

그래프 1

최백준 choi@startlink.io



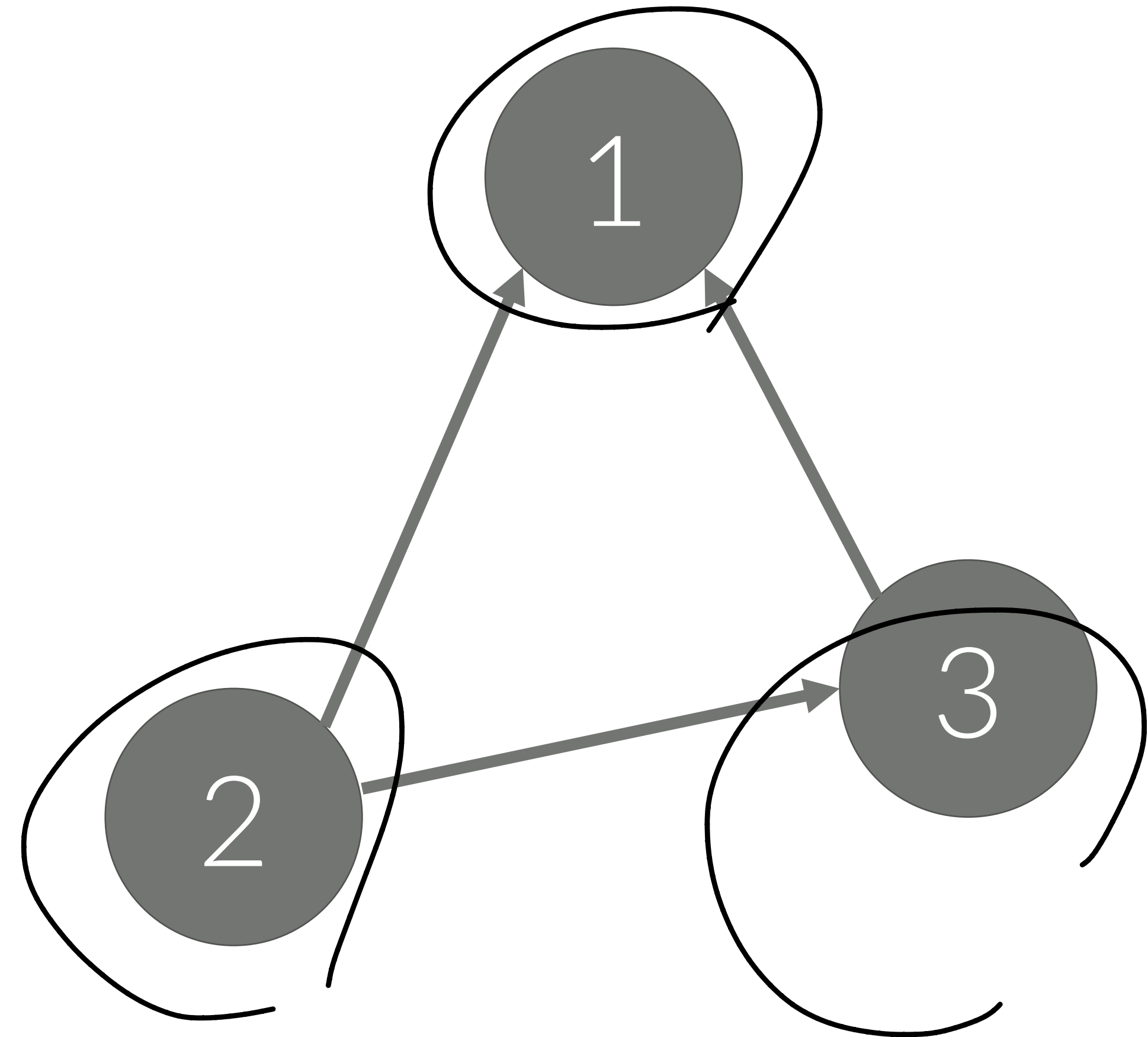
그래프

그래프

Graph

3

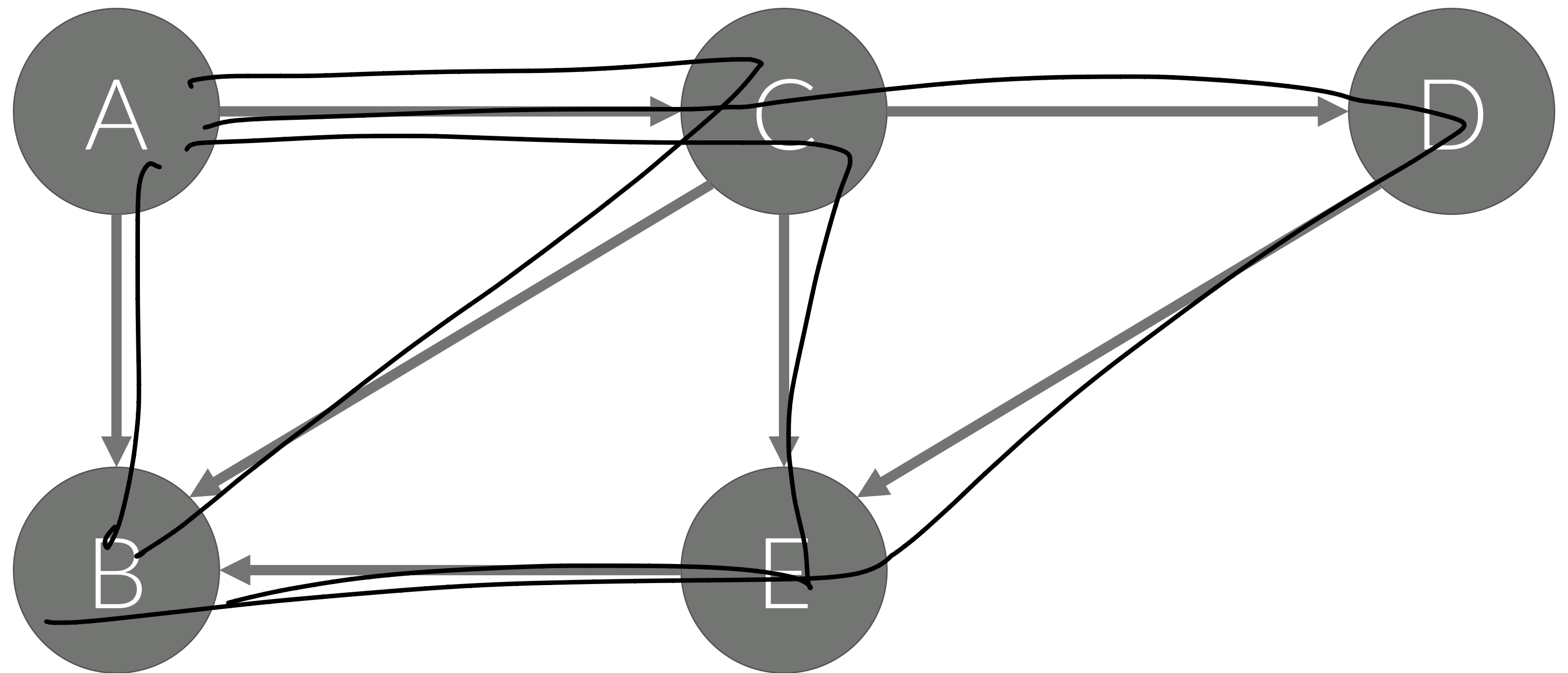
- 자료구조의 일종
- 정점 (Node, Vertex)
- 간선 (Edge): 정점간의 관계를 나타낸다.
- $G = (V, E)$ 로 나타낸다.



경로

Path

- 정점 A에서 B로 가는 경로
- $A \rightarrow C \rightarrow D \rightarrow E \rightarrow B$
- $A \rightarrow B$
- $A \rightarrow C \rightarrow B$
- $A \rightarrow C \rightarrow E \rightarrow B$

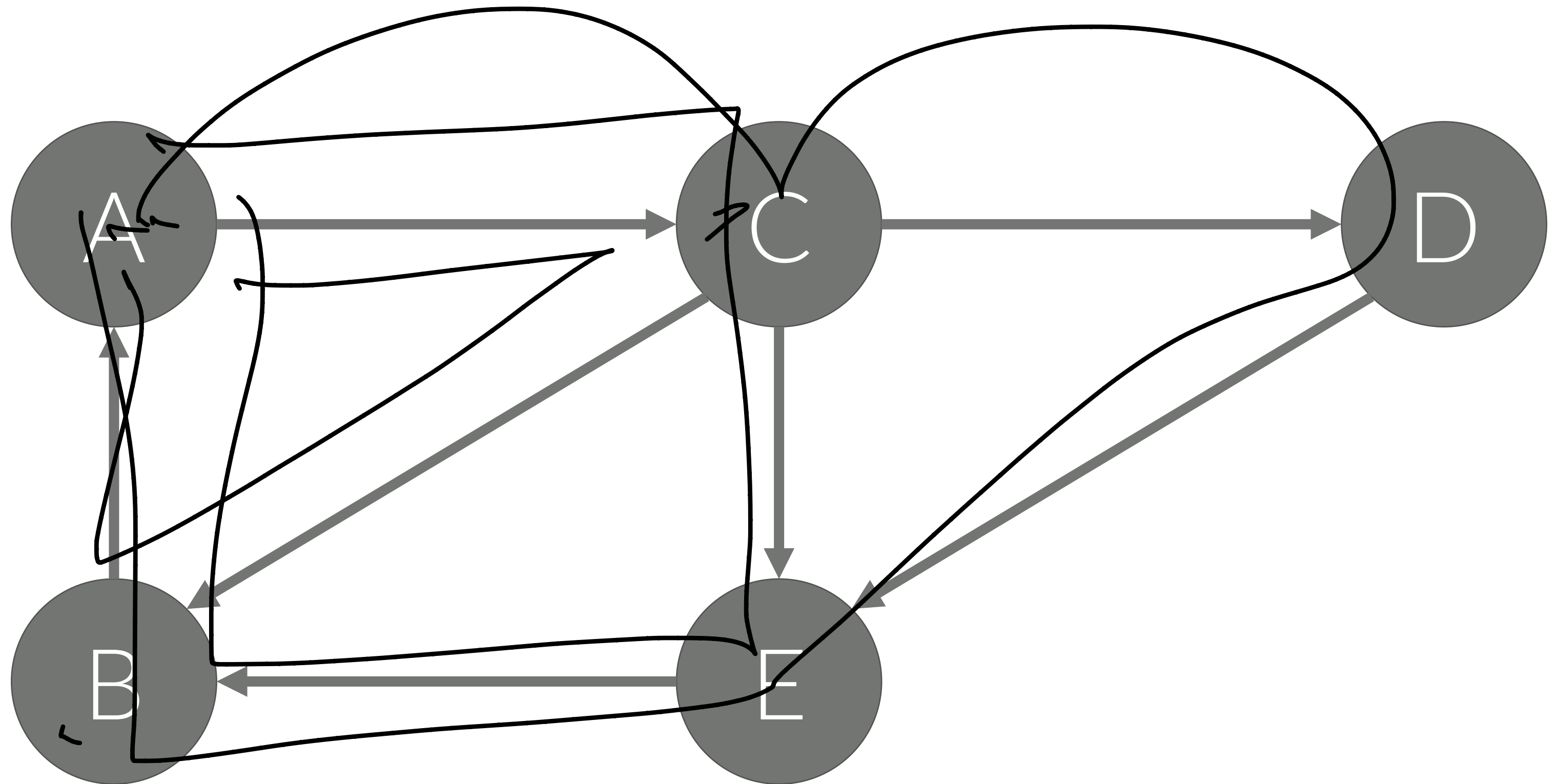


사이클

Path

정점 A에서 다시 A로 돌아오는 경로

- $A \rightarrow C \rightarrow B \rightarrow A$
- $A \rightarrow C \rightarrow E \rightarrow B \rightarrow A$
- $A \rightarrow C \rightarrow D \rightarrow E \rightarrow B \rightarrow A$



단순 경로와 단순 사이클

Simple Path and Simple Cycle

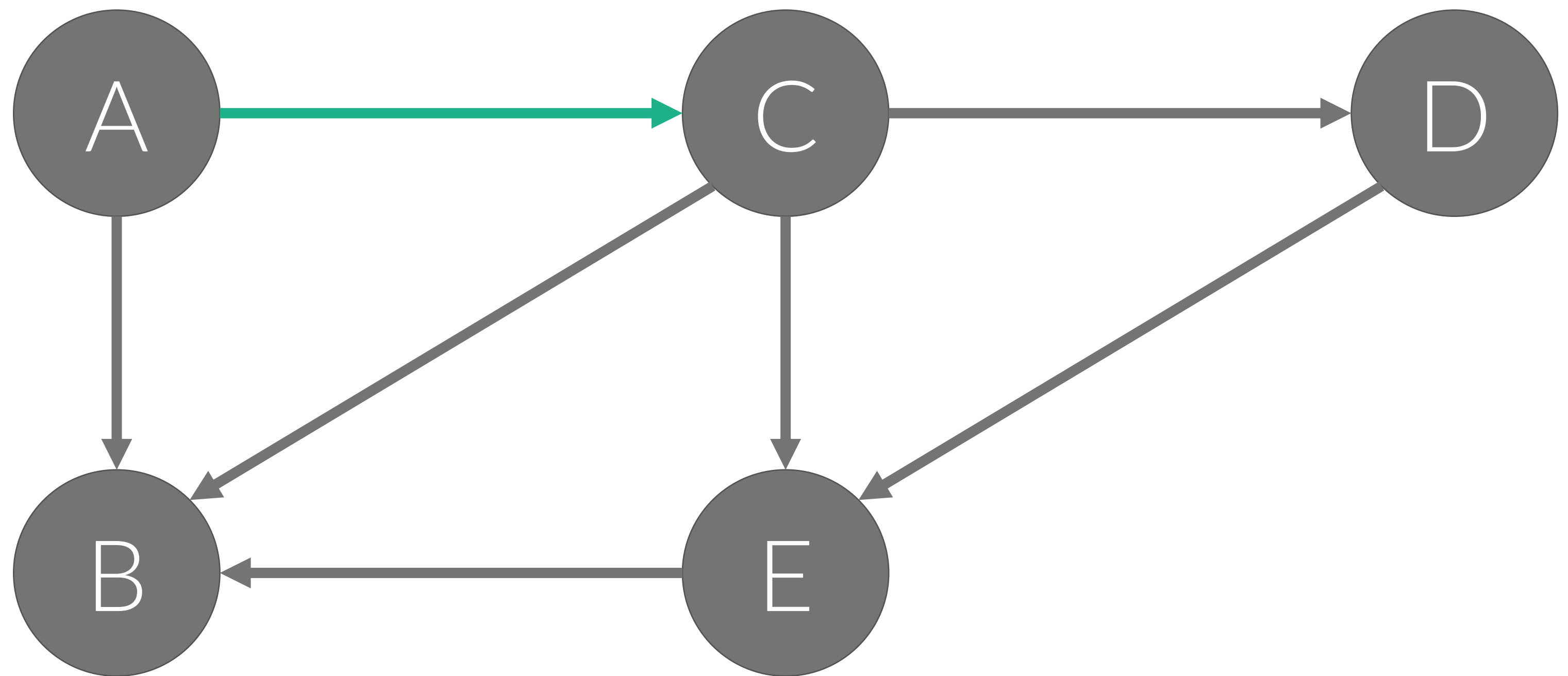
- 경로/사이클에서 같은 정점을 두 번 이상 방문하지 않는 경로/사이클
- 특별한 말이 없으면, 일반적으로 사용하는 경로와 사이클은 단순 경로/사이클을 말한다.

방향 있는 그래프

Directed Graph

- $A \rightarrow C$ 와 같이 간선에 방향이 있다.
- $A \rightarrow C$ 는 있지만, $C \rightarrow A$ 는 없다.

$A \rightarrow C$



방향 없는 그래프

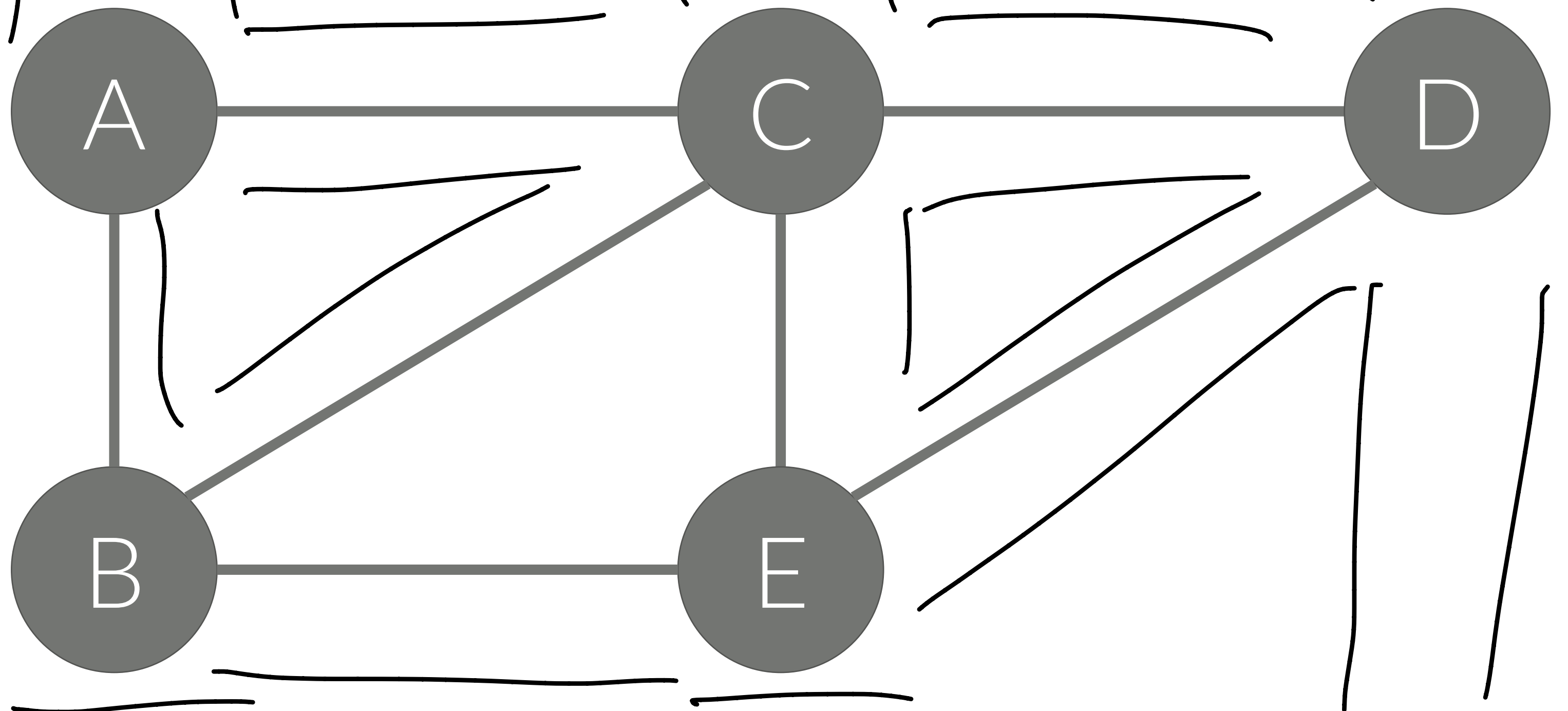
Undirected Graph

- $A - C$ 와 같이 간선에 방향이 없다.
- $A - C$ 는 $A \rightarrow C$ 와 $C \rightarrow A$ 를 나타낸다.
- 양방향 그래프 (Bidirection Graph) 라고도 한다.

$A - C$

$A \rightarrow C$

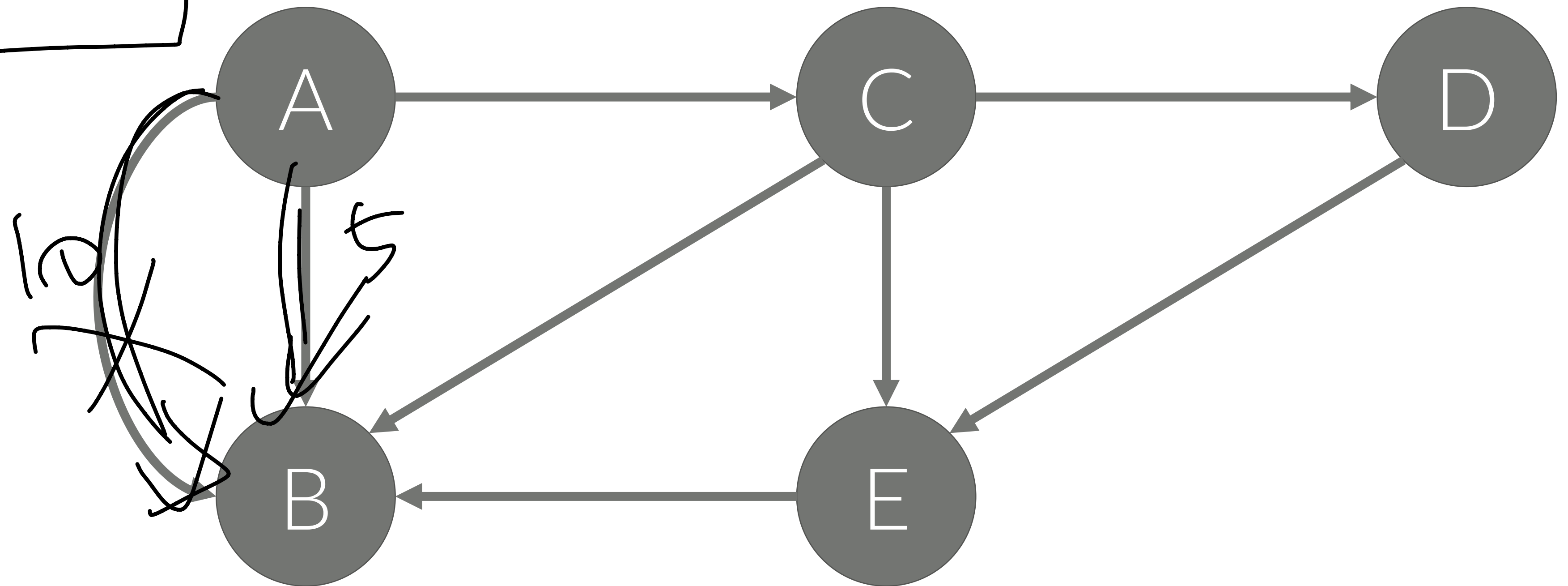
$C \rightarrow A$



간선 여러개

Multiple Edge

- 두 정점 사이에 간선이 여러 개일 수도 있다.
- 아래 그림의 A-B는 연결하는 간선이 2개이다.
- 두 간선은 서로 다른 간선이다

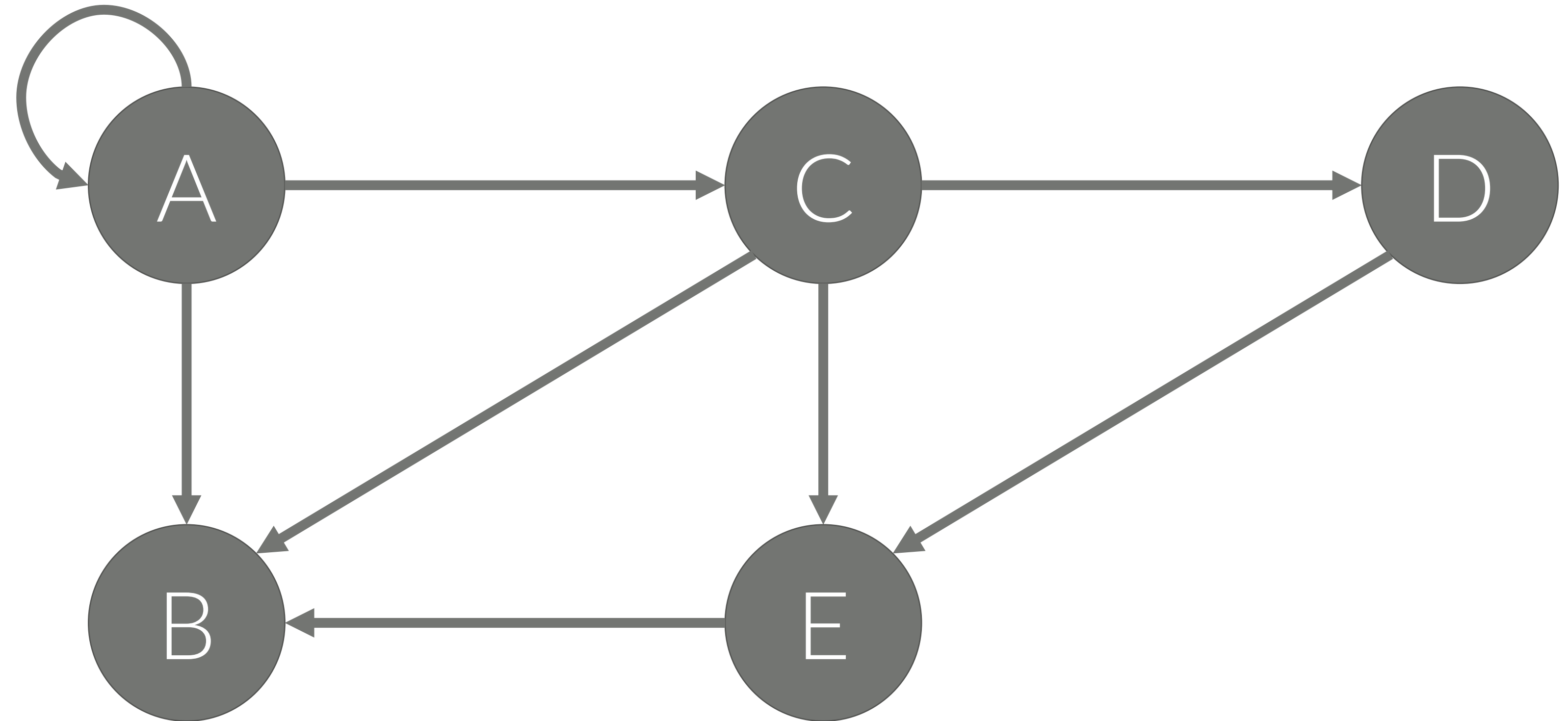


루프

10

Loop

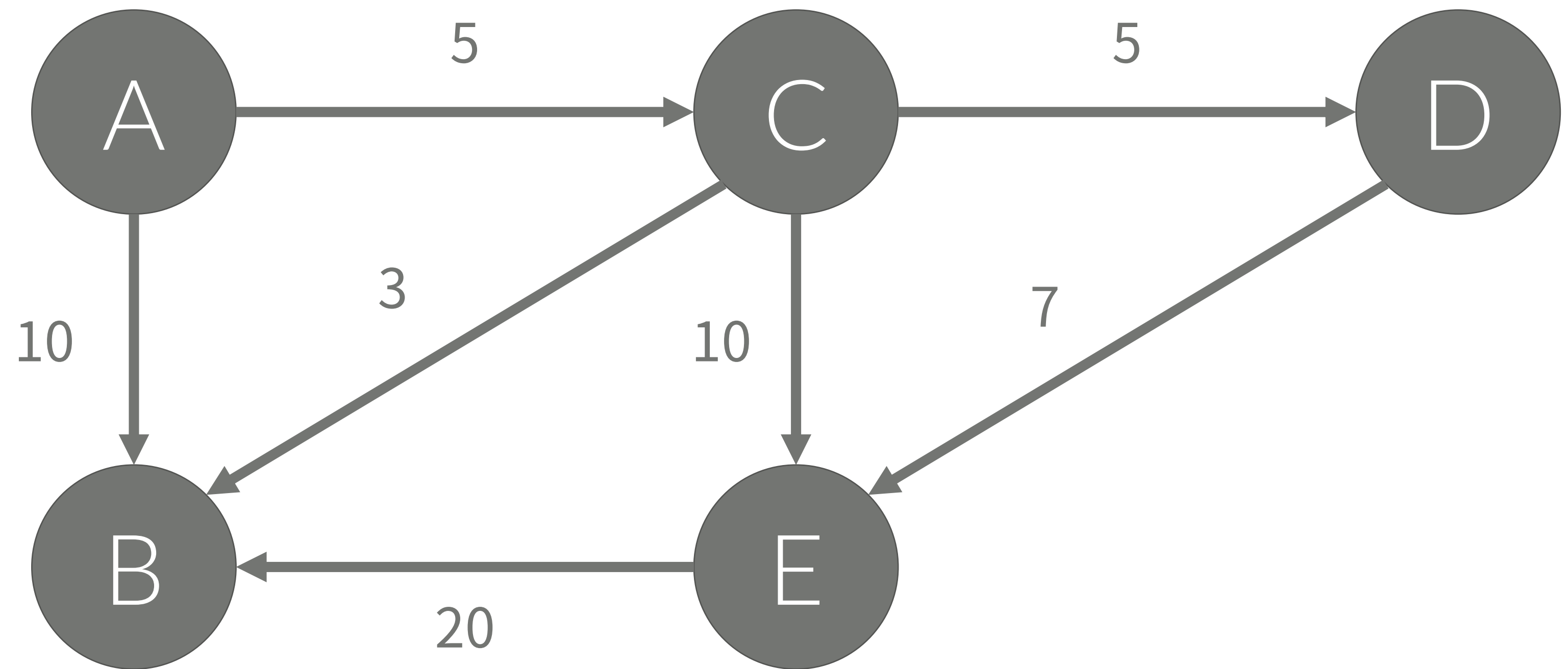
- 간선의 양 끝 점이 같은 경우가 있다.
- $A \rightarrow A$



가중치

Weight

- 간선에 가중치가 있는 경우에는
- A에서 B로 이동하는 거리, 이동하는데 필요한 시간, 이동하는데 필요한 비용 등등등...

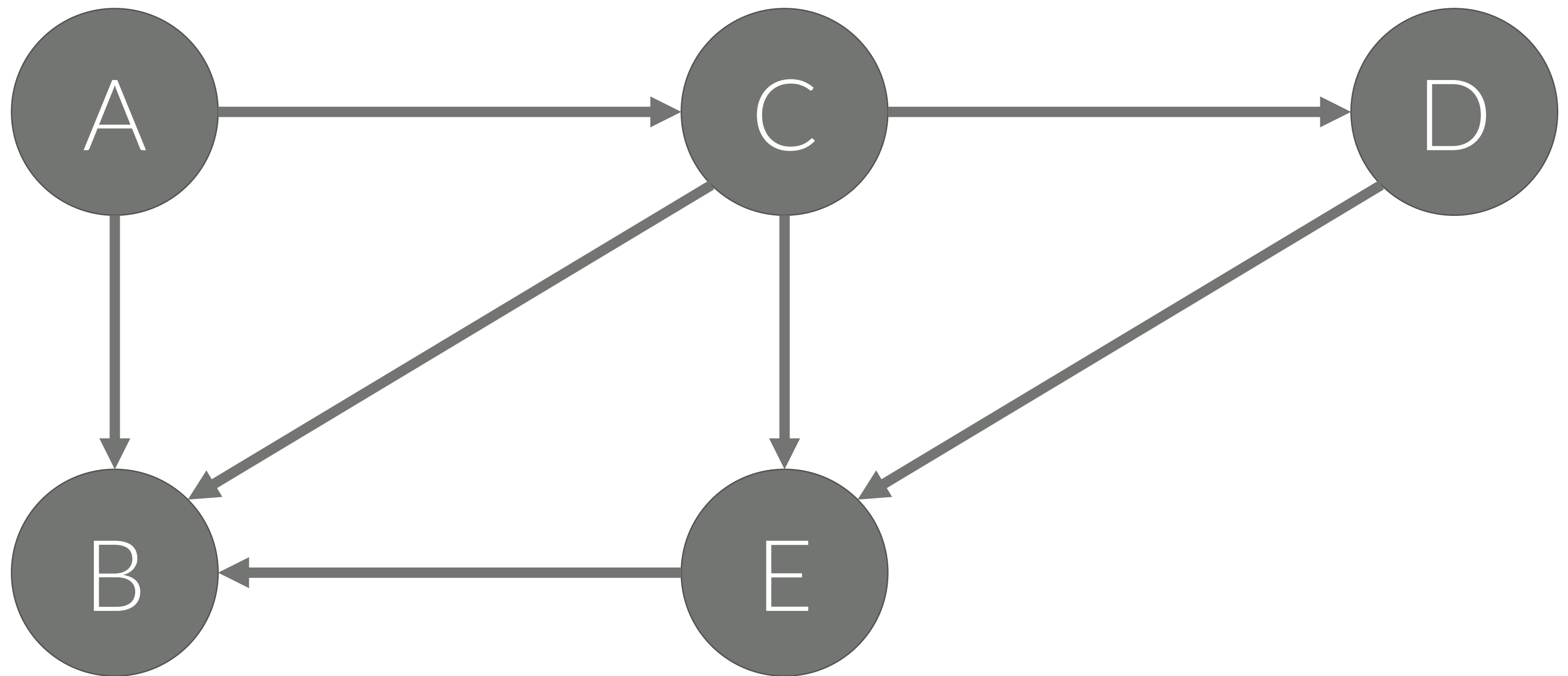


가중치

Weight

12

- 가중치가 없는 경우에는 1이라고 생각하면 된다

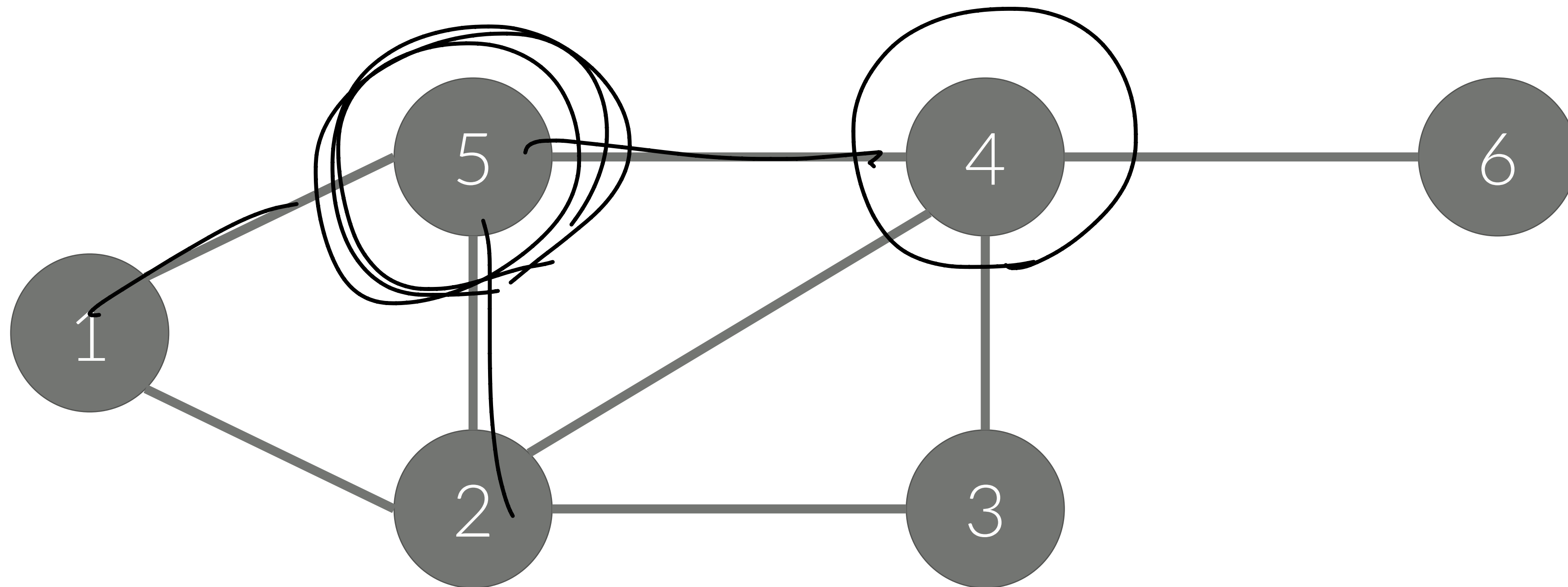


차수

Degree

13

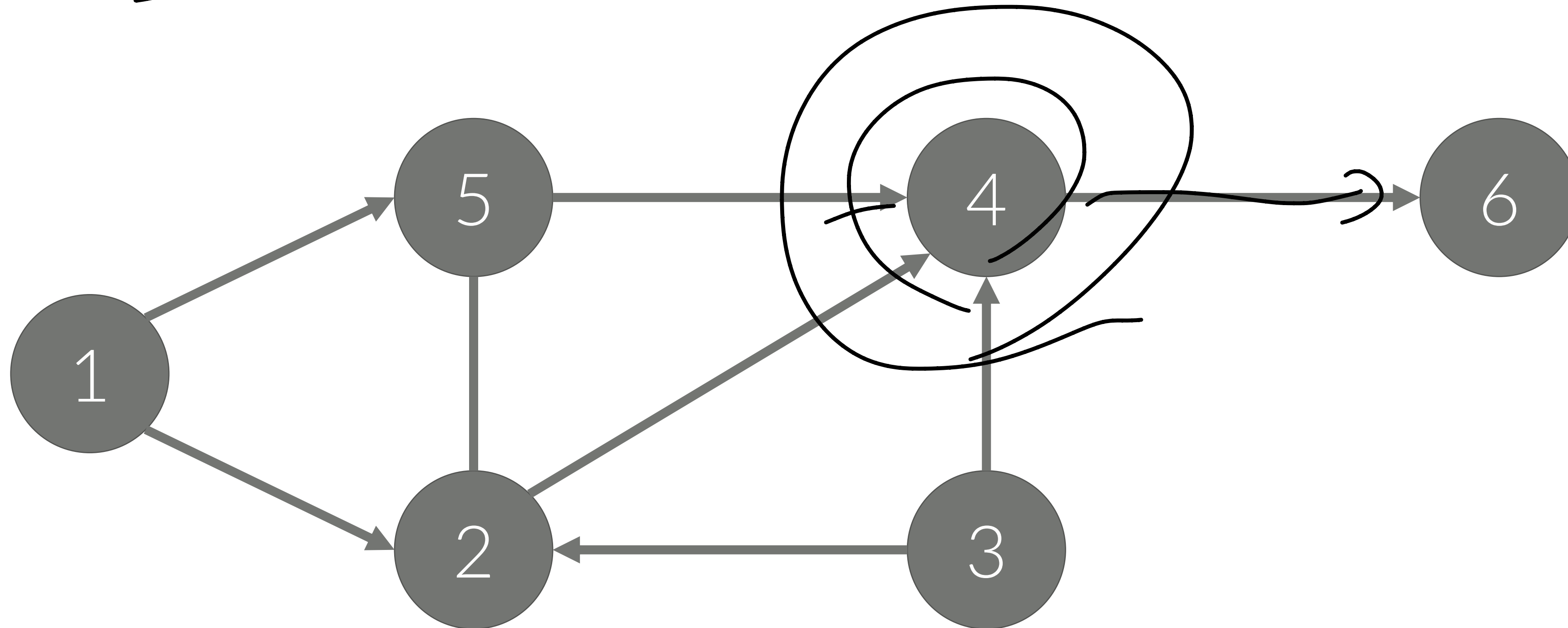
- 정점과 연결되어 있는 간선의 개수
- 5의 차수: 3
- 4의 차수: 4



차수

Degree

- 방향 그래프의 경우에는 In-degree, Out-degree로 나누어서 차수를 계산한다
- 4의 In-degree: 3
- 4의 Out-degree: 1



그래프 저장 방법

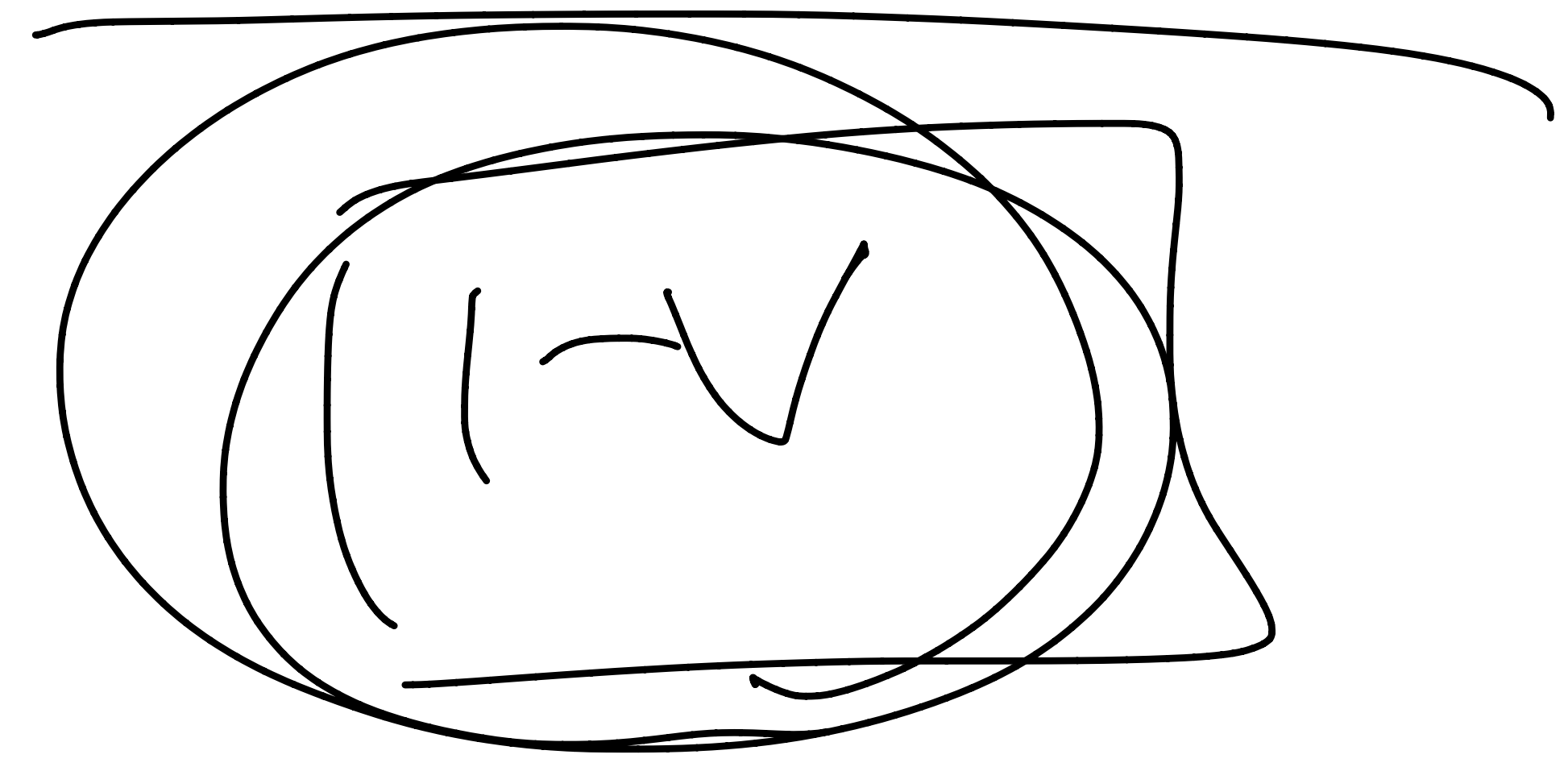
- ① 인접 행렬
- ② 인접 리스트

그래프의 표현

2022

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

한 정점 x 와 연결된 정점들
→
이웃 정점들



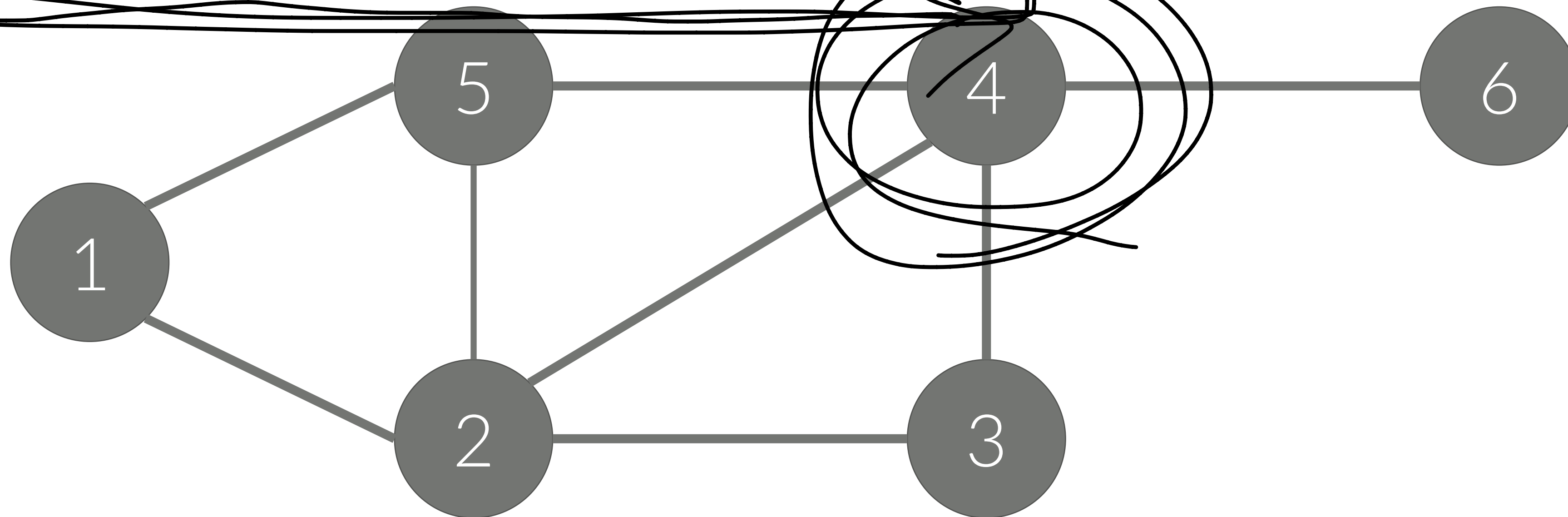
그래프의 표현

Representation of Graph

정점: 6개, 간선: 8개

16

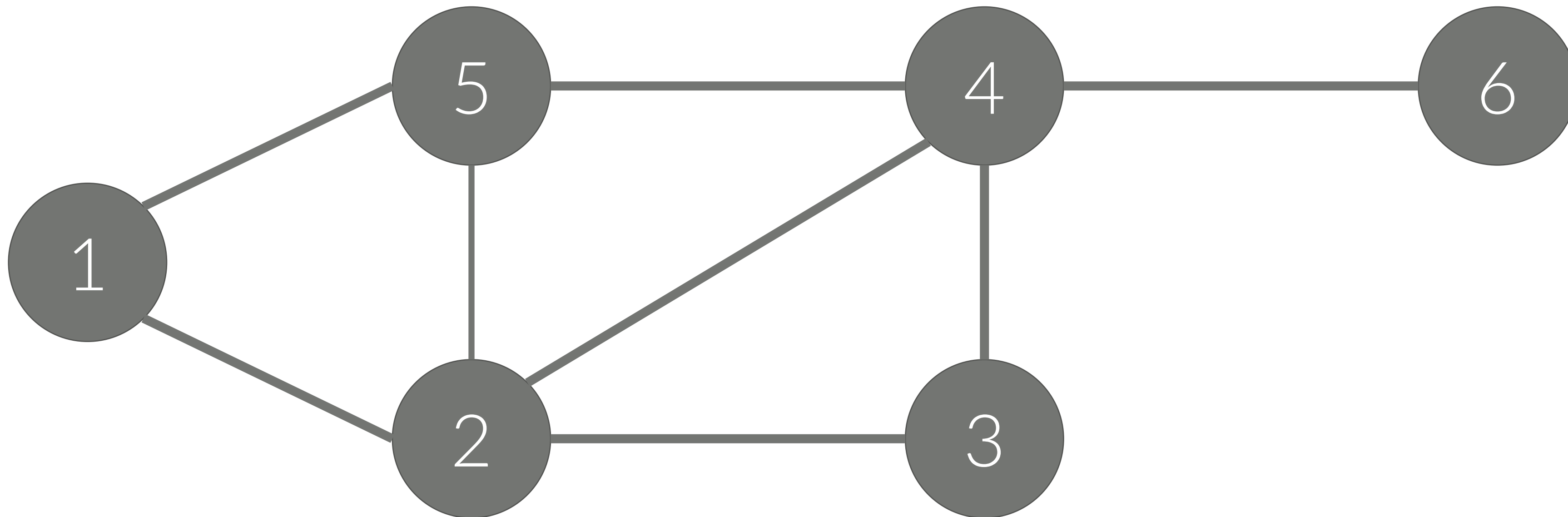
- 아래와 같은 그래프는 정점이 6개, 간선이 8개 있다.
- 간선에 방향이 없기 때문에, 방향이 없는 그래프이다.
- 정점: {1, 2, 3, 4, 5, 6}
- 간선: {(1, 2), (1, 5), (2, 5), (2, 3), (3, 4), (2, 4), (4, 5), (4, 6)}



인접 행렬

Adjacency-matrix

- 정점의 개수를 V 라고 했을 때
- $V \times V$ 크기의 이차원 배열을 이용한다
- $A[i][j] = 1$ ($i \rightarrow j$ 간선이 있을 때), 0 (없을 때)

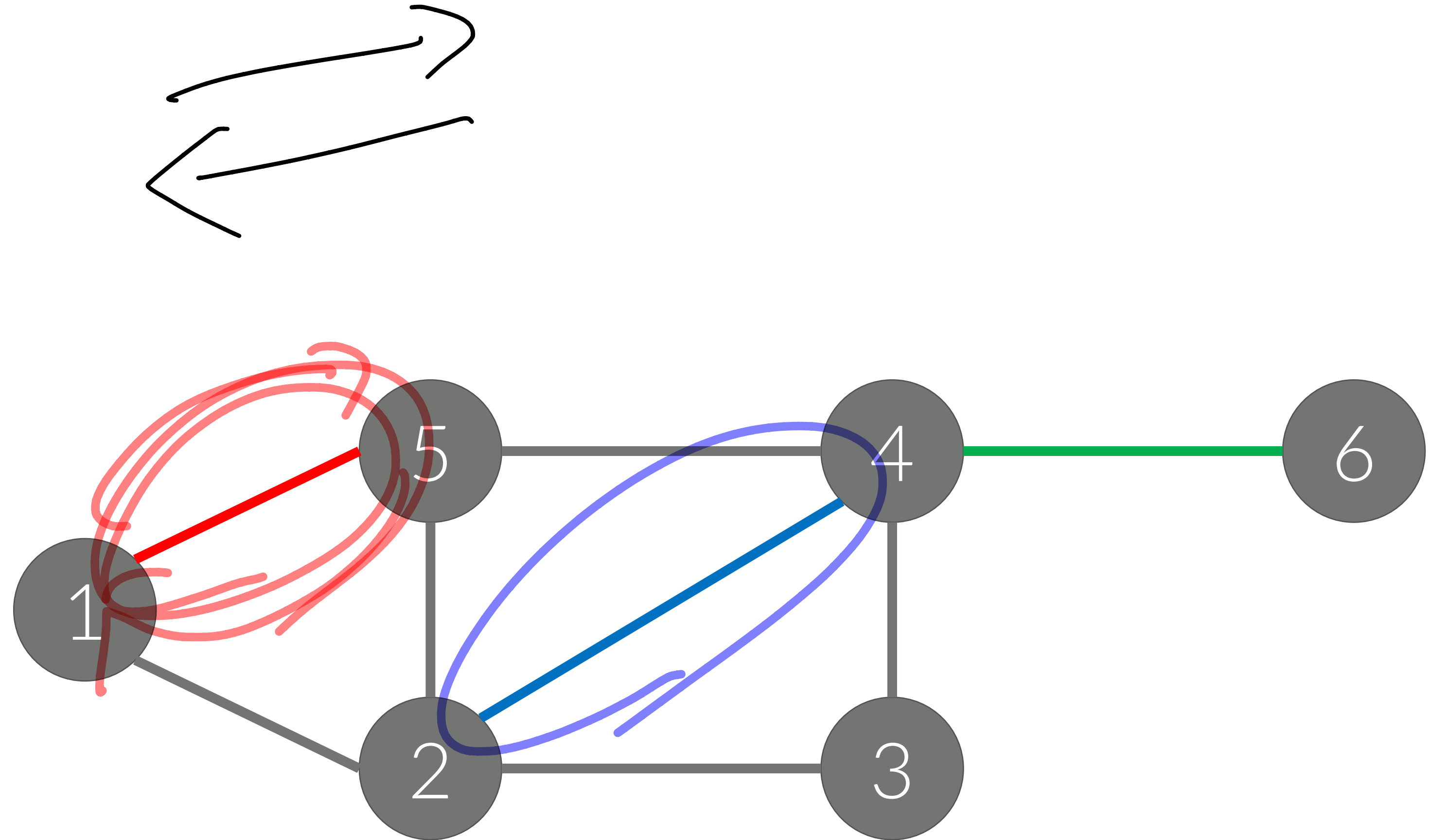


인접 행렬

Adjacency-matrix

18

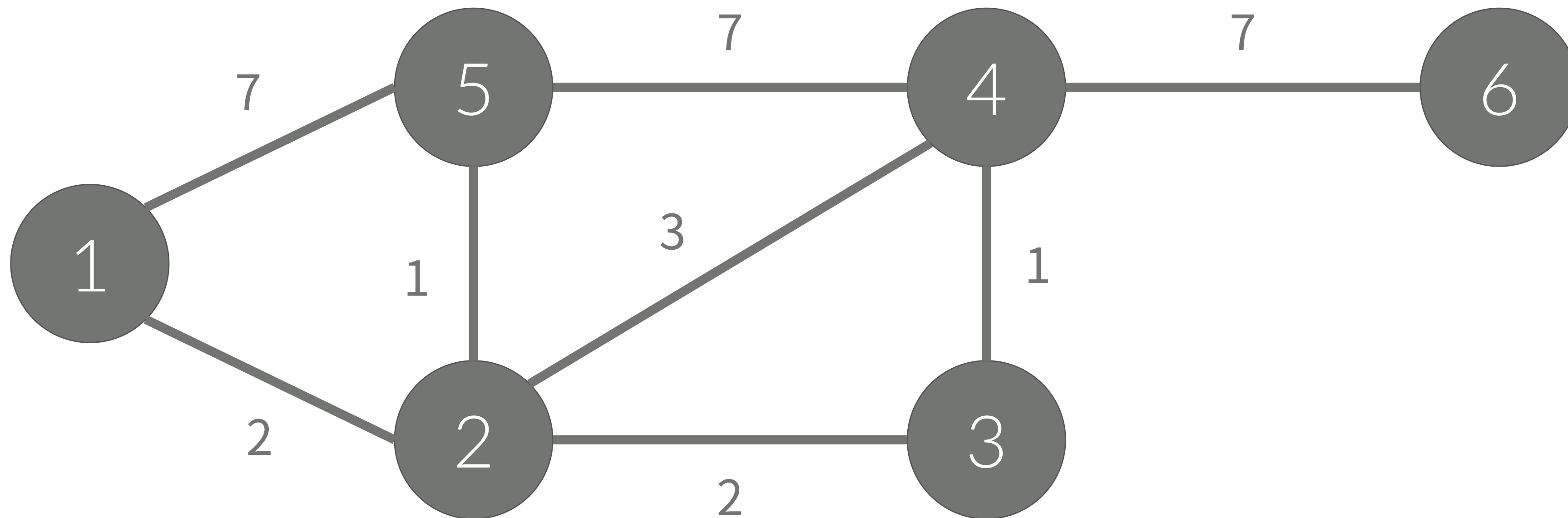
	1	2	3	4	5	6
1	0	1	0	0	1	0
2	1	0	1	1	1	0
3	0	1	0	1	0	0
4	0	1	1	0	1	1
5	1	1	0	1	0	0
6	0	0	0	1	0	0



인접 행렬

Adjacency-matrix

- 정점의 개수를 V 라고 했을 때
- $V \times V$ 크기의 이차원 배열을 이용한다
- $A[i][j] = w$ ($i \rightarrow j$ 간선이 있을 때, 그 가중치), 0 (없을 때)



인접 행렬

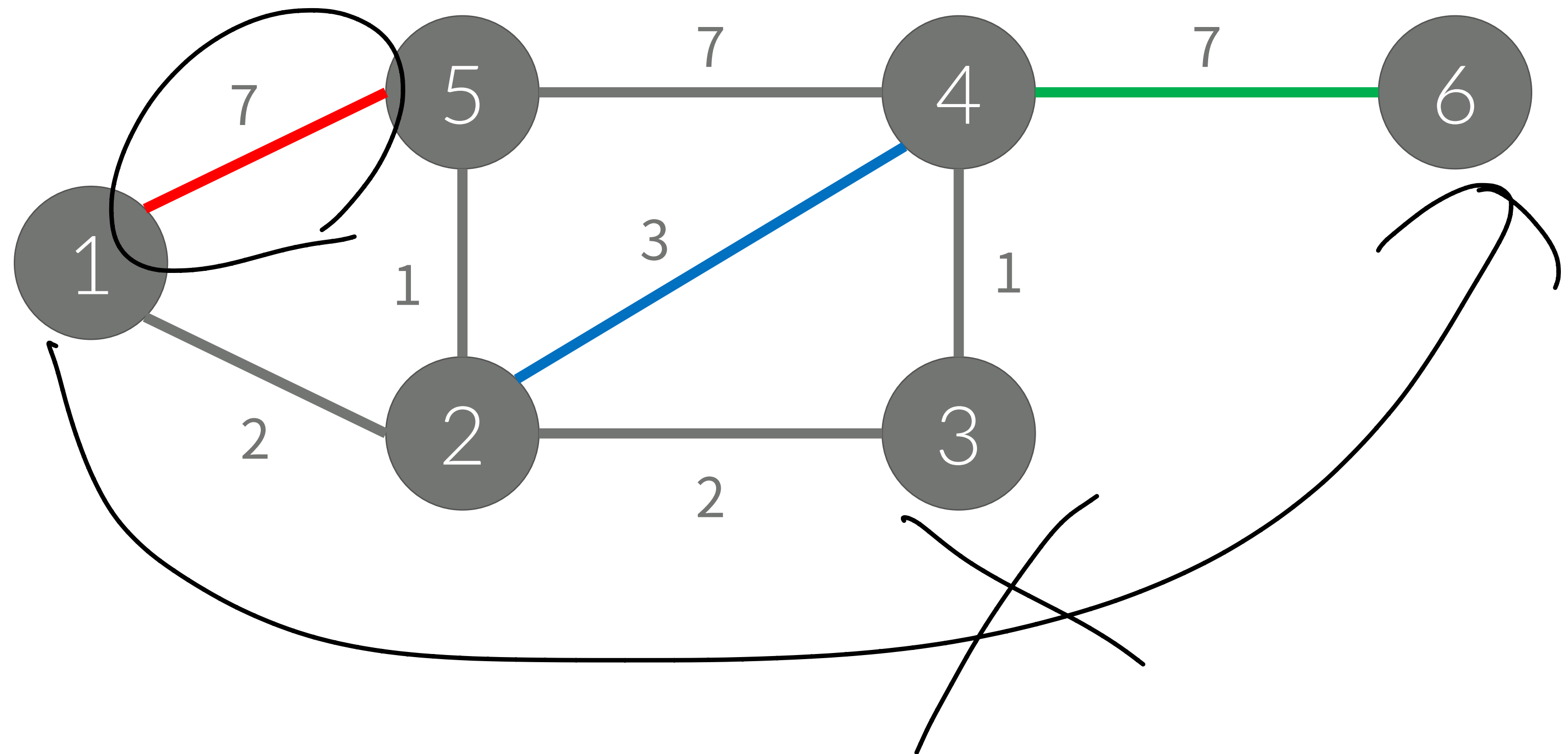
Adjacency matrix

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

정점 V , 간선 E

공간: V^2

한 정점 연결된 모든 간선: $O(V)$



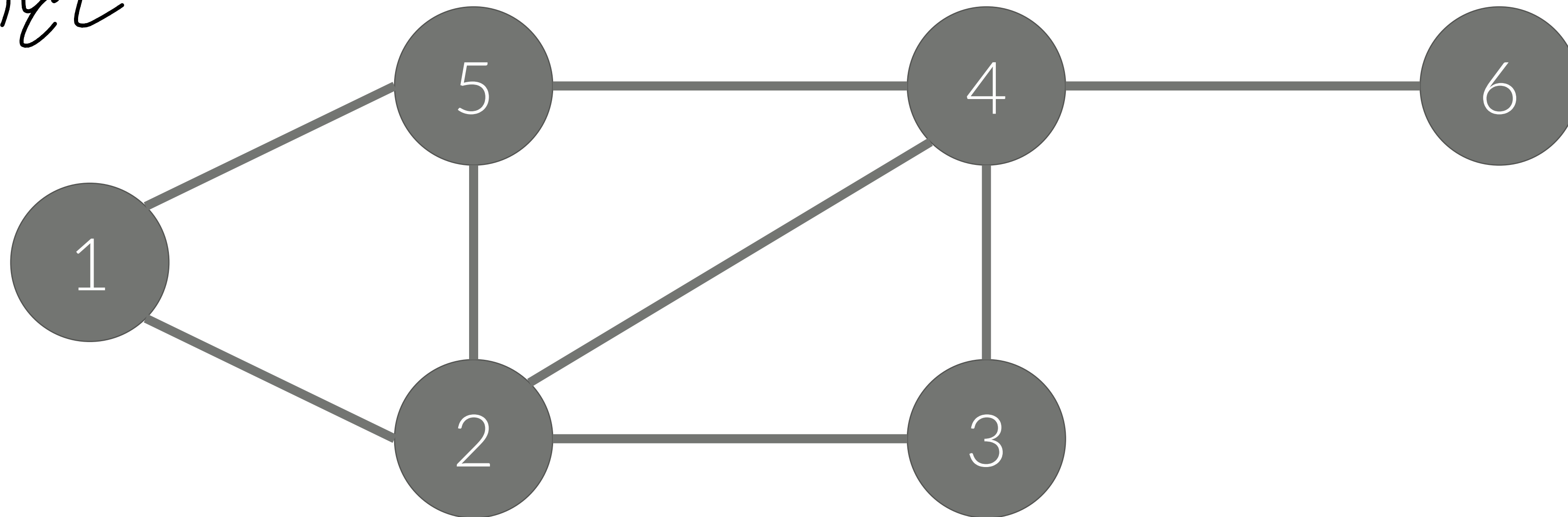
인접 리스트

Adjacency-list

- 리스트를 이용해서 구현한다.

• A[i] : i와 연결된 정점을 리스트로 포함하고 있음

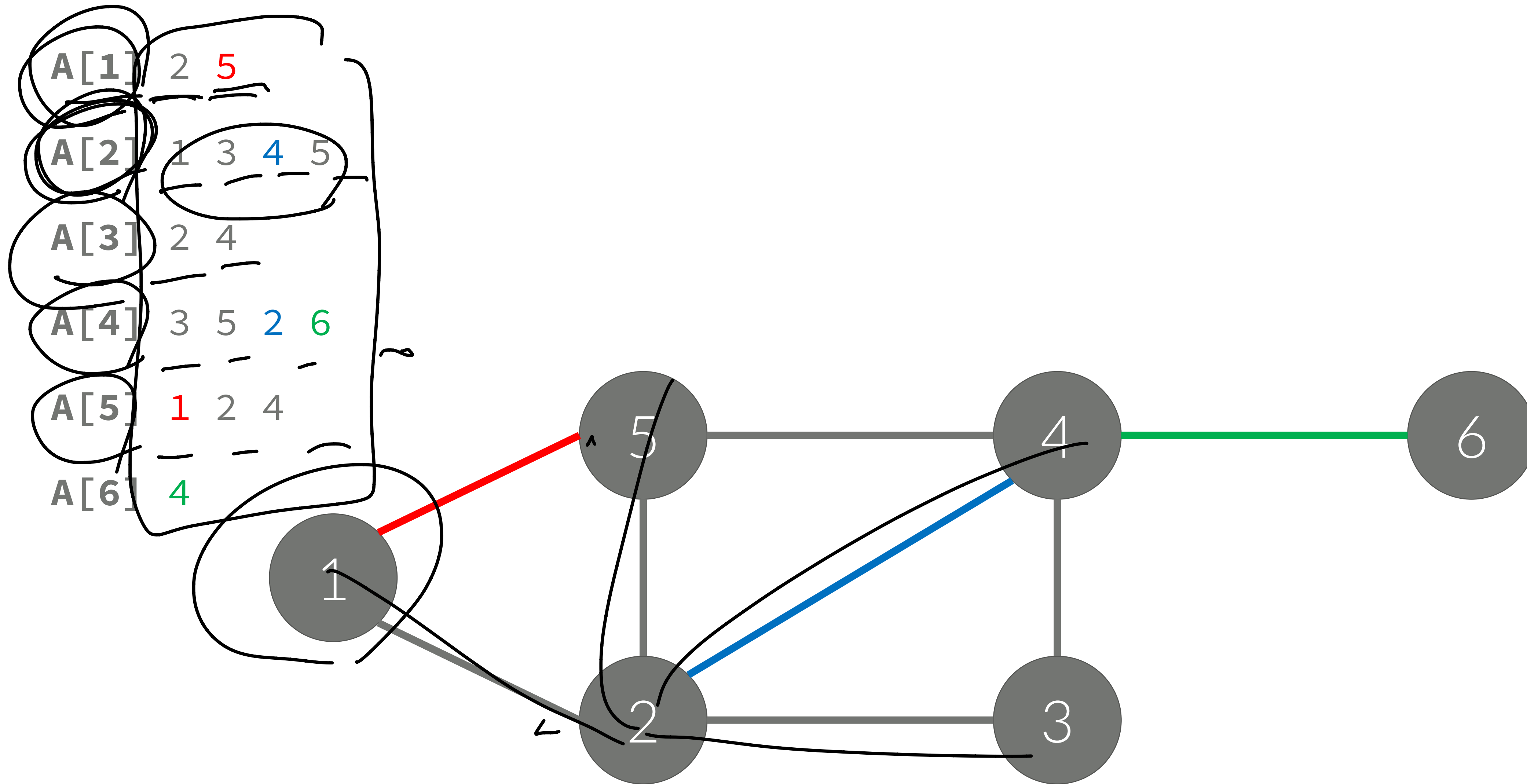
1-2
가늠



인접 리스트

Adjacency-list

22



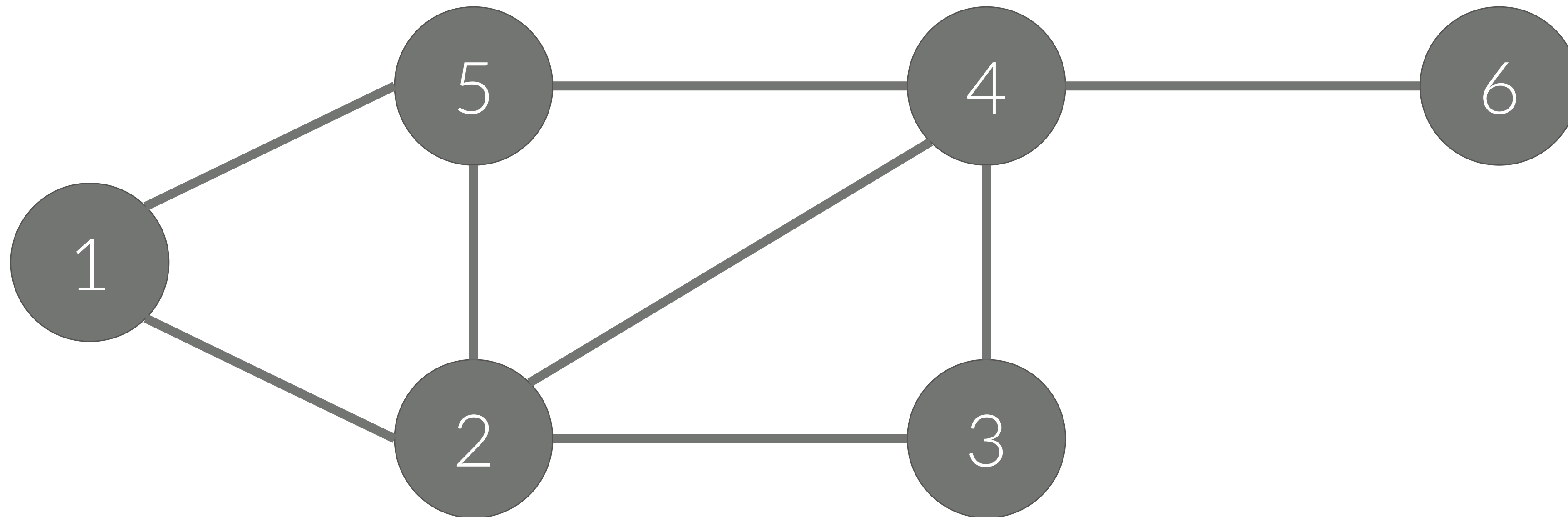
인접 리스트

Adjacency-list

배열, 리스트

- 리스트는 크기를 동적으로 변경할 수 있어야 한다.
- 링크드 리스트나 길이를 동적으로 변경할 수 있는 배열을 사용한다.

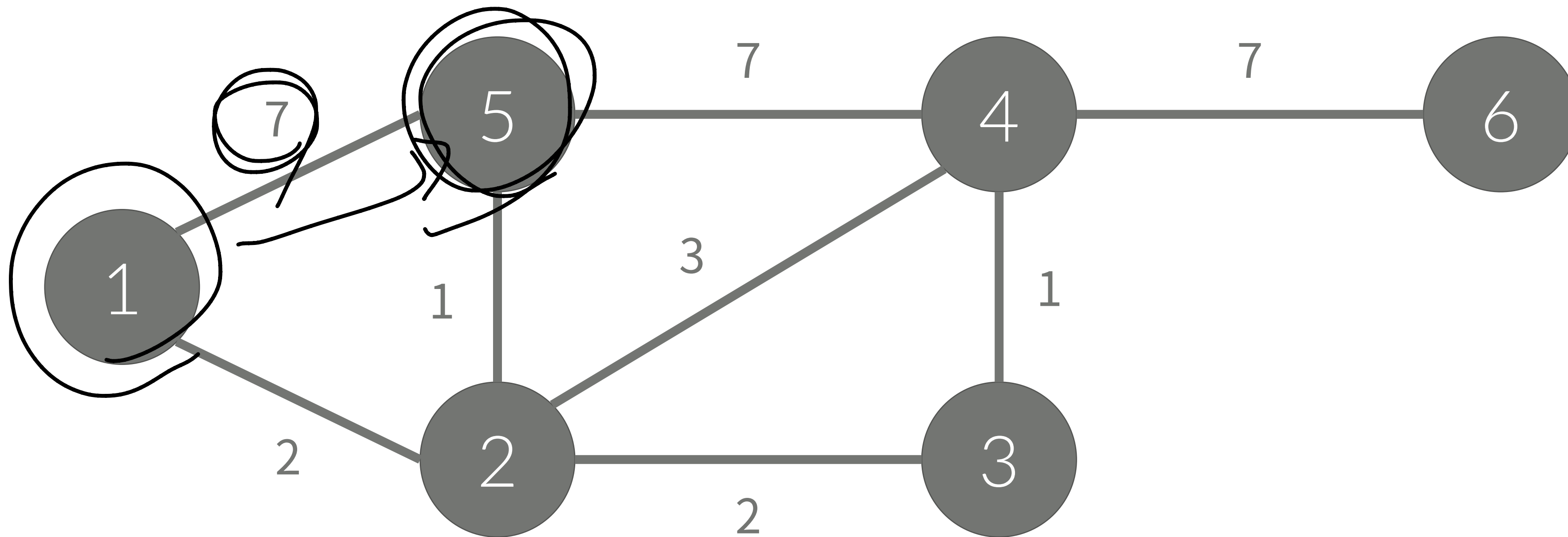
C++ → vector
Java → ArrayList
Python → list



인접 리스트

Adjacency-list

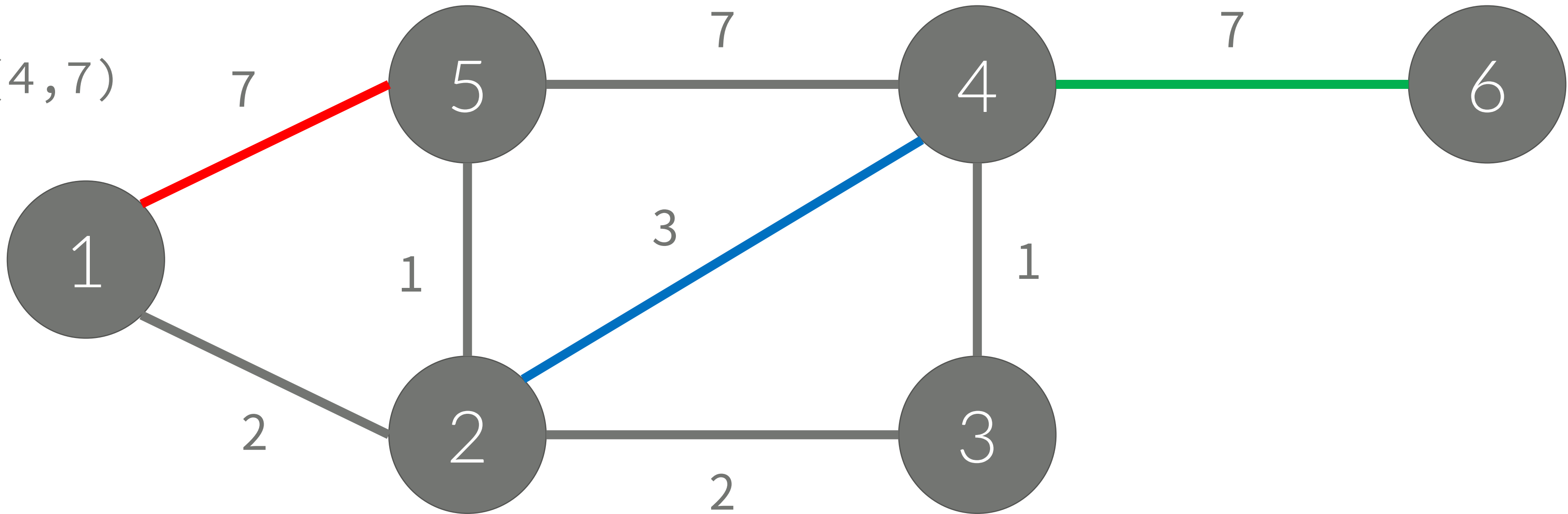
- 리스트를 이용해서 구현한다.
- $A[i] = i$ 와 연결된 정점과 그 간선의 가중치를 리스트로 포함하고 있음



인접 리스트

Adjacency-list

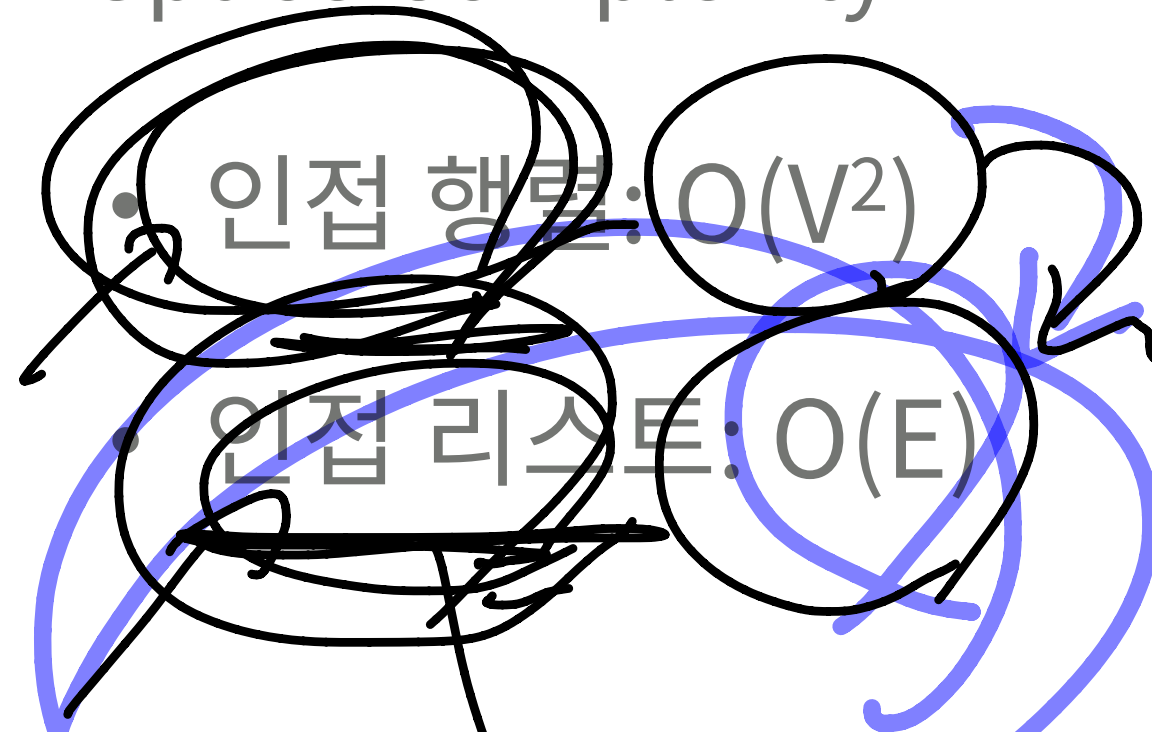
- A[1] (2, 2) (5, 7)
- A[2] (1, 2) (3, 2) (4, 3) (5, 1)
- A[3] (2, 2) (4, 1)
- A[4] (3, 1) (5, 7) (2, 3) (6, 7)
- A[5] (1, 7) (2, 1) (4, 7)
- A[6] (4, 7)



공간 복잡도

Space Complexity

26



이차원

$\overline{T} - \overline{S}$



한 정점과 인접한 모든 정점을

찾는 시간 $O(\text{차수})$

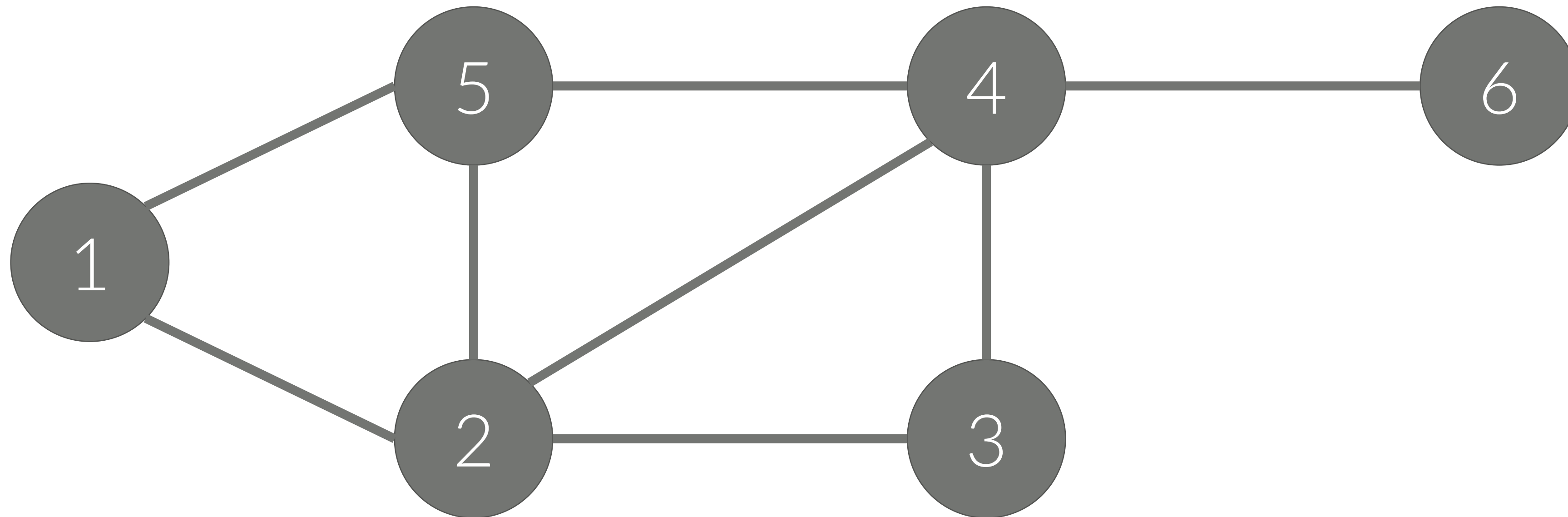
(u, v) 간선 있는지 있는지 $\frac{O(1)}{O(\text{차수})}$

간선 리스트

Edge List

27

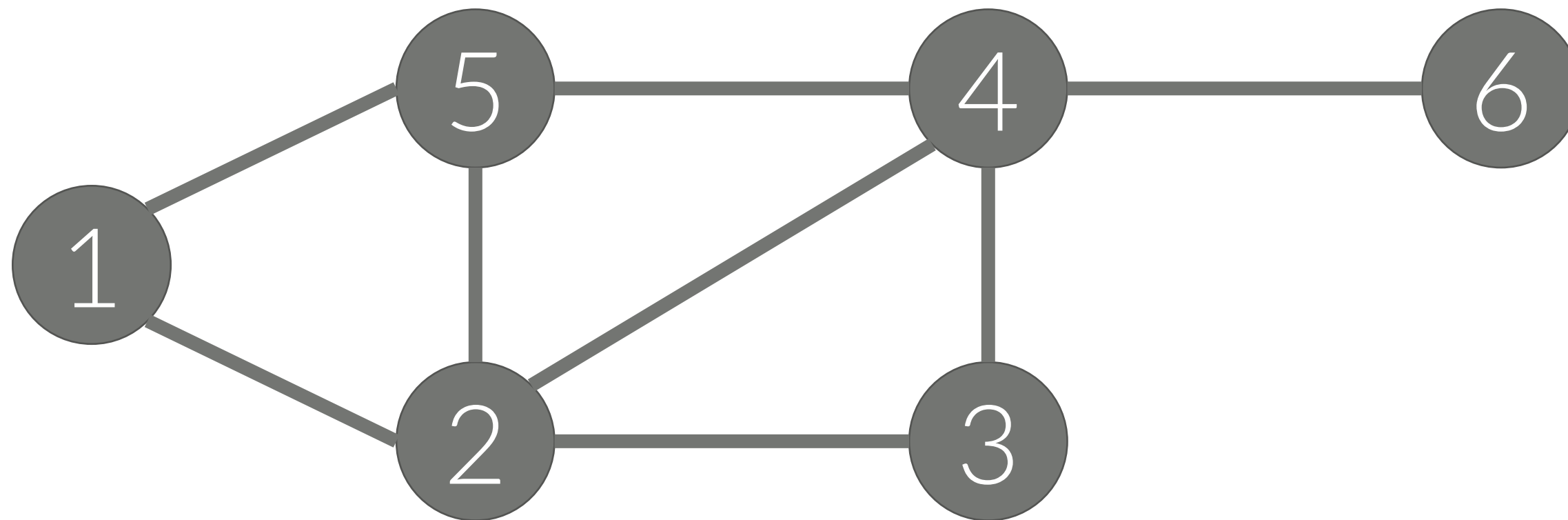
- 배열을 이용해서 구현한다.
- 간선을 모두 저장하고 있다.



간선 리스트

Edge List

- 배열을 이용해서 구현한다.
- 간선을 모두 저장하고 있다.
- E라는 배열에 간선을 모두 저장



$$E[0] = 1 \ 2$$

$$E[1] = 1 \ 5$$

$$E[2] = 2 \ 3$$

$$E[3] = 2 \ 4$$

$$E[4] = 2 \ 5$$

$$E[5] = 5 \ 4$$

$$E[6] = 4 \ 3$$

$$E[7] = 4 \ 6$$

$$E[8] = 2 \ 1$$

$$E[9] = 5 \ 1$$

$$E[10] = 3 \ 2$$

$$E[11] = 4 \ 2$$

$$E[12] = 5 \ 2$$

$$E[13] = 4 \ 5$$

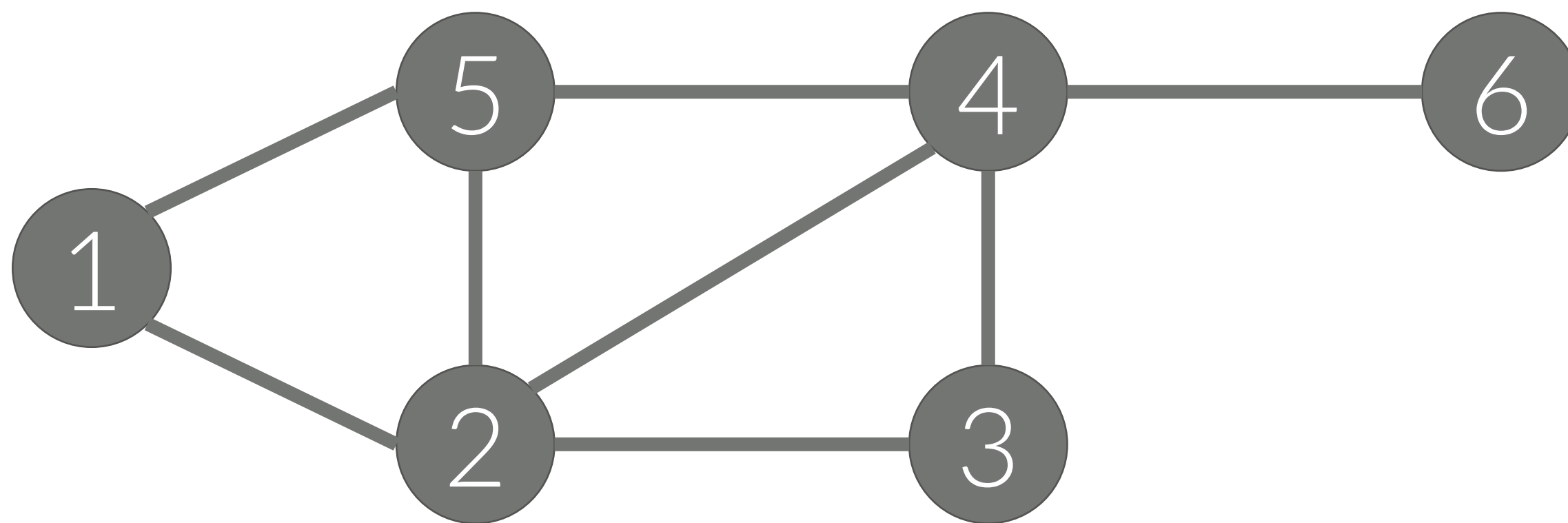
$$E[14] = 3 \ 4$$

$$E[15] = 6 \ 4$$

간선 리스트

Edge List

- 각 간선의 앞 정점을 기준으로 개수를 센다.



i	0	1	2	3	4	5	6
cnt[i]	0	2	4	2	4	3	1

$$E[0] = 1\ 2$$

$$E[1] = 1\ 5$$

$$E[2] = 2\ 1$$

$$E[3] = 2\ 3$$

$$E[4] = 2\ 4$$

$$E[5] = 2\ 5$$

$$E[6] = 3\ 2$$

$$E[7] = 3\ 4$$

$$E[8] = 4\ 2$$

$$E[9] = 4\ 3$$

$$E[10] = 4\ 5$$

$$E[11] = 4\ 6$$

$$E[12] = 5\ 1$$

$$E[13] = 5\ 2$$

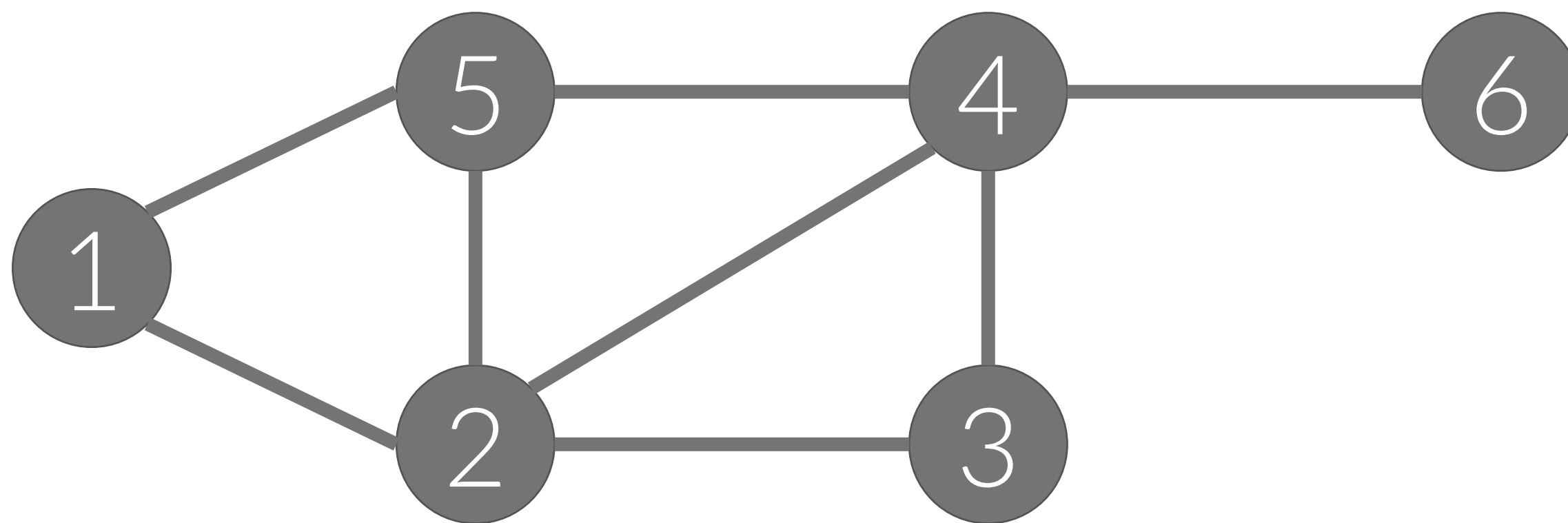
$$E[14] = 5\ 4$$

$$E[15] = 6\ 4$$

간선 리스트

Edge List

```
for (int i=0; i<m; i++) {  
    cnt[e[i][0]] += 1;  
}
```



i	0	1	2	3	4	5	6
cnt[i]	0	2	4	2	4	3	1

0 2 6 8 12 15 16

E[0] = 1 2

E[1] = 1 5

E[2] = 2 1

E[3] = 2 3

E[4] = 2 4

E[5] = 2 5

E[6] = 3 2

E[7] = 3 4

E[8] = 4 2

E[9] = 4 3

E[10] = 4 5

E[11] = 4 6

E[12] = 5 1

E[13] = 5 2

E[14] = 5 4

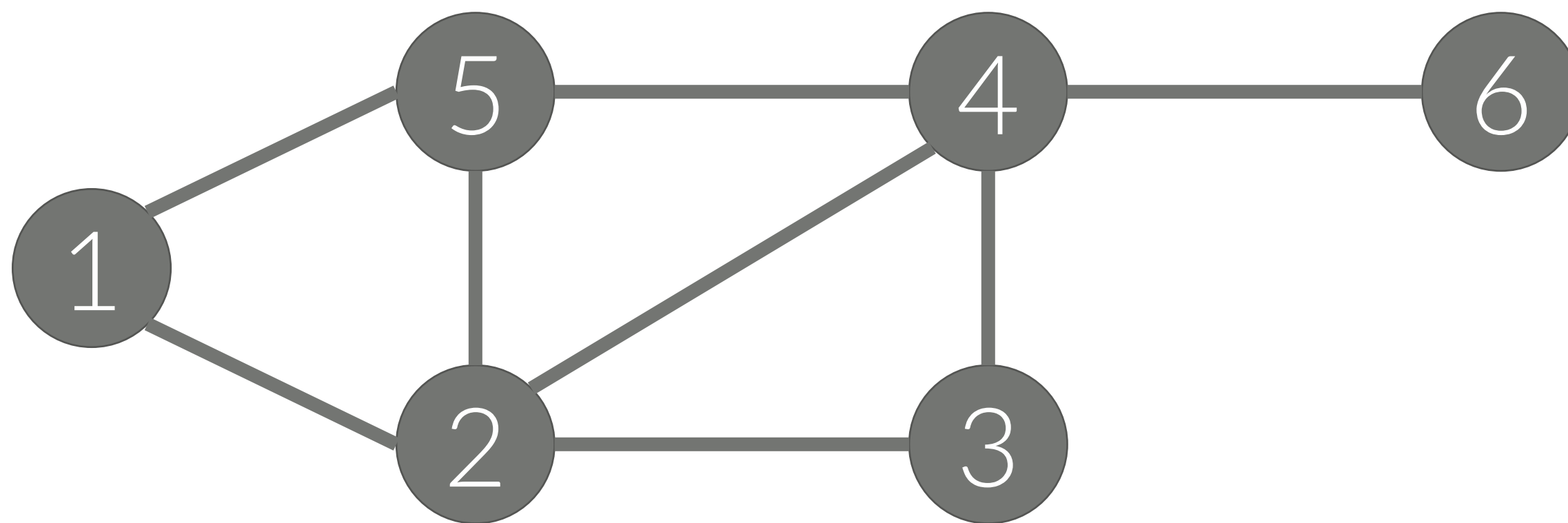
E[15] = 6 4

간선 리스트

Edge List

31

```
for (int i=1; i<=n; i++) {  
    cnt[i] = cnt[i-1] + cnt[i];  
}
```



i	0	1	2	3	4	5	6
cnt[i]	0	2	6	8	12	15	16

112

E[0] = 1 2
E[1] = 1 5
E[2] = 2 1
E[3] = 2 3
E[4] = 2 4
E[5] = 2 5
E[6] = 3 2
E[7] = 3 4

E[8] = 4 2
E[9] = 4 3
E[10] = 4 5
E[11] = 4 6
E[12] = 5 1
E[13] = 5 2
E[14] = 5 4
E[15] = 6 4

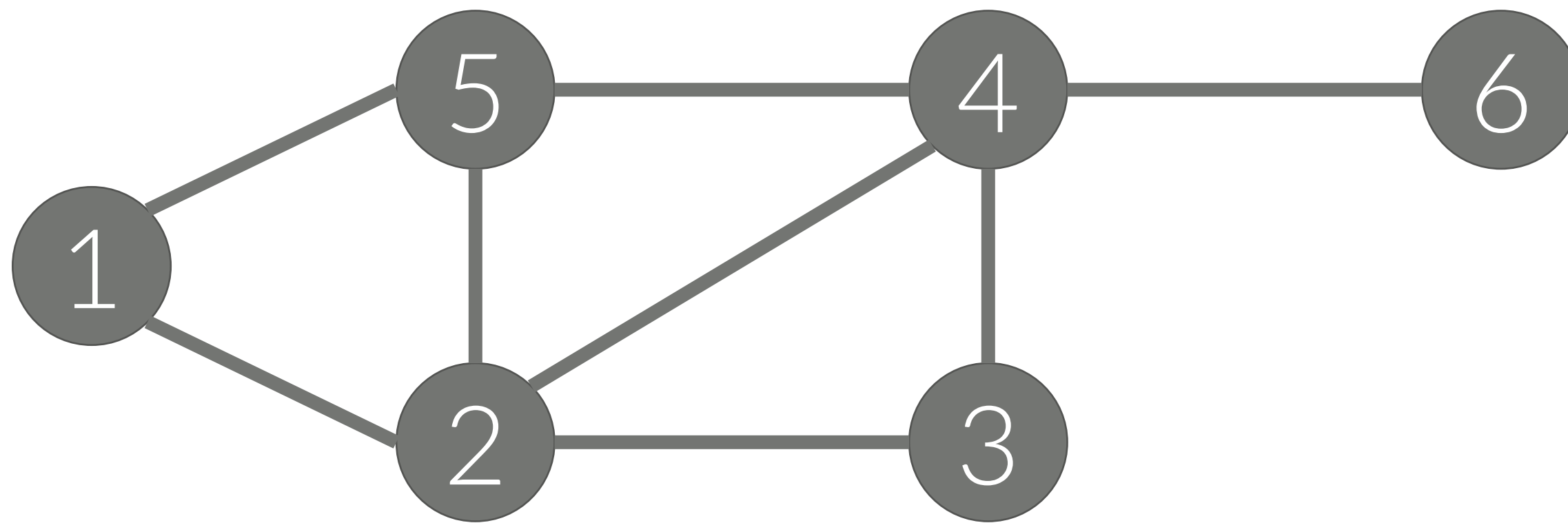
간선 리스트

Edge List

32

$O(\text{차수})$

- i 번 정점과 연결된 간선은
- E배열에서 $\text{cnt}[i-1]$ 부터 $\text{cnt}[i]-1$ 까지이다.



$E[0] = 1\ 2$

$E[1] = 1\ 5$

$E[2] = 2\ 1$

$E[3] = 2\ 3$

$E[4] = 2\ 4$

$E[5] = 2\ 5$

$E[6] = 3\ 2$

$E[7] = 3\ 4$

$E[8] = 4\ 2$

$E[9] = 4\ 3$

$E[10] = 4\ 5$

$E[11] = 4\ 6$

$E[12] = 5\ 1$

$E[13] = 5\ 2$

$E[14] = 5\ 4$

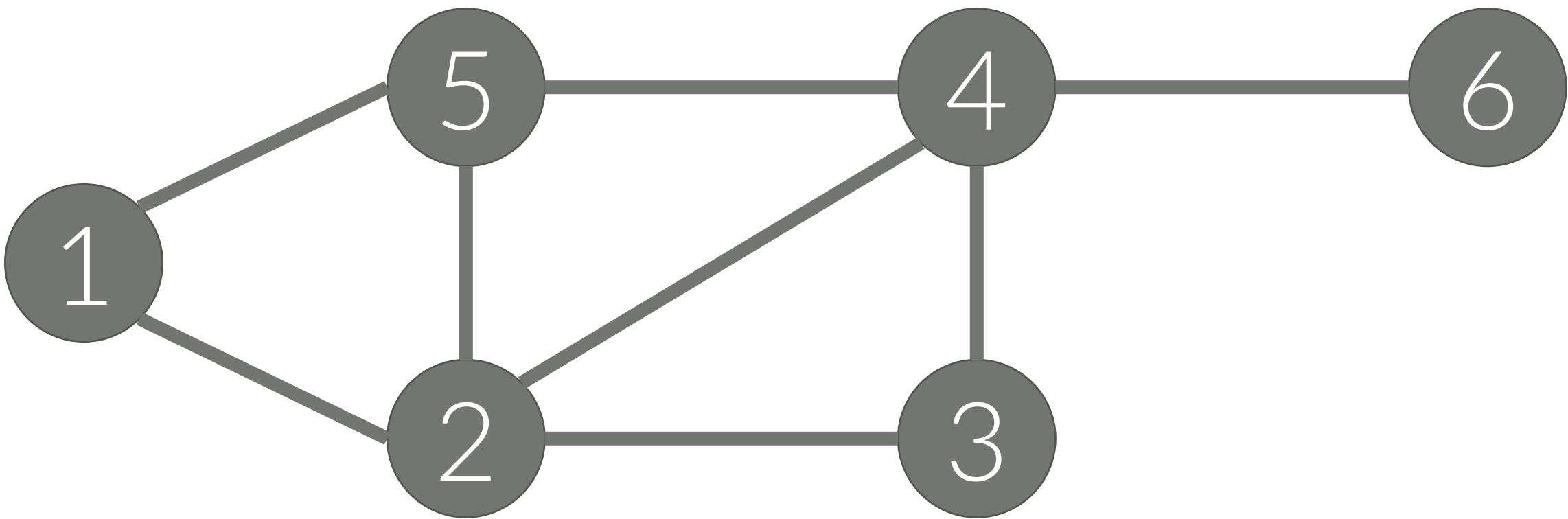
$E[15] = 6\ 4$

i	0	1	2	3	4	5	6
cnt[i]	0	2	6	8	12	15	16

간선 리스트

Edge List

- 3번 정점: cnt[2] ~ cnt[3]-1



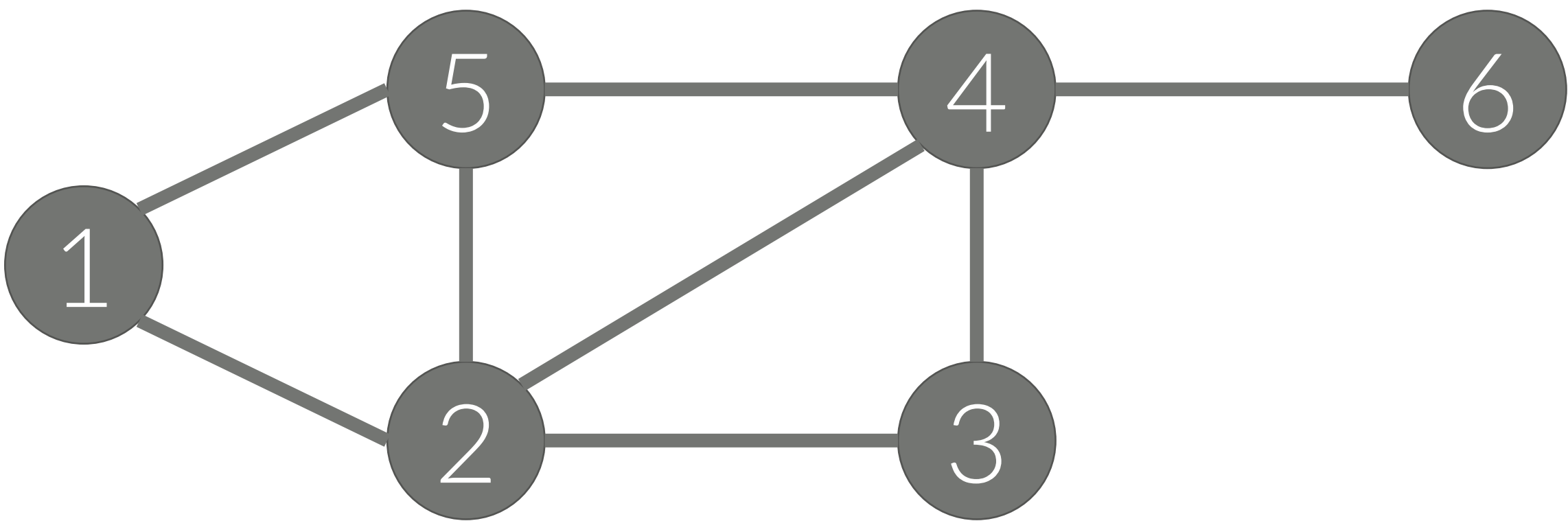
E[0]	=	1	2	E[8]	=	4	2
E[1]	=	1	5	E[9]	=	4	3
E[2]	=	2	1	E[10]	=	4	5
E[3]	=	2	3	E[11]	=	4	6
E[4]	=	2	4	E[12]	=	5	1
E[5]	=	2	5	E[13]	=	5	2
E[6]	=	3	2	E[14]	=	5	4
E[7]	=	3	4	E[15]	=	6	4

i	0	1	2	3	4	5	6
cnt[i]	0	2	6	8	12	15	16

간선 리스트

Edge List

- 4번 정점: cnt[3] ~ cnt[4]-1



i	0	1	2	3	4	5	6
cnt[i]	0	2	6	8	12	15	16

$E[0] = 1\ 2$

$E[1] = 1\ 5$

$E[2] = 2\ 1$

$E[3] = 2\ 3$

$E[4] = 2\ 4$

$E[5] = 2\ 5$

$E[6] = 3\ 2$

$E[7] = 3\ 4$

$E[8] = 4\ 2$

$E[9] = 4\ 3$

$E[10] = 4\ 5$

$E[11] = 4\ 6$

$E[12] = 5\ 1$

$E[13] = 5\ 2$

$E[14] = 5\ 4$

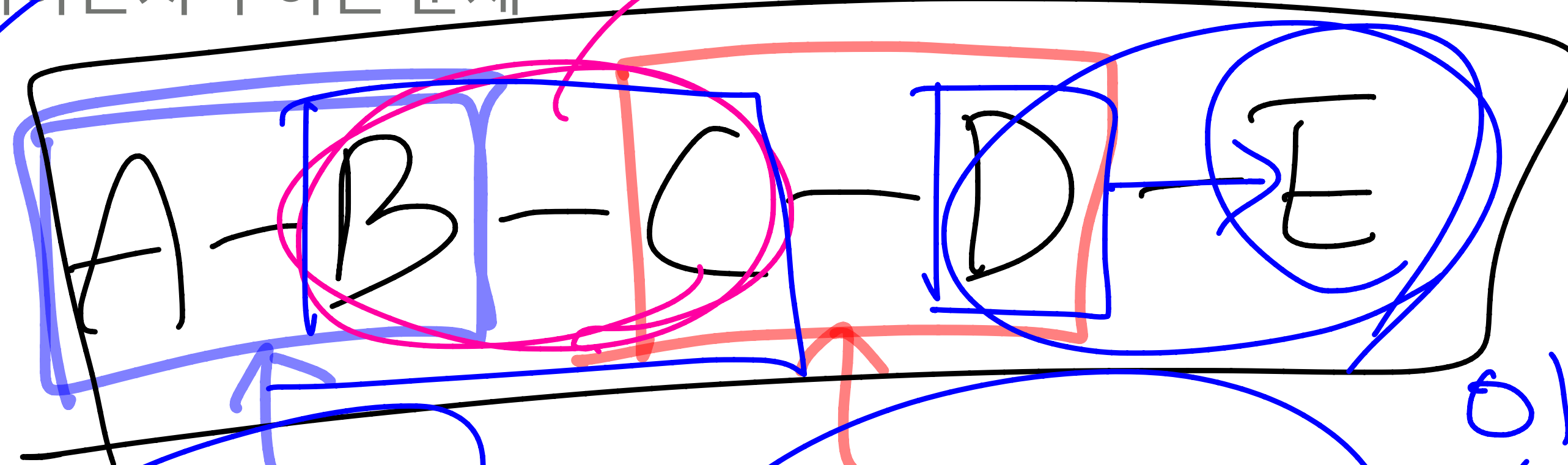
$E[15] = 6\ 4$

Ans: 282

2/2

이제 형제

- 총 N명의 친구 관계가 주어졌을 때
 - 다음과 같은 친구 관계가 존재하는지 구하는 문제
 - A는 B와 친구다.
 - B는 C와 친구다.
 - C는 D와 친구다.
 - D는 E와 친구다.
-



01762156

간지년

간헐적 단식

ABCDE

<https://www.acmicpc.net/problem/13023>

- $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$
- $A \rightarrow B$
- $B \rightarrow C$
- $C \rightarrow D$
- 에서 길이가 4인 단순 경로를 찾고
- $D \rightarrow E$ 를 찾는다.

ABCDE

<https://www.acmicpc.net/problem/13023>

- $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$
- $A \rightarrow B$
- $C \rightarrow D$
- 는 그냥 간선이기 때문에, 간선 리스트로 찾을 수 있다
- $B \rightarrow C$ 는 인접 행렬로 찾을 수 있다
- $D \rightarrow E$ 는 인접 리스트로 찾는다

ABCDE

<https://www.acmicpc.net/problem/13023>

- 소스: <http://codeplus.codes/9e95120c2a744876813ae5198c84993b>

목적: 임의의 정점에서 시작
연결되어 있는 모든 정점을
방문하는 것 (백신)

그래프의 탐색

DFS
BFS

그래프의 탐색

40

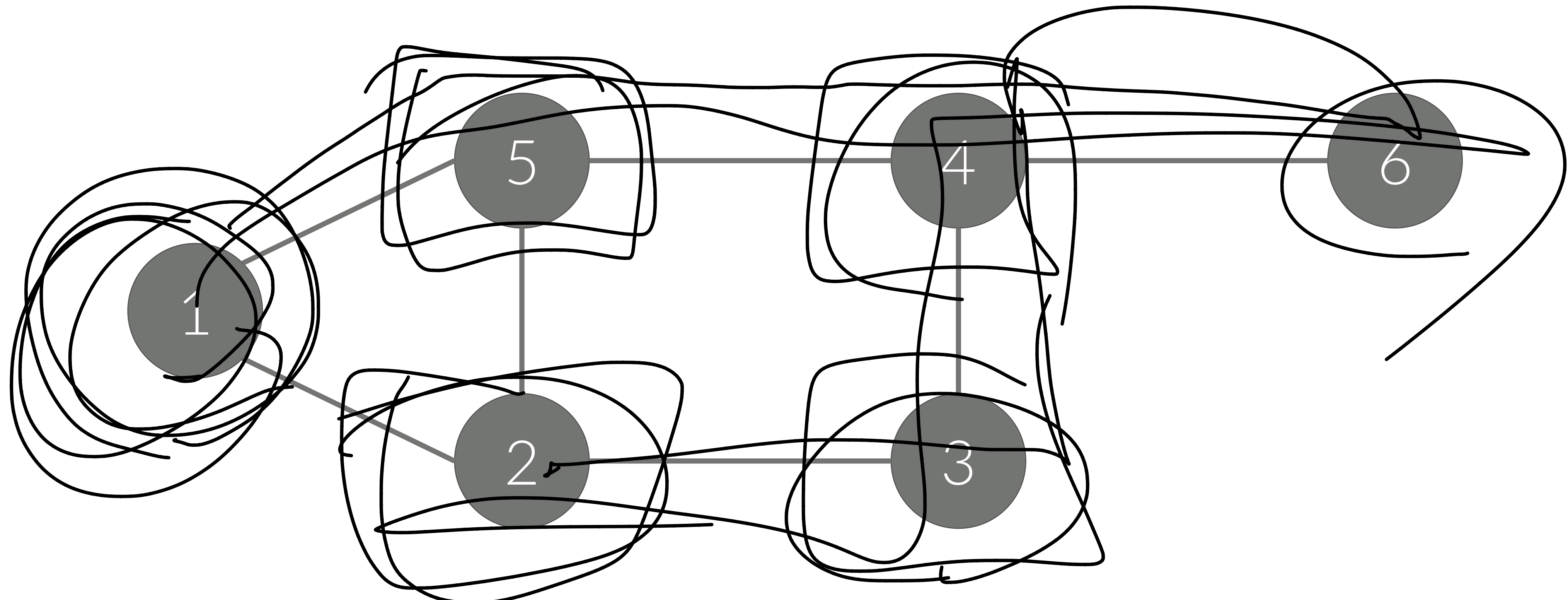
DFS, BFS

- DFS: 깊이 우선 탐색

→ Stack

- BFS: 너비 우선 탐색

→ Queue

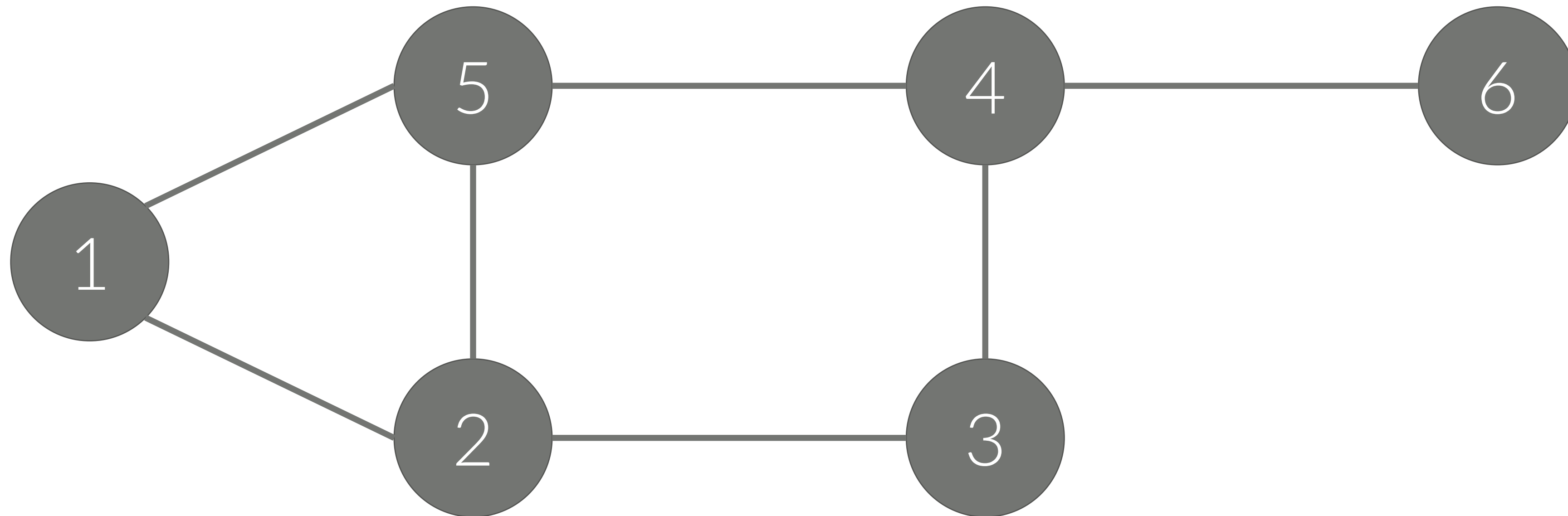


깊이 우선 탐색

Depth First Search

41

- 스택을 이용해서 갈 수 있는 만큼 최대한 많이 가고
- 갈 수 없으면 이전 정점으로 돌아간다.



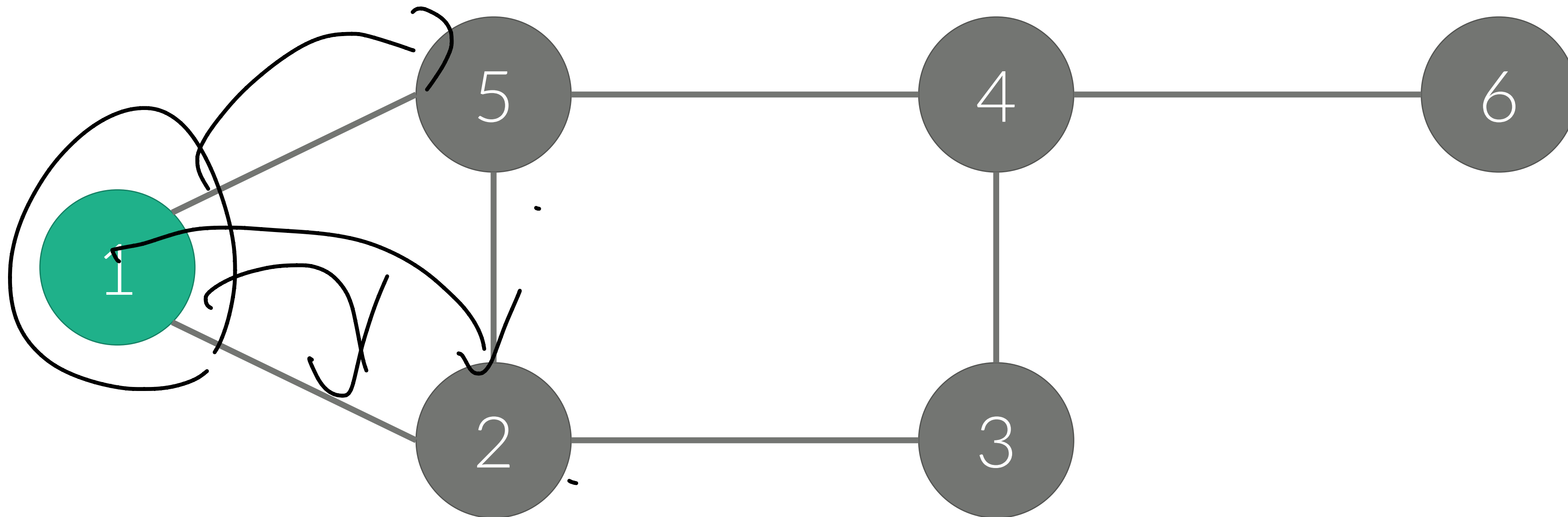
깊이 우선 탐색

42

Depth First Search

- 현재 정점: 1
- 순서: 1
- 스택: 1

i	1	2	3	4	5	6
check[i]	1	0	0	0	0	0



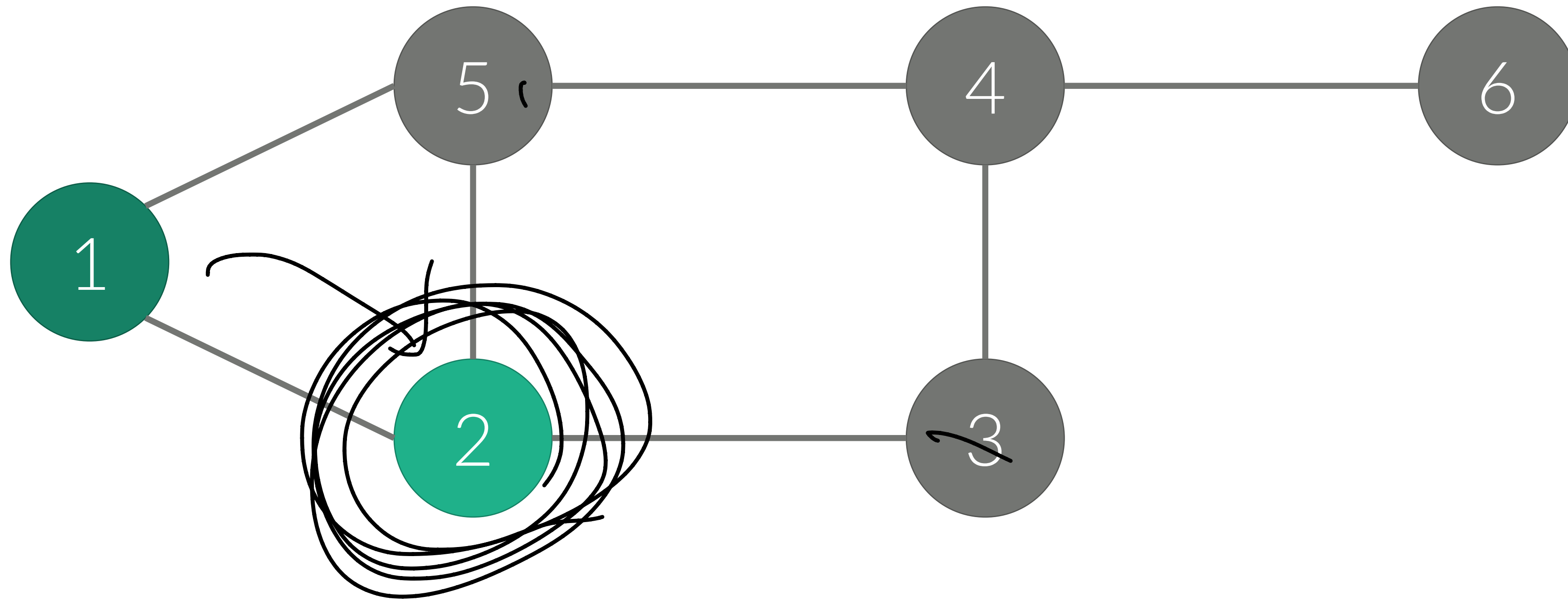
깊이 우선 탐색

43

Depth First Search

- 현재 정점: 2
- 순서: 1 2
- 스택: 1 2

i	1	2	3	4	5	6
check[i]	1	1	0	0	0	0



깊이 우선 탐색

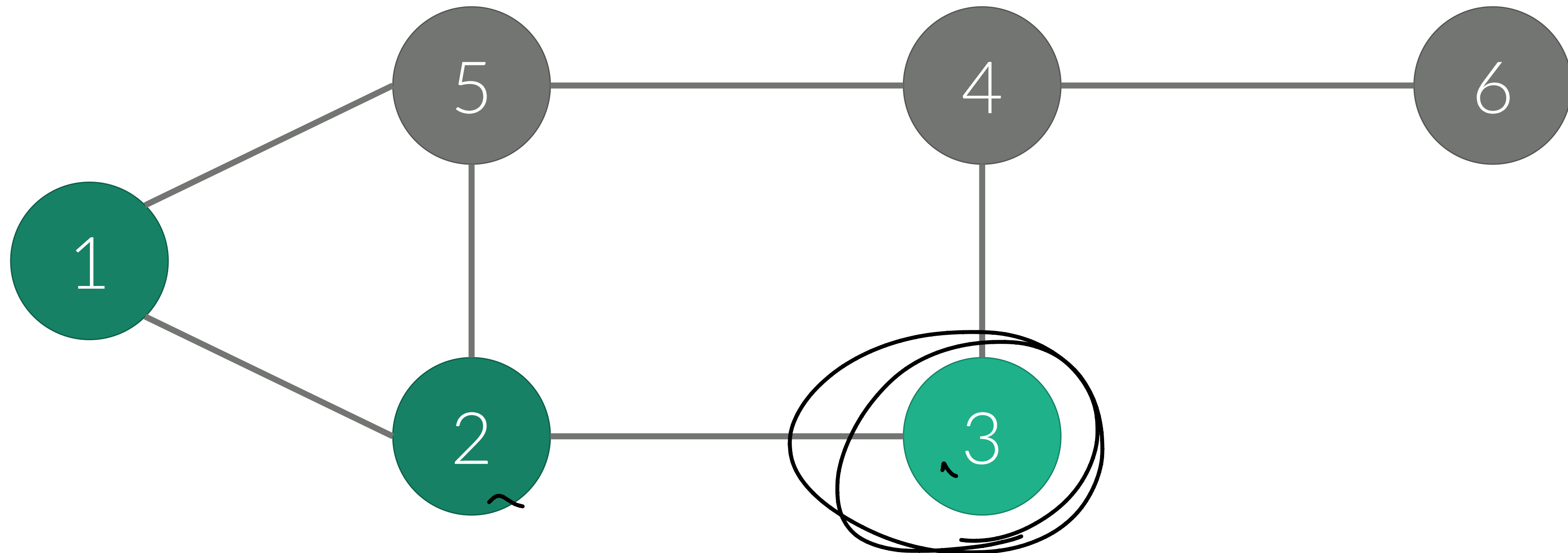
44

Depth First Search

- 현재 정점: 3
- 순서: 1 2 3
- 스택: 1 2 3



i	1	2	3	4	5	6
check[i]	1	1	1	0	0	0



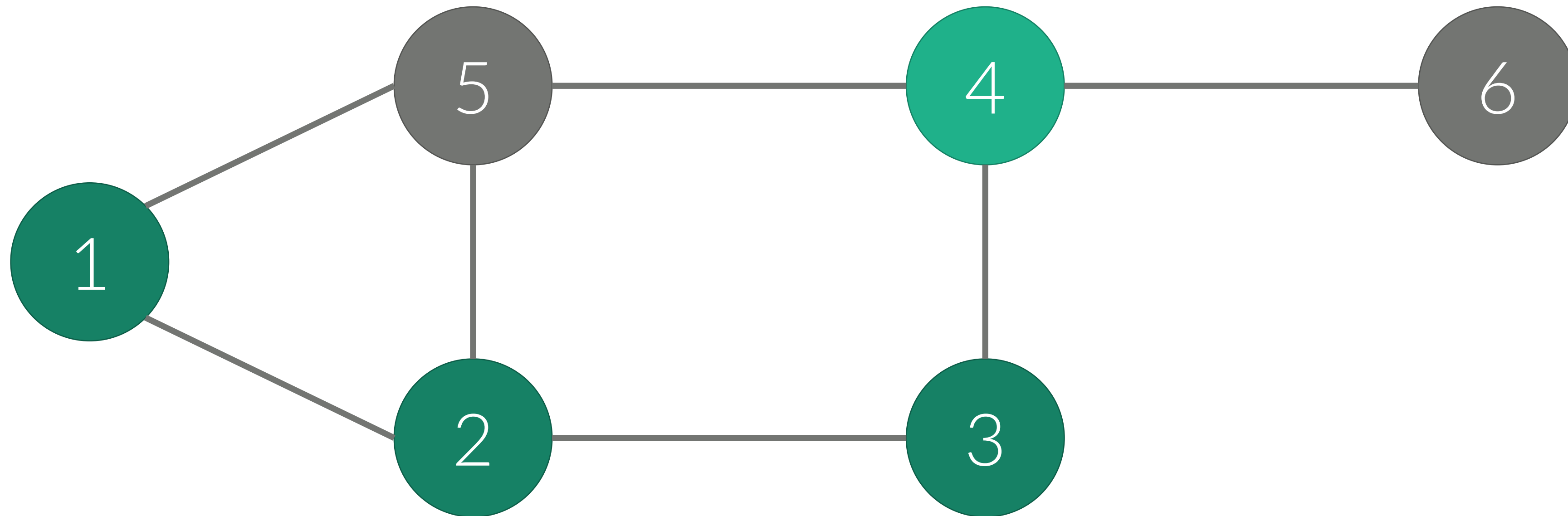
깊이 우선 탐색

45

Depth First Search

- 현재 정점: 4
- 순서: 1 2 3 4
- 스택: 1 2 3 4

i	1	2	3	4	5	6
check[i]	1	1	1	1	0	0



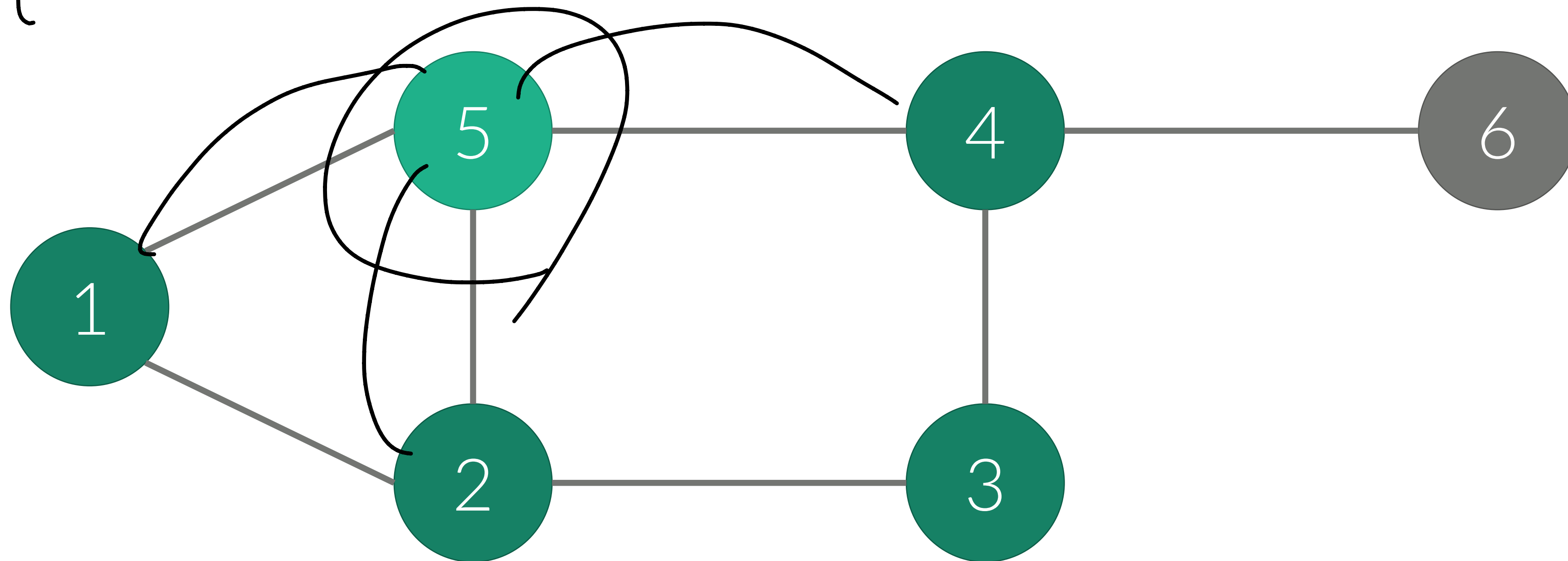
깊이 우선 탐색

46

Depth First Search

- 현재 정점: 5
- 순서: 1 2 3 4 5
- 스택: 1 2 3 4 5

i	1	2	3	4	5	6
check[i]	1	1	1	1	1	0



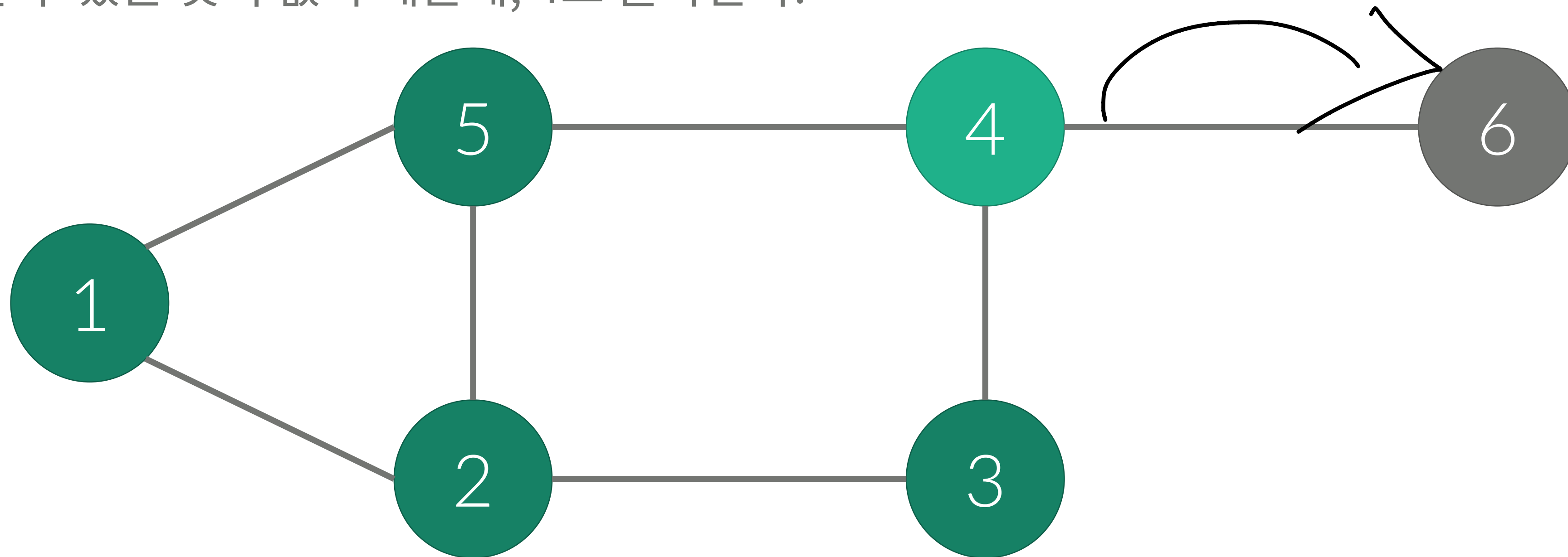
깊이 우선 탐색

47

Depth First Search

- 현재 정점: 4
- 순서: 1 2 3 4 5
- 스택: 1 2 3 4
- 5에서 더 갈 수 있는 것이 없기 때문에, 4로 돌아간다.

i	1	2	3	4	5	6
check[i]	1	1	1	1	1	0



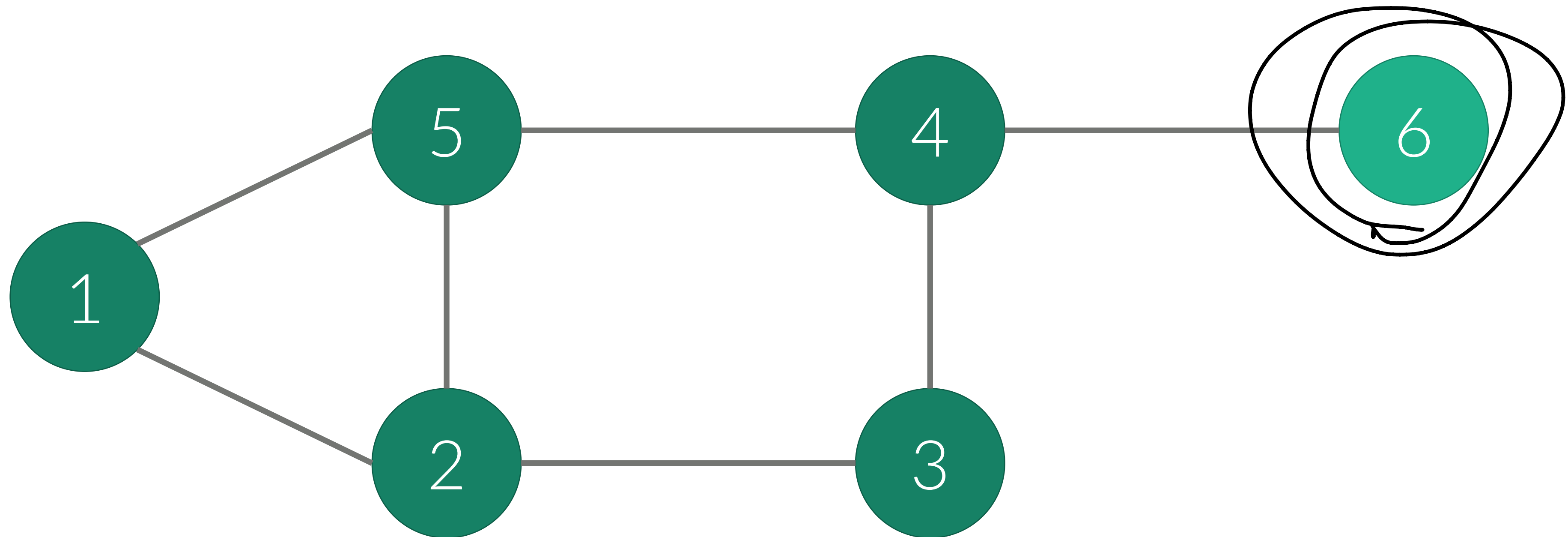
깊이 우선 탐색

48

Depth First Search

- 현재 정점: 6
- 순서: 1 2 3 4 5 6
- 스택: 1 2 3 4 6

i	1	2	3	4	5	6
check[i]	1	1	1	1	1	1



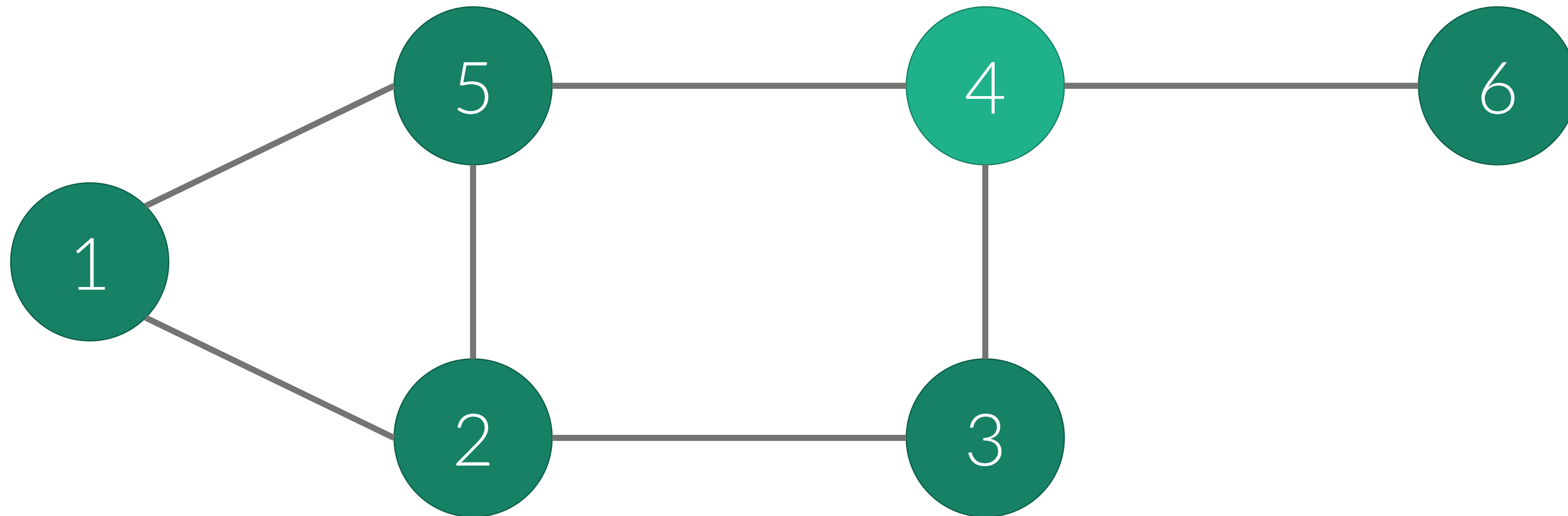
깊이 우선 탐색

49

Depth First Search

- 현재 정점: 4
- 순서: 1 2 3 4 5 6
- 스택: 1 2 3 4
- 6에서 갈 수 있는 것이 없기 때문에 4로 돌아간다.

i	1	2	3	4	5	6
check[i]	1	1	1	1	1	1



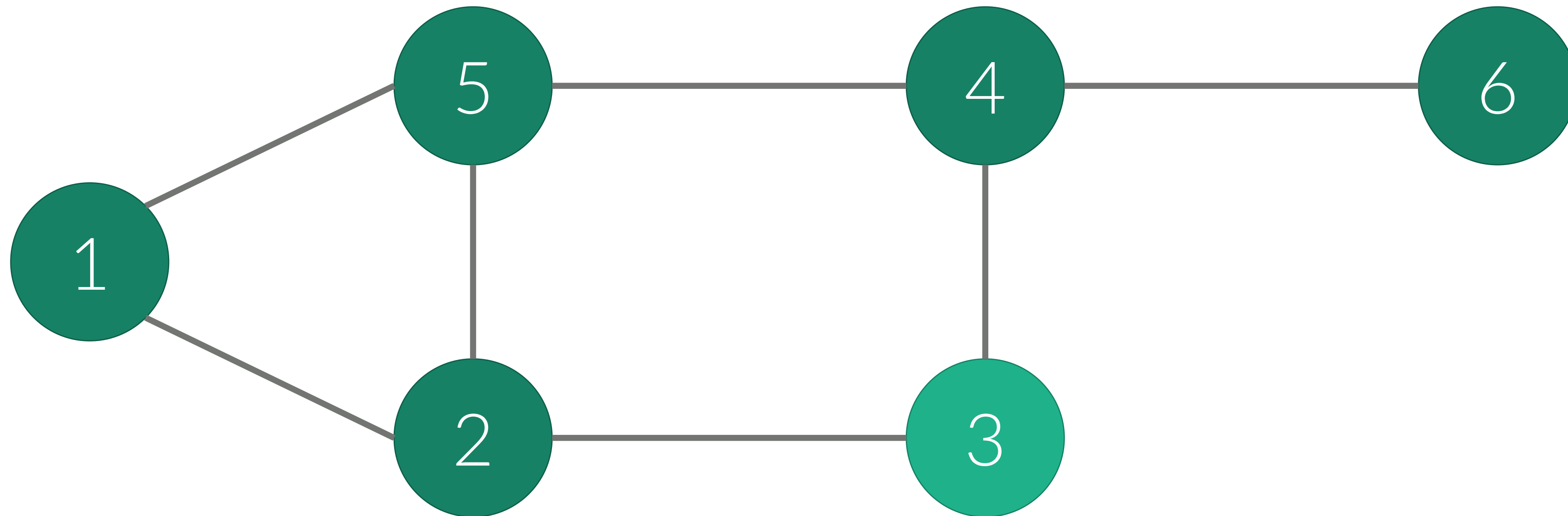
깊이 우선 탐색

50

Depth First Search

- 현재 정점: 3
- 순서: 1 2 3 4 5 6
- 스택: 1 2 3
- 4에서 갈 수 있는 것이 없기 때문에 3으로 돌아간다.

i	1	2	3	4	5	6
check[i]	1	1	1	1	1	1



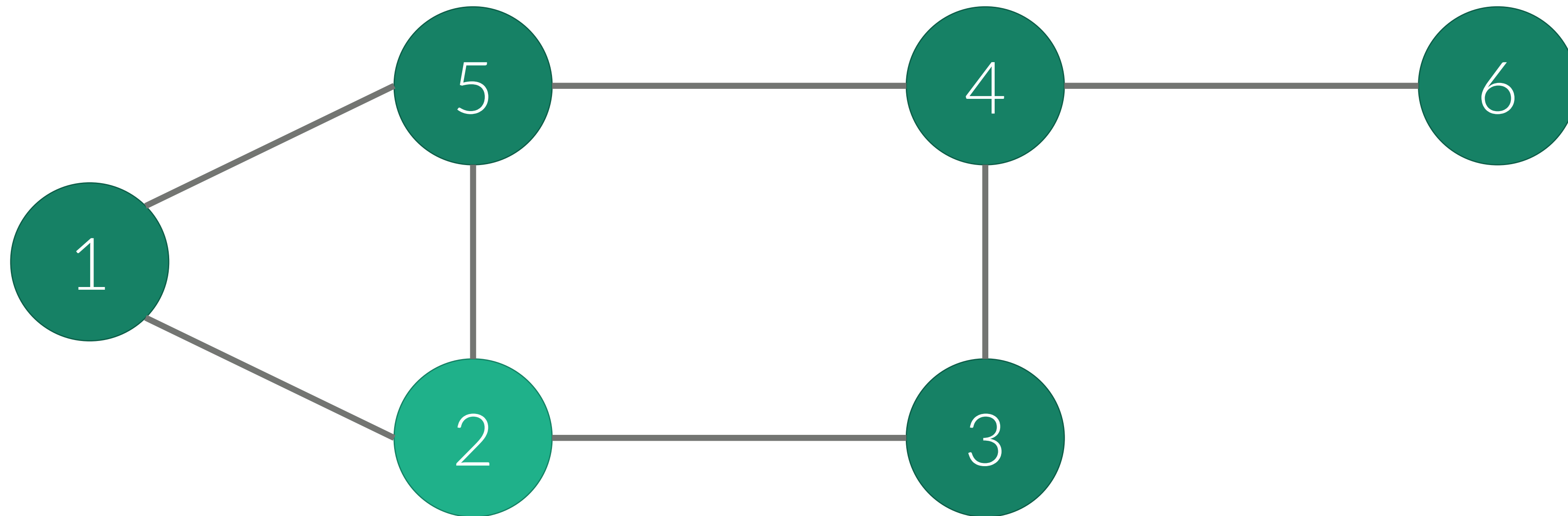
깊이 우선 탐색

51

Depth First Search

- 현재 정점: 2
- 순서: 1 2 3 4 5 6
- 스택: 1 2
- 3에서 갈 수 있는 것이 없기 때문에 2으로 돌아간다.

i	1	2	3	4	5	6
check[i]	1	1	1	1	1	1



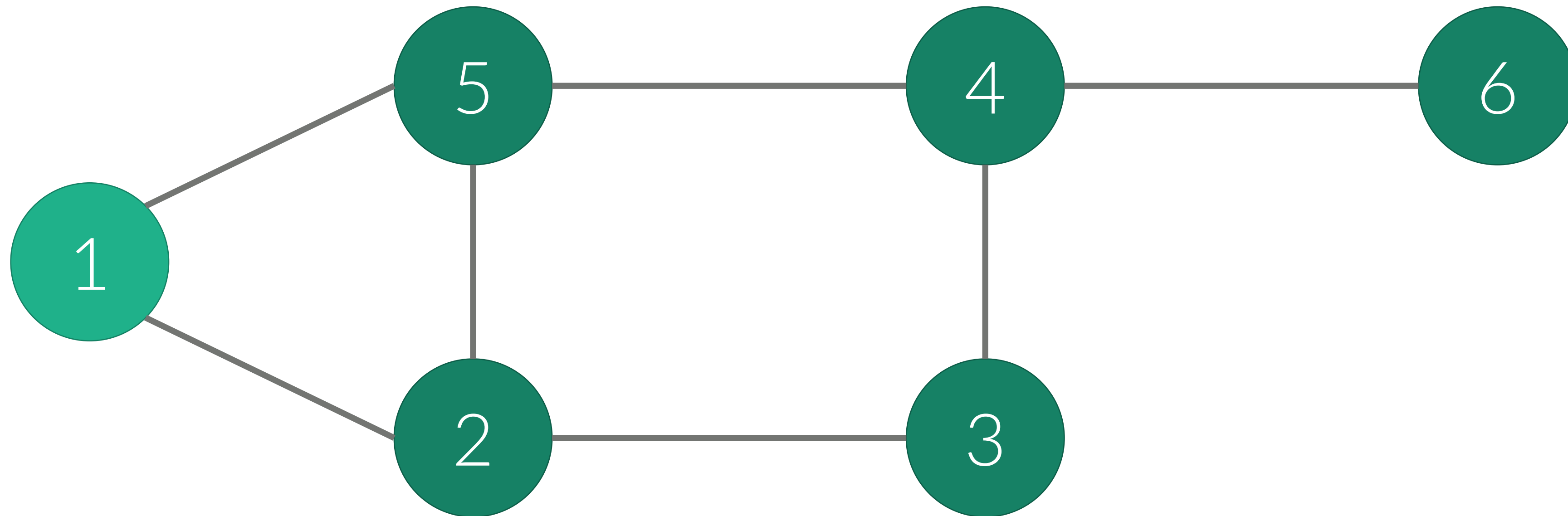
깊이 우선 탐색

52

Depth First Search

- 현재 정점: 1
- 순서: 1 2 3 4 5 6
- 스택: 1
- 2에서 갈 수 있는 것이 없기 때문에 1으로 돌아간다.

i	1	2	3	4	5	6
check[i]	1	1	1	1	1	1



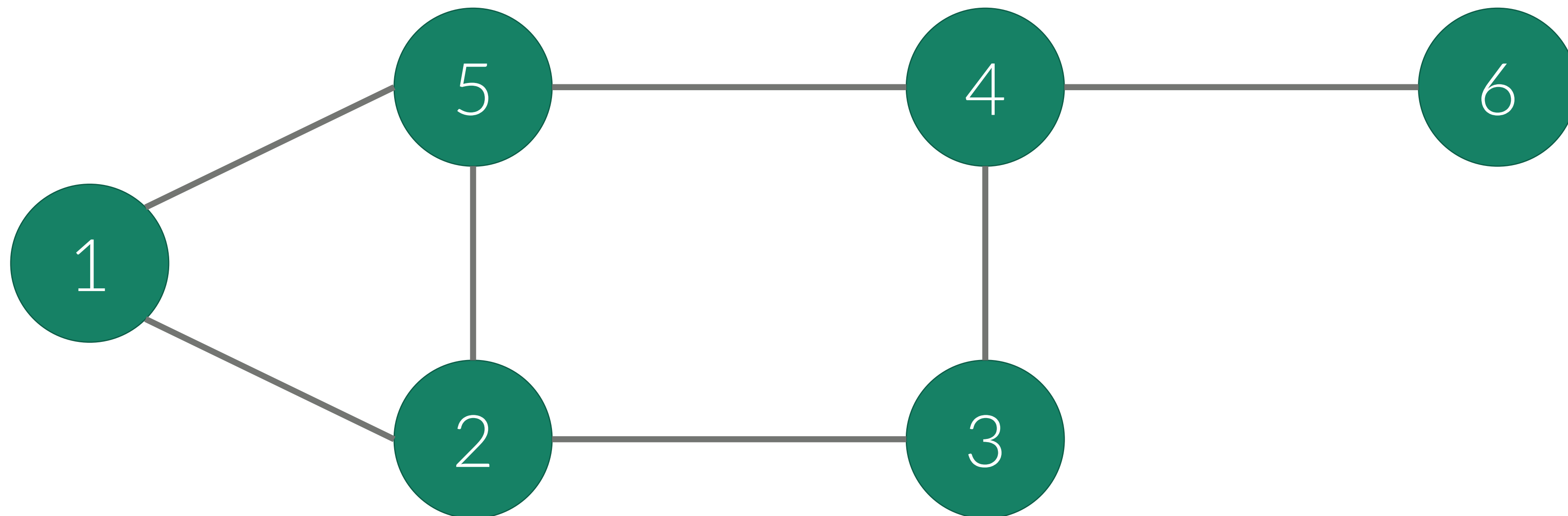
깊이 우선 탐색

53

Depth First Search

- 현재 정점:
- 순서: 1 2 3 4 5 6
- 스택:
- 탐색 종료

i	1	2	3	4	5	6
check[i]	1	1	1	1	1	1



깊이 우선 탐색

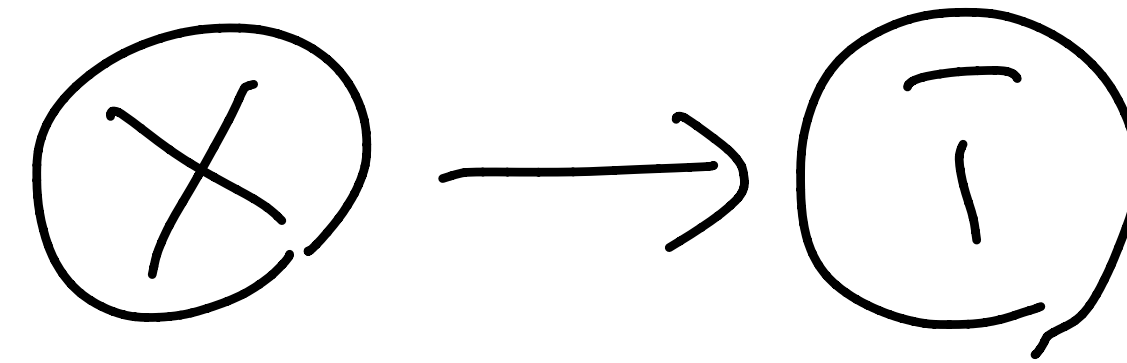
Depth First Search

- 재귀 호출을 이용해서 구현할 수 있다.

```
void dfs(int x) {
    check[x] = true;
    for (int i=1; i<=n; i++) {
        if (a[x][i] == 1 && check[i] == false) {
            dfs(i);
        }
    }
}
```

- 인접 행렬을 이용한 구현

dfs(x): x를 방문



$V \times \mathcal{O}(V)$

$\mathcal{O}(V^2)$

깊이 우선 탐색

Depth First Search

55

$$V \leq E$$

- 재귀 호출을 이용해서 구현할 수 있다.

```
void dfs(int x) {  
    check[x] = true;  
    for (int i=0; i<a[x].size(); i++) {  
        int y = a[x][i];  
        if (check[y] == false) {  
            dfs(y);  
        }  
    }  
}
```

$$O(V+E)$$

←
재귀
→ $O(E)$

- 인접 리스트를 이용한 구현

너비 우선 탐색

Breadth First Search

56

- 큐를 이용해서 지금 위치에서 갈 수 있는 것을 모두 큐에 넣는 방식
- 큐에 넣을 때 방문했다고 체크해야 한다.

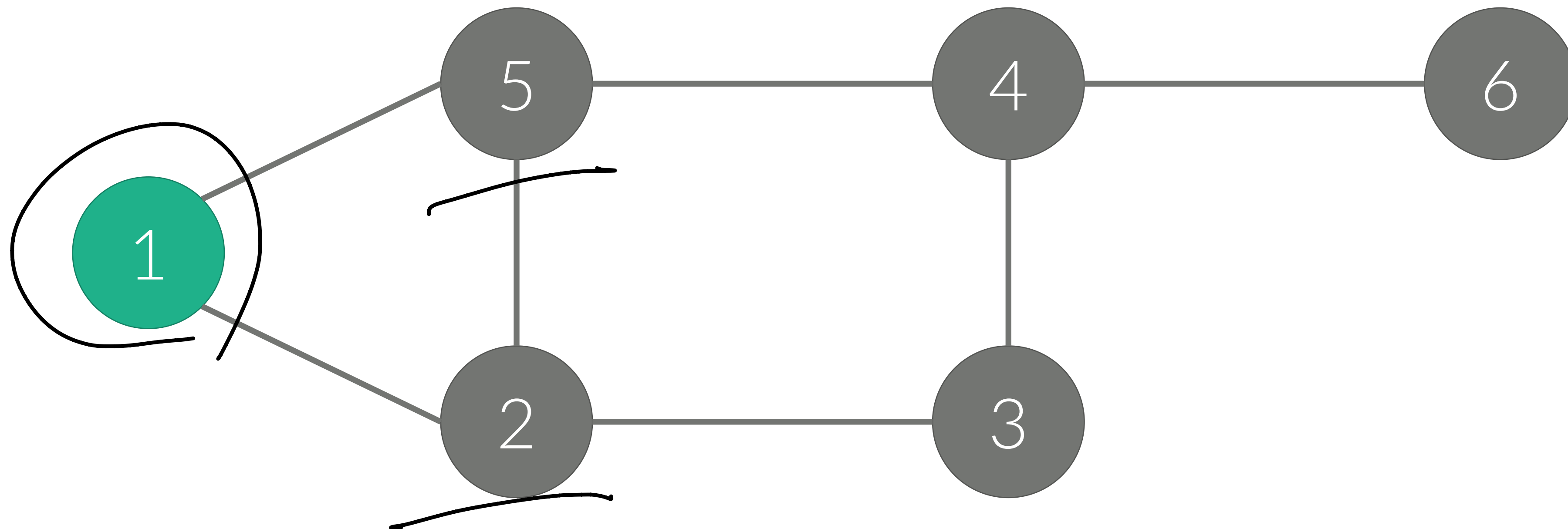
너비 우선 탐색

57

Breadth First Search

- 현재 정점: 1
- 순서: 1
- 큐: 1

i	1	2	3	4	5	6
check[i]	1	0	0	0	0	0



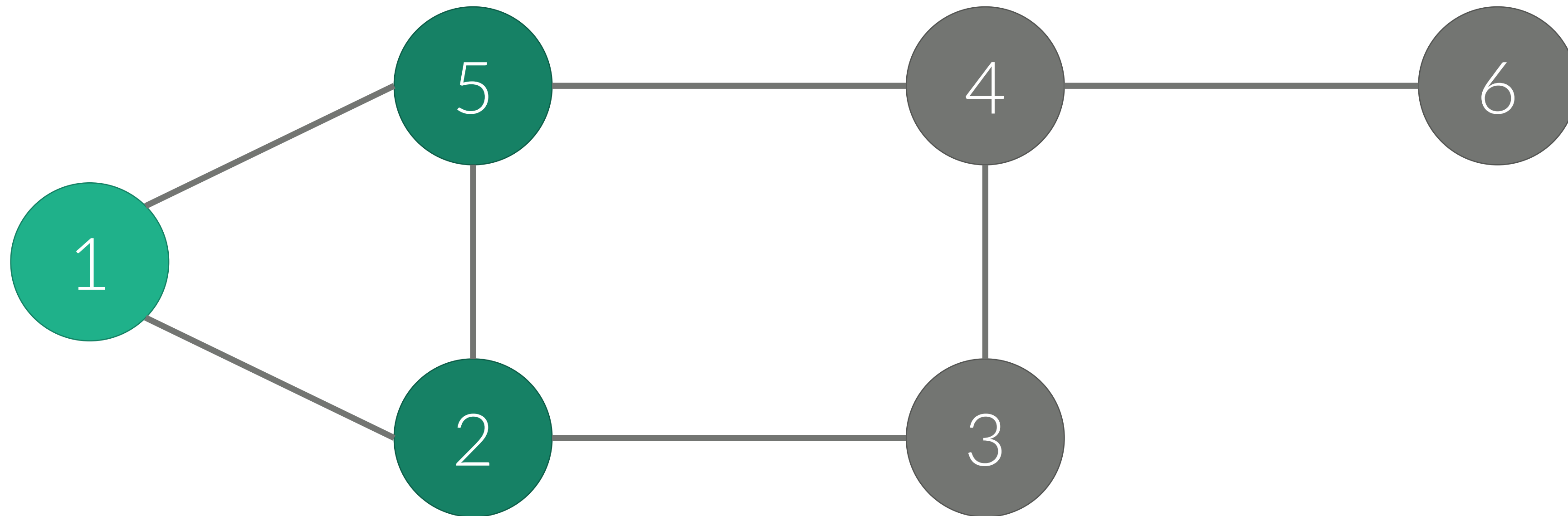
너비 우선 탐색

58

Breadth First Search

- 현재 정점: 1
- 순서: 1 2 5
- 큐: 1 2 5

i	1	2	3	4	5	6
check[i]	1	1	0	0	1	0



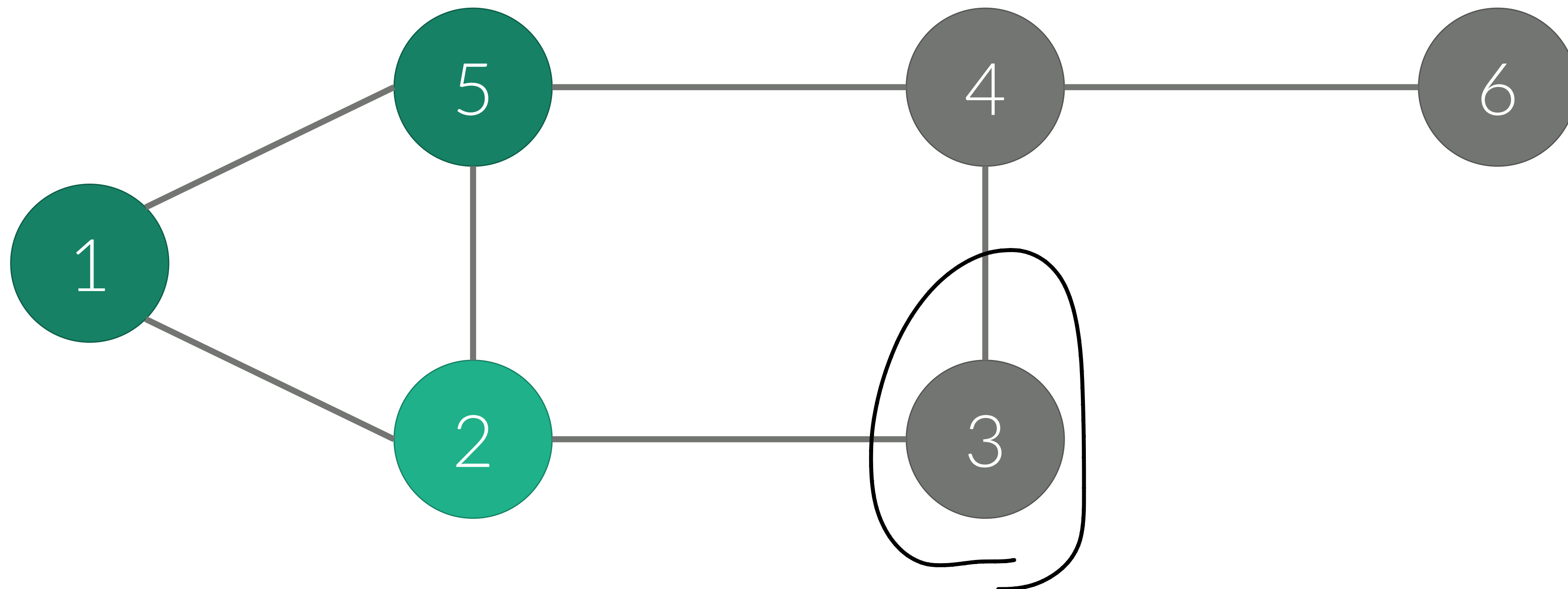
너비 우선 탐색

59

Breadth First Search

- 현재 정점: 2
- 순서: 1 2 5
- 큐: 2 5

i	1	2	3	4	5	6
check[i]	1	1	0	0	1	0



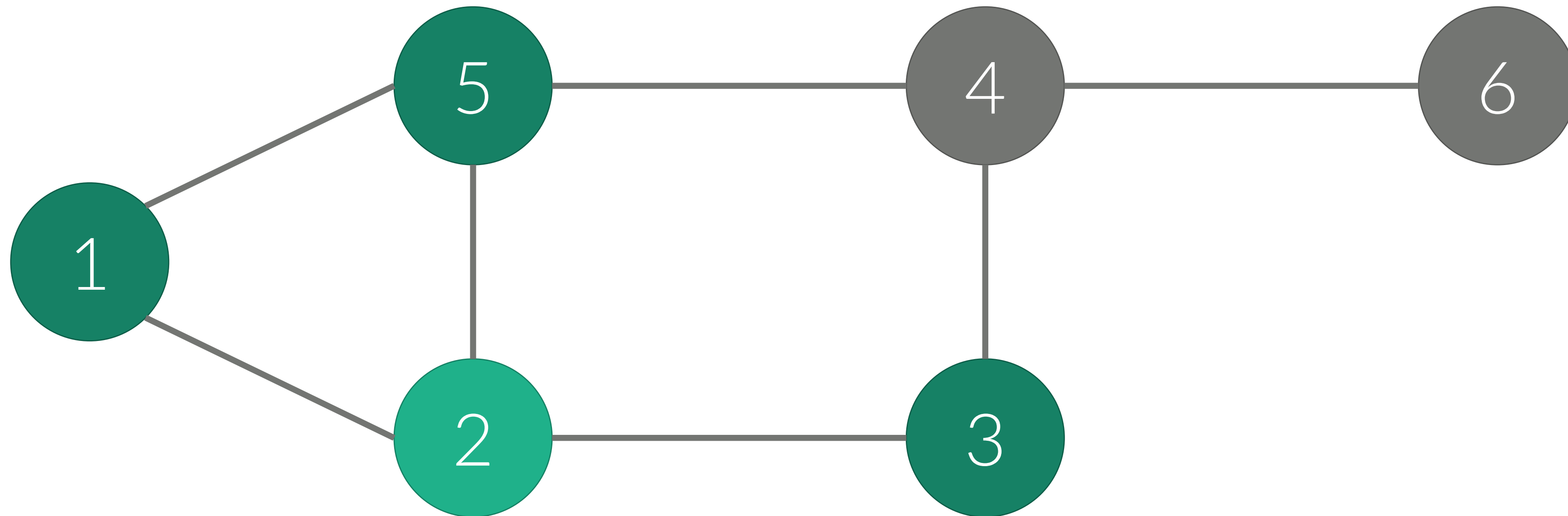
너비 우선 탐색

60

Breadth First Search

- 현재 정점: 2
- 순서: 1 2 5 3
- 큐: 2 5 3

i	1	2	3	4	5	6
check[i]	1	1	1	0	1	0



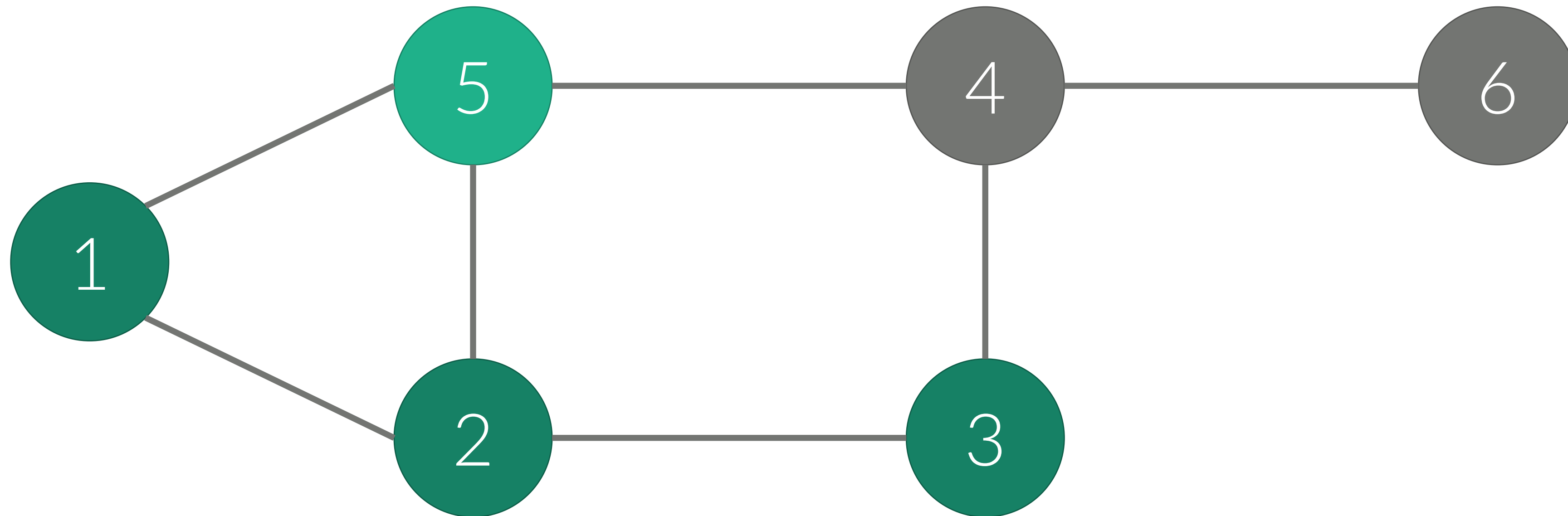
너비 우선 탐색

61

Breadth First Search

- 현재 정점: 5
- 순서: 1 2 5 3
- 큐: 5 3

i	1	2	3	4	5	6
check[i]	1	1	1	0	1	0



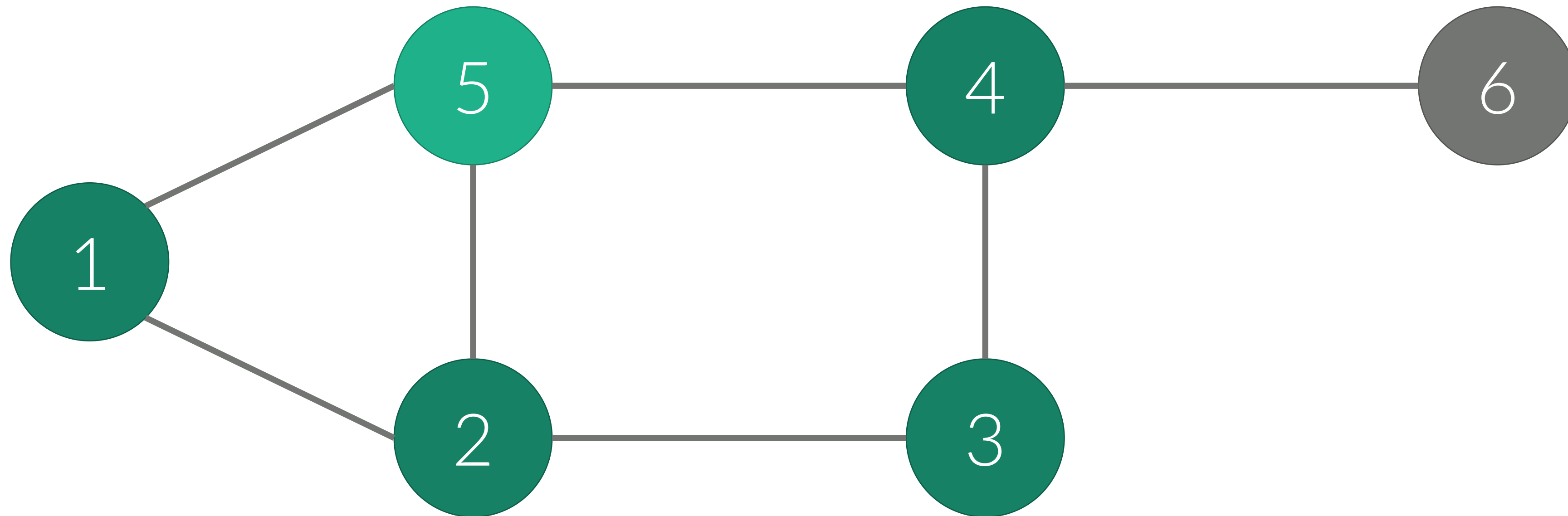
너비 우선 탐색

62

Breadth First Search

- 현재 정점: 5
- 순서: 1 2 5 3 4
- 큐: 5 3 4

i	1	2	3	4	5	6
check[i]	1	1	1	1	1	0



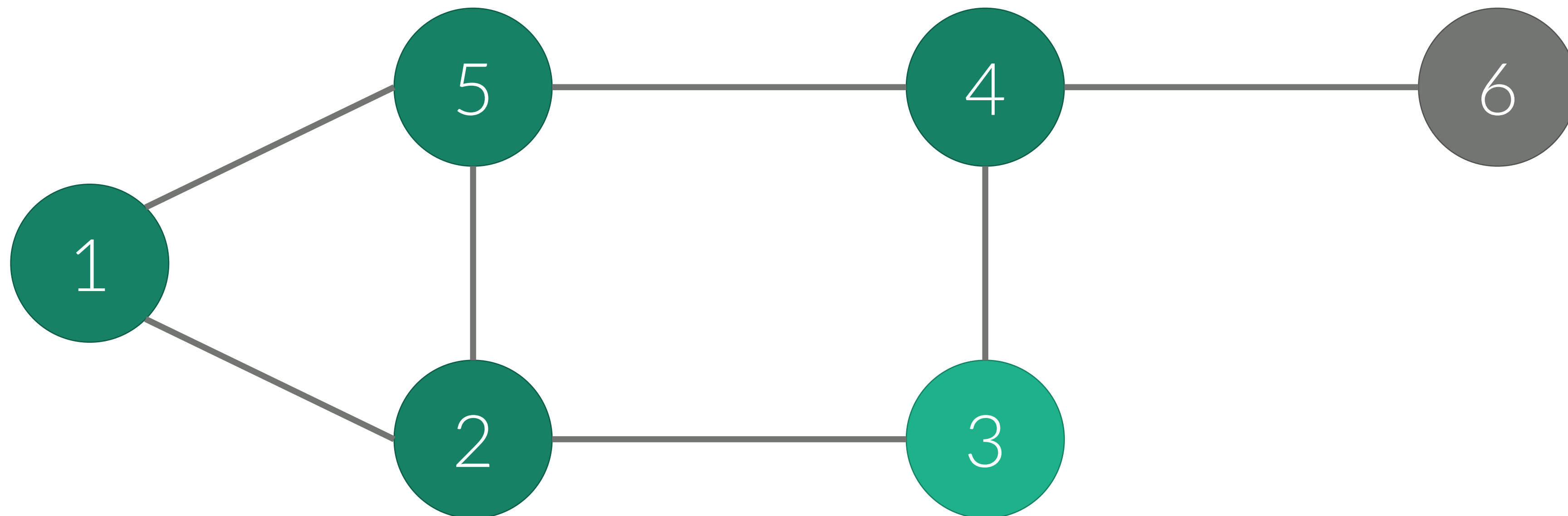
너비 우선 탐색

63

Breadth First Search

- 현재 정점: 3
- 순서: 1 2 5 3 4
- 큐: 3 4

i	1	2	3	4	5	6
check[i]	1	1	1	1	1	0



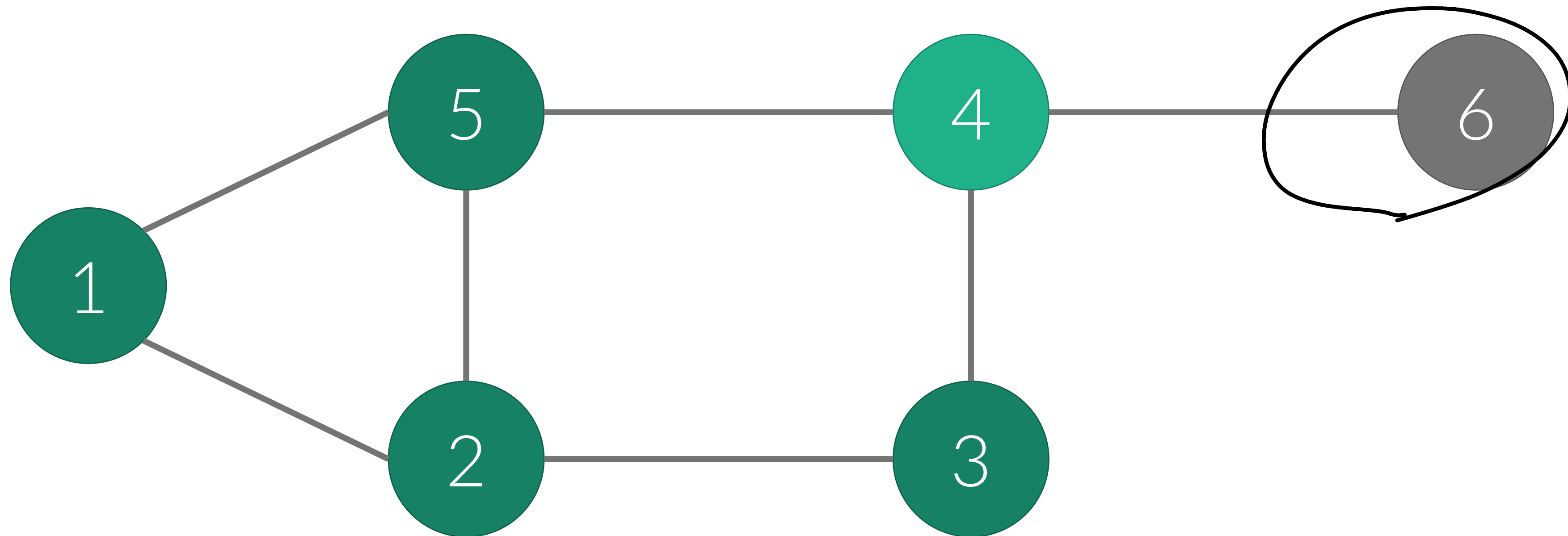
너비 우선 탐색

64

Breadth First Search

- 현재 정점: 4
- 순서: 1 2 5 3 4
- 큐: 4

i	1	2	3	4	5	6
check[i]	1	1	1	1	1	0



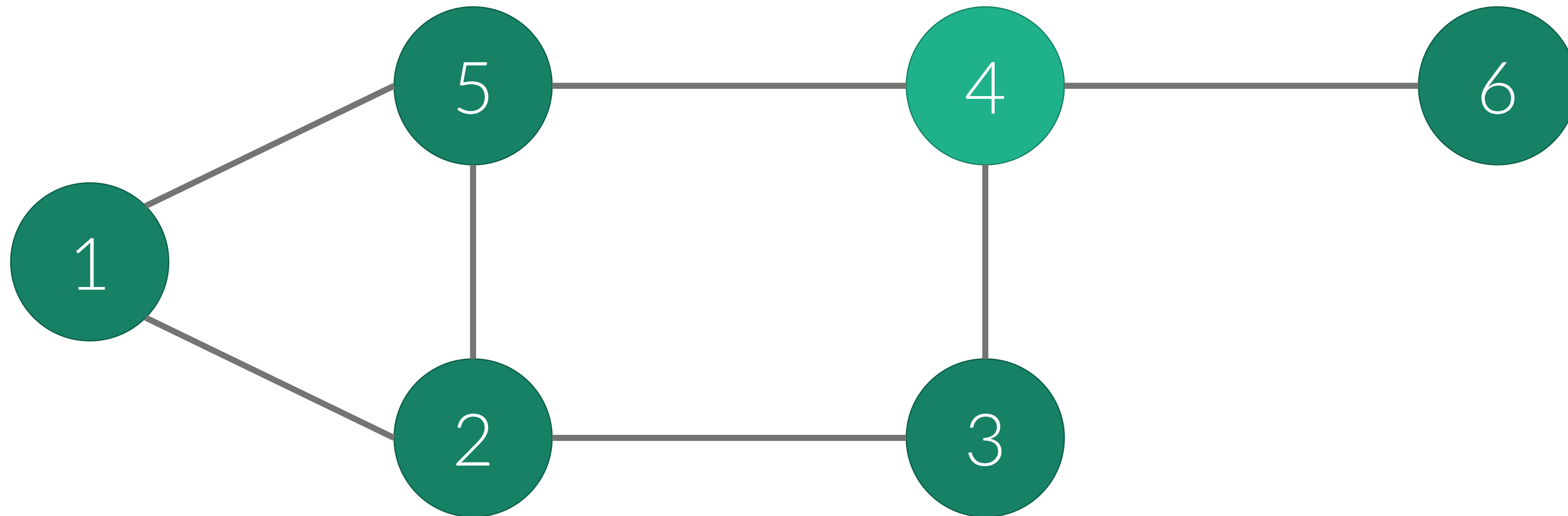
너비 우선 탐색

65

Breadth First Search

- 현재 정점: 4
- 순서: 1 2 5 3 4 6
- 큐: 4 6

i	1	2	3	4	5	6
check[i]	1	1	1	1	1	1



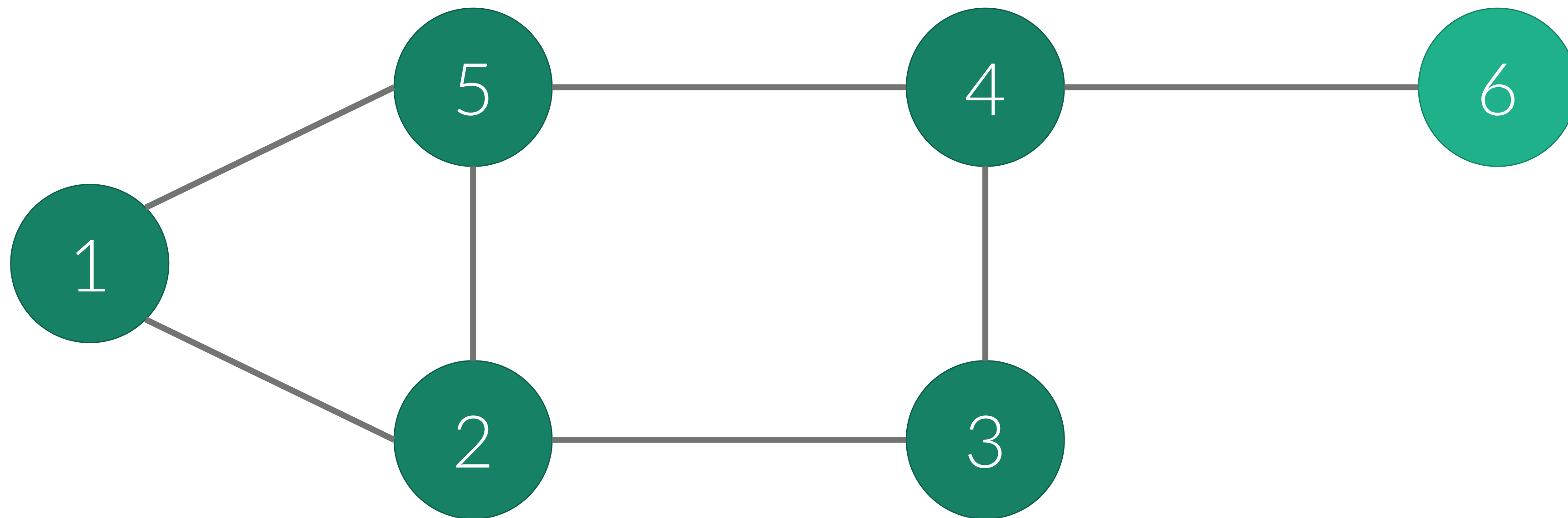
너비 우선 탐색

66

Breadth First Search

- 현재 정점: 6
- 순서: 1 2 5 3 4 6
- 큐: 6

i	1	2	3	4	5	6
check[i]	1	1	1	1	1	1



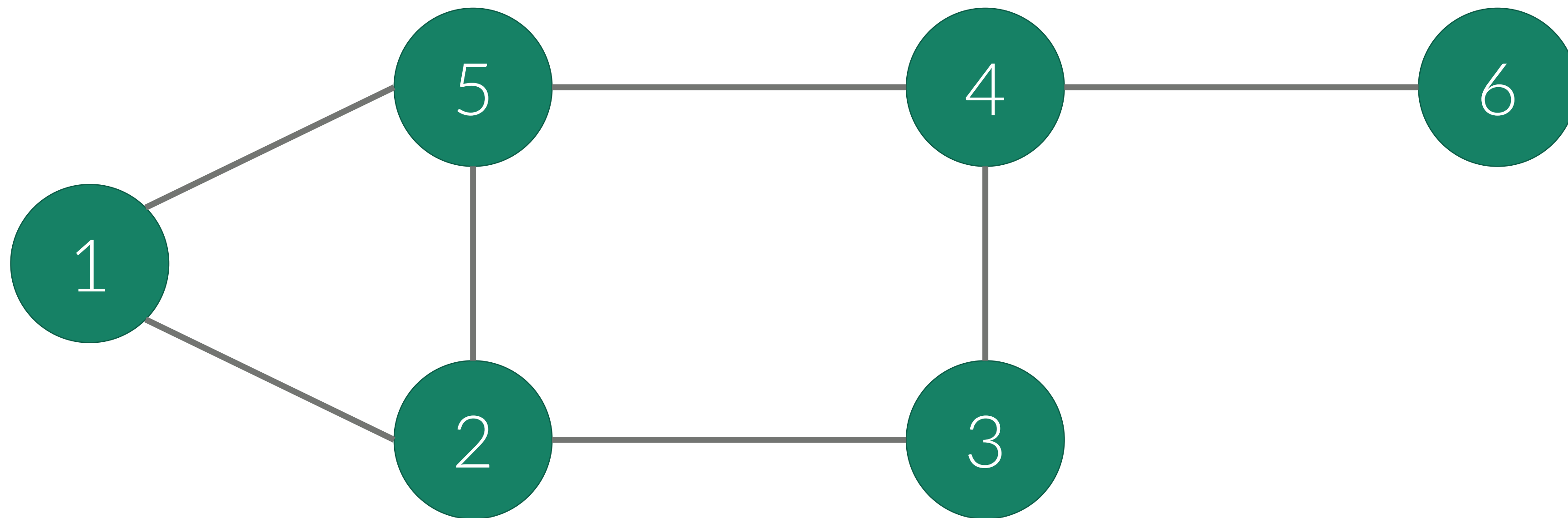
너비 우선 탐색

67

Breadth First Search

- 현재 정점: 6
- 순서: 1 2 5 3 4 6
- 큐:
- 탐색 완료

i	1	2	3	4	5	6
check[i]	1	1	1	1	1	1



너비 우선 탐색

Breadth First Search

- BFS의 구현은 큐를 이용해서 할 수 있다. (인접 행렬)

$$O(V^2)$$

```
queue<int> q; →
```

```
check[1] = true; q.push(1); → 시작
```

```
while (!q.empty()) {
```

```
    int x = q.front(); q.pop();
```

```
    for (int i=1; i<=n; i++) {
```

```
        if (a[x][i] == 1 && check[i] == false) {
```

```
            check[i] = true;
```

```
            q.push(i);
```

```
        }
```

```
    }
```

```
}
```

⊗

⊗ — ⊗

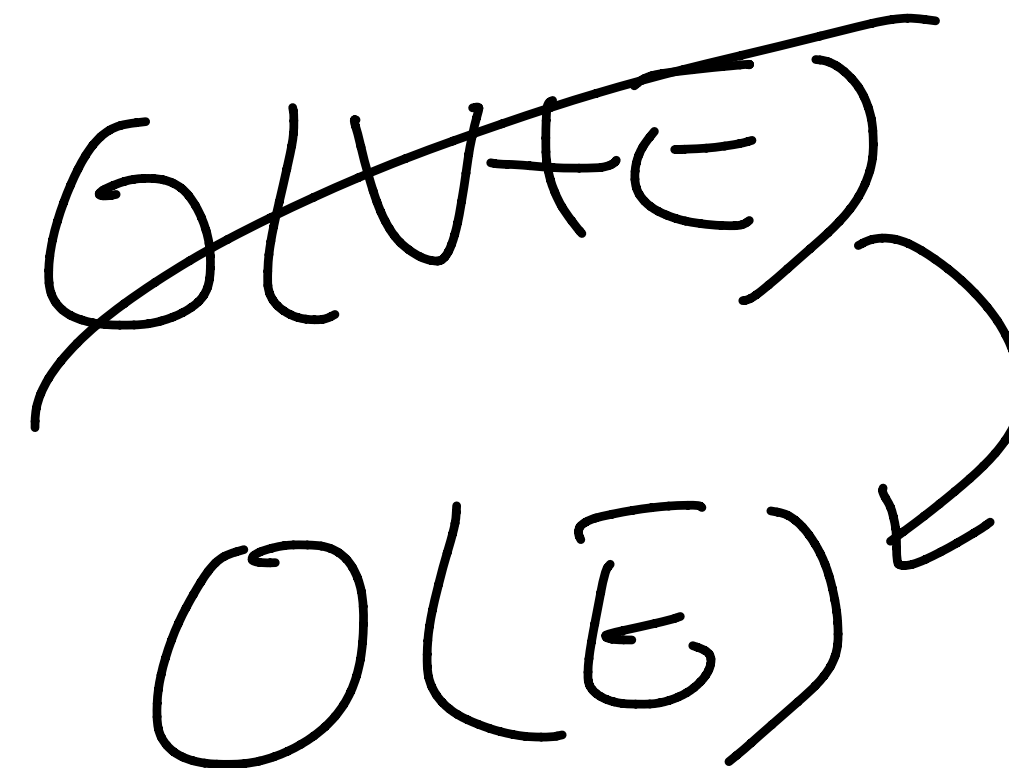
너비 우선 탐색

Breadth First Search

- BFS의 구현은 큐를 이용해서 할 수 있다. (인접 리스트)

```
queue<int> q;
check[1] = true; q.push(1);
while (!q.empty()) {
    int x = q.front(); q.pop();
    for (int i=0; i<a[x].size(); i++) {
        int y = a[x][i];
        if (check[y] == false) {
            check[y] = true; q.push(y);
        }
    }
}
```

~~$O(V+E)$~~
 $O(E)$



시간 복잡도

Time Complexity

- 인접 행렬: $O(V^2)$
- 인접 리스트: $O(V+E)$

DFS와 BFS

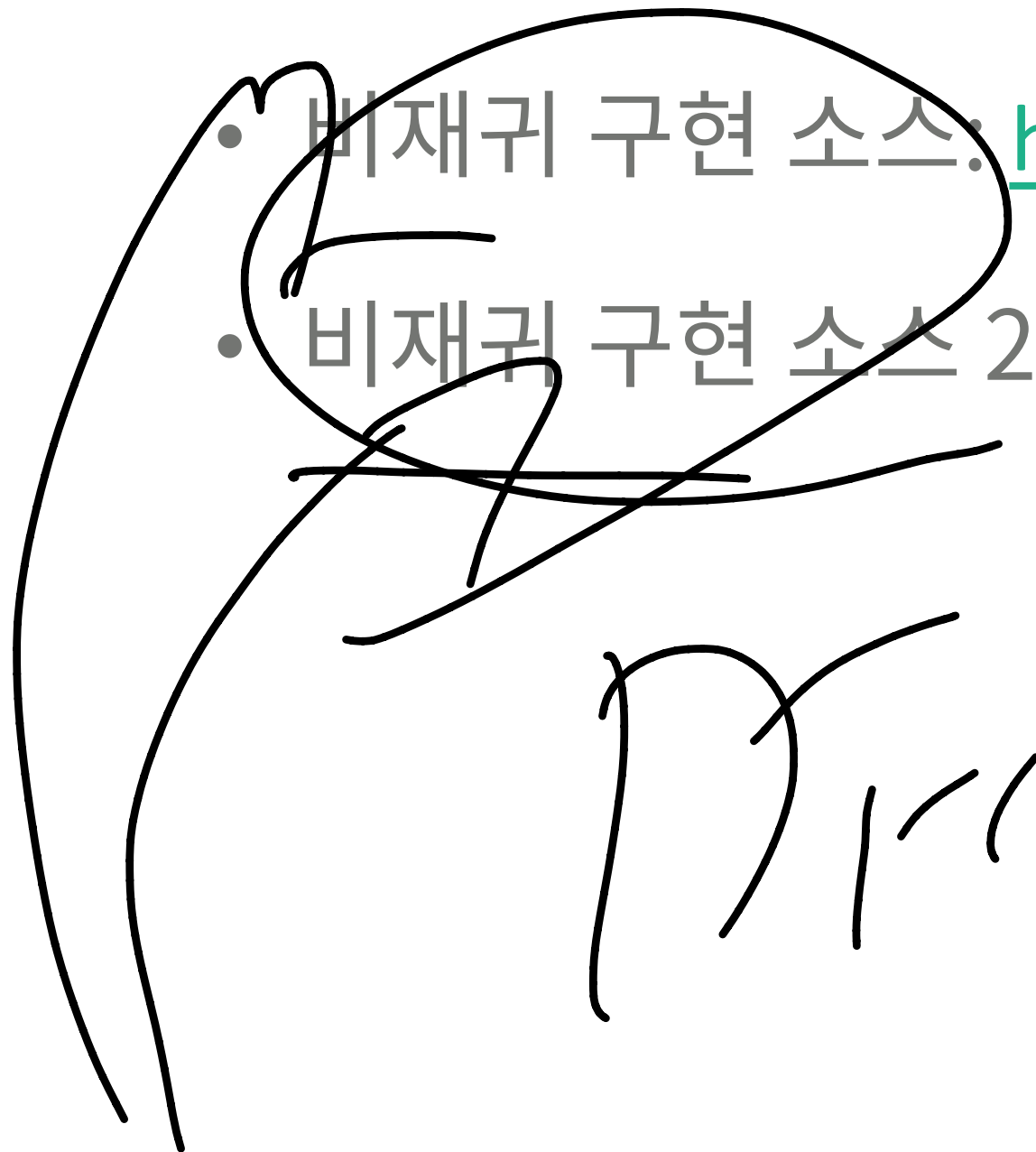
<https://www.acmicpc.net/problem/1260>

- 그래프를 DFS로 탐색한 결과와 BFS로 탐색한 결과를 출력하는 문제

DFS와 BFS

<https://www.acmicpc.net/problem/1260>

- 인접 리스트 소스: <http://codeplus.codes/8749c68651d64a4993d3e47c476886e4>
- 간선 리스트 소스: <http://codeplus.codes/304dc84b3eb04e1b9832505dae72a0fc>
- 비재귀 구현 소스: <http://codeplus.codes/3844f91fdbbc424ca8122481dfdf479b>
- 비재귀 구현 소스 2: <http://codeplus.codes/f020aad01e0c426ca73624bf3580d376>



연결 요소

연결 요소

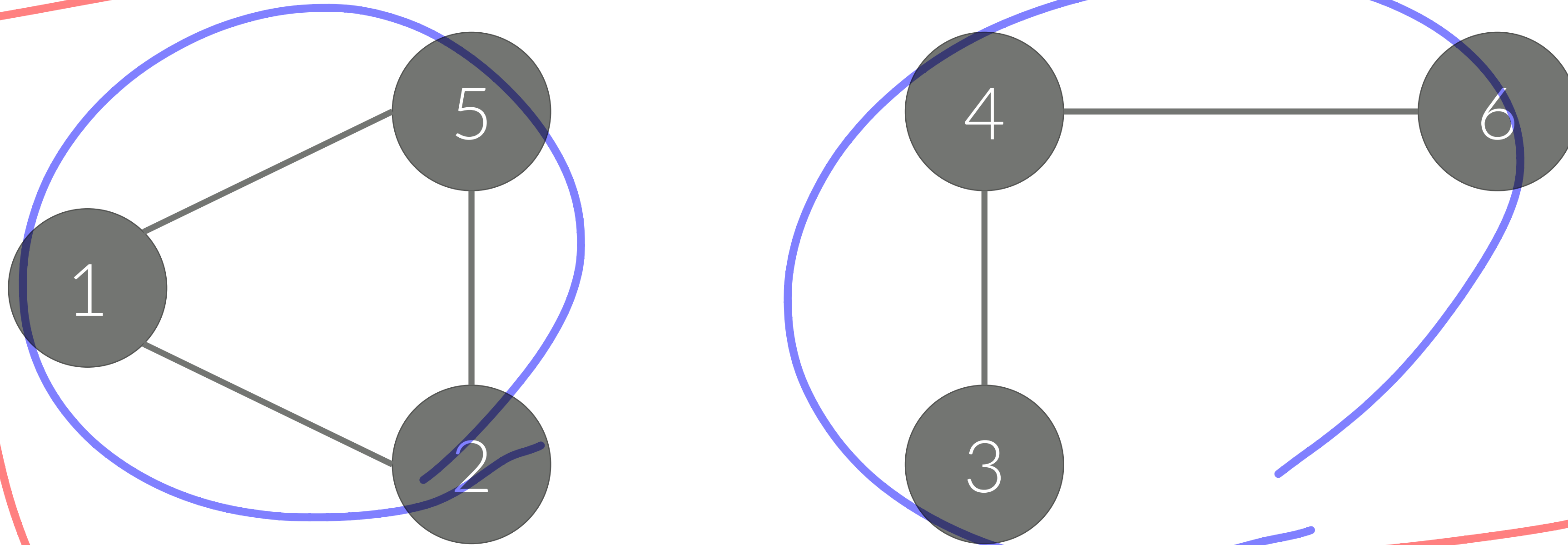
Connected Component

2개
[개]

연결 그래프

74

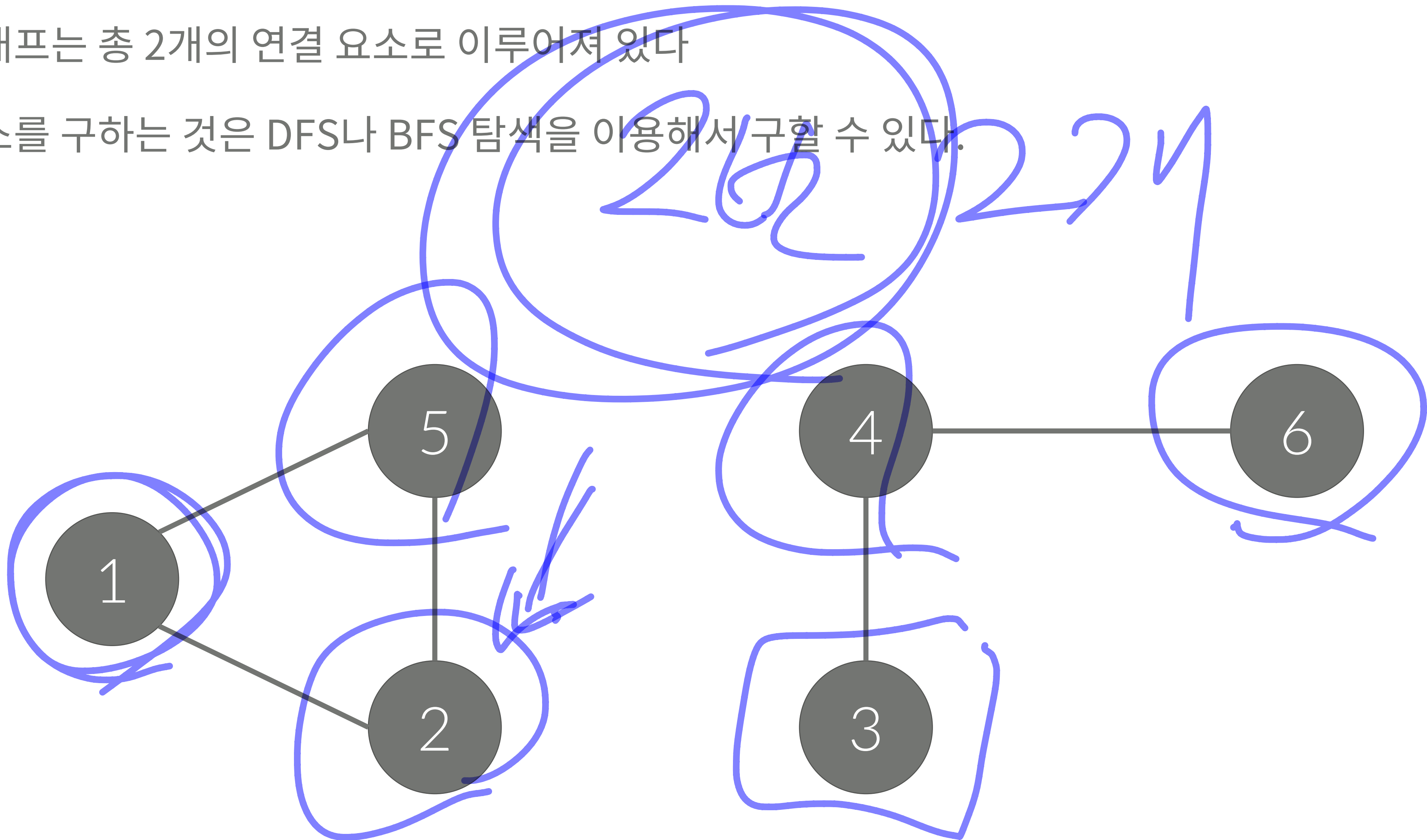
- 그래프가 아래 그림과 같이 나누어져 있지 않은 경우가 있을 수도 있다
- 이렇게 나누어진 각각의 그래프를 연결 요소라고 한다.
- 연결 요소에 속한 모든 정점을 연결하는 경로가 있어야 한다
- 또, 다른 연결 요소에 속한 정점과 연결하는 경로가 있으면 안된다



연결 요소

Connected Component

- 아래 그래프는 총 2개의 연결 요소로 이루어져 있다
- 연결 요소를 구하는 것은 DFS나 BFS 탐색을 이용해서 구할 수 있다.



연결 요소

<https://www.acmicpc.net/problem/11724>

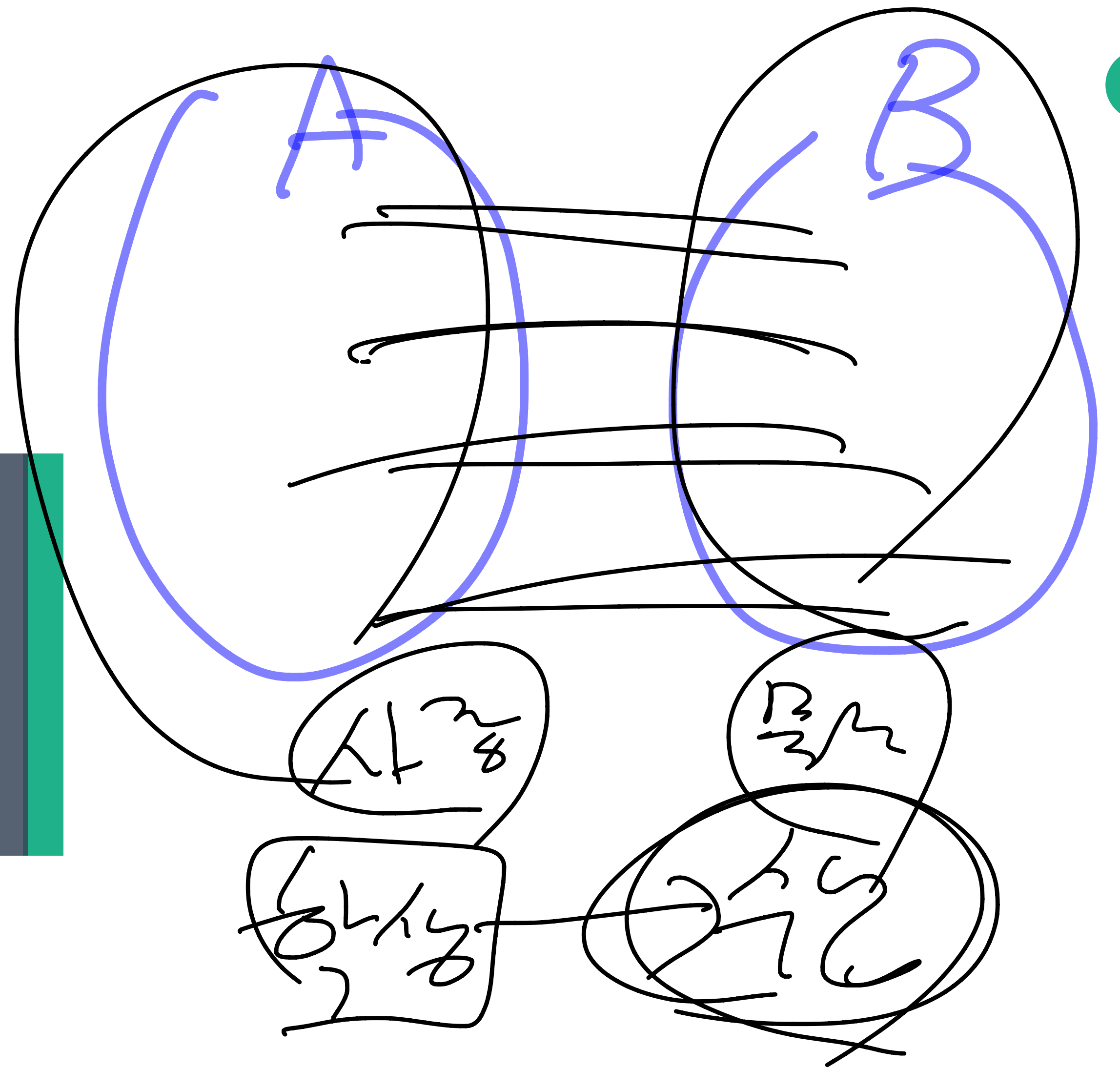
- 연결 요소의 개수를 구하는 문제

연결 요소

<https://www.acmicpc.net/problem/11724>

- 소스: <http://codeplus.codes/dbcc0c3d8022443d9a6319e16a1e2c99>

이분 그래프



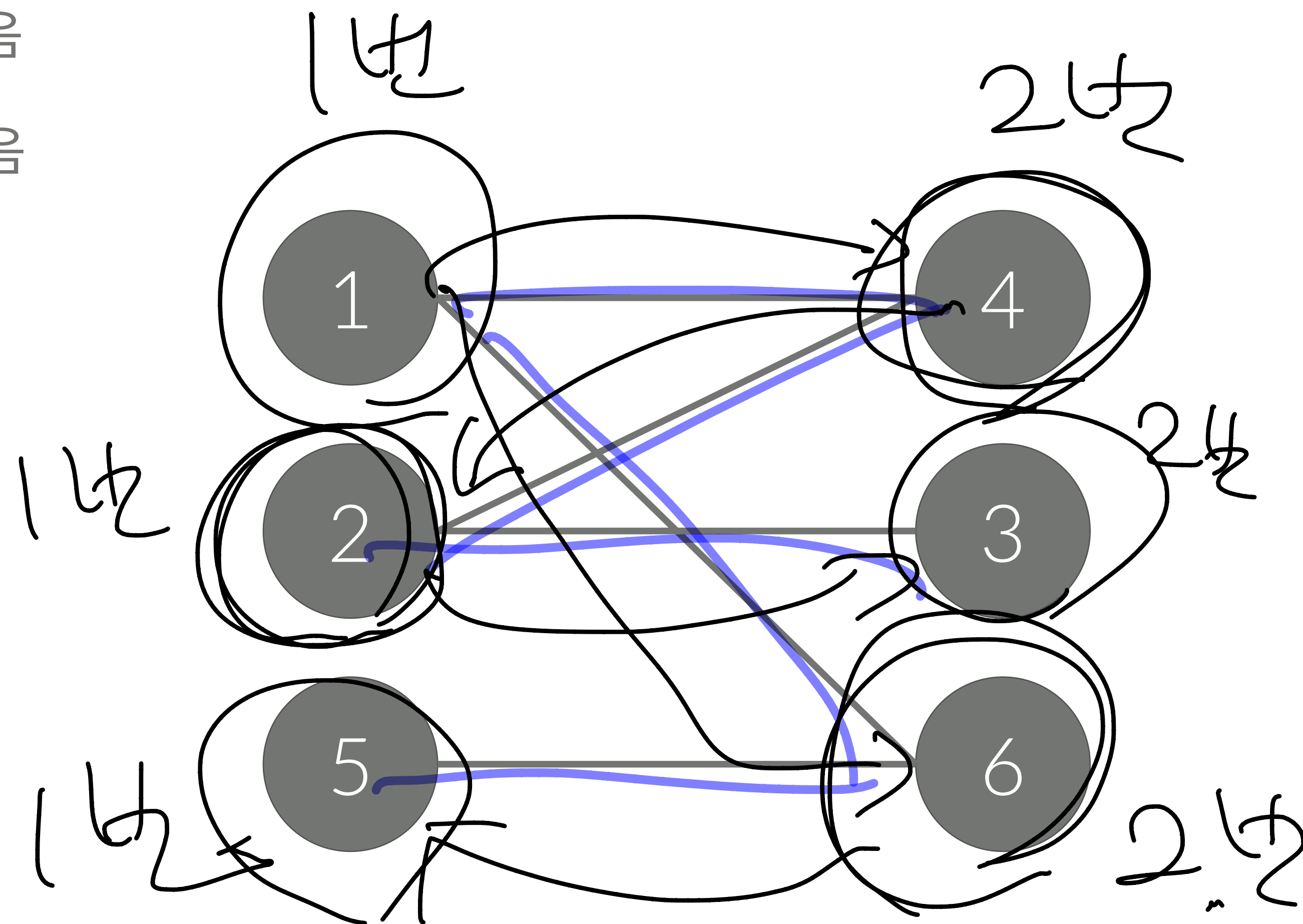
이분 그래프

Bipartite Graph

그림 ①, ②

79

- 그래프를 다음과 같이 A와 B로 나눌 수 있으면 이분 그래프라고 한다.
- A에 포함되어 있는 정점끼리 연결된 간선이 없음
- B에 포함되어 있는 정점끼리 연결된 간선이 없음
- 모든 간선의 한 끝 점은 A에, 다른 끝 점은 B에

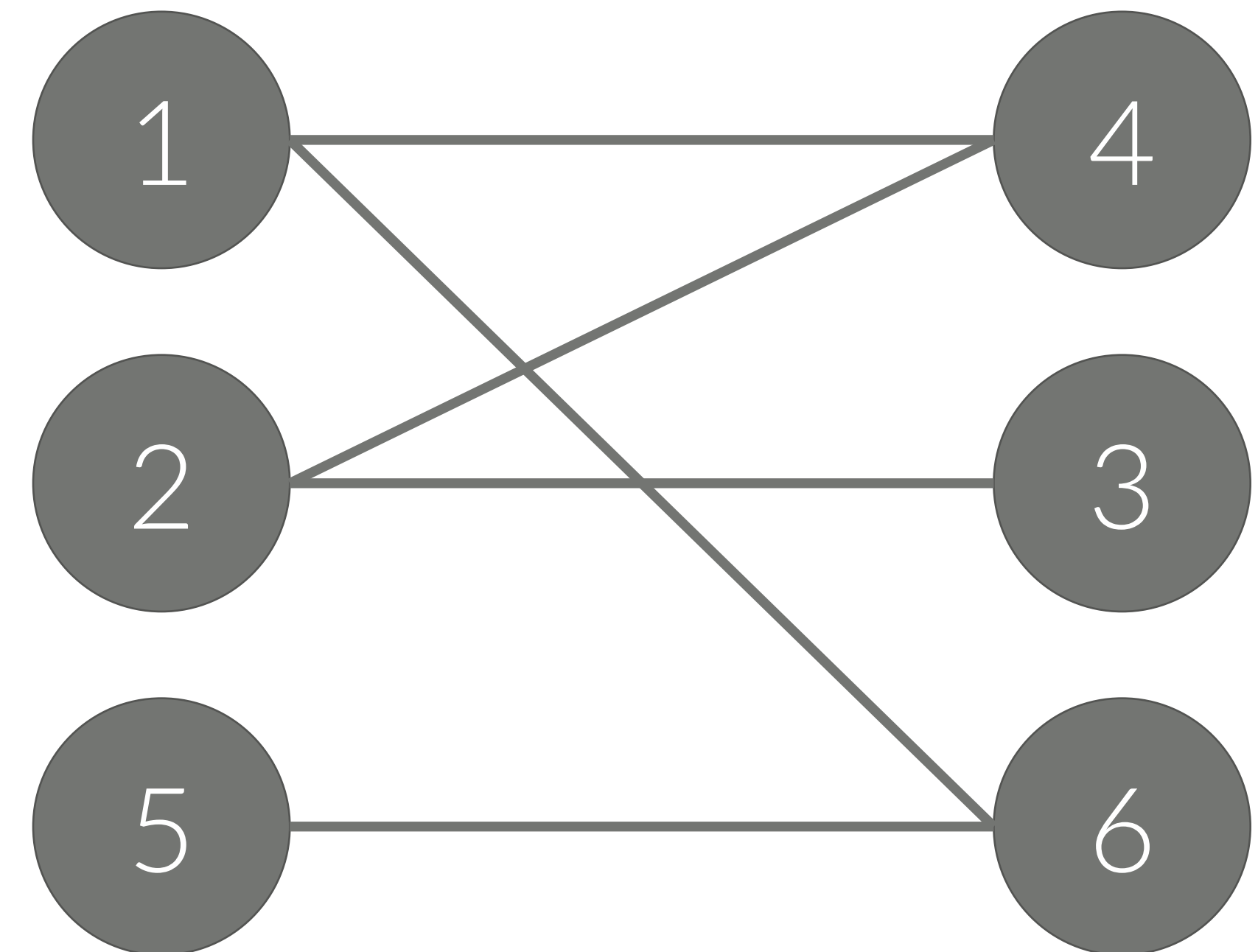


이분 그래프

Bipartite Graph

80

- 그래프를 DFS또는 BFS 탐색으로 이분 그래프인지 아닌지 알아낼 수 있다.



이분 그래프

<https://www.acmicpc.net/problem/1707>

- 그래프가 이분 그래프인지 아닌지 판별하는 문제

이분 그래프

<https://www.acmicpc.net/problem/1707>

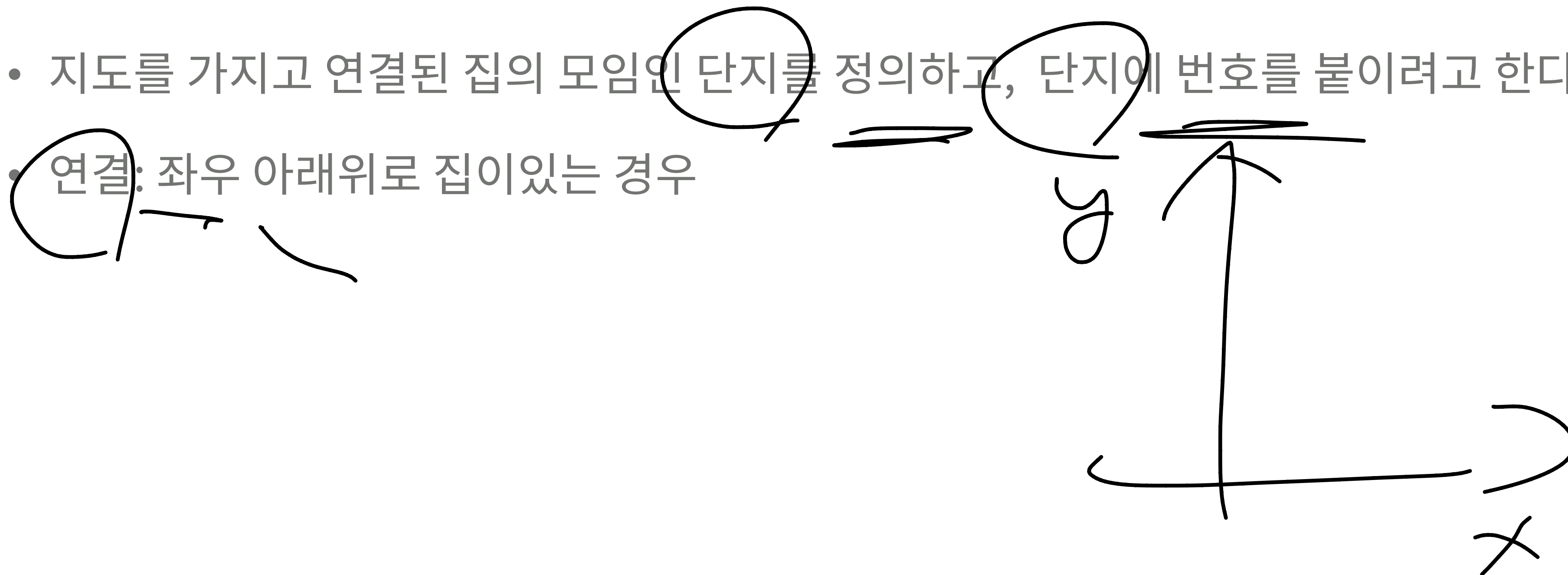
- 소스: <http://codeplus.codes/3766db41630d45e5a2a9edba40557a5c>
- 소스 2: <http://codeplus.codes/96cd1686b52f4740b72e52310bb01041>

그래프 문제

단지번호붙이기

<https://www.acmicpc.net/problem/2667>

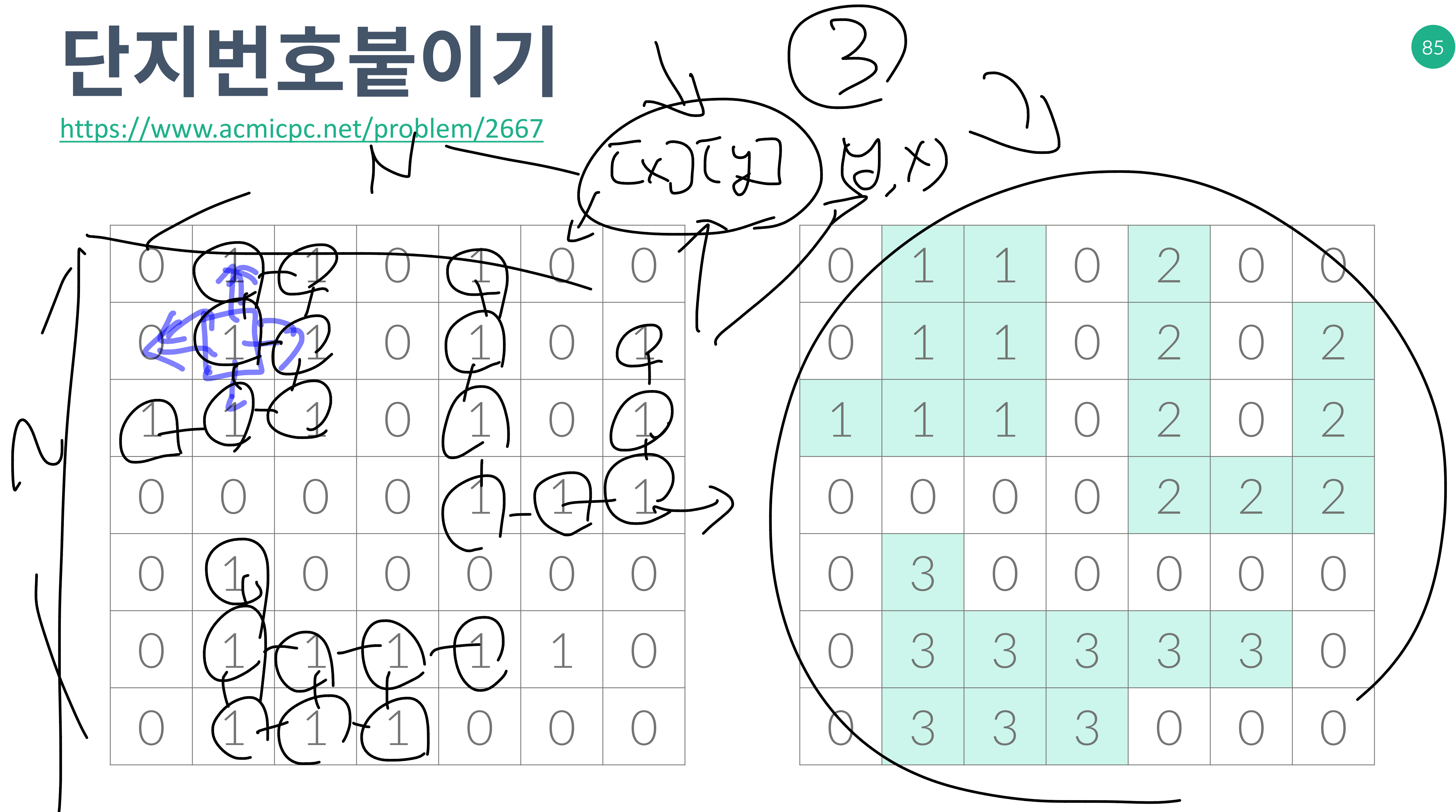
- 정사각형 모양의 지도가 있다
- 0은 집이 없는 곳, 1은 집이 있는 곳
- 지도를 가지고 연결된 집의 모임인 단지를 정의하고, 단지에 번호를 붙이려고 한다
- 연결: 좌우 아래위로 집이있는 경우



단지번호붙이기

<https://www.acmicpc.net/problem/2667>

85



단지번호붙이기

<https://www.acmicpc.net/problem/2667>

- DFS나 BFS 알고리즘을 이용해서 어떻게 이어져있는지 확인할 수 있다.
- $d[i][j] = (i, j)$ 를 방문안했으면 0, 했으면 단지 번호

단지번호붙이기

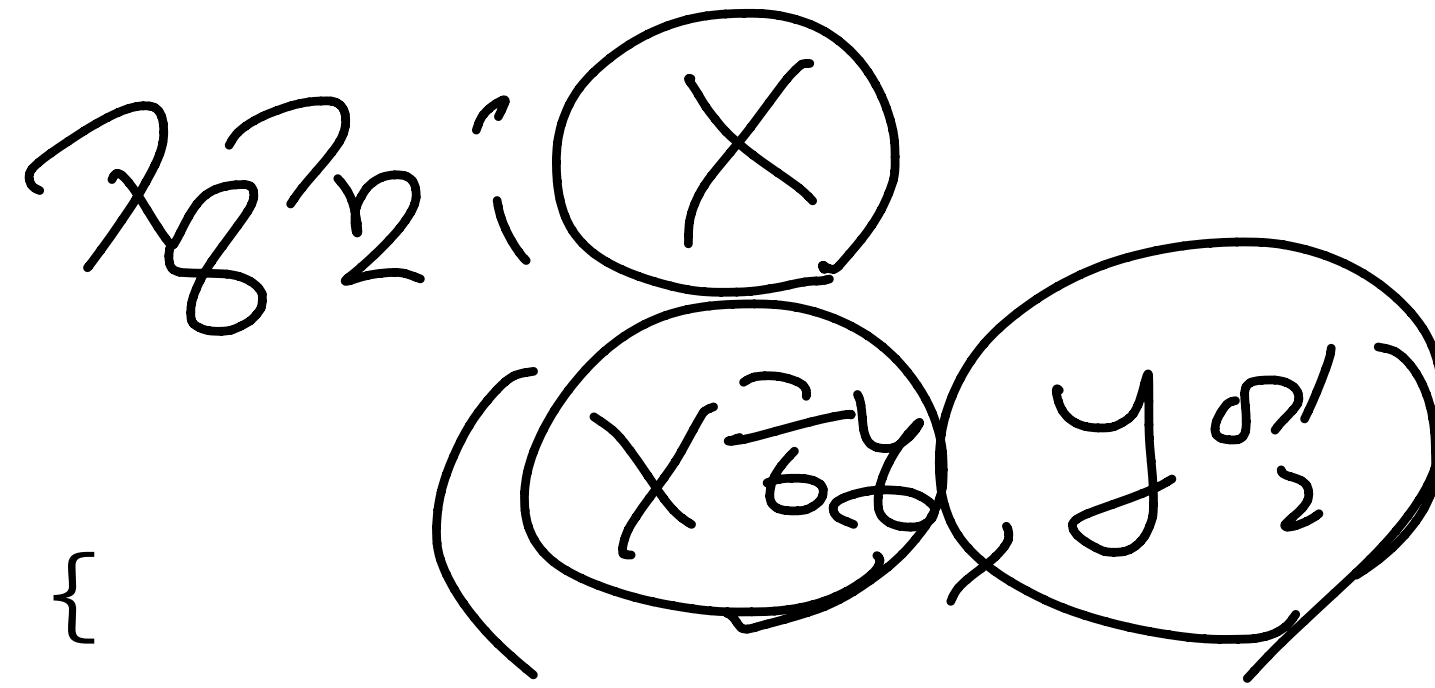
<https://www.acmicpc.net/problem/2667>

```
int cnt = 0;
for (int i=0; i<n; i++) {
    for (int j=0; j<n; j++) {
        if (a[i][j] == 1 && d[i][j] == 0) {
            bfs(i, j, ++cnt);
        }
    }
}
```

단지번호붙이기

<https://www.acmicpc.net/problem/2667>

88



```
void bfs(int x, int y, int cnt) {
```

```
    queue<pair<int, int>> q; q.push(make_pair(x, y)); d[x][y] = cnt;
```

```
    while (!q.empty()) {
```

Handwritten note: $x = q.front().first$, $y = q.front().second$; $q.pop()$;

```
        for (int k=0; k<4; k++) {
```

```
            int nx = x+dx[k], ny = y+dy[k];
```

```
            if (0 <= nx && nx < n && 0 <= ny && ny < n) {
```

```
                if (a[nx][ny] == 1 && d[nx][ny] == 0) {
```

```
                    q.push(make_pair(nx, ny)); d[nx][ny] = cnt;
```

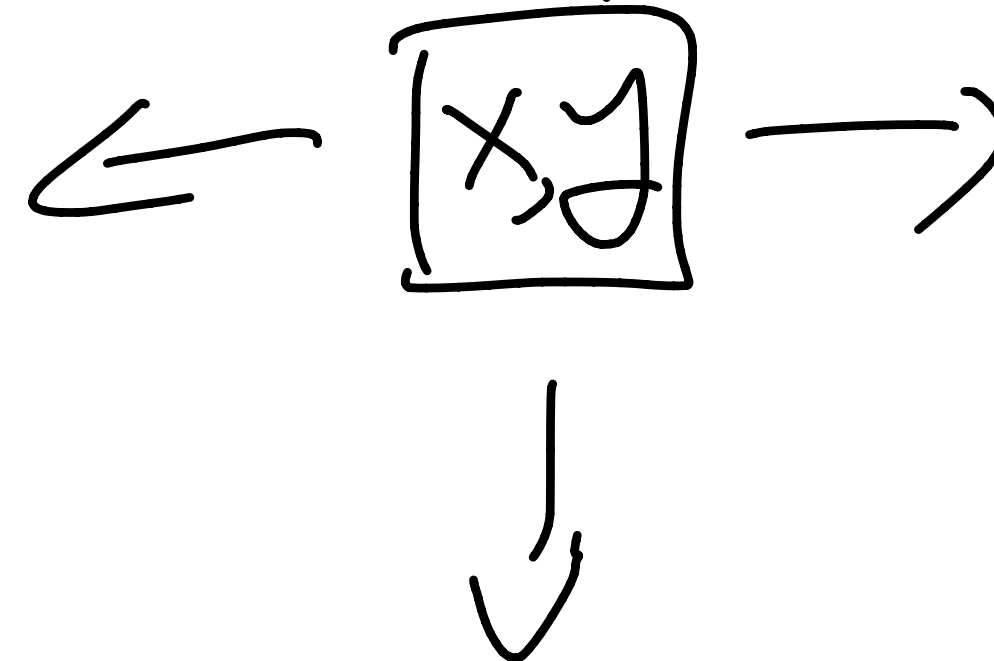
```
                }
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

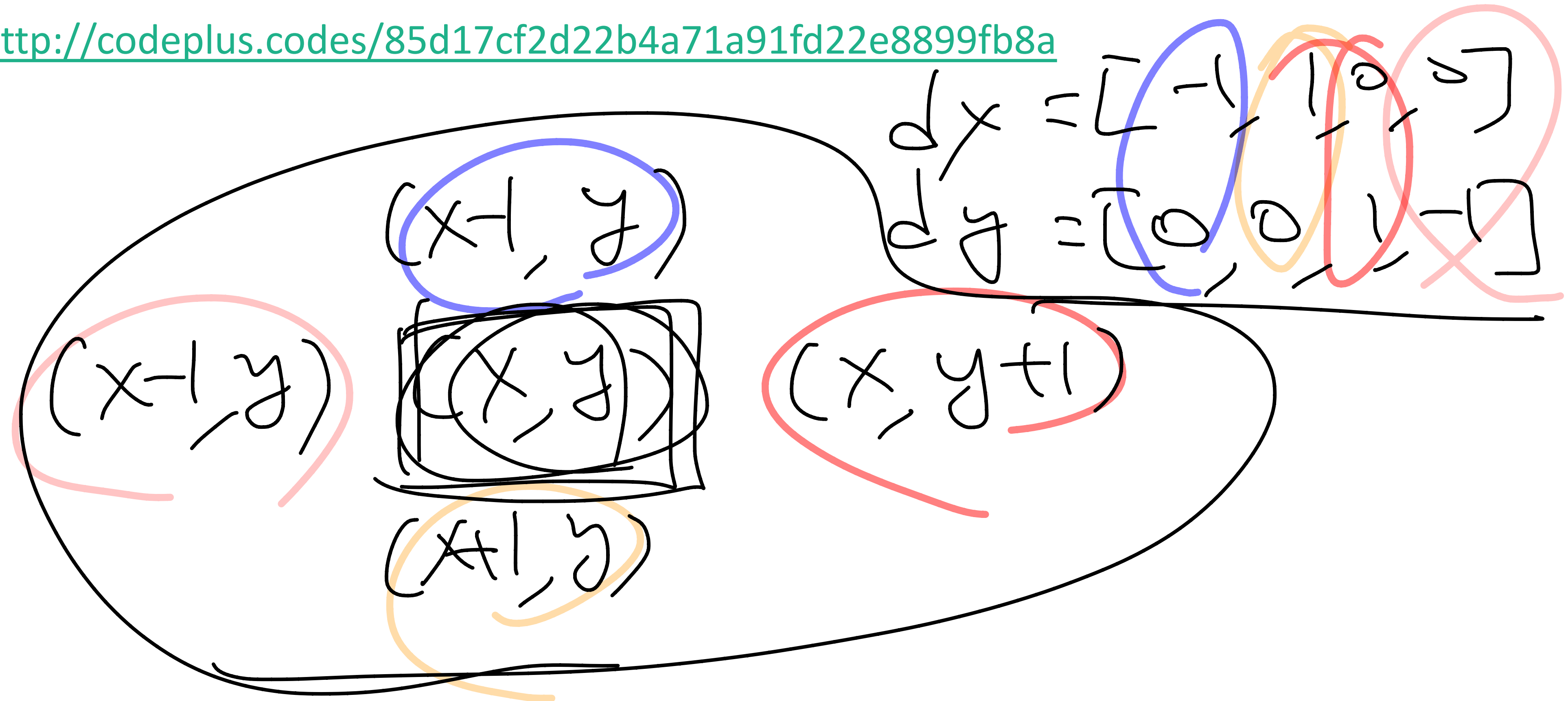


단지번호붙이기

89

<https://www.acmicpc.net/problem/2667>

- BFS 소스: <http://codeplus.codes/fc2c76b7935048969af1aff47e3cb75e>
- DFS 소스: <http://codeplus.codes/85d17cf2d22b4a71a91fd22e8899fb8a>

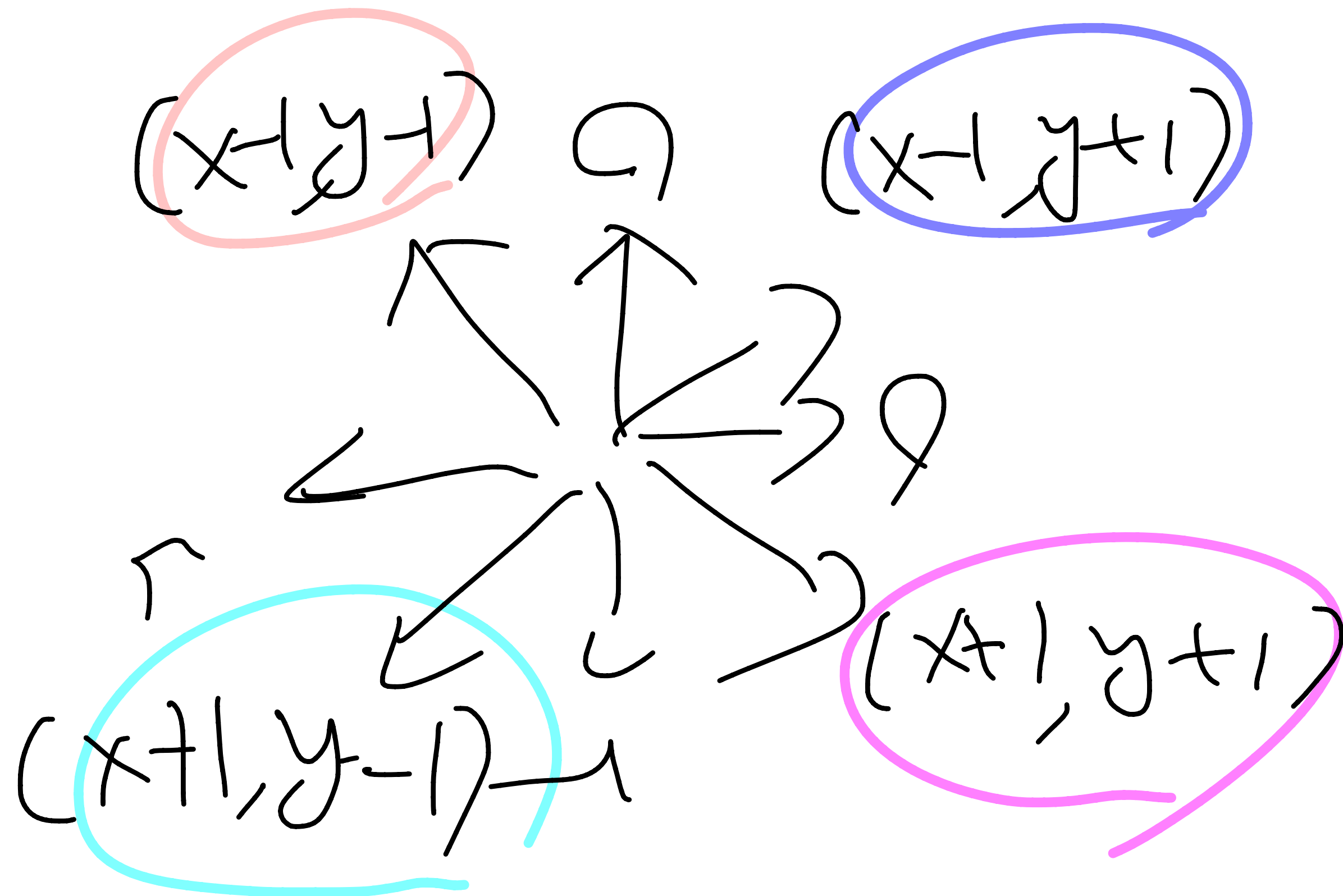


섬의 개수

90

<https://www.acmicpc.net/problem/4963>

- 소스: <http://codeplus.codes/c77e2c04ccca42beab30e743fdab2282>

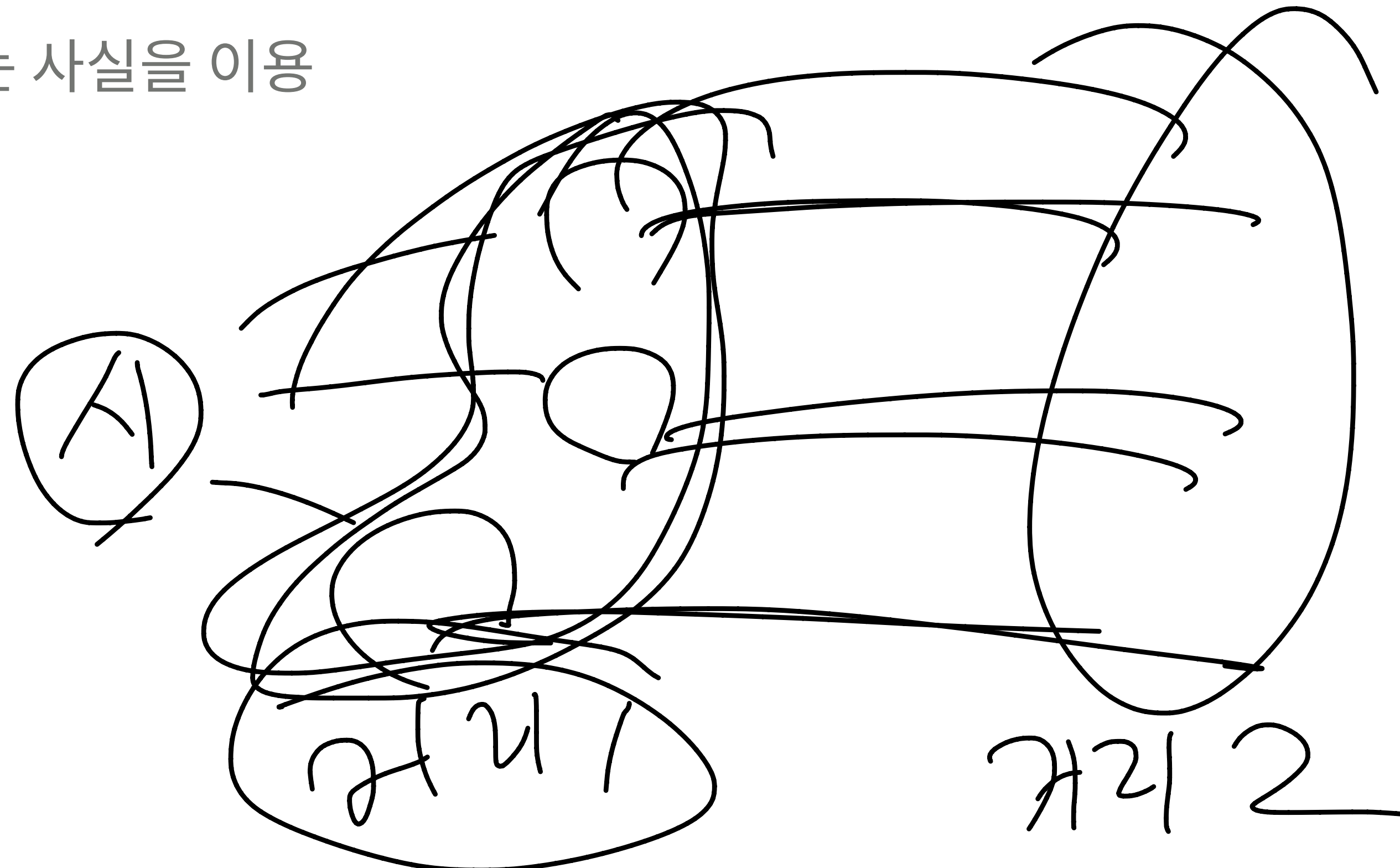


$$dx = [1, -1, 0, 0, -1, -1, 1, 1]$$
$$dy = [0, 0, 1, -1, -1, 1, 1, -1]$$

미로 탐색

<https://www.acmicpc.net/problem/2178>

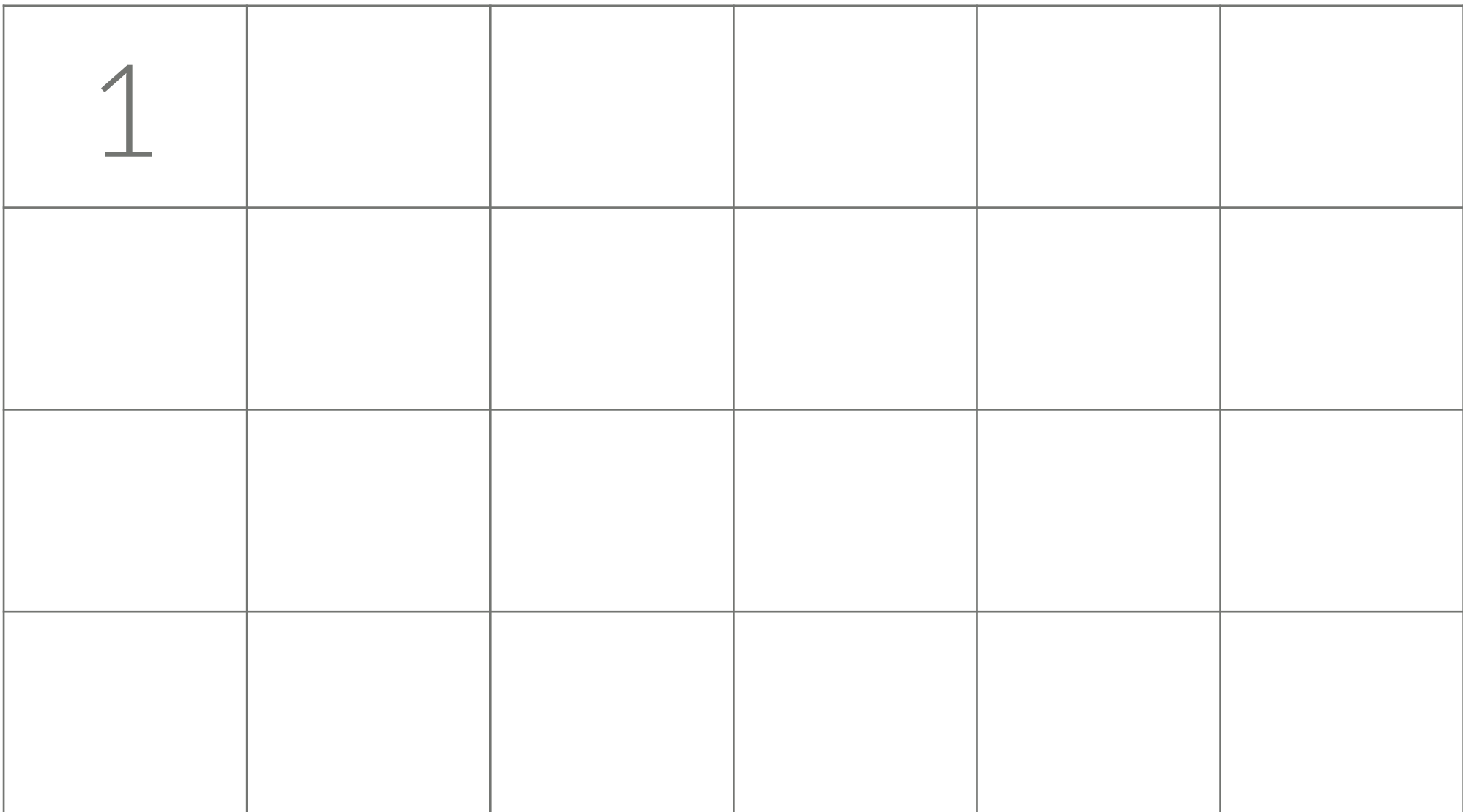
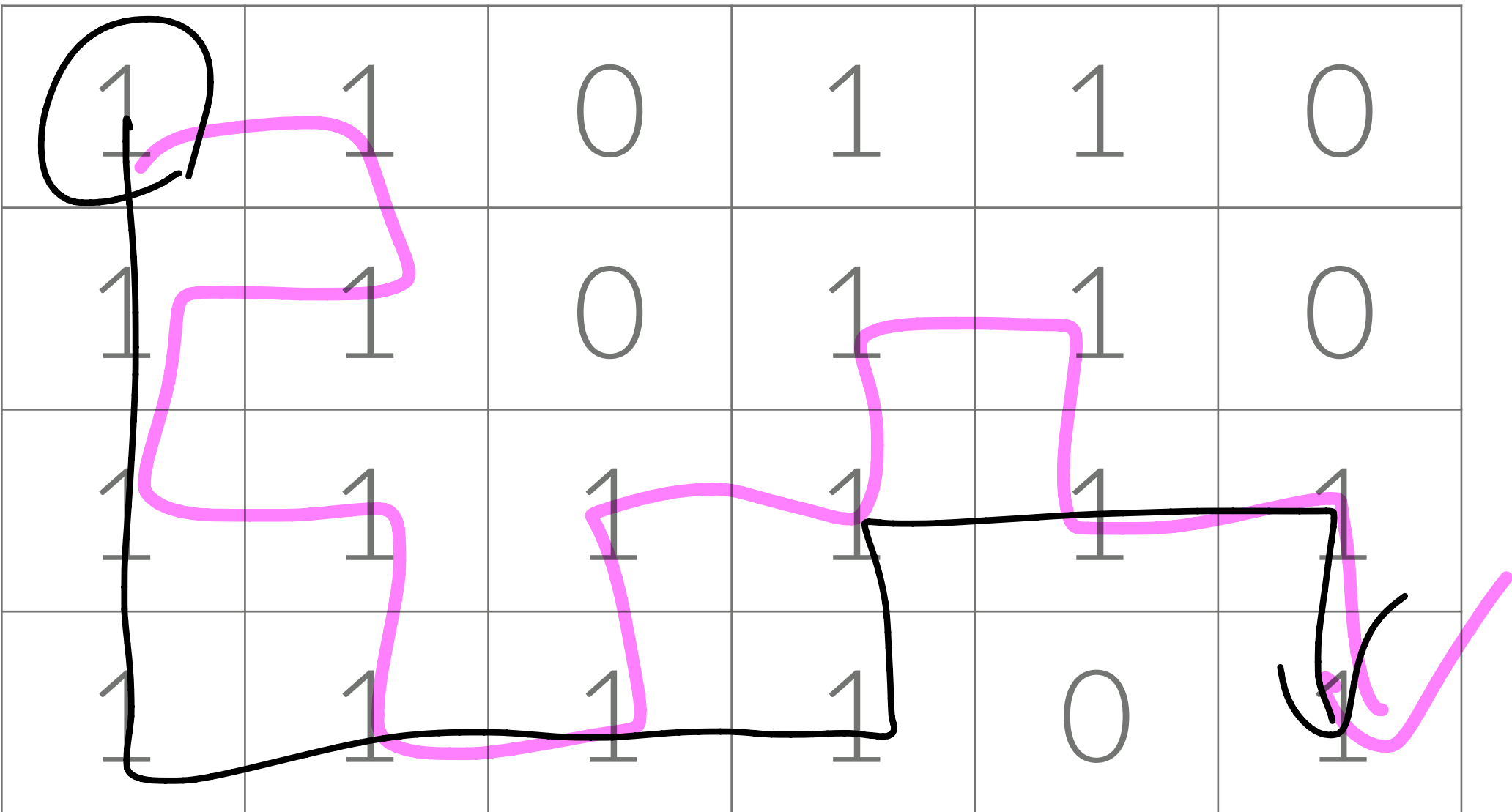
- (1, 1) 에서 (N, M)으로 가는 가장 빠른 길을 구하는 문제
- DFS 탐색으로는 문제를 풀 수 없다.
- BFS 탐색을 사용해야 한다.
- BFS는 단계별로 진행된다는 사실을 이용



미로 탐색

<https://www.acmicpc.net/problem/2178>

- (1, 1) 에서 (N, M)으로 가는 가장 빠른 길을 구하는 문제



미로 탐색

<https://www.acmicpc.net/problem/2178>

- (1, 1) 에서 (N, M)으로 가는 가장 빠른 길을 구하는 문제

1	1	0	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1
1	1	1	1	0	1

1	2				
2					

미로 탐색

<https://www.acmicpc.net/problem/2178>

- (1, 1) 에서 (N, M)으로 가는 가장 빠른 길을 구하는 문제

1	1	0	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1
1	1	1	1	0	1

1	2				
2	3				
3					

미로 탐색

<https://www.acmicpc.net/problem/2178>

- (1, 1) 에서 (N, M)으로 가는 가장 빠른 길을 구하는 문제

1	1	0	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1
1	1	1	1	0	1

1	2				
2	3				
3	4				
4					

미로 탐색

<https://www.acmicpc.net/problem/2178>

- (1, 1) 에서 (N, M)으로 가는 가장 빠른 길을 구하는 문제

1	1	0	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1
1	1	1	1	0	1

1	2				
2	3				
3	4	5			
4	5				

미로 탐색

<https://www.acmicpc.net/problem/2178>

- (1, 1) 에서 (N, M)으로 가는 가장 빠른 길을 구하는 문제

1	1	0	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1
1	1	1	1	0	1

1	2				
2	3				
3	4	5	6		
4	5	6			

미로 탐색

<https://www.acmicpc.net/problem/2178>

- (1, 1) 에서 (N, M)으로 가는 가장 빠른 길을 구하는 문제

1	1	0	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1
1	1	1	1	0	1

1	2				
2	3		7		
3	4	5	6	7	
4	5	6	7		

미로 탐색

<https://www.acmicpc.net/problem/2178>

- (1, 1) 에서 (N, M)으로 가는 가장 빠른 길을 구하는 문제

1	1	0	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1
1	1	1	1	0	1

1	2		8		
2	3		7	8	
3	4	5	6	7	8
4	5	6	7		

미로 탐색

<https://www.acmicpc.net/problem/2178>

$N \times M$
3

- (1, 1) 에서 (N, M)으로 가는 가장 빠른 길을 구하는 문제

1	1	0	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1
1	1	1	1	0	1

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

미로 탐색

<https://www.acmicpc.net/problem/2178>

- 소스: <http://codeplus.codes/baaf6b955c45466cacdda1f2e01cf08b>

토마토

<https://www.acmicpc.net/problem/7576>

- 하루가 지나면, 익은 토마토의 인접한 곳에 있는 익지 않은 토마토들이 익게 된다
- 인접한 곳: 앞, 뒤, 왼쪽, 오른쪽
- 토마토가 저절로 익는 경우는 없다
- 상자안의 익은 토마토와 익지 않은 토마토가 주어졌을 때, 며칠이 지나면 토마토가 모두 익는지 구하는 문제

토마토

103

<https://www.acmicpc.net/problem/7576>

- BFS 탐색을 하면서, 거리를 재는 방식으로 진행한다

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	1

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

토마토

<https://www.acmicpc.net/problem/7576>

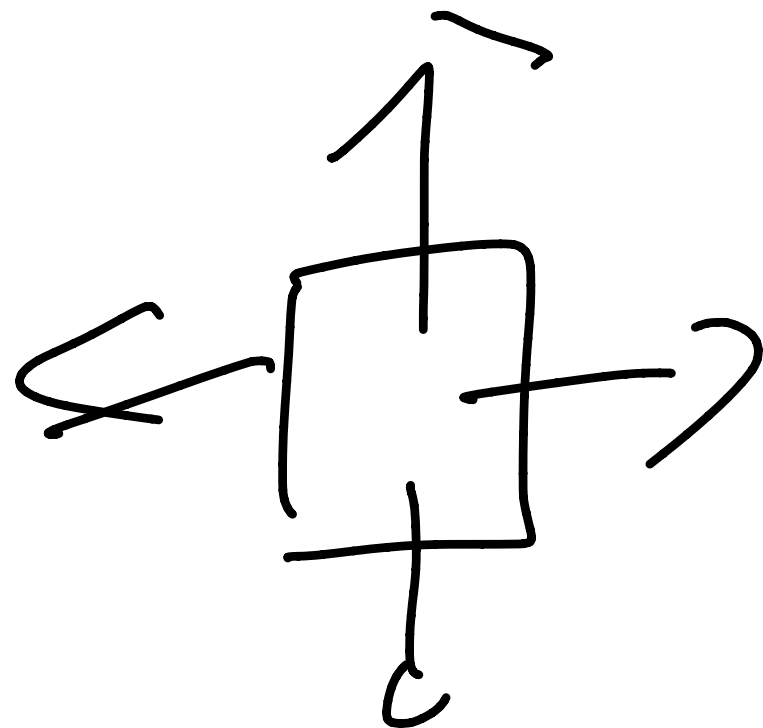
- 소스: <http://codeplus.codes/ad34bc6da295433f9b11bba2fd593c7e>

나이트의 이동

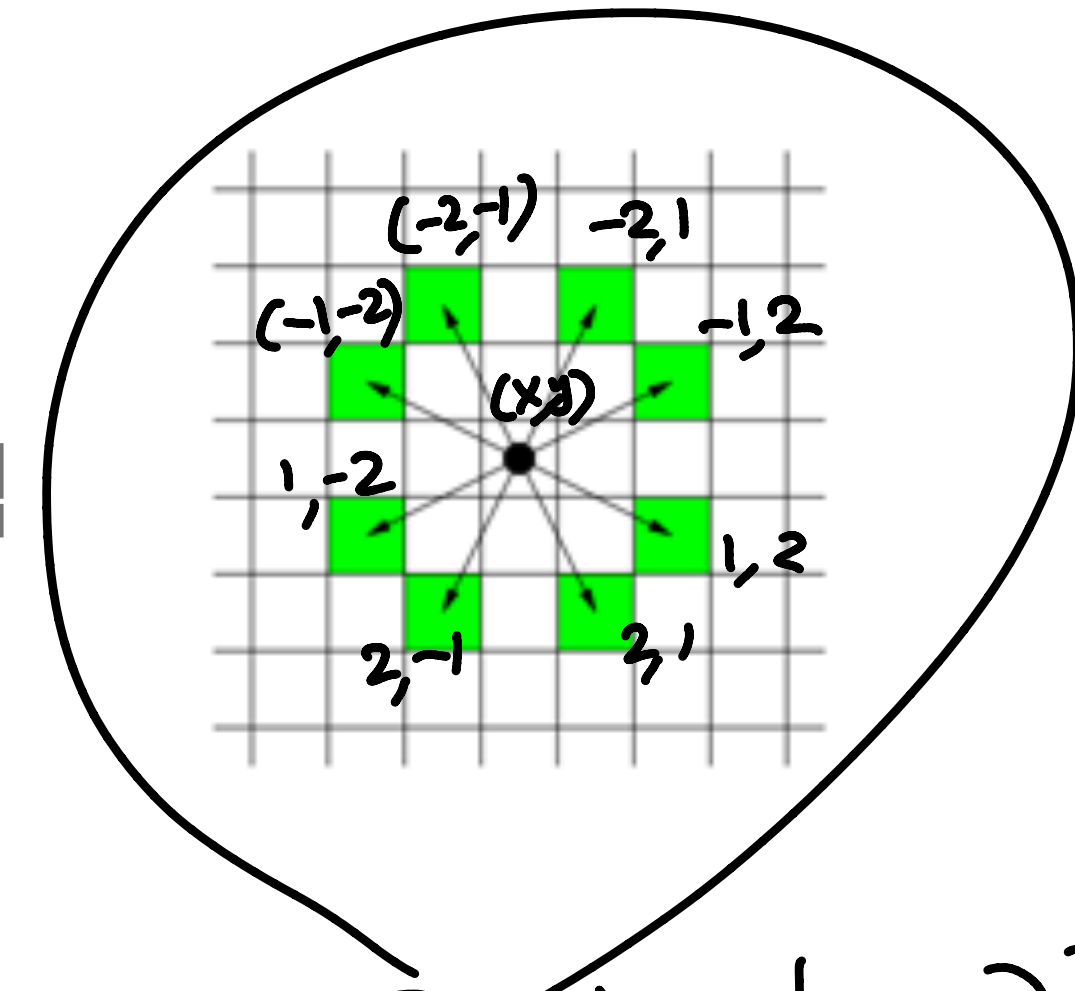
<https://www.acmicpc.net/problem/7562>

105

- 체스판 위에 한 나이트가 있을 때, 나이트가 이동하려고 하는 칸이 주어짐
- 이 때, 최소 몇 번만에 이동할 수 있는지 구하는 문제



8칸



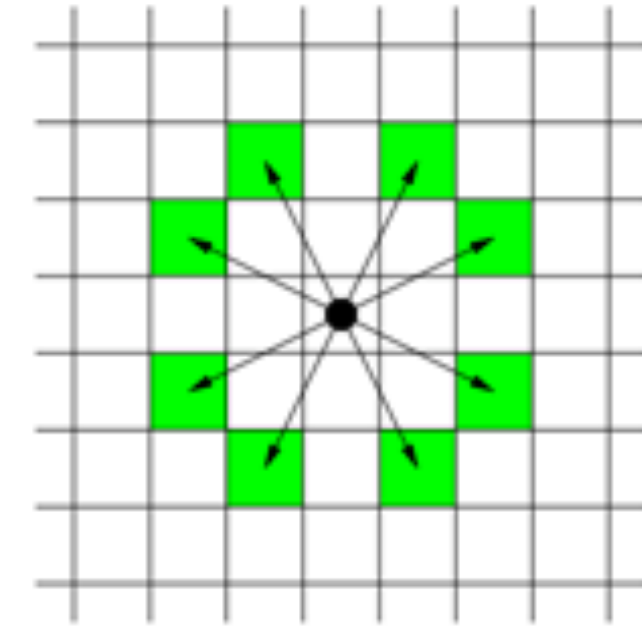
$$dx = [-2, -1, 1, 2, 2, 1, -1, -2]$$

$$dy = [1, 2, 2, 1, -1, -2, -2, -1]$$

나이트의 이동

<https://www.acmicpc.net/problem/7562>

- 체스판의 한 변의 길이 L ($4 \leq L \leq 300$)
- 체스판 칸의 개수 = $L^2 = 90,000$
- 개수가 많지 않기 때문에, 모든 칸을 다 방문해보면 됨



나이트의 이동

107

<https://www.acmicpc.net/problem/7562>

- 소스: <http://codeplus.codes/22e081fad6ab4a088f6c4f05851aeeb1>

끝

코드 플러스

<https://code.plus>

- 슬라이드에 포함된 소스 코드를 보려면 "정보 수정 > 백준 온라인 저지 연동"을 통해 연동한 다음, "백준 온라인 저지"에 로그인해야 합니다.
- 강의 내용에 대한 질문은 코드 플러스의 "질문 게시판"에서 할 수 있습니다.
- 문제와 소스 코드는 슬라이드에 첨부된 링크를 통해서 볼 수 있으며, "백준 온라인 저지"에서 서비스됩니다.
- 슬라이드와 동영상 강의는 코드 플러스 사이트를 통해서만 볼 수 있으며, 동영상 강의의 녹화와 다운로드, 배포와 유통은 저작권법에 의해서 금지되어 있습니다.
- 다른 경로로 이 슬라이드나 동영상 강의를 본 경우에는 codeplus@startlink.io 로 이메일 보내주세요.
- 강의 내용, 동영상 강의, 슬라이드, 첨부되어 있는 소스 코드의 저작권은 스타트링크와 최백준에게 있습니다.