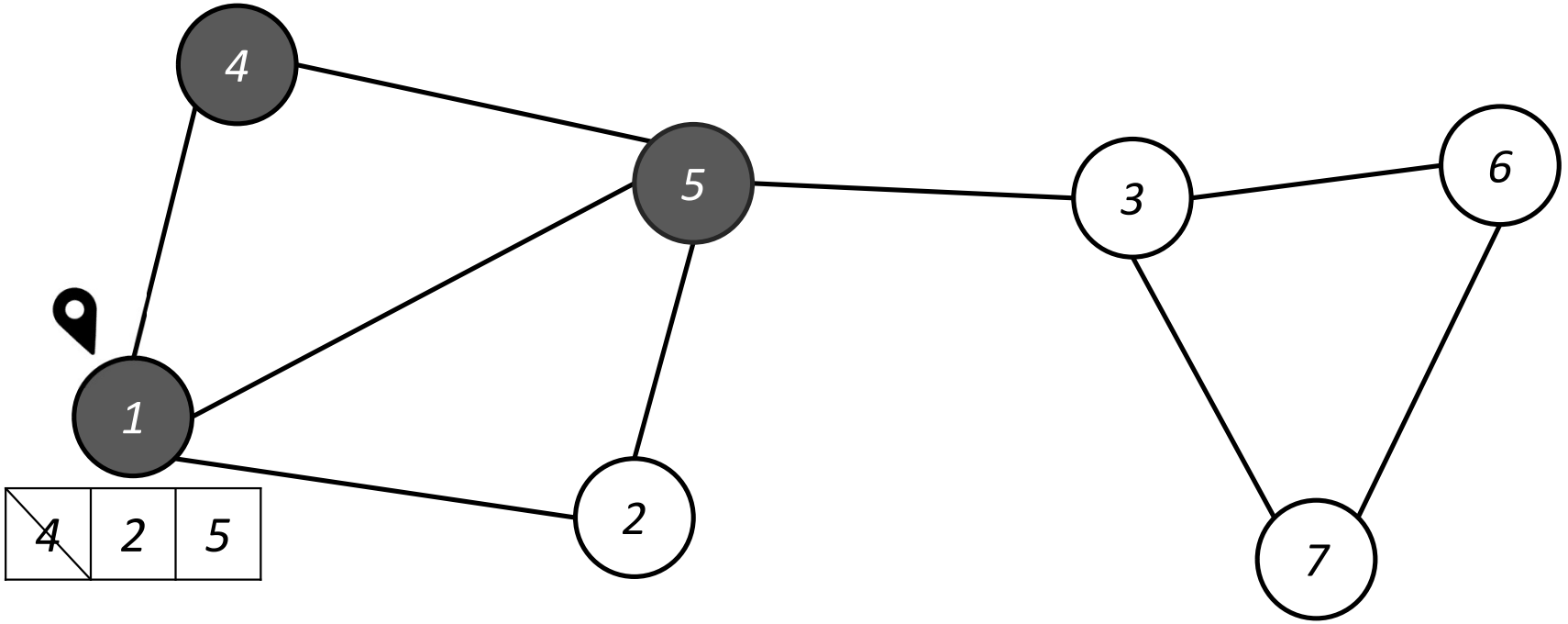
I 깊이 우선 탐색(*DFS, Depth First Search*)



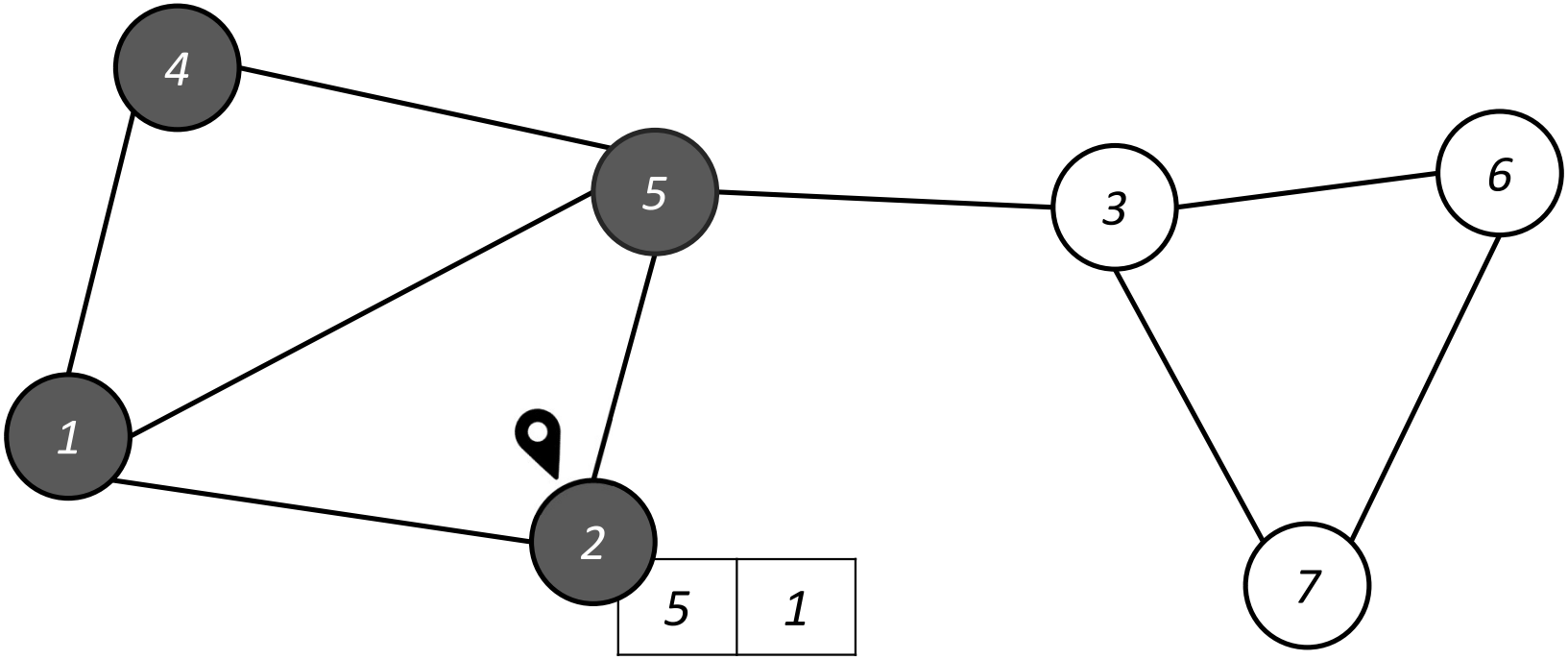
I 깊이 우선 탐색(*DFS, Depth First Search*)



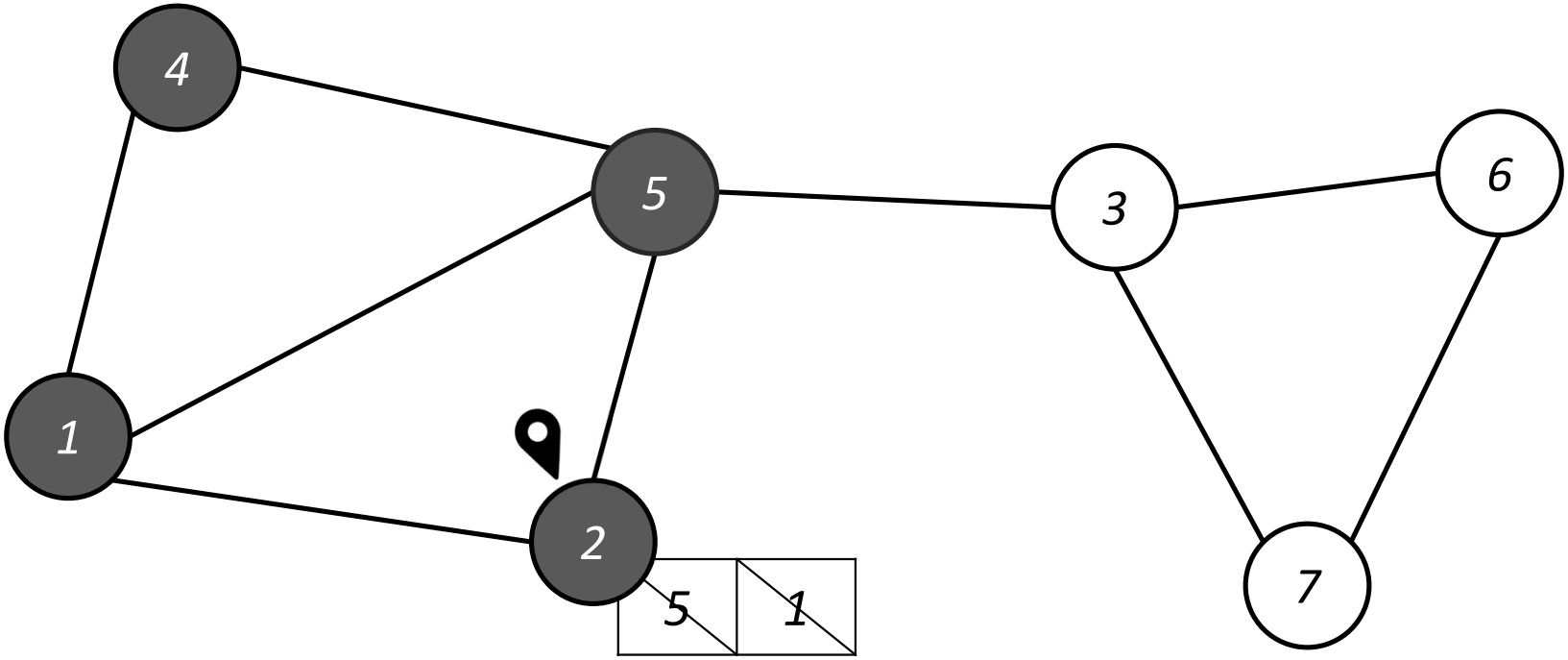
I 깊이 우선 탐색(*DFS, Depth First Search*)



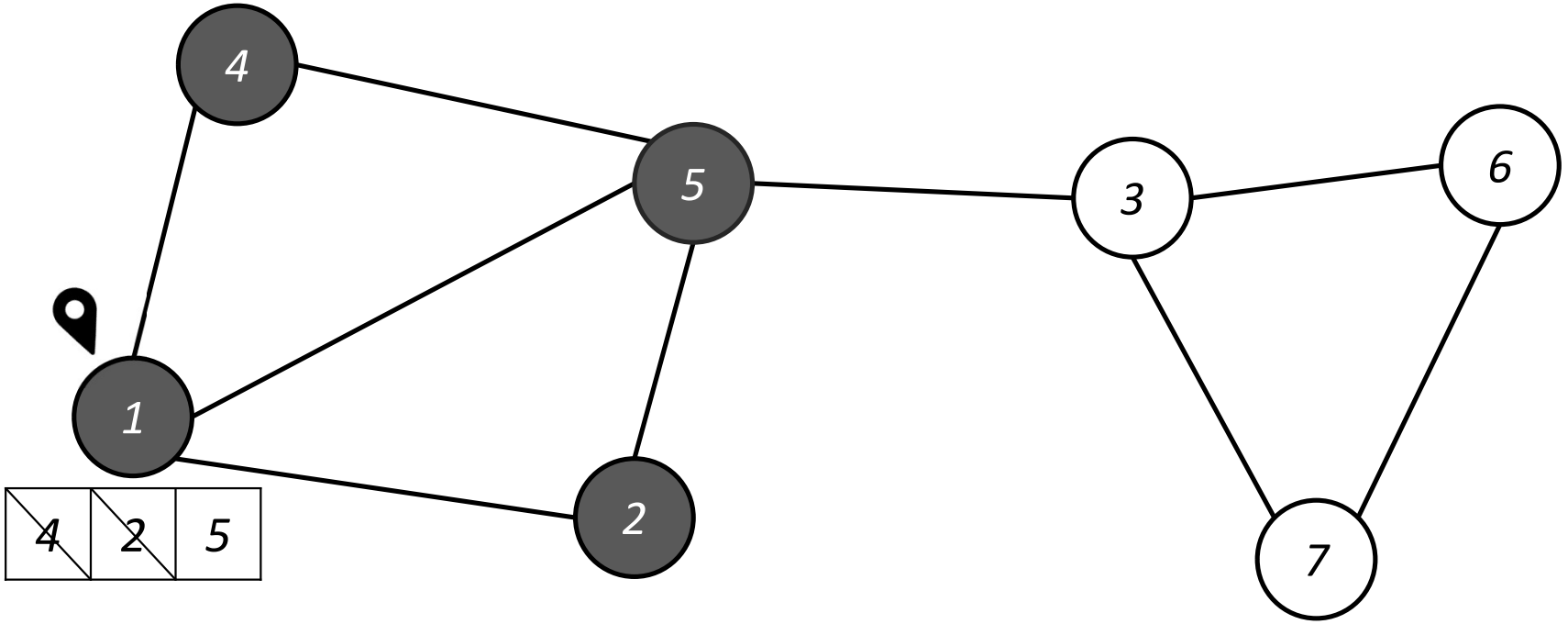
I 깊이 우선 탐색(*DFS, Depth First Search*)



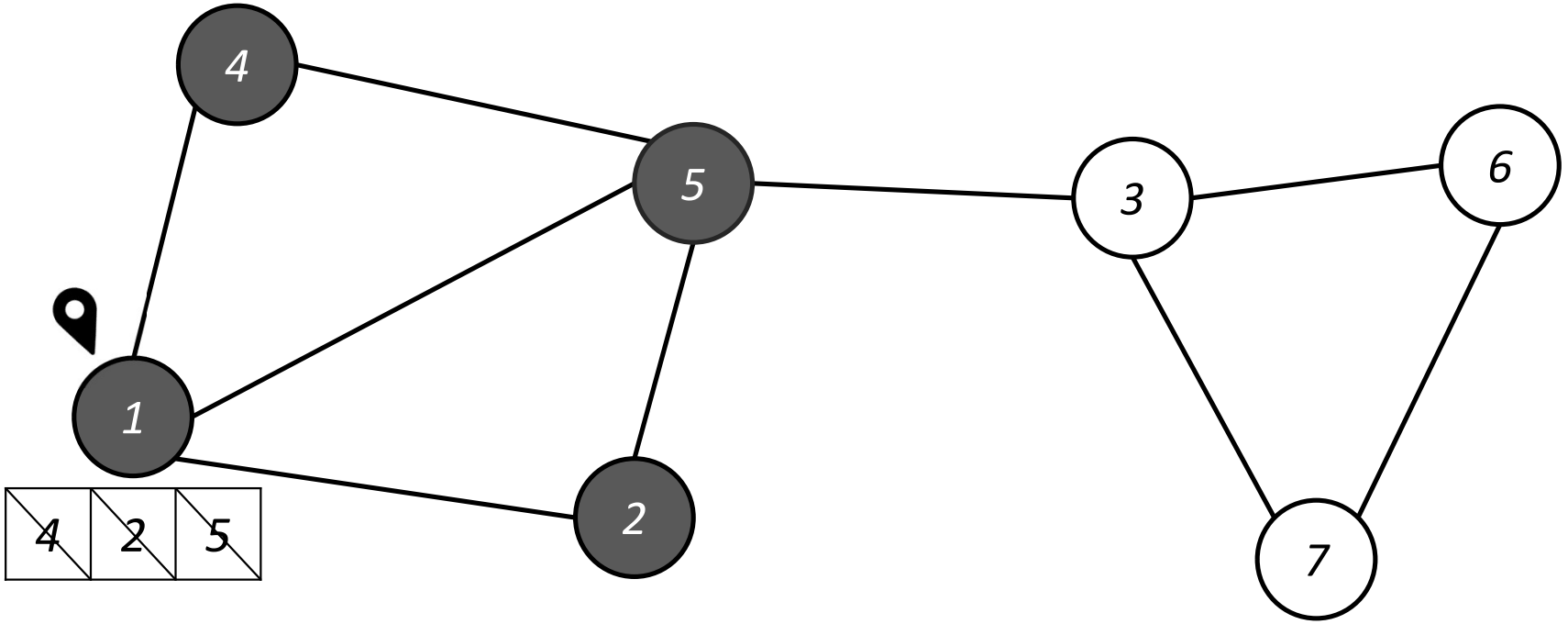
I 깊이 우선 탐색(*DFS, Depth First Search*)



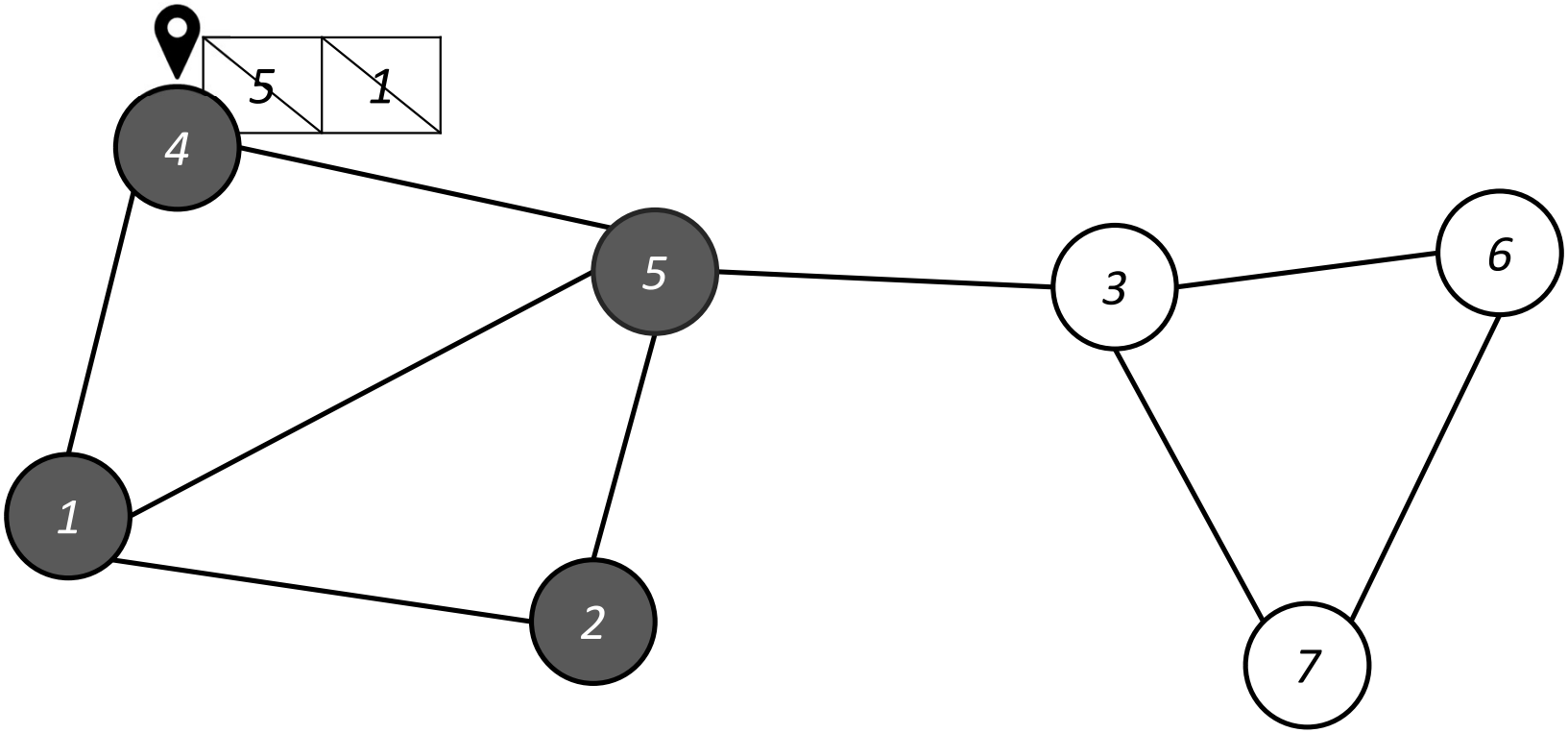
I 깊이 우선 탐색(*DFS, Depth First Search*)



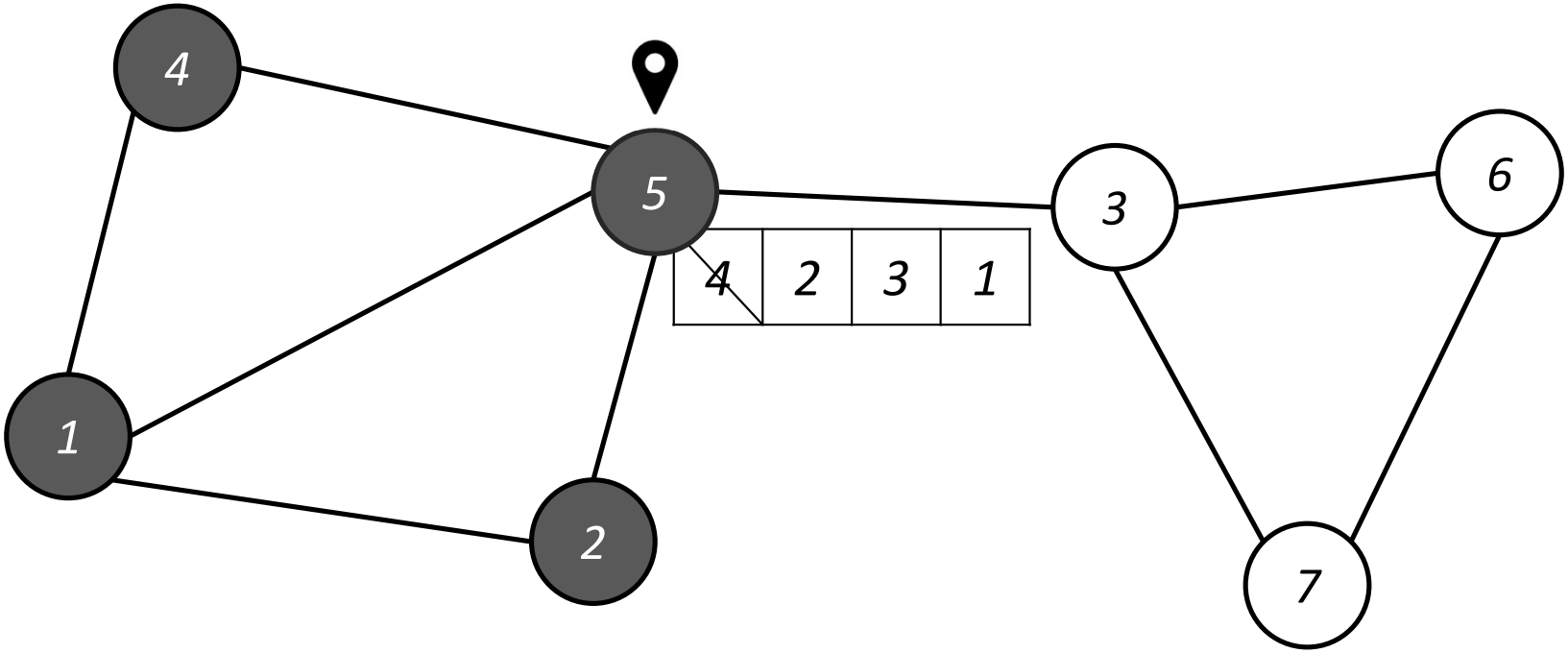
I 깊이 우선 탐색(*DFS, Depth First Search*)



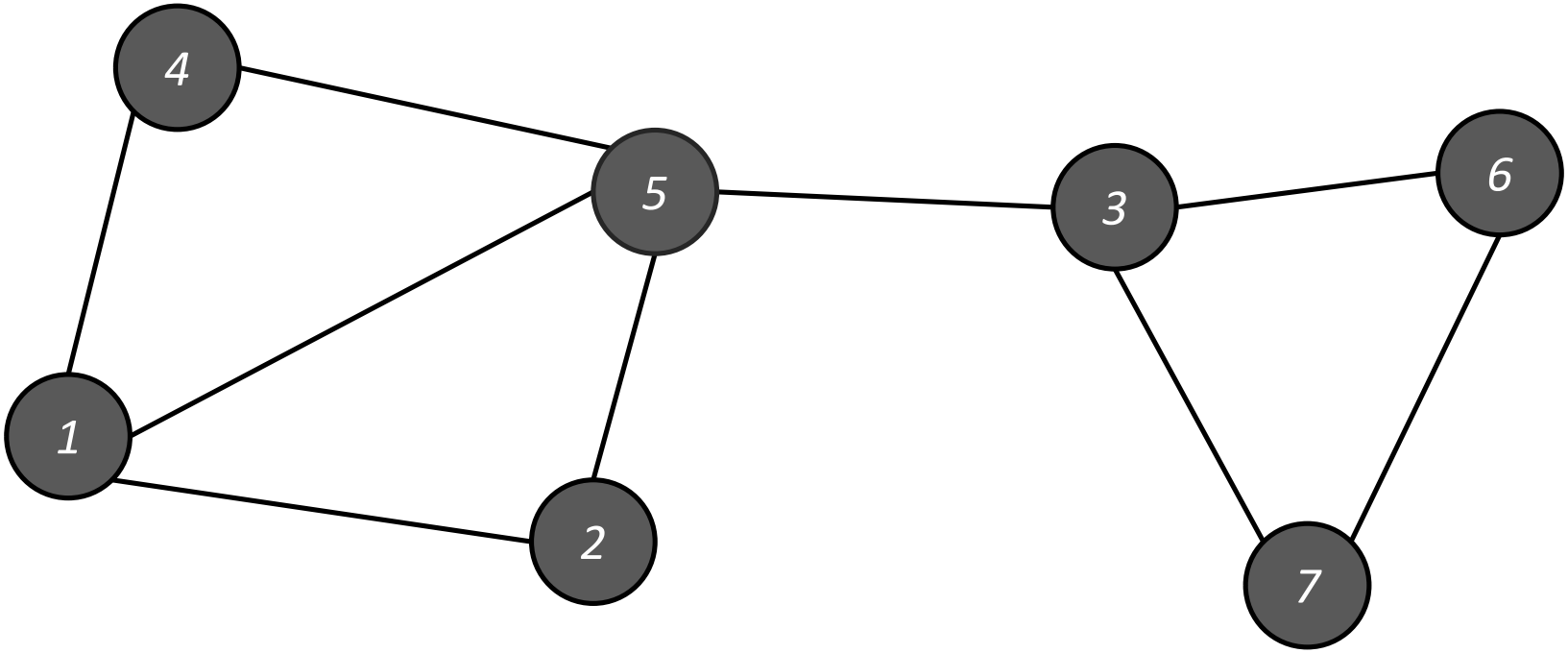
I 깊이 우선 탐색(*DFS, Depth First Search*)



I 깊이 우선 탐색(*DFS, Depth First Search*)



I 깊이 우선 탐색(*DFS, Depth First Search*)



I 깊이 우선 탐색(*DFS, Depth First Search*)

```
// x 를 갈 수 있다는 걸 알고 방문한 상태
static void dfs(int x){
    // x 를 방문했다.
    visit[x] = true;

    // x 에서 갈 수 있는 곳들을 모두 방문한다.
    for (int y: x 에서 갈 수 있는 점들){
        if (visit[y])  // y 를 이미 갈 수 있다는 사실을 안다면, 굳이 갈 필요 없다.
            continue;

        // y에서 갈 수 있는 곳들도 확인 해보자
        dfs(y);
    }
}

main(){
    dfs(5);
}
```

모든 정점이 x 로 한 번씩만 등장한다. $O(V)$

인접 행렬 $O(V)$ / 인접 리스트 $O(\deg(x))$

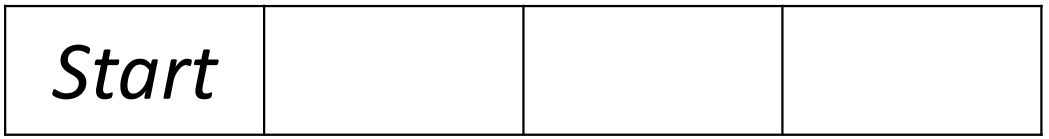
인접 행렬 ➔ $O(V^2)$

인접 리스트 ➔ $O(\deg(1) + \deg(2) + \cdots + \deg(V)) = O(E)$

I 너비 우선 탐색(*BFS, Breadth First Search*)

```java
// start 에서 시작해서 갈 수 있는 정점들을 모두 탐색하기
static void bfs(int start) {
    Queue<Integer> que = new LinkedList<>();

    // start는 방문 가능한 점이므로 que에 넣어준다.
    que.add(start);
    visit[start] = true;  // start를 갈 수 있다고 표시하기 (중요!!!)

    while (!que.isEmpty()) {  // 더 확인할 점이 없다면 정지
        int x = que.poll();

        for (int y: x 에서 갈 수 있는 점들){
            if (visit[y]) continue;  // x 에서 y 를 갈 수는 있지만, 이미 탐색한 점이면 무시

            // y를 갈 수 있으니까 que에 추가하고, visit 처리 하기!
            que.add(y);
            visit[y] = true;
        }
    }
}
```

I 너비 우선 탐색(*BFS, Breadth First Search*)



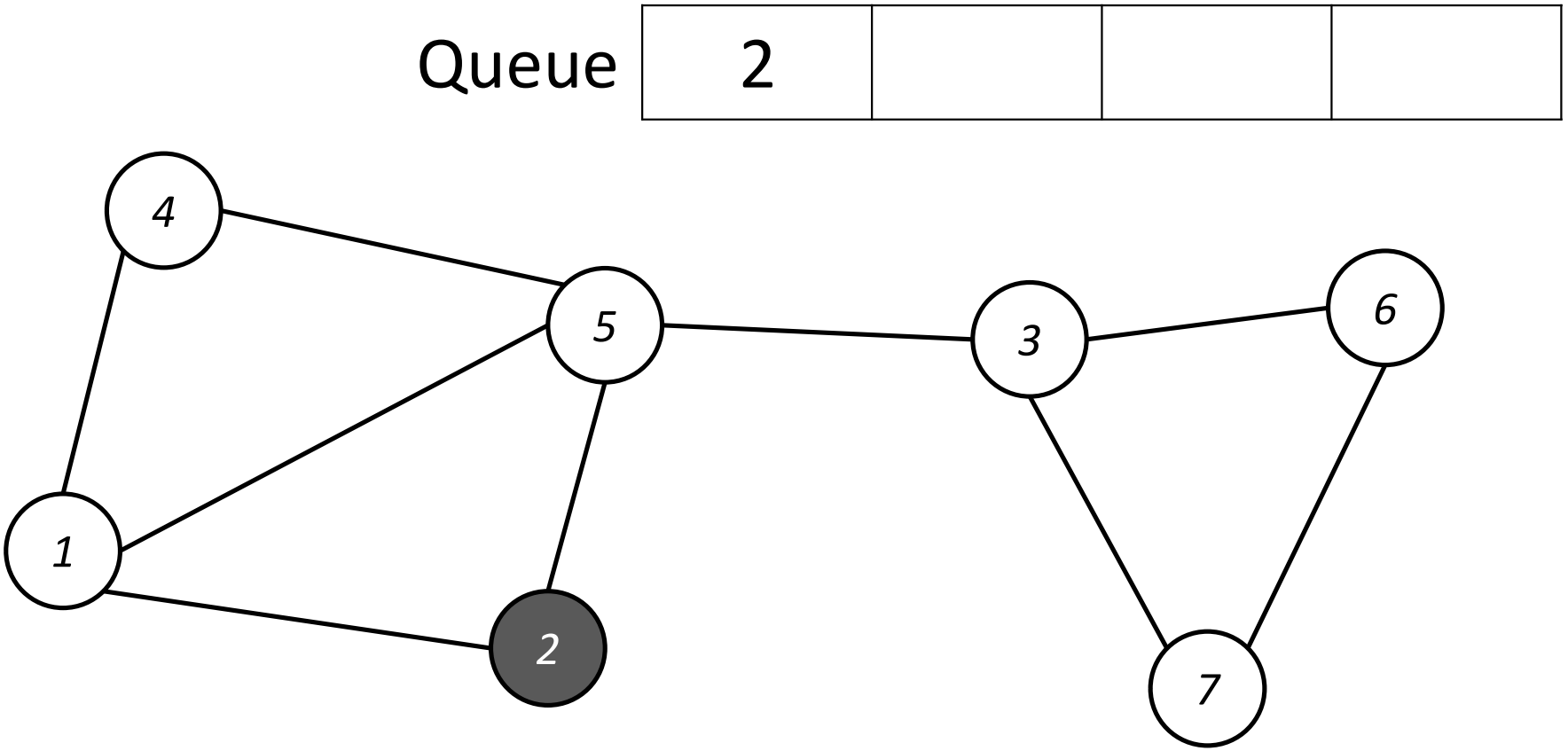
<Queue가 들고 있는 자료의 의미>

방문이 가능한 정점들을 찾을 때, Queue에 해당 정점을 넣는다.

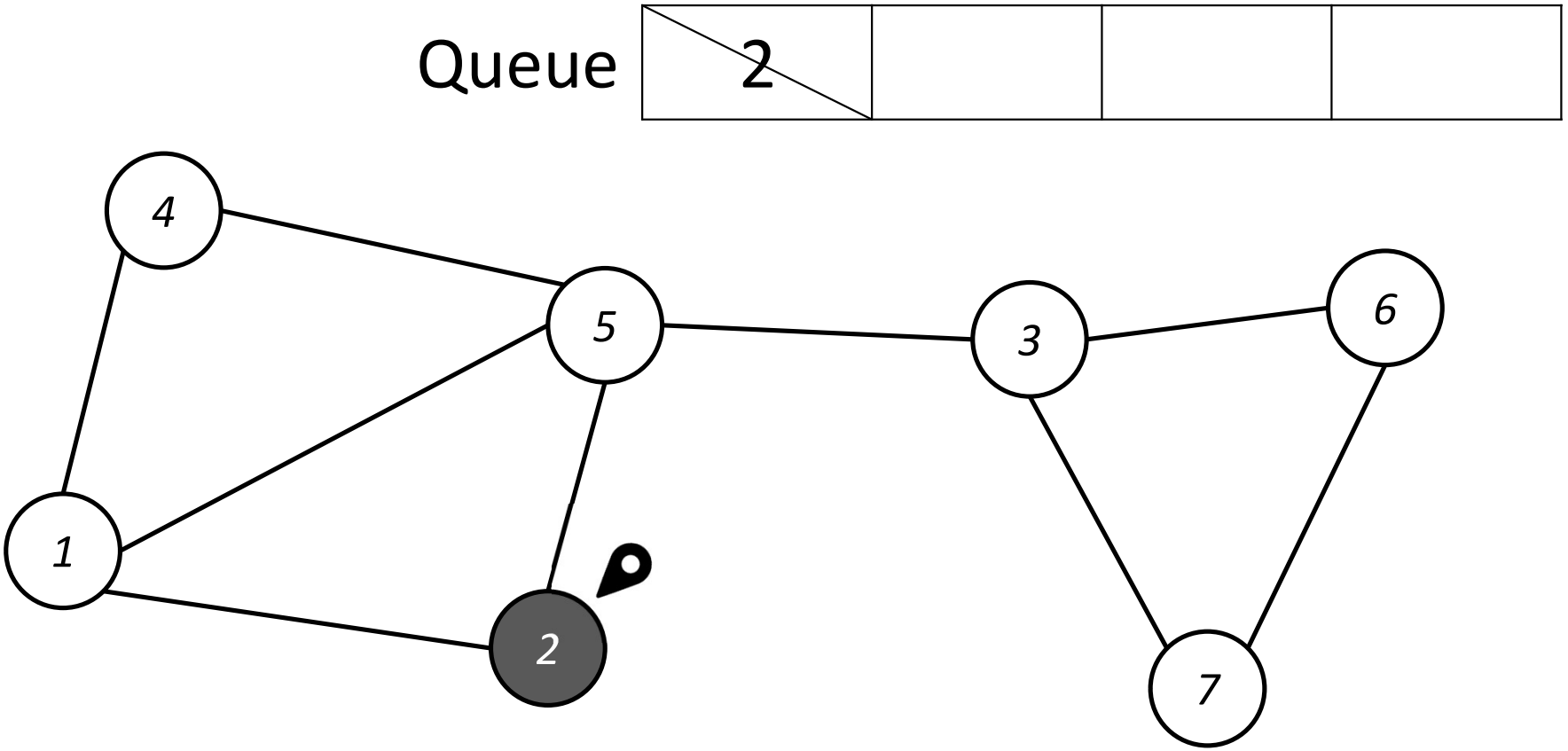
Queue에 정점이 남았다 ➔ 아직 방문 가능한 점이 남아있다. or 탐색 중이다.

Queue가 비어있다 ➔ 시작점에서 갈 수 있는 모든 점을 찾아냈다! or 탐색이 끝났다!

I 너비 우선 탐색(*BFS, Breadth First Search*)

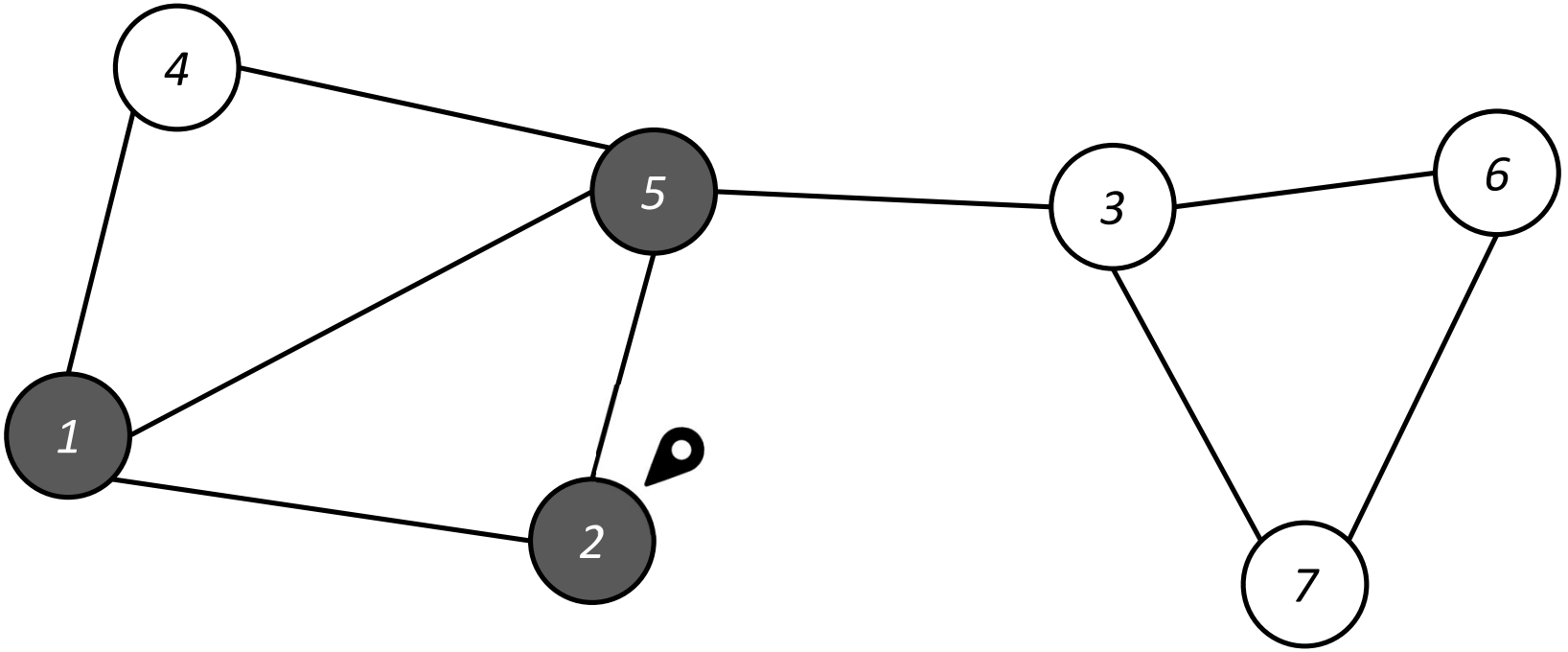


I 너비 우선 탐색(*BFS, Breadth First Search*)

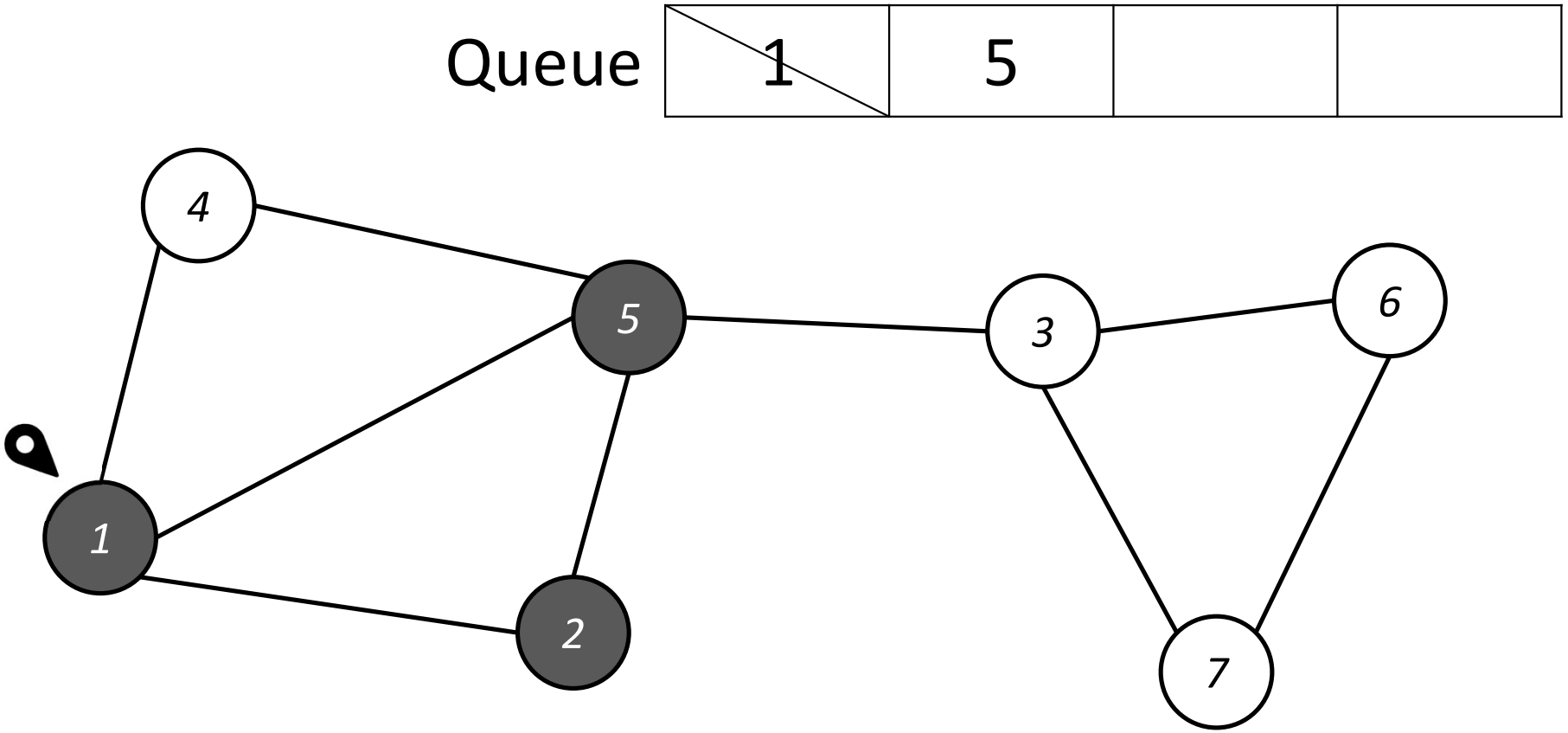


I 너비 우선 탐색(*BFS, Breadth First Search*)

Queue	1	5		

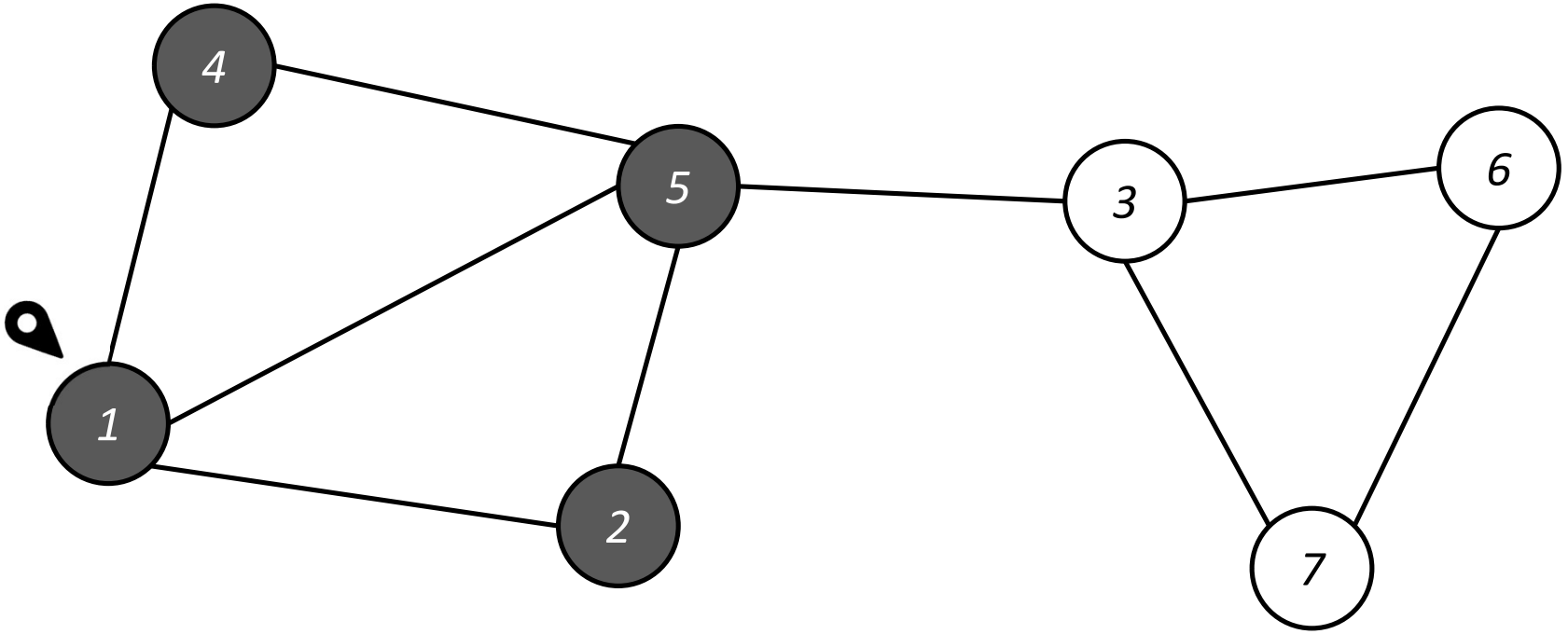


I 너비 우선 탐색(*BFS, Breadth First Search*)

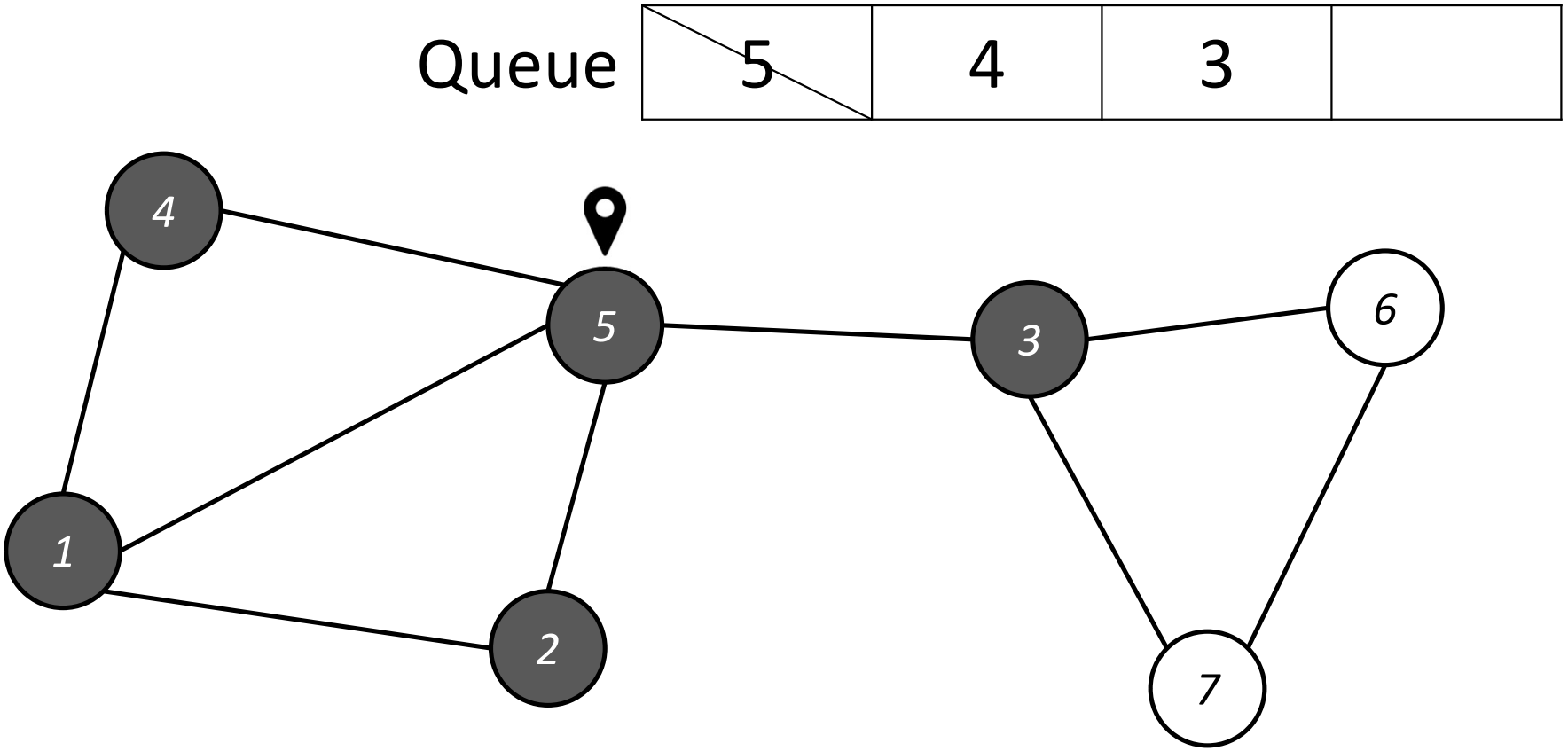


I 너비 우선 탐색(*BFS, Breadth First Search*)

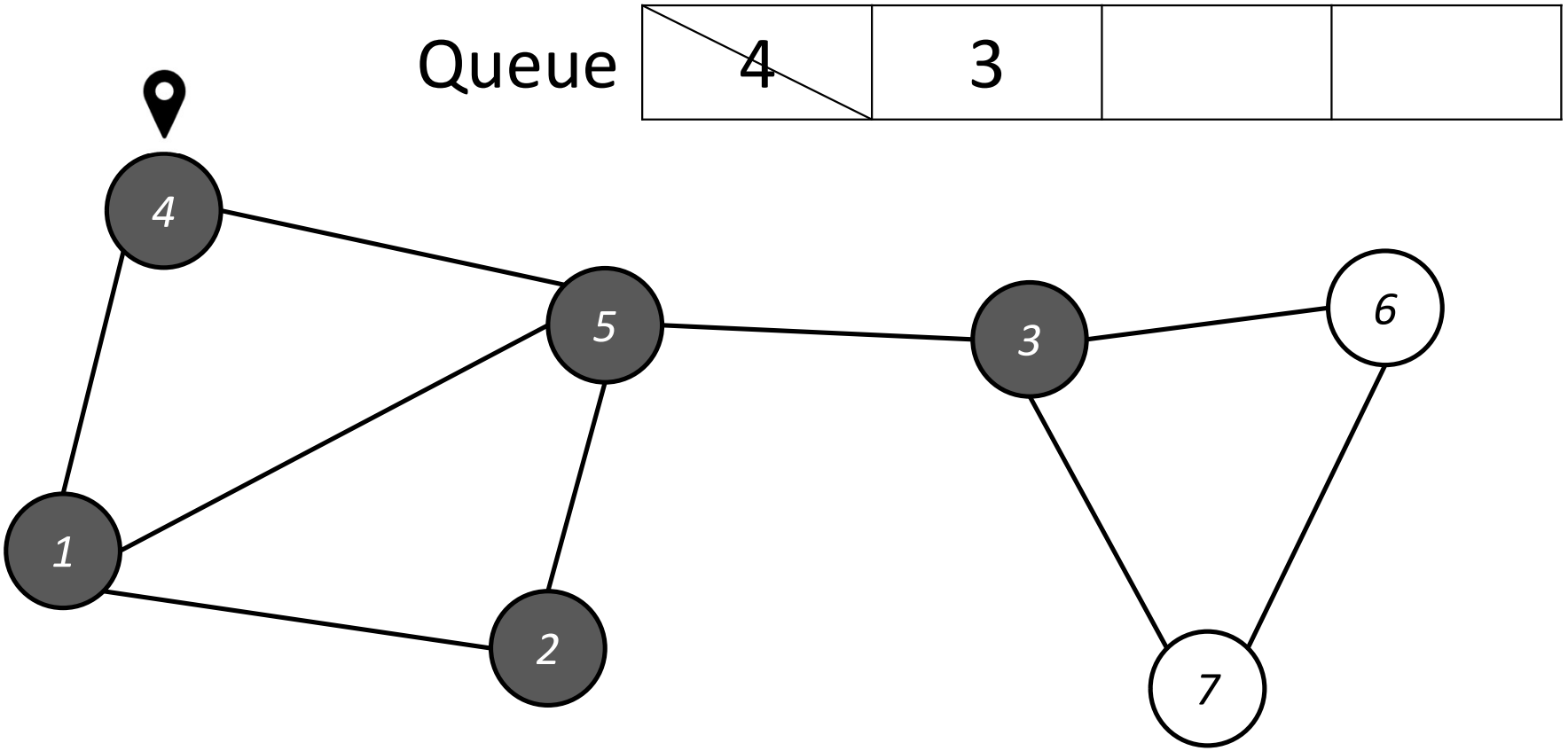
Queue	5	4		



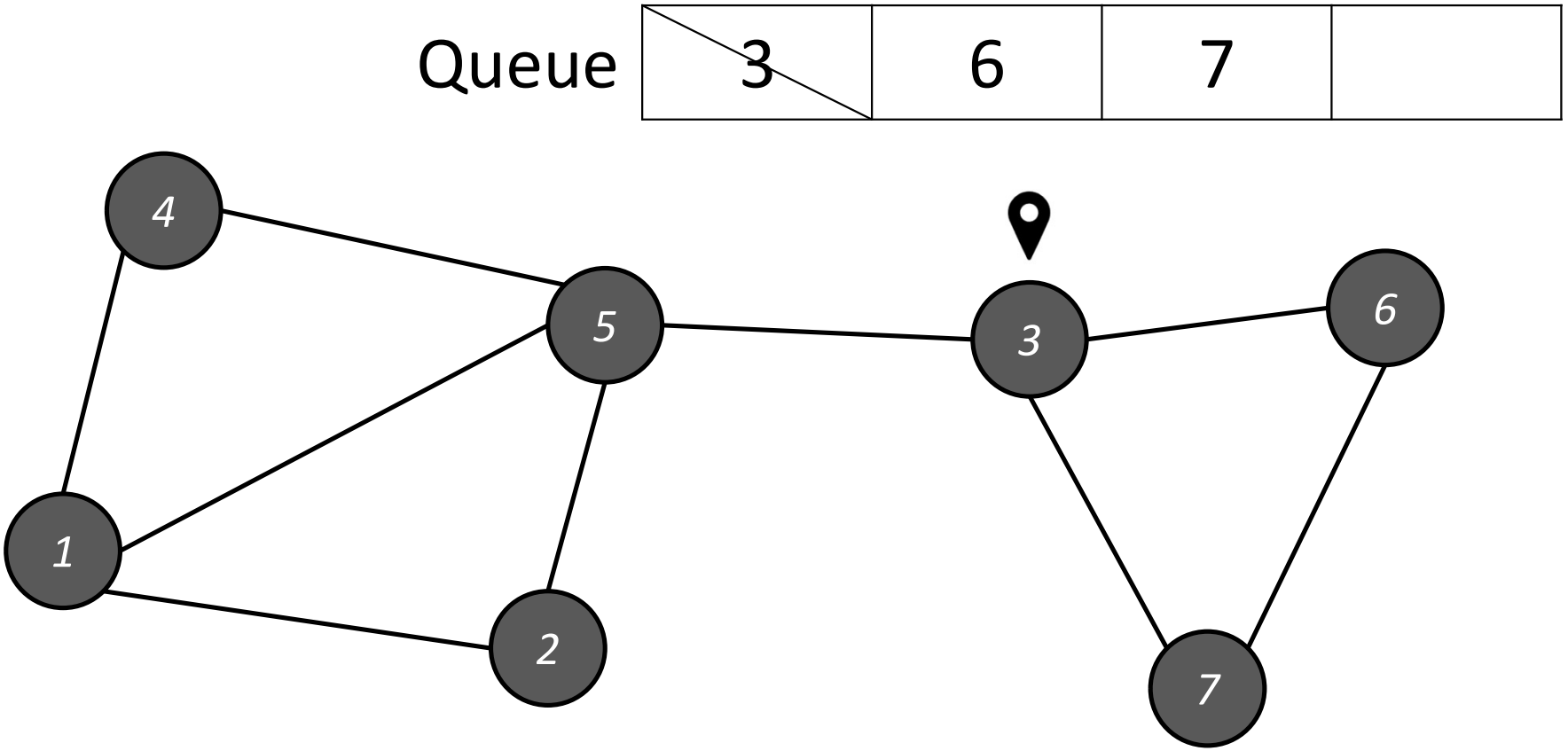
I 너비 우선 탐색(*BFS, Breadth First Search*)



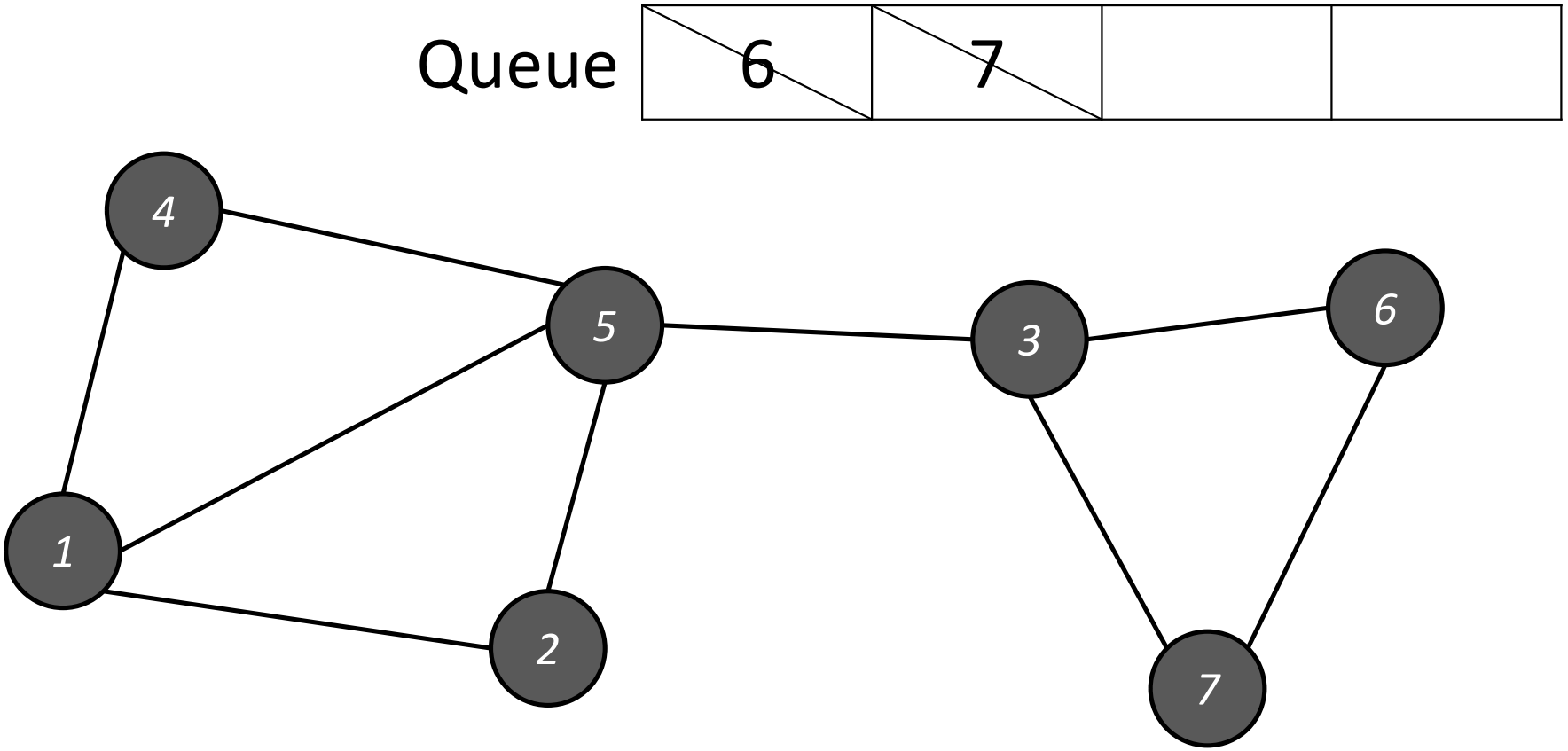
I 너비 우선 탐색(*BFS, Breadth First Search*)



I 너비 우선 탐색(*BFS, Breadth First Search*)



I 너비 우선 탐색(*BFS, Breadth First Search*)



I 너비 우선 탐색(*BFS, Breadth First Search*)

```java
// start 에서 시작해서 갈 수 있는 정점들을 모두 탐색하기
static void bfs(int start) {
    Queue<Integer> que = new LinkedList<>();

    // start는 방문 가능한 점이므로 que에 넣어준다.
    que.add(start);
    visit[start] = true;  // start를 갈 수 있다고 표시하기 (중요!!!)

    while (!que.isEmpty()) {  // 더 확인할 점이 없다면 정지
        int x = que.poll();

        for (int y: x 에서 갈 수 있는 점들){
            if (visit[y]) continue;  // x 에서 y 를 갈 수는 있지만, 이미 탐색한 점이면

            // y를 갈 수 있으니까 que에 추가하고, visit 처리 하기!
            que.add(y);
            visit[y] = true;
        }
    }
}
```

모든 정점이 x 로 한 번씩만 등장한다. $O(V)$

인접 행렬 $O(V)$ / 인접 리스트 $O(\deg(x))$

인접 행렬 ➔ $O(V^2)$

인접 리스트 ➔ $O(\deg(1) + \deg(2) + \cdots + \deg(V)) = O(E)$

BOJ 1260 – DFS와 BFS

난이도: 2

$1 \leq$ 정점 개수, $N < 1{,}000$

$1 \leq$ 간선 개수, $M \leq 10{,}000$

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.

I 접근 – 유일한 차이

단, 방문할 수 있는 정점이 여러 개인 경우에는 정점 번호가 작은 것을 먼저 방문하고,

adj	*1*	*2*	*3*	*4*	*5*
1	0	0	1	1	1
2	0	0	0	0	1
3	1	0	0	0	0
4	1	0	0	0	1
5	1	1	0	1	0

adj			
1	5	3	4
2	5		
3	1		
4	1	5	
5	2	1	4

입력 순서대로 저장하면 작은 번호부터 보기 위해 많은 시간이 필요!
$O(\deg(x)^2)$

I 접근 – 유일한 차이

단, 방문할 수 있는 정점이 여러 개인 경우에는 정점 번호가 작은 것을 먼저 방문하고,

adj	1	2	3	4	5
1	0	0	1	1	1
2	0	0	0	0	1
3	1	0	0	0	0
4	1	0	0	0	1
5	1	1	0	1	0

adj			
1	3	4	5
2	5		
3	1		
4	1	5	
5	1	2	4

만약 초기에 정렬을 해 놓는다면?

$O(\deg(x)\log(\deg(x)))$

I 시간, 공간 복잡도 계산하기

<인접 행렬>

시간: $O(V\text{^}2)$
공간: $O(V\text{^}2)$

<인접 리스트>
시간: $O(E \log E)$
공간: $O(E)$

I 구현

```java
// x 를 갈 수 있다는 걸 알고 방문한 상태
static void dfs(int x) {
    /* TODO */
}

// start 에서 시작해서 갈 수 있는 정점들을 모두 탐색하기
static void bfs(int start) {
    Queue<Integer> que = new LinkedList<>();
    /* TODO */
}

static void pro() {
    // 모든 x에 대해서 adj[x] 정렬하기
    /* TODO */

    // DFS, BFS 결과 구하기
    /* TODO */

    System.out.println(sb);
}
```