그래프 1

최백준 choi@startlink.io

그래프

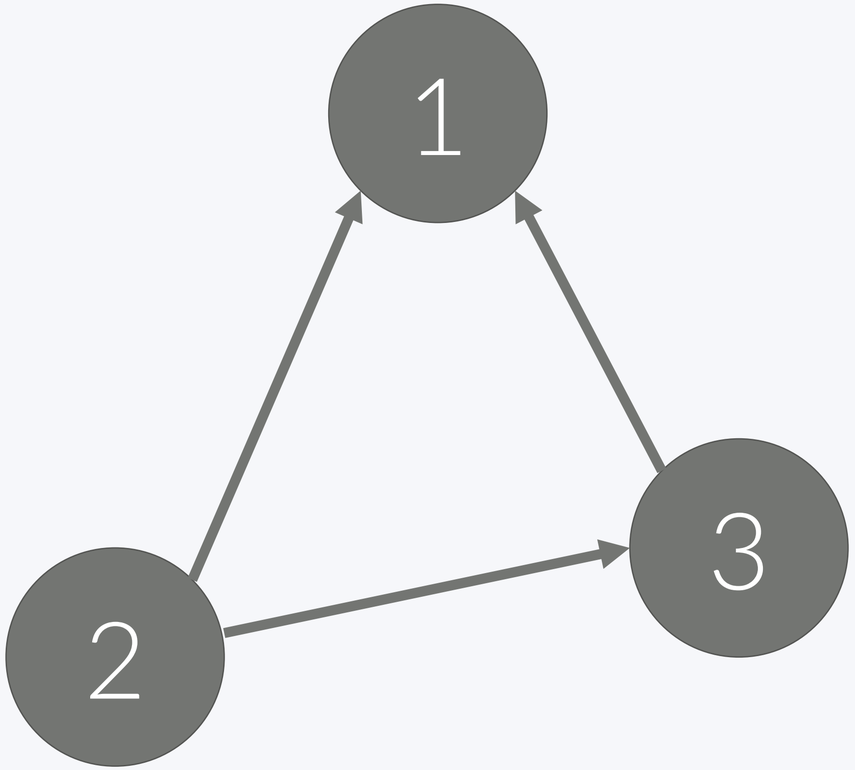
그래프

Graph

- 자료구조의 일종

- 정점 (Node, Vertex)
- 간선 (Edge): 정점간의 관계를 나타낸다.

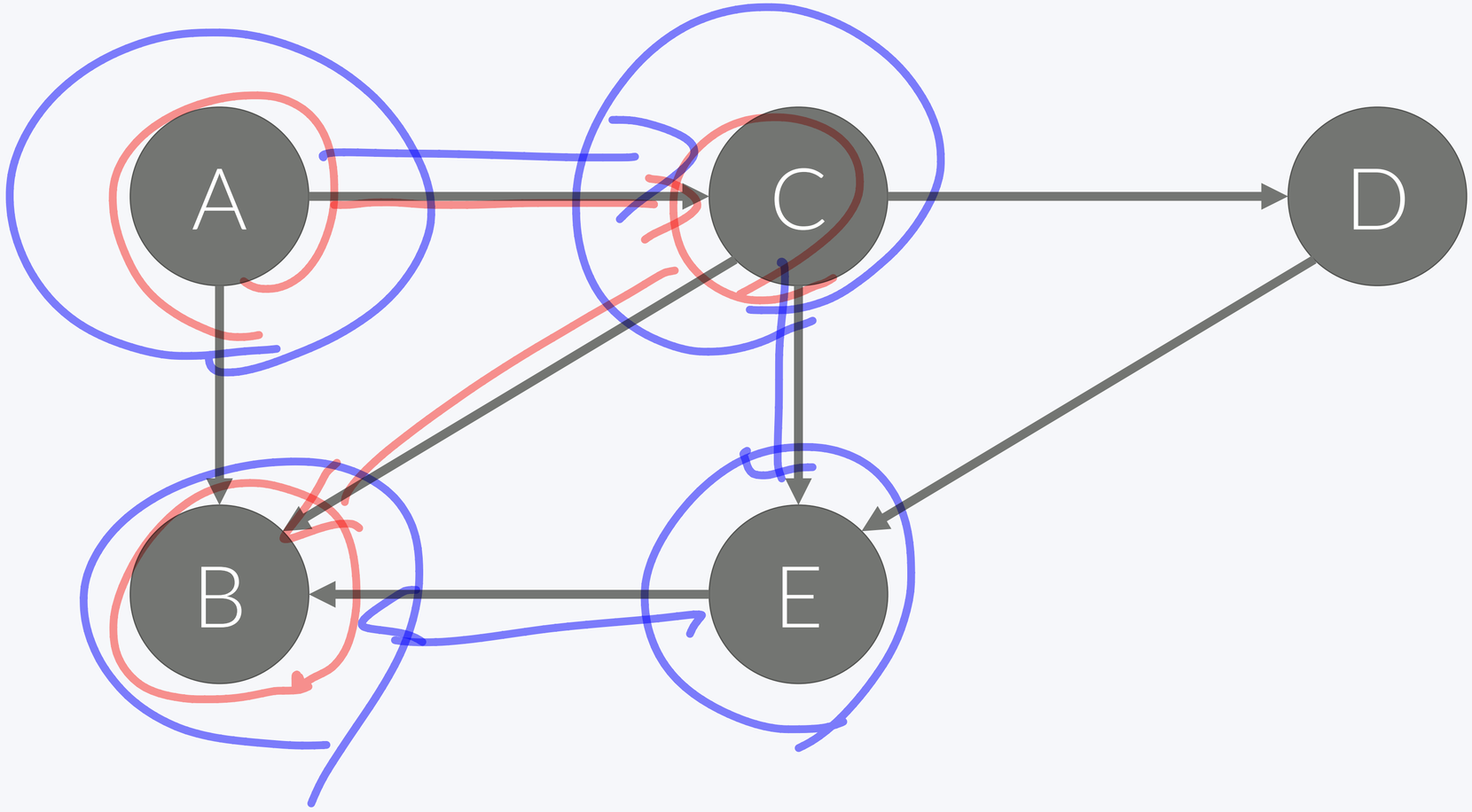
- G = (V, E)로 나타낸다.



경로

Path

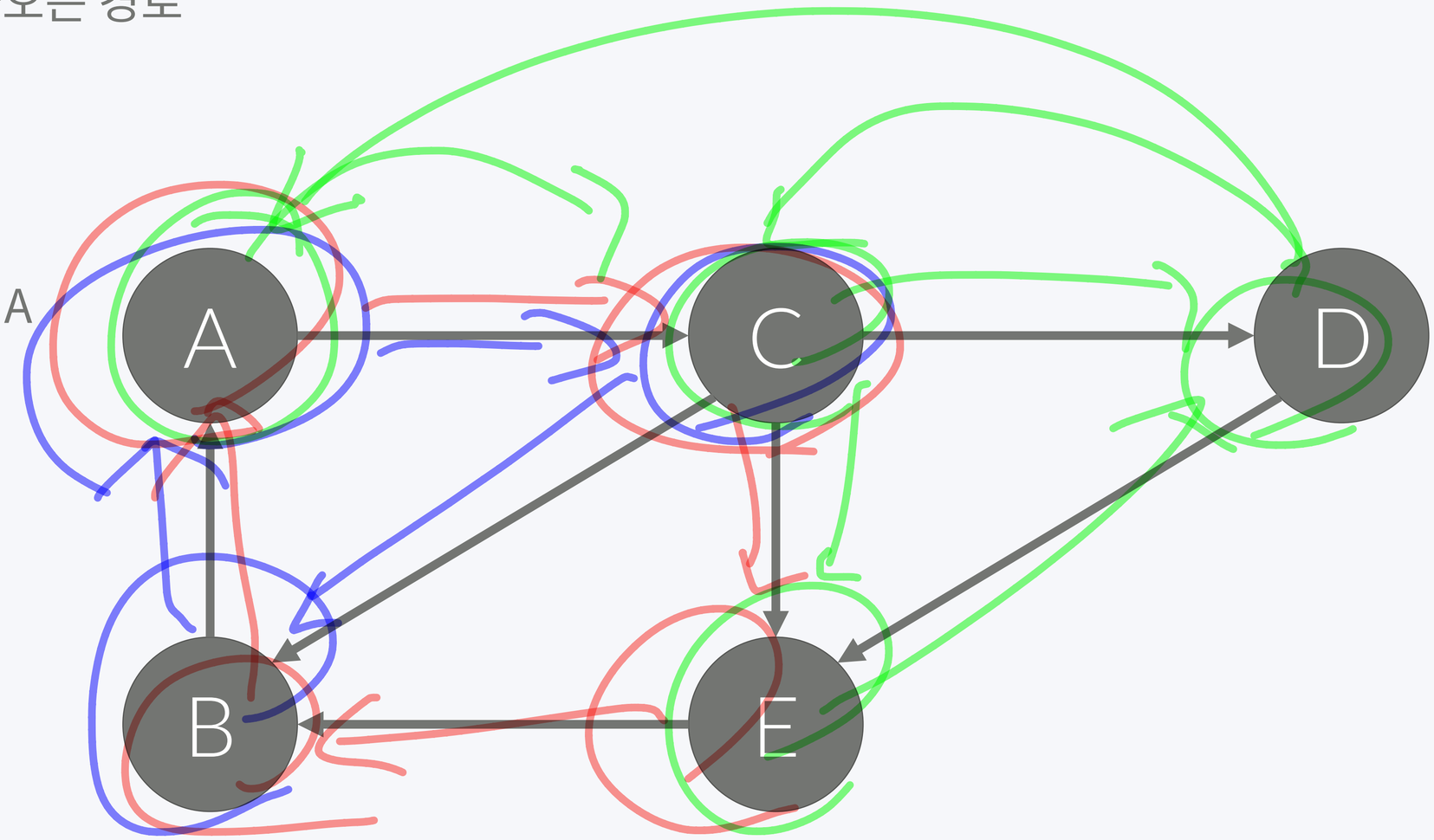
- 정점 A에서 B로 가는 경로
- A → C → D → E → B
- A → B
- A → C → B
- A → C → E → B



사이클

Path

- 정점 A에서 다시 A로 돌아오는 경로
- A → C → B → A
- A → C → E → B → A
- A → C → D → E → B → A



단순 경로와 단순 사이클

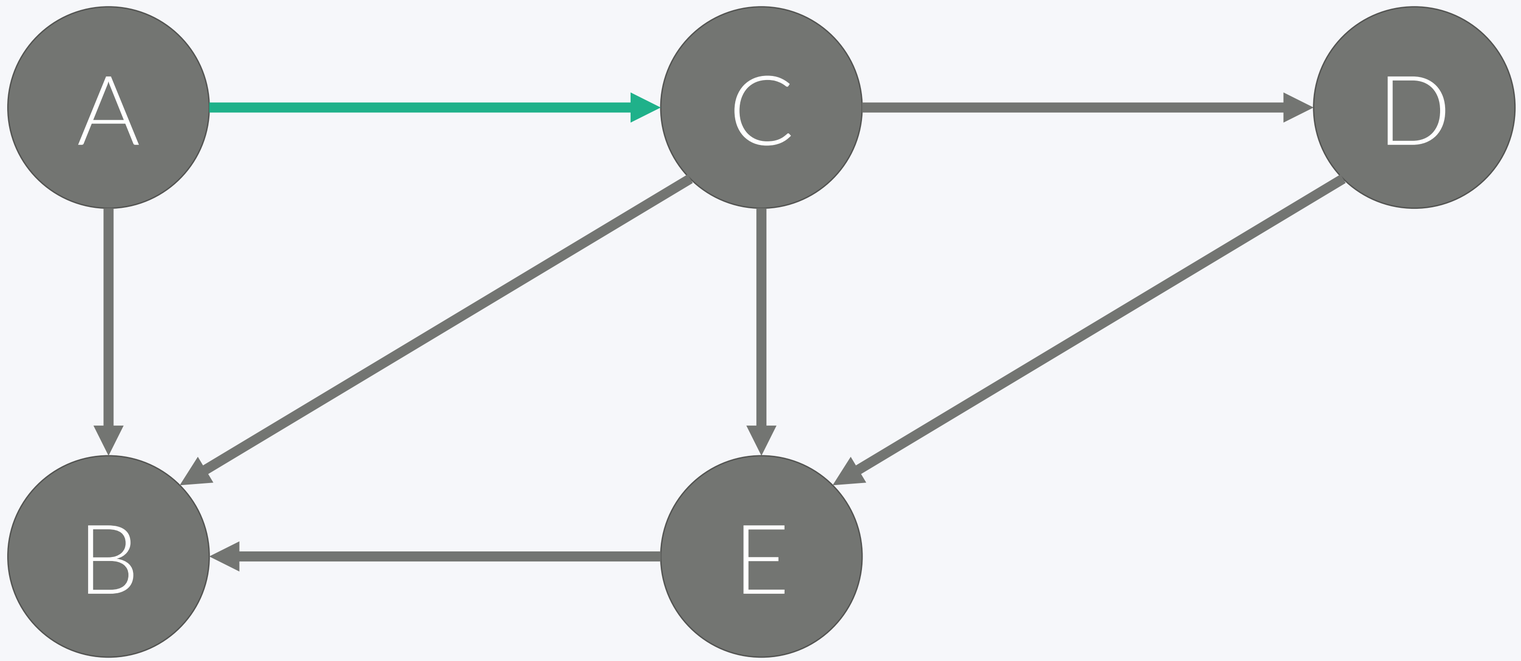
Simple Path and Simple Cycle

- 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

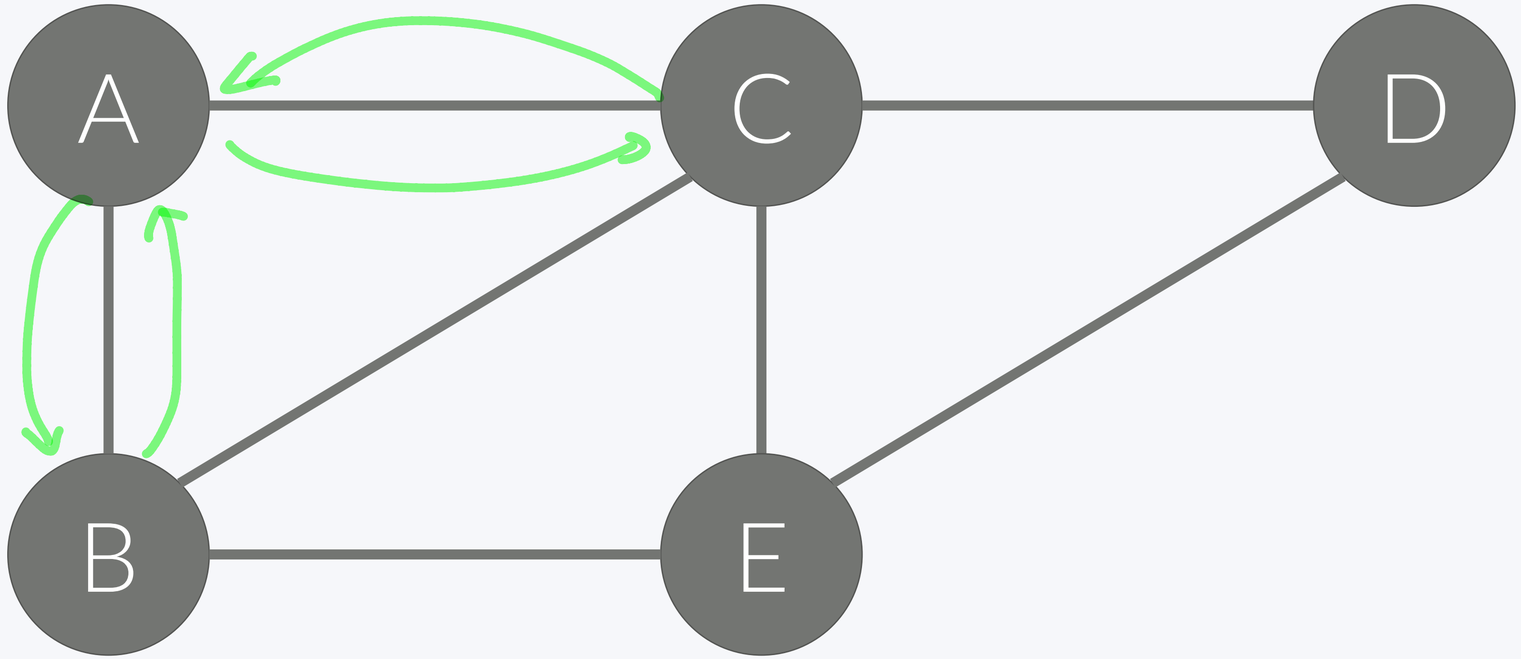
- A → C 와 같이 간선에 방향이 있다.
- A → C는 있지만, C → A는 없다.



방향 없는 그래프

Undirected Graph

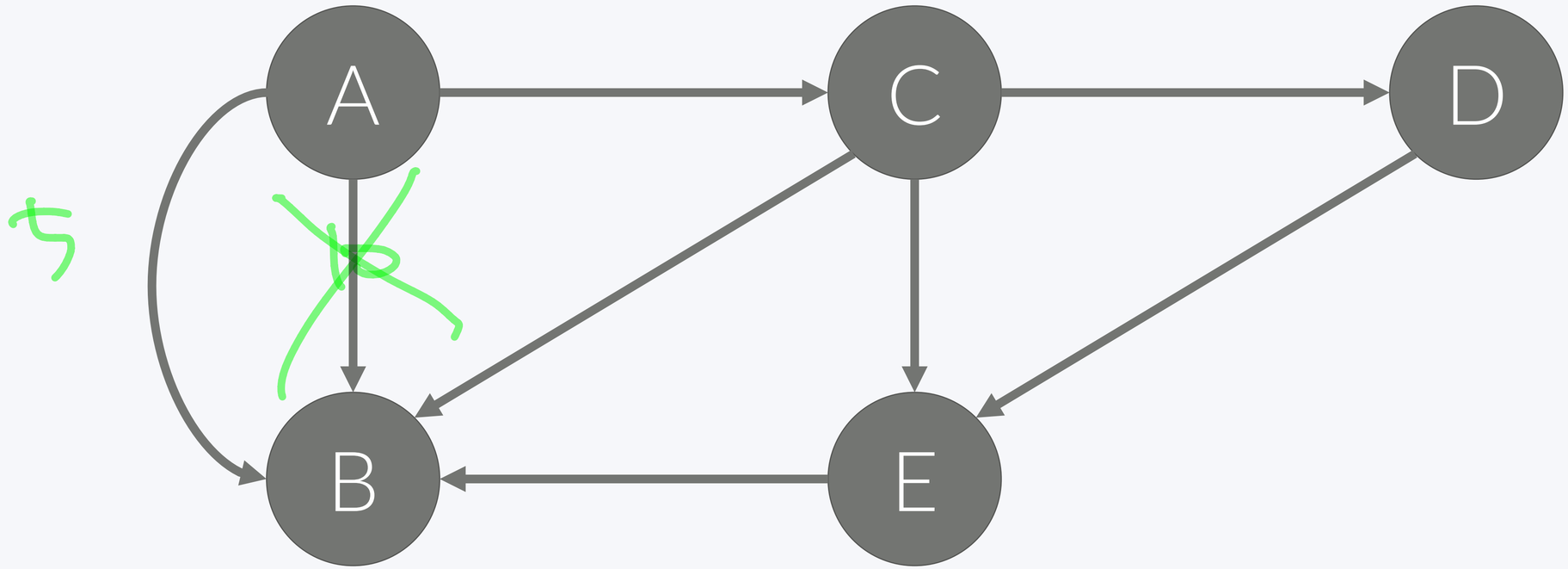
- A – C 와 같이 간선에 방향이 없다.
- A – C는 A → C와 C → A를 나타낸다.
- 양방향 그래프 (Bidirection Graph) 라고도 한다.



간선 여러개

Multiple Edge

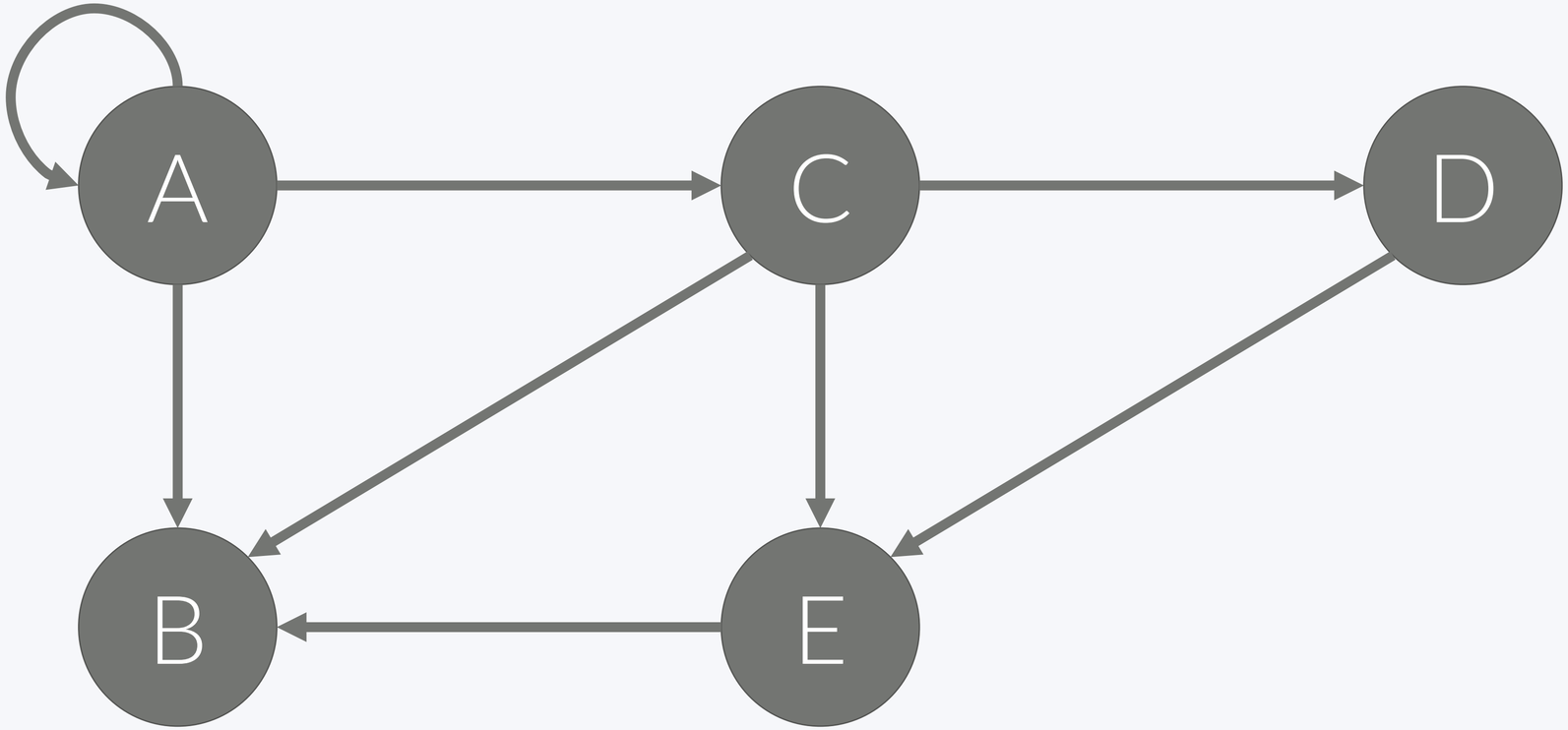
- 두 정점 사이에 간선이 여러 개일 수도 있다.
- 아래 그림의 A-B는 연결하는 간선이 2개이다.
- 두 간선은 서로 다른 간선이다



루프

Loop

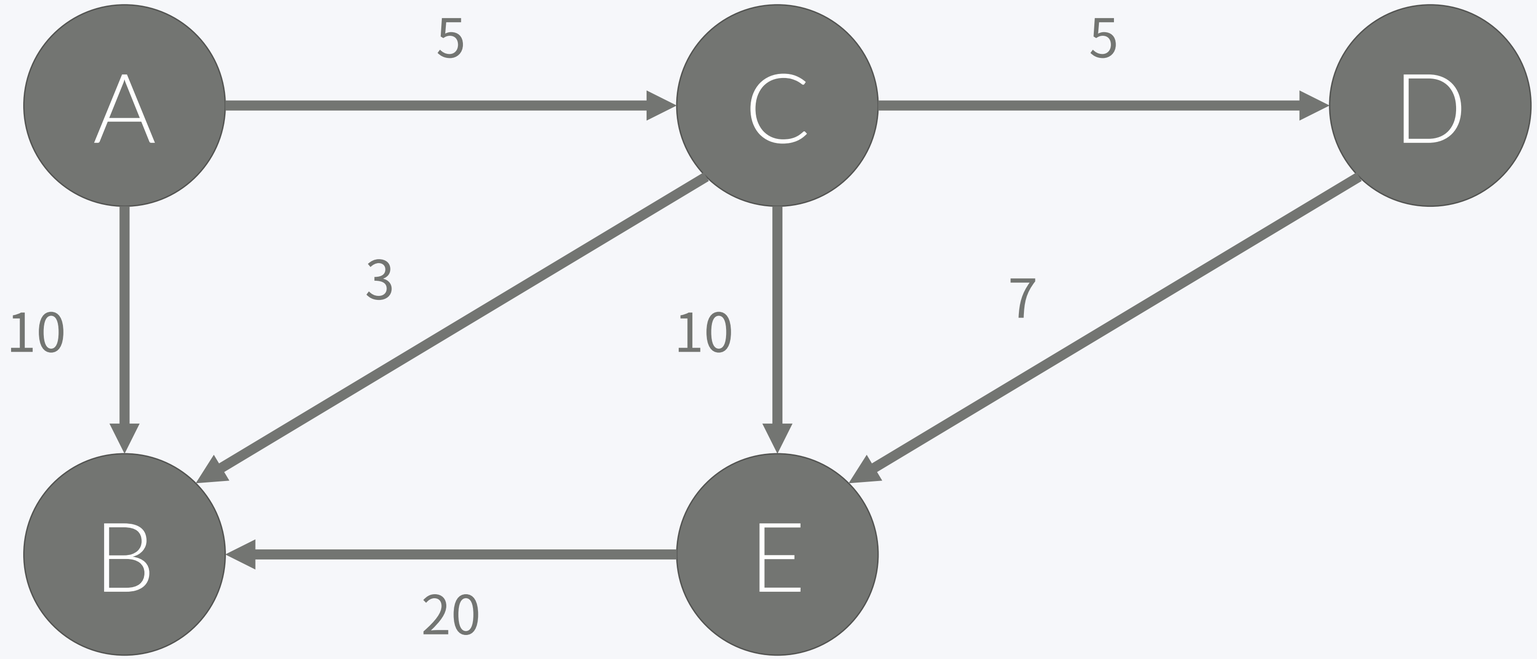
- 간선의 양 끝 점이 같은 경우가 있다.
- A → A



가중치

Weight

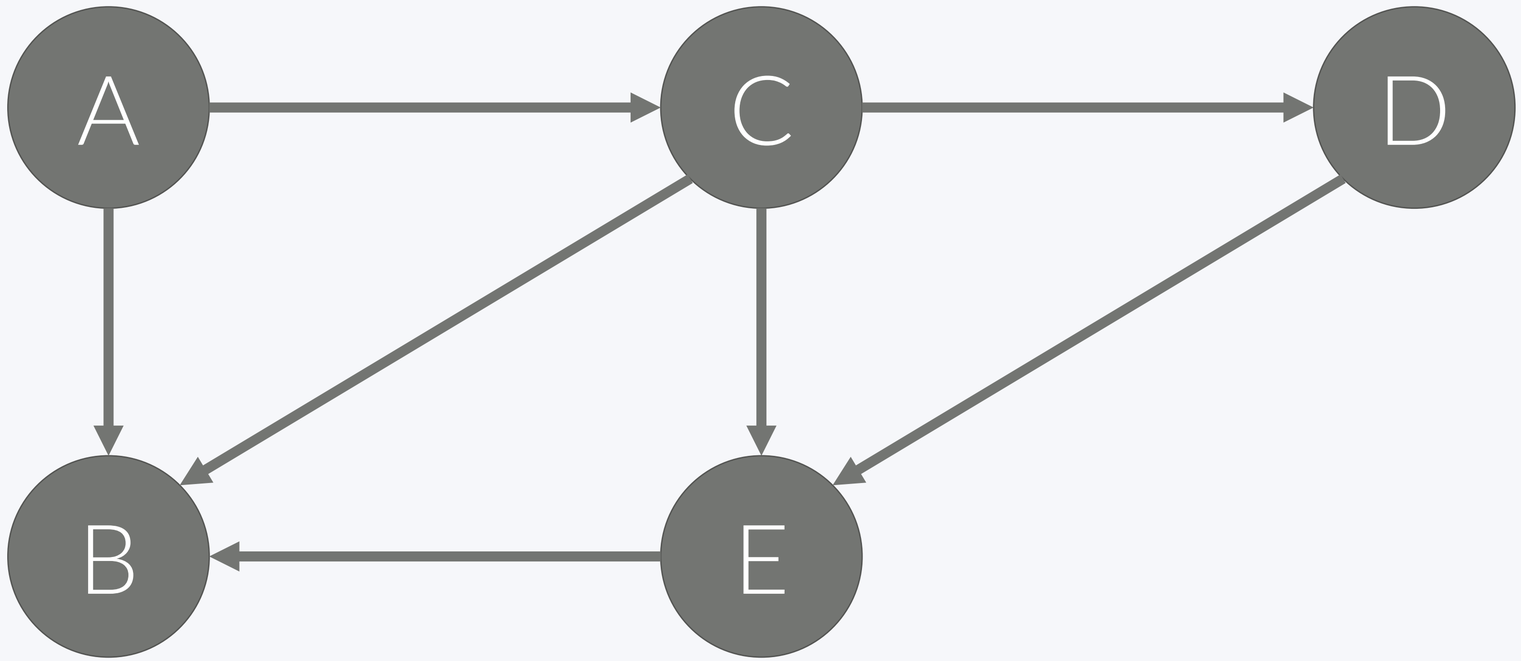
- 간선에 가중치가 있는 경우에는
- A에서 B로 이동하는 거리, 이동하는데 필요한 시간, 이동하는데 필요한 비용 등등등…



가중치

Weight

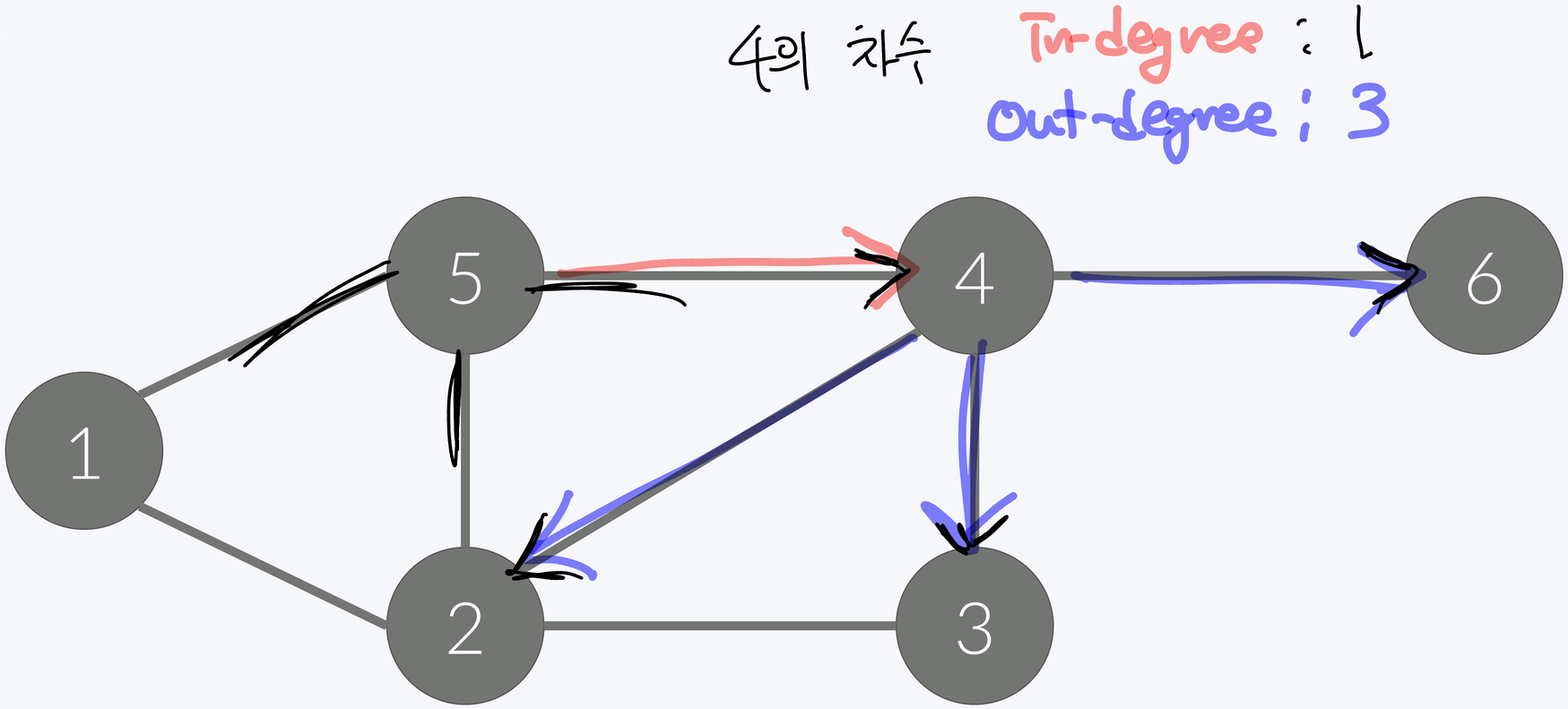
- 가중치가 없는 경우에는 1이라고 생각하면 된다



차수

Degree

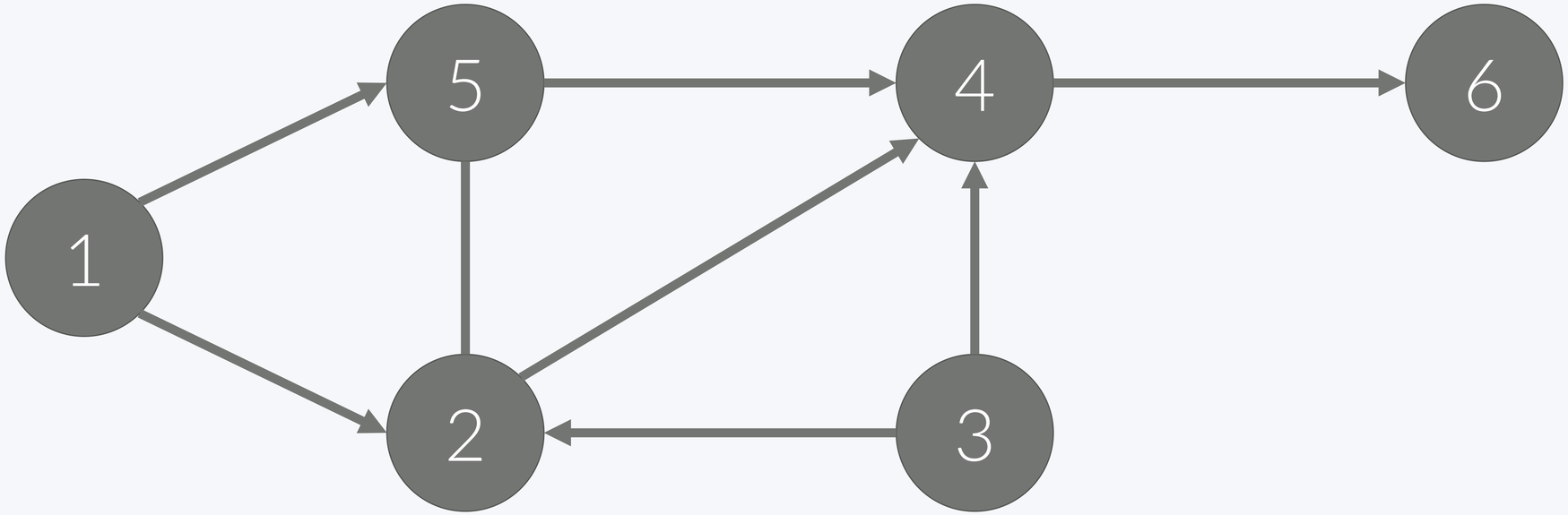
- 정점과 연결되어 있는 간선의 개수
- 5의 차수: 3
- 4이 차수: 4



차수

Degree

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

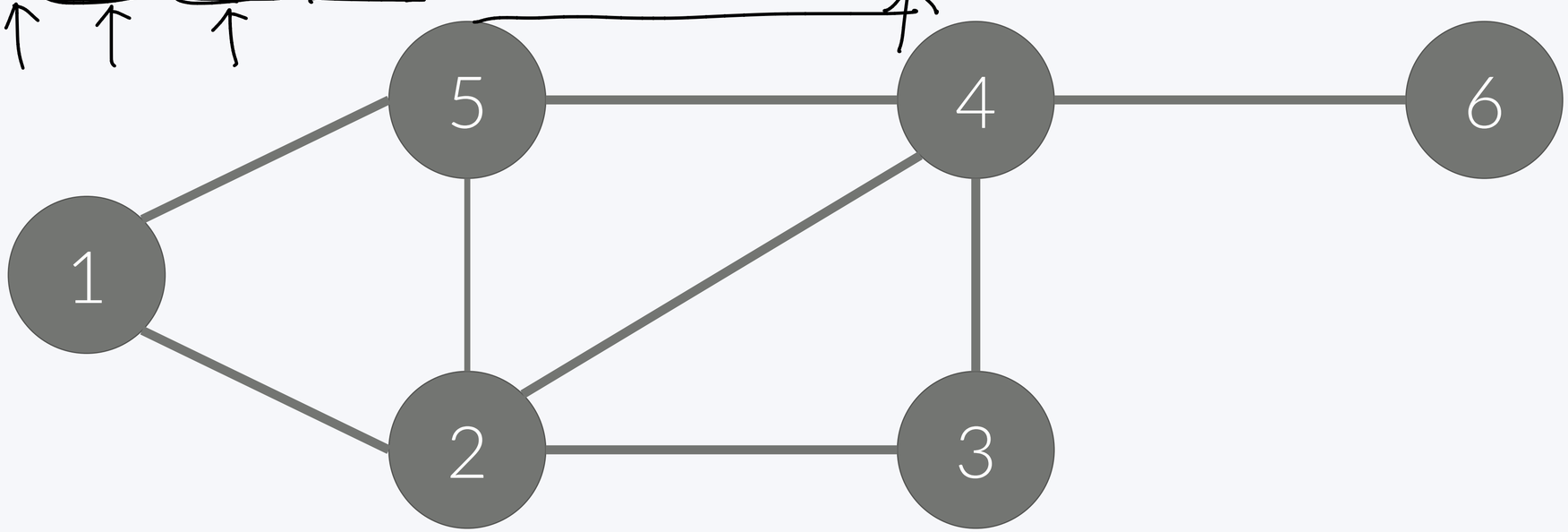


그래프의 표현

그래프의 표현

Representation of Graph

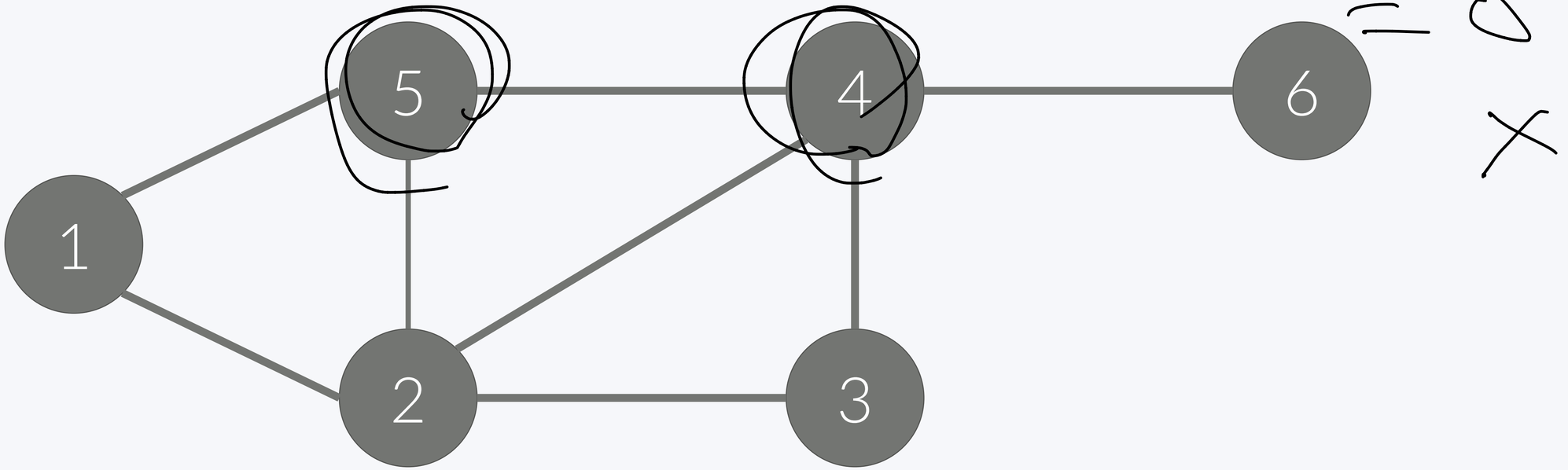
- 아래와 같은 그래프는 정점이 6개, 간선이 8개 있다.
- 간선에 방향이 없기 때문에, 방향이 없는 그래프이다.
- 정점: {1, 2, 3, 4, 5, 6}
- 간선: {(1, 2), (1, 5), (2, 5), (2, 3), (3, 4), (2, 4), (4, 5), (4, 6)}



인접 행렬

Adjacency-matrix

- 정점의 개수를 V이라고 했을 때
- V×V 크기의 이차원 배열을 이용한다
- A[i][j] = 1 (i -> j 간선이 있을 때), 0 (없을 때)



인접 행렬

Adjacency-matrix

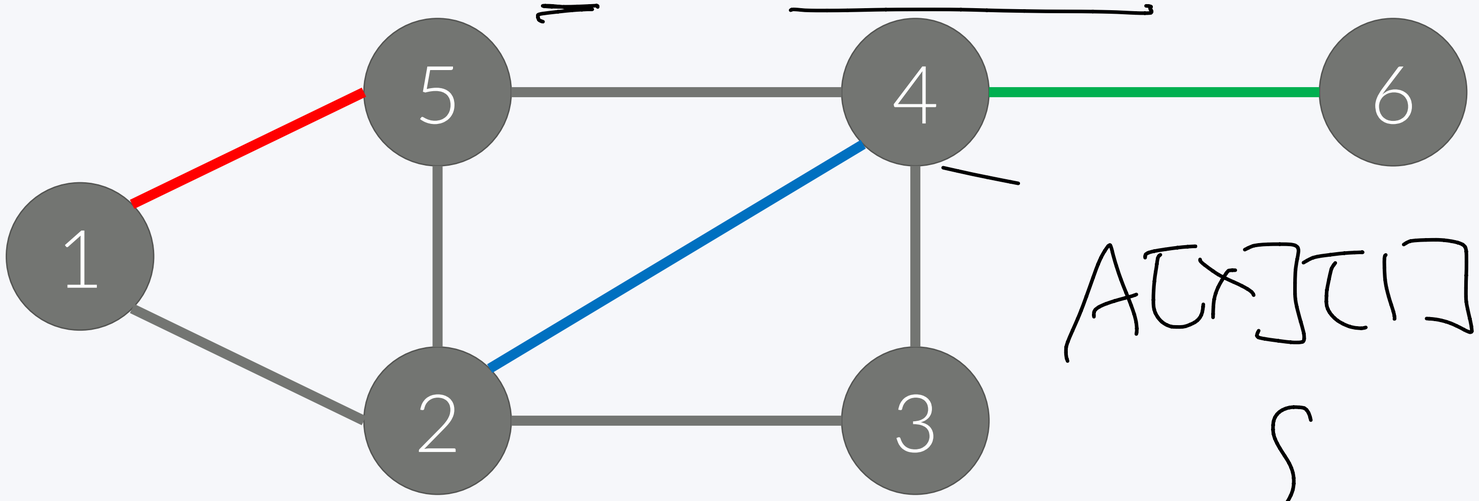
정점 V. 간선 E

공간: V^2

두 정점 연결? O(1)

X와 연결된 모든 정점 O(V)

	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



A[X][1]

⋮

A[X][V]

인접 행렬

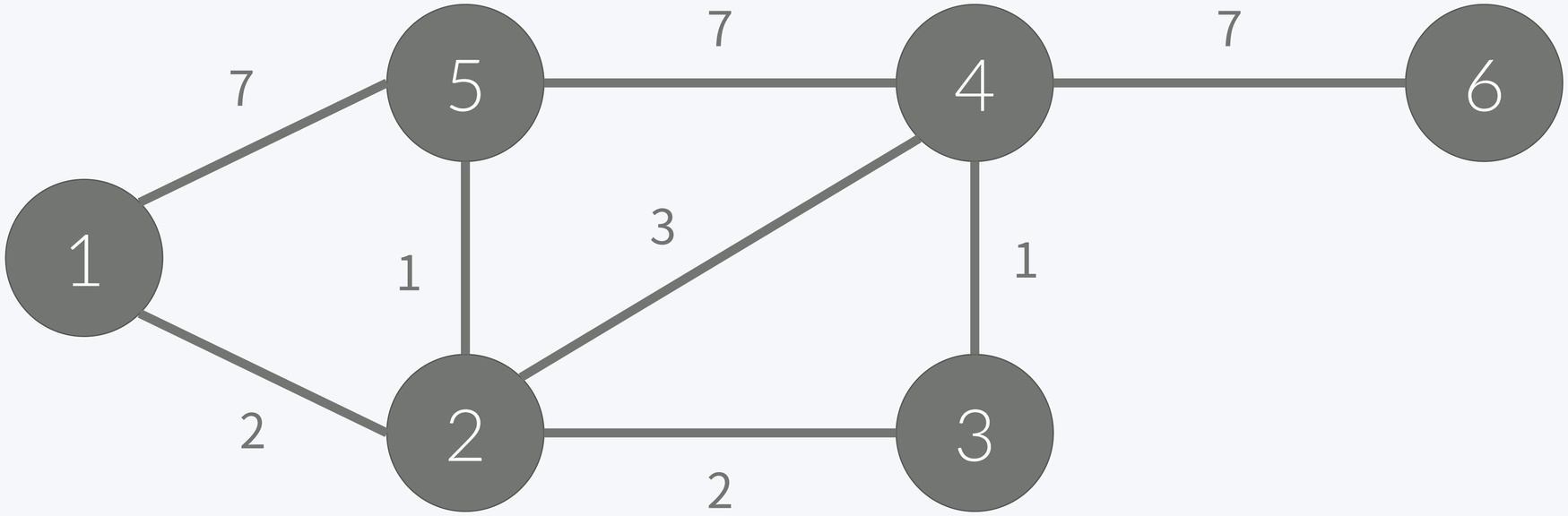
Adjacency-matrix


```cpp
#include <cstdio>
#include <vector>
int a[10][10];
int main() {
    int n, m;
    scanf("%d %d",&n,&m);
    for (int i=0; i<m; i++) {
        int u,v;
        scanf("%d %d",&u,&v);
        a[u][v] = a[v][u] = 1;
    }
}
```

인접 행렬

Adjacency-matrix

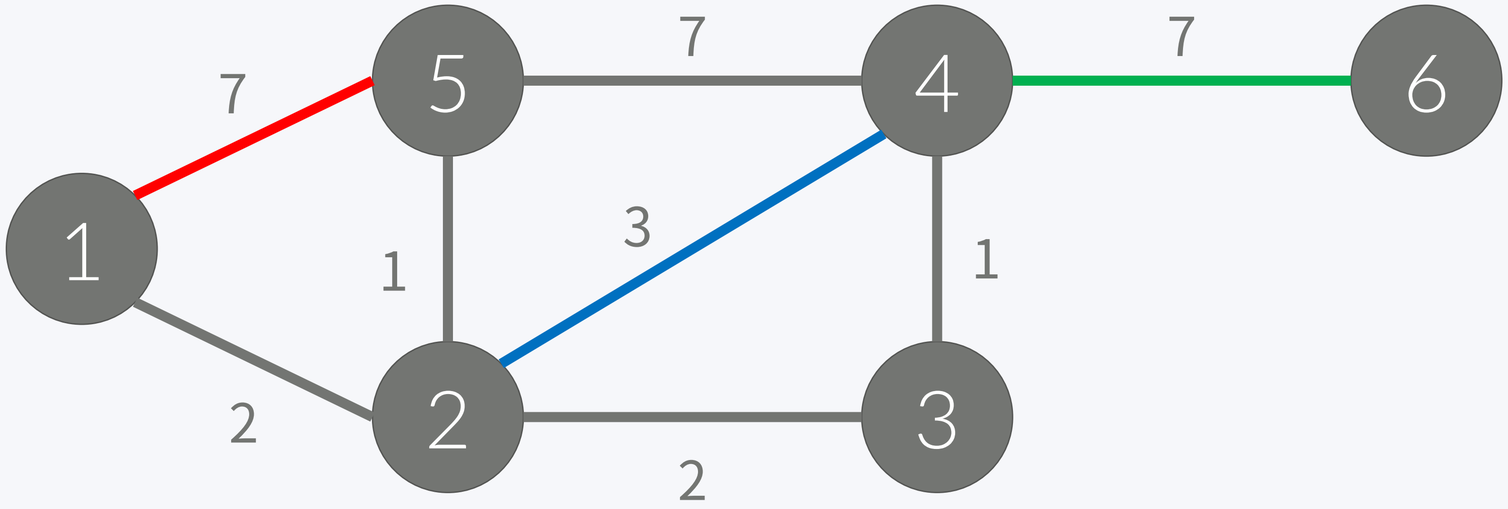
- 정점의 개수를 N이라고 했을 때
- N×N 크기의 이차원 배열을 이용한다
- A[i][j] = w (i -> j 간선이 있을 때, 그 가중치), 0 (없을 때)



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



인접 행렬

Adjacency-matrix

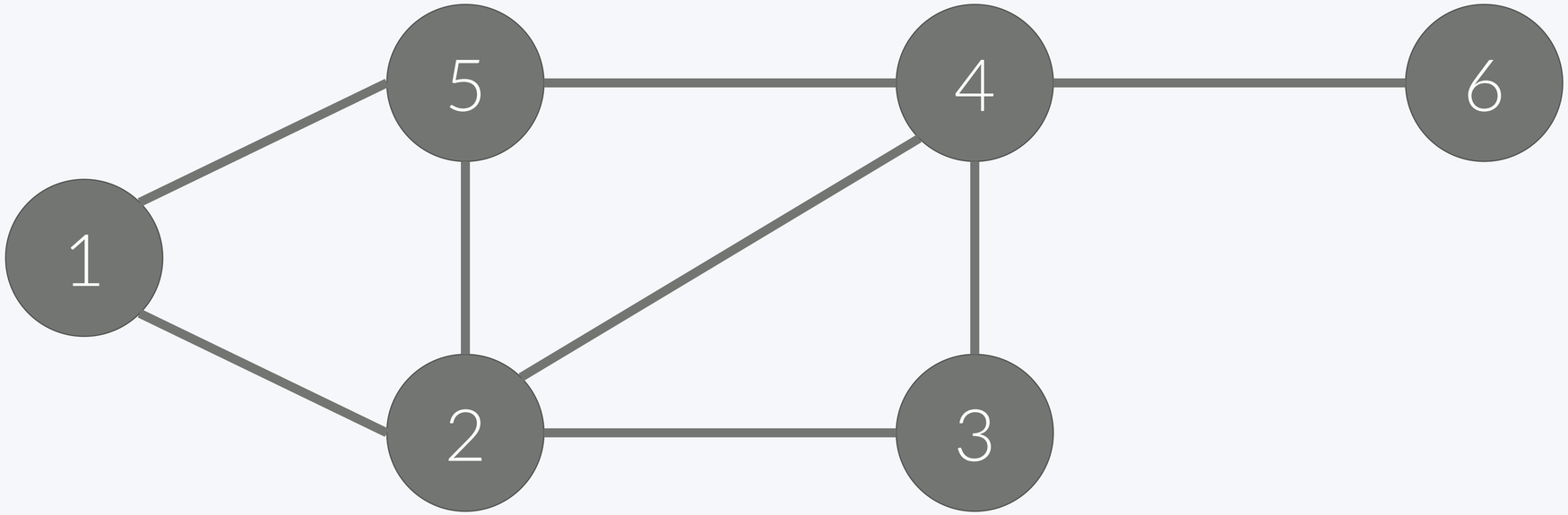

```
#include <cstdio>
#include <vector>
int a[10][10];
int main() {
    int n, m;
    scanf("%d %d",&n,&m);
    for (int i=0; i<m; i++) {
        int u,v,w;
        scanf("%d %d %d",&u,&v,&w);
        a[u][v] = a[v][u] = w;
    }
}
```

인접 리스트

Adjacency-list

A[i] = i와 연결된 모든 정점 저장

- 리스트를 이용해서 구현한다.
- A[i] = i와 연결된 정점을 리스트로 포함하고 있음



인접 리스트

Adjacency-list

A[1] 2 5

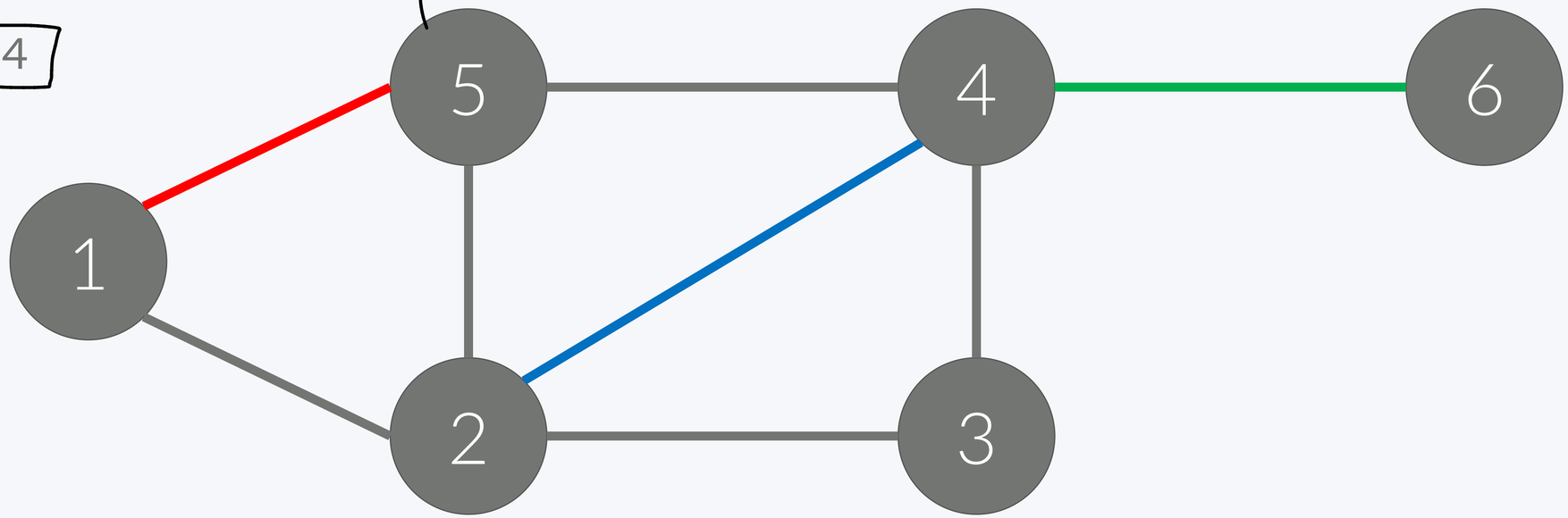
A[2] 1 3 4 5

A[3] 2 4

A[4] 3 5 2 6

A[5] 1 2 4

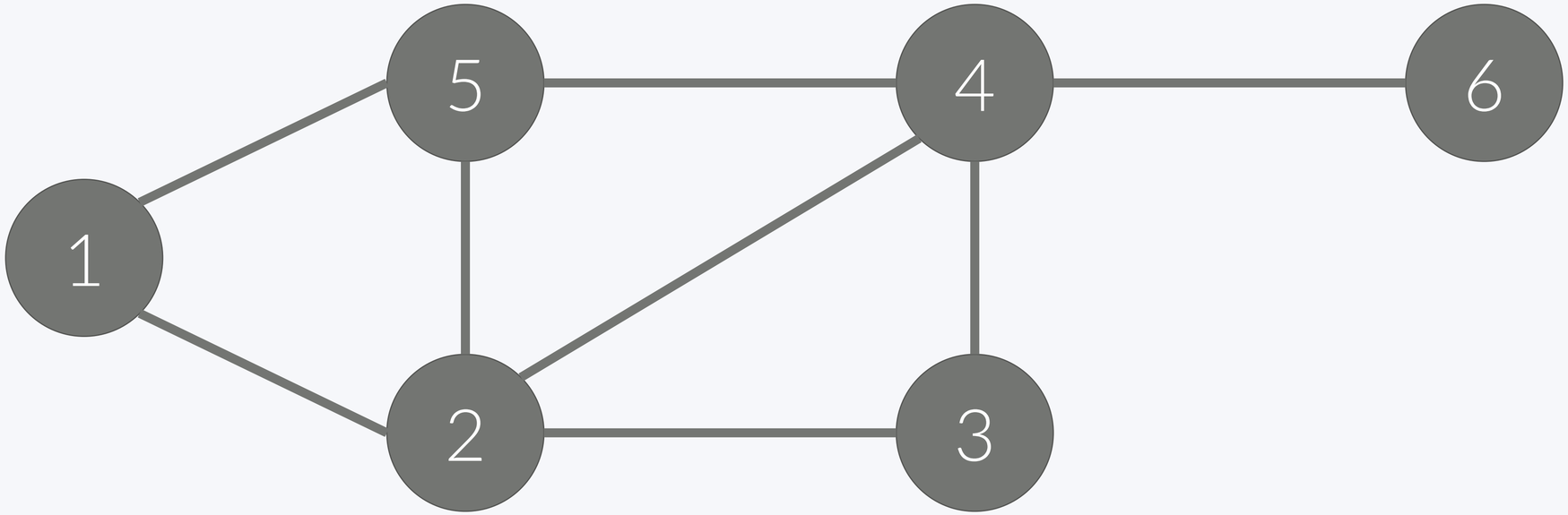
A[6] 4



인접 리스트

Adjacency-list

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



인접 리스트

Adjacency-list

```
#include <cstdio>
#include <vector>
using namespace std;
vector<int> a[10];
int main() {
    int n, m;
    scanf("%d %d",&n,&m);
    for (int i=0; i<m; i++) {
        int u,v;
        scanf("%d %d",&u,&v);
        a[u].push_back(v); a[v].push_back(u);
    }
}
```

인접 리스트

Adjacency-list


```
#include <cstdio>
#include <vector>
using namespace std;
int main() {
    int n, m;
    scanf("%d %d",&n,&m);
    vector<vector<int>> a(n+1);
    for (int i=0; i<m; i++) {
        int u,v;
        scanf("%d %d",&u,&v);
        a[u].push_back(v); a[v].push_back(u);
    }
}
```

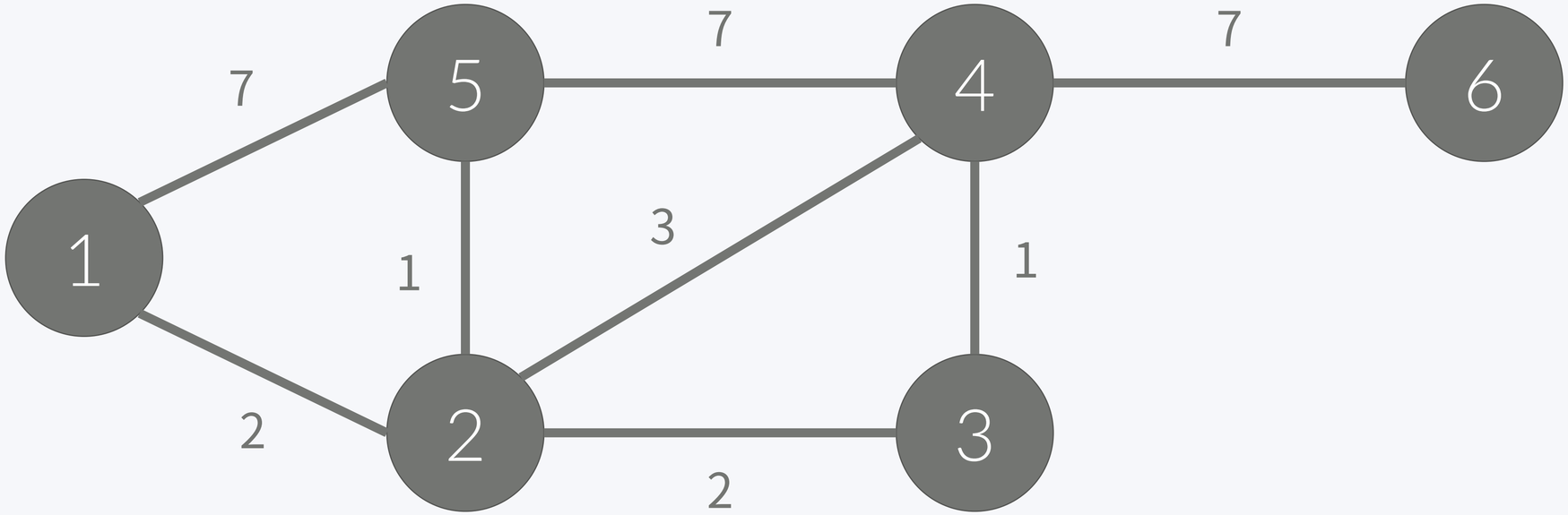
인접 리스트

Adjacency-list

간선: u → v (w)

(|u|(v, w))

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

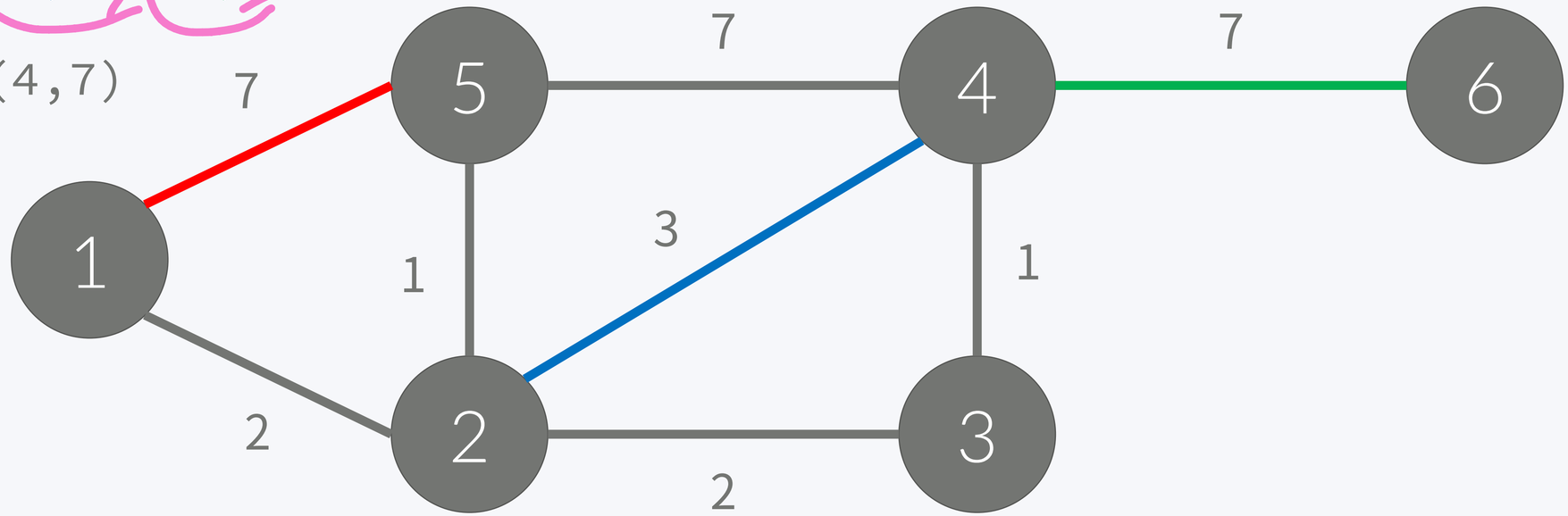


인접 리스트

Adjacency-list

V+E

```
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)
```



인접 리스트

Adjacency-list

```
#include <cstdio>
#include <vector>
using namespace std;
vector<pair<int,int>> a[10];
int main() {
    int n,m;
    scanf("%d %d",&n,&m);
    for (int i=0; i<m; i++) {
        int u,v,w;
        scanf("%d %d %d",&u,&v,&w);
        a[u].push_back(make_pair(v,w)); a[v].push_back(make_pair(u,w));
    }
}
```

공간 복잡도

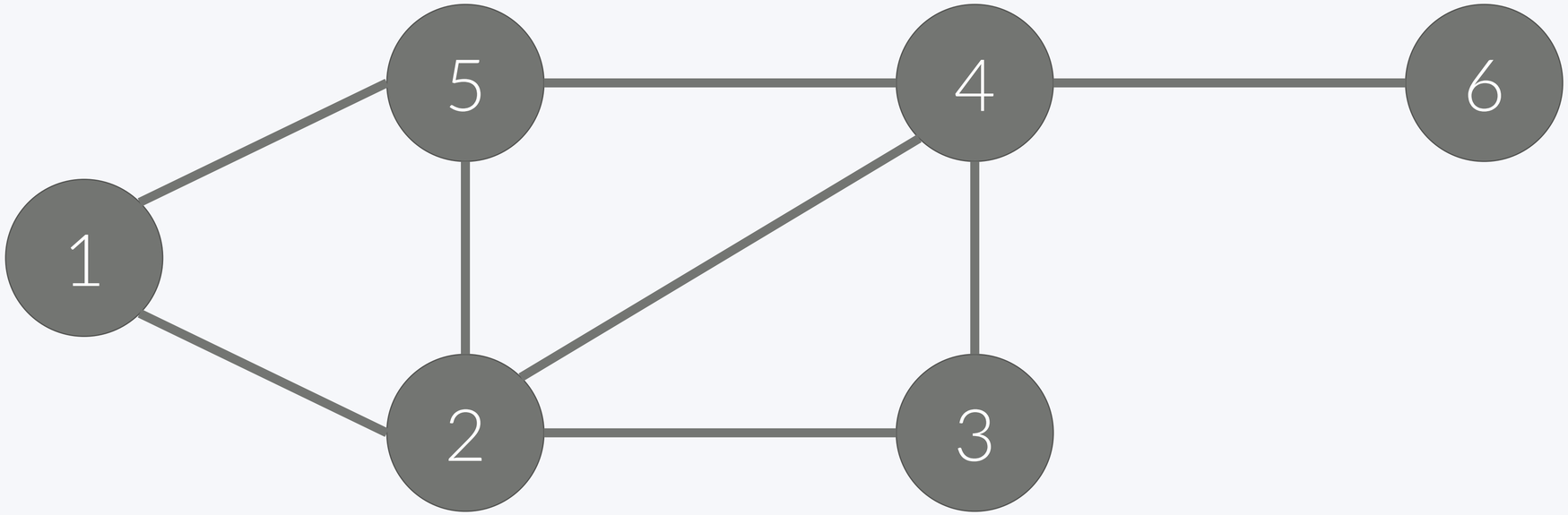
Space Complexity

- 인접 행렬: O(V^2)
- 인접 리스트: O(E)

간선 리스트

Edge-list

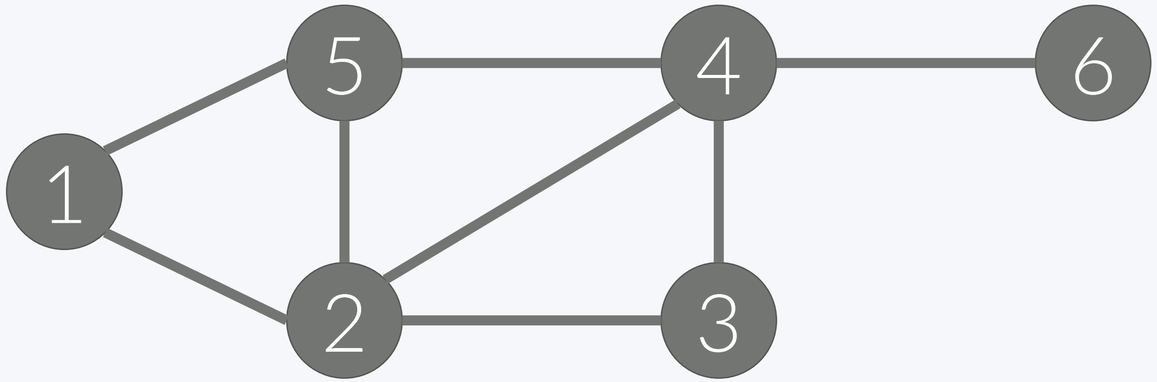
- 배열을 이용해서 구현한다.
- 간선을 모두 저장하고 있다.



간선 리스트

Edge List

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

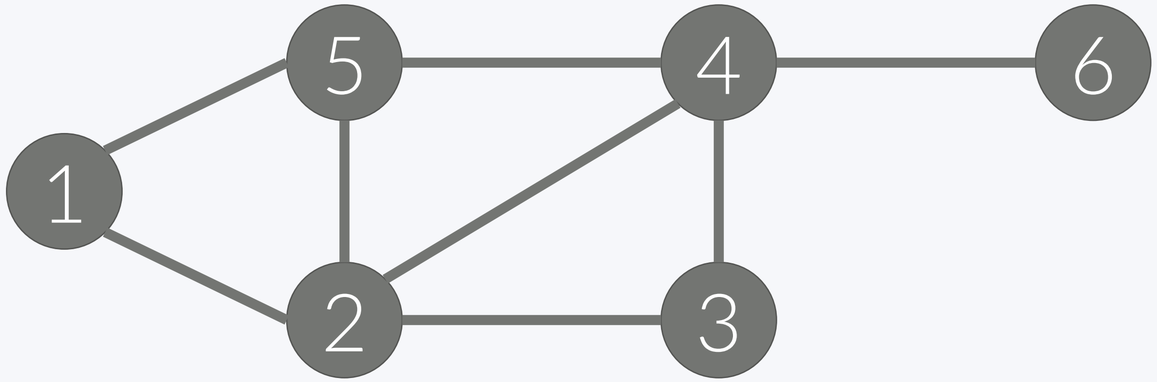



```
E[0] = 1 2
E[1] = 1 5
E[2] = 2 3
E[3] = 2 4
E[4] = 2 5
E[5] = 5 4
E[6] = 4 3
E[7] = 4 6
E[8] = 2 1
E[9] = 5 1
E[10] = 3 2
E[11] = 4 2
E[12] = 5 2
E[13] = 4 5
E[14] = 3 4
E[15] = 6 4
```

간선 리스트

Edge List

- 각 간선의 앞 정점을 기준으로 개수를 센다.



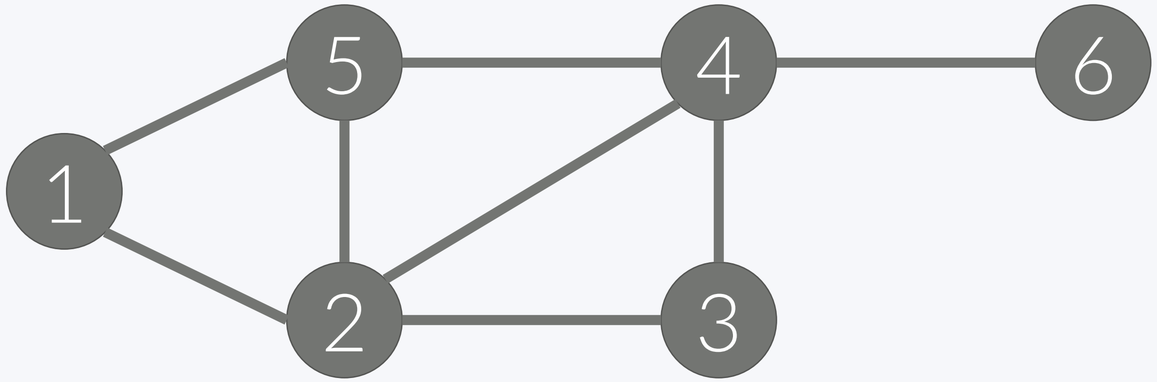
i	0	1	2	3	4	5	6
cnt[i]	0	2	4	2	4	3	1

```
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;
}
```



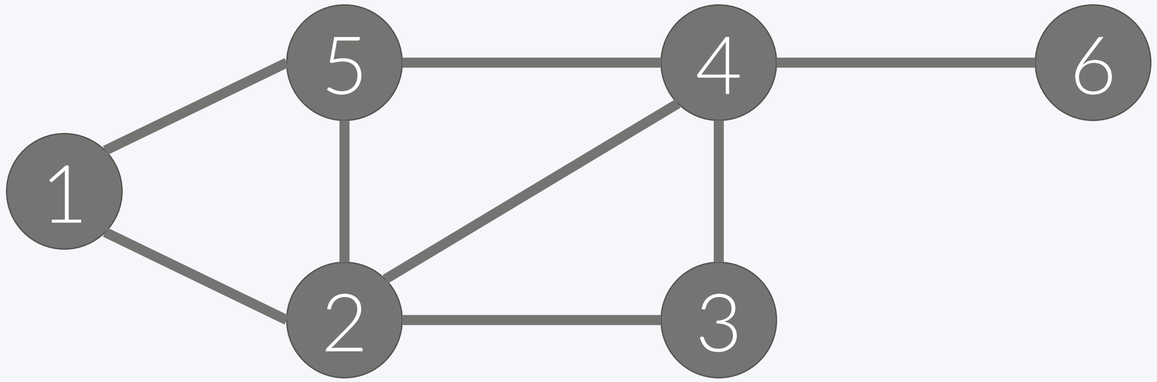
i	0	1	2	3	4	5	6
cnt[i]	0	2	6	8	12	15	16

```
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

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



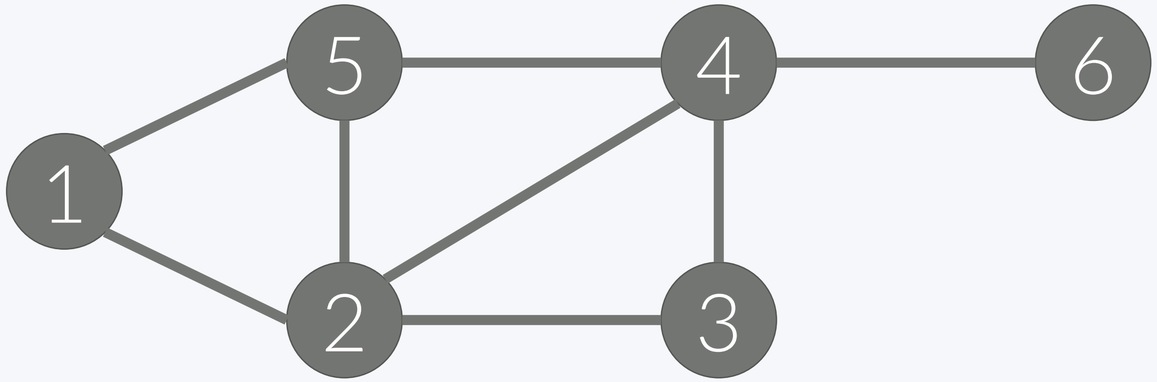
i	0	1	2	3	4	5	6
cnt[i]	0	2	6	8	12	15	16

```
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

- i번 정점과 연결된 간선은
- E배열에서 cnt[i-1]부터 cnt[i]-1 까지이다.



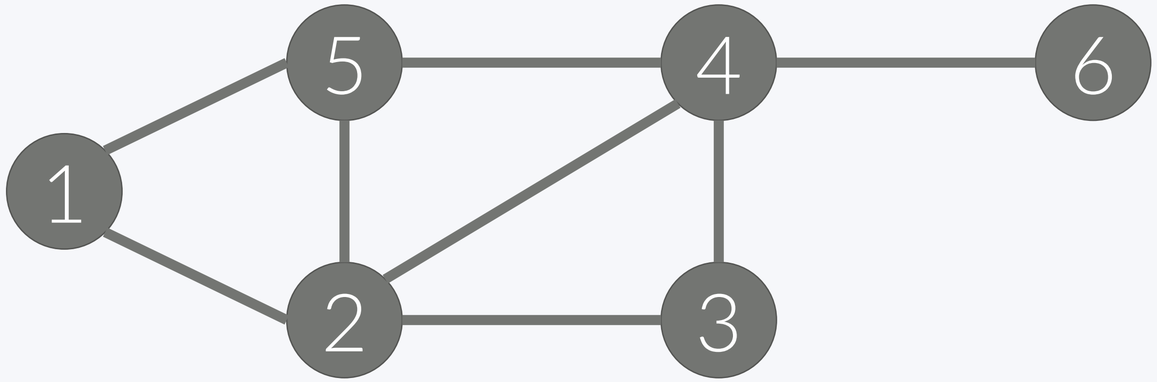
i	0	1	2	3	4	5	6
cnt[i]	0	2	6	8	12	15	16

```
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
```

간선 리스트

Edge List

- 3번 정점: cnt[2] ~ cnt[3]-1



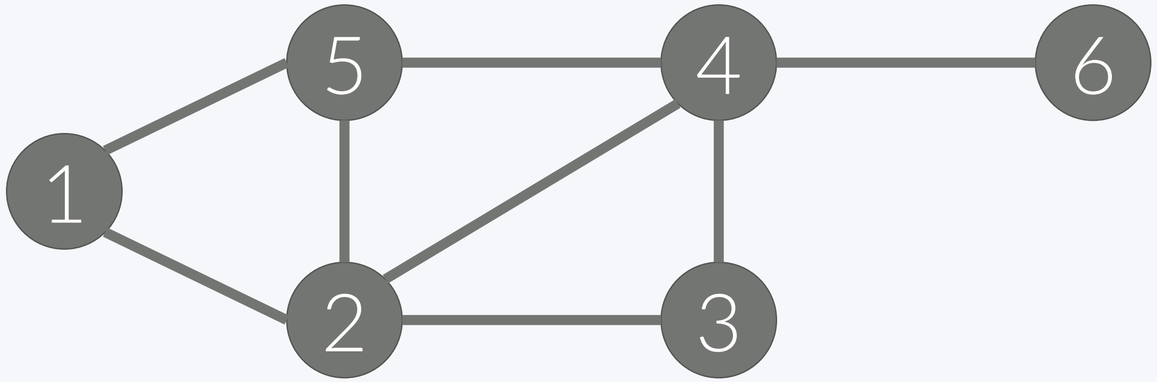
i	0	1	2	3	4	5	6
cnt[i]	0	2	6	8	12	15	16

```
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
```

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
```

ABCDE

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

- 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.

ABCDE

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

- A -> B -> C -> D -> E

- A -> B
- B -> C
- C -> D
- 에서 길이가 4인 단순 경로를 찾고
- D -> E를 찾는다.

ABCDE

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

- A -> B -> C -> D -> E

- A -> B
- C -> D
- 는 그냥 간선이기 때문에, 간선 리스트로 찾을 수 있다
- B -> C는 인접 행렬로 찾을 수 있다
- D -> E는 인접 리스트로 찾는다

ABCDE

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

- 소스: http://boj.kr/b451bfacded14e8d9a7792dda4fdcc2e

목적 : 시작점에서 출발해서
연결된 모든 정점을 1번씩 방문

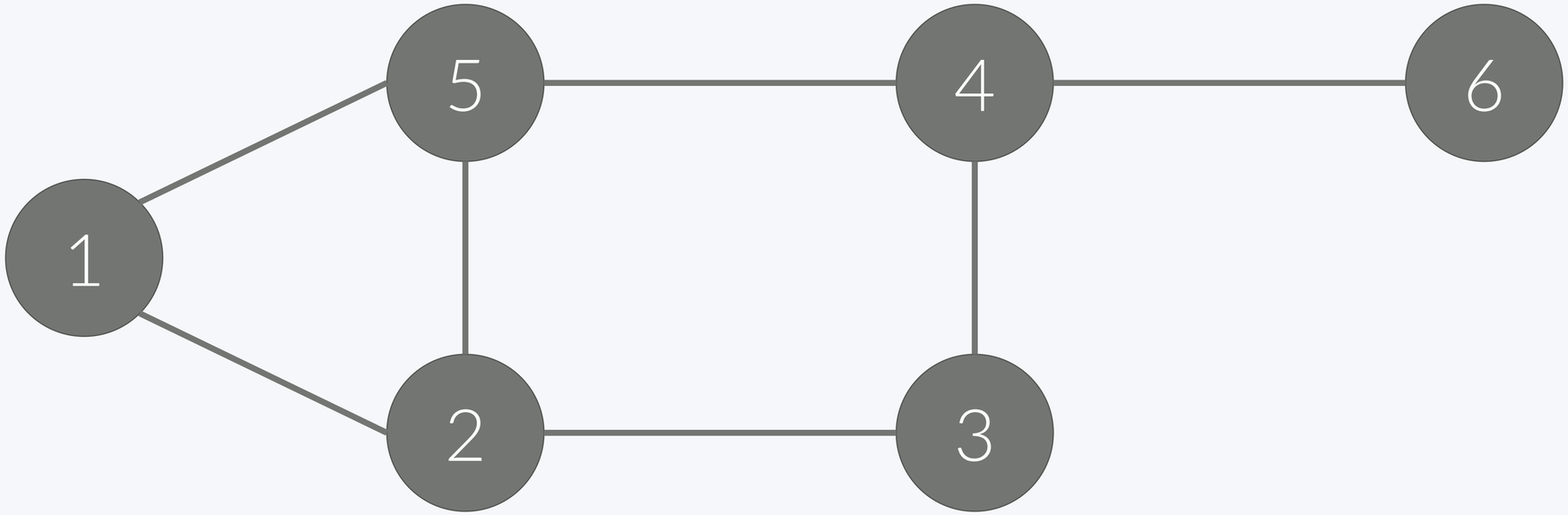
DFS, BFS

그래프의 탐색

그래프의 탐색

DFS, BFS

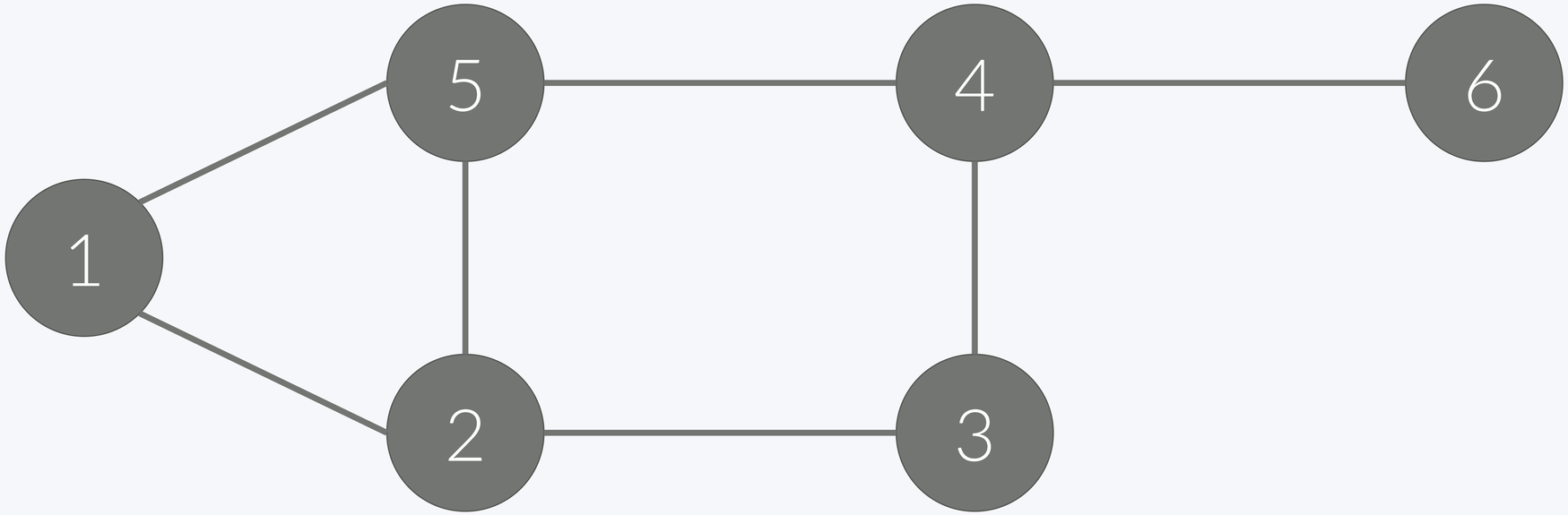
- DFS: 깊이 우선 탐색 First Search
- BFS: 너비 우선 탐색



깊이 우선 탐색

Depth First Search

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

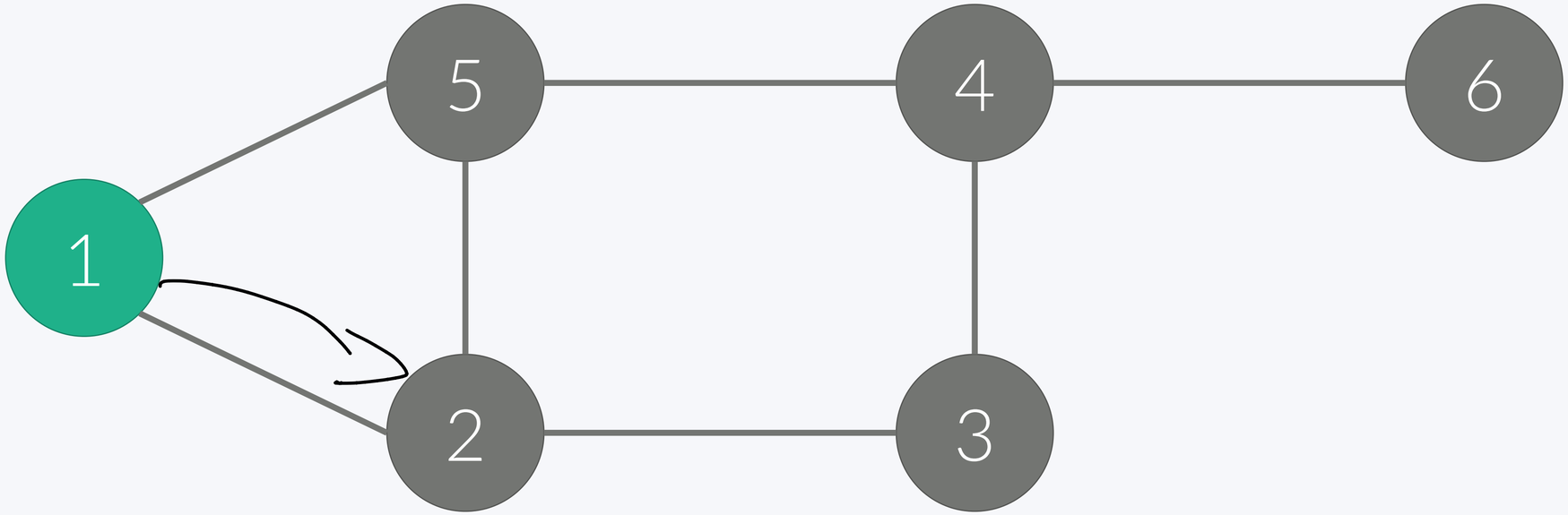


깊이 우선 탐색

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

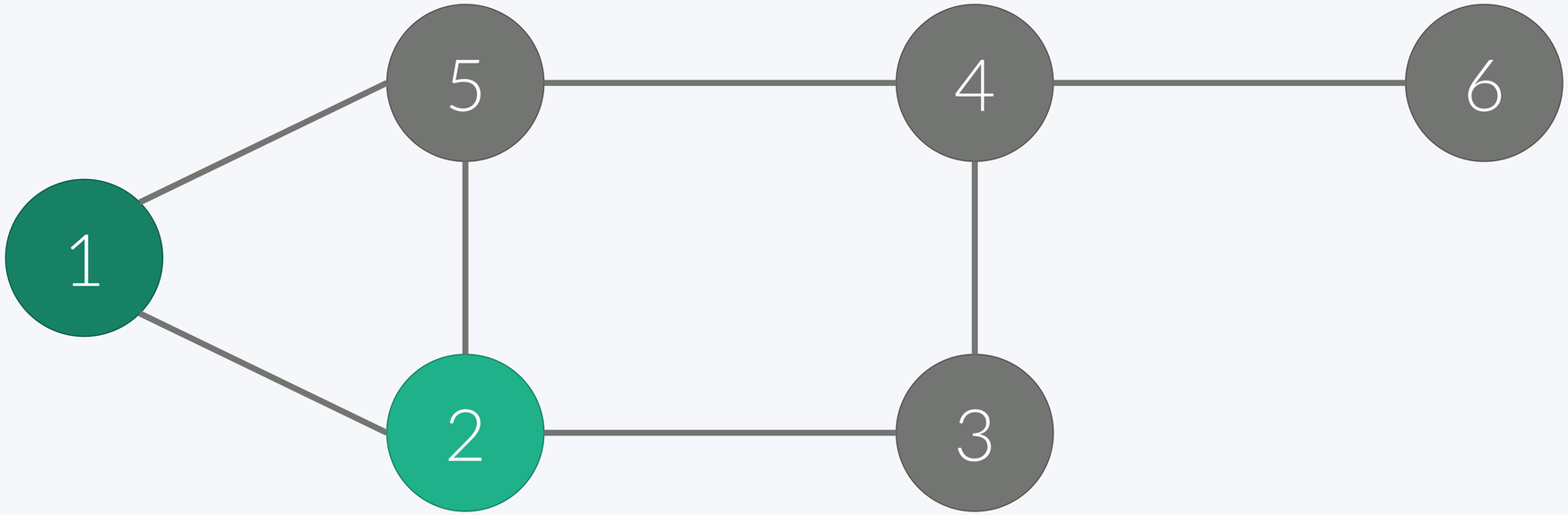


깊이 우선 탐색

Depth First Search

- 현재 정점: 2
- 순서: 1 2
- 스택: 1 2 3

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

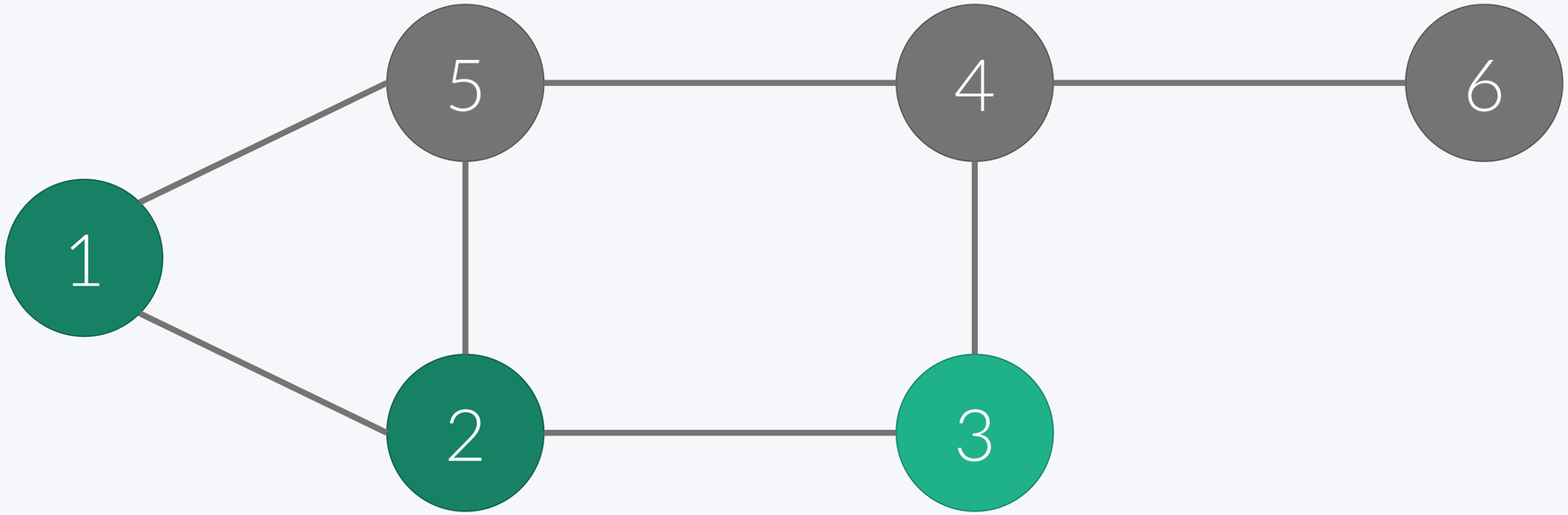


깊이 우선 탐색

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

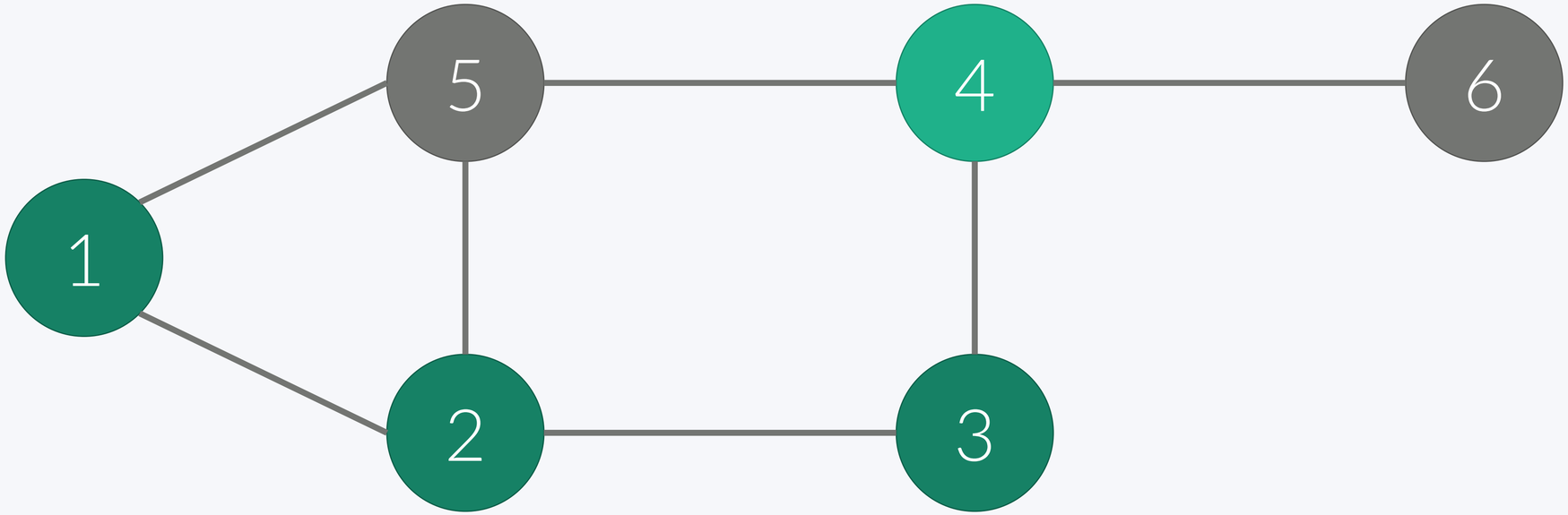


깊이 우선 탐색

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

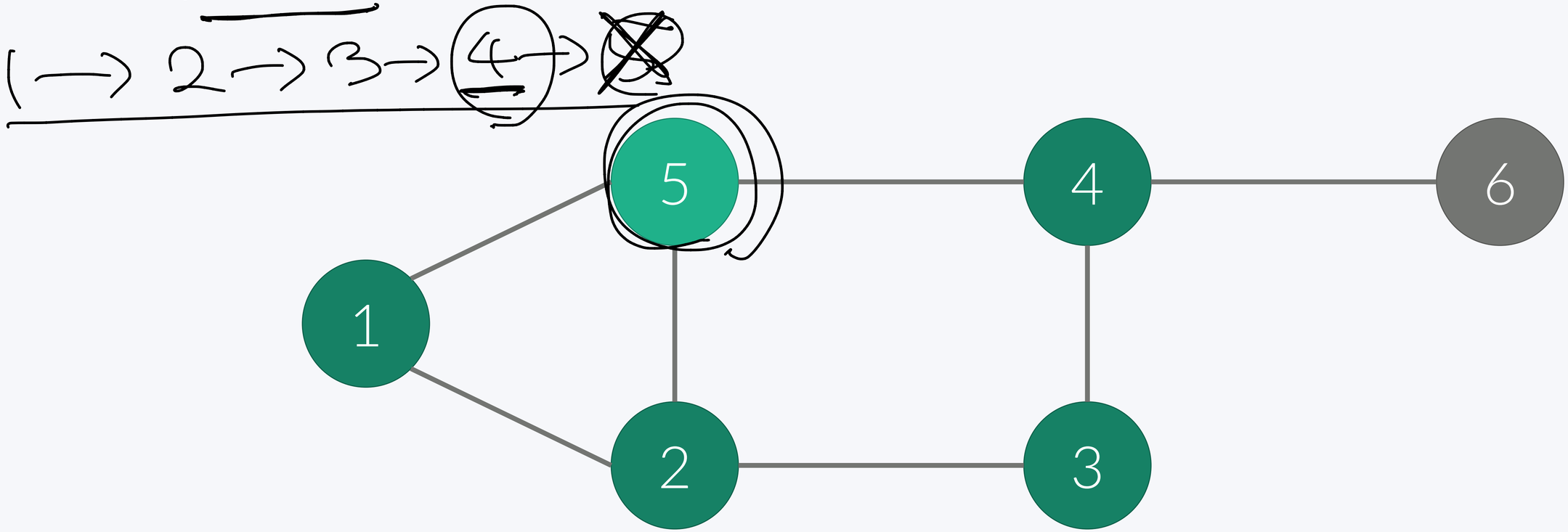


깊이 우선 탐색

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

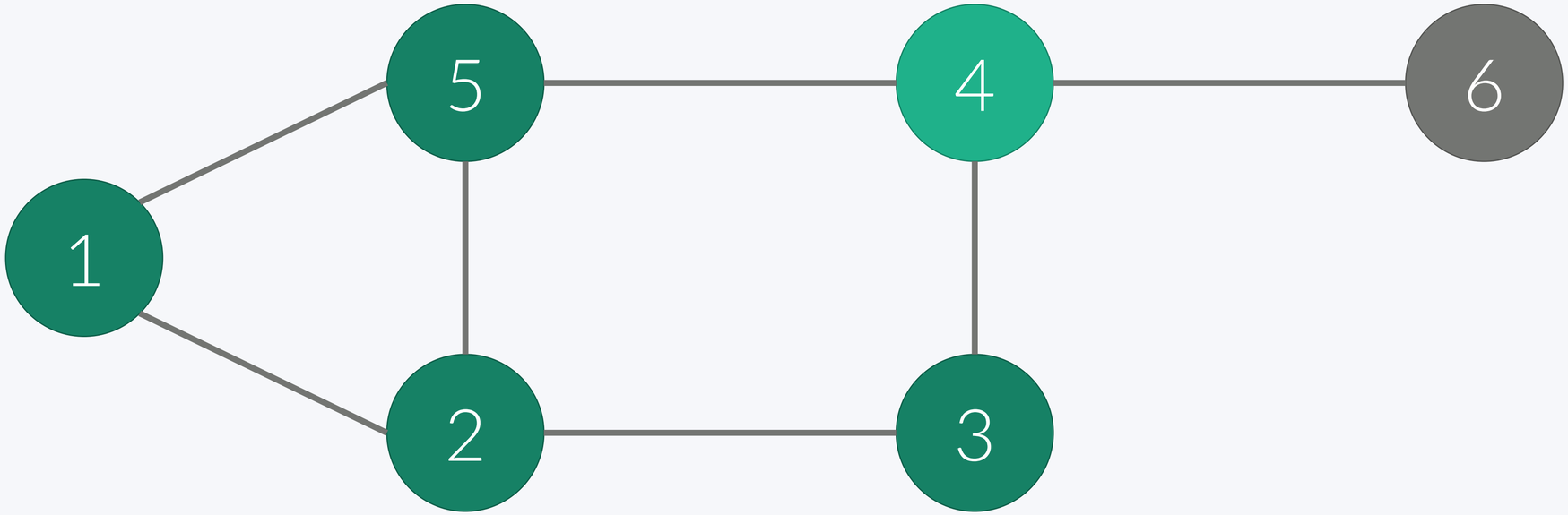


깊이 우선 탐색

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

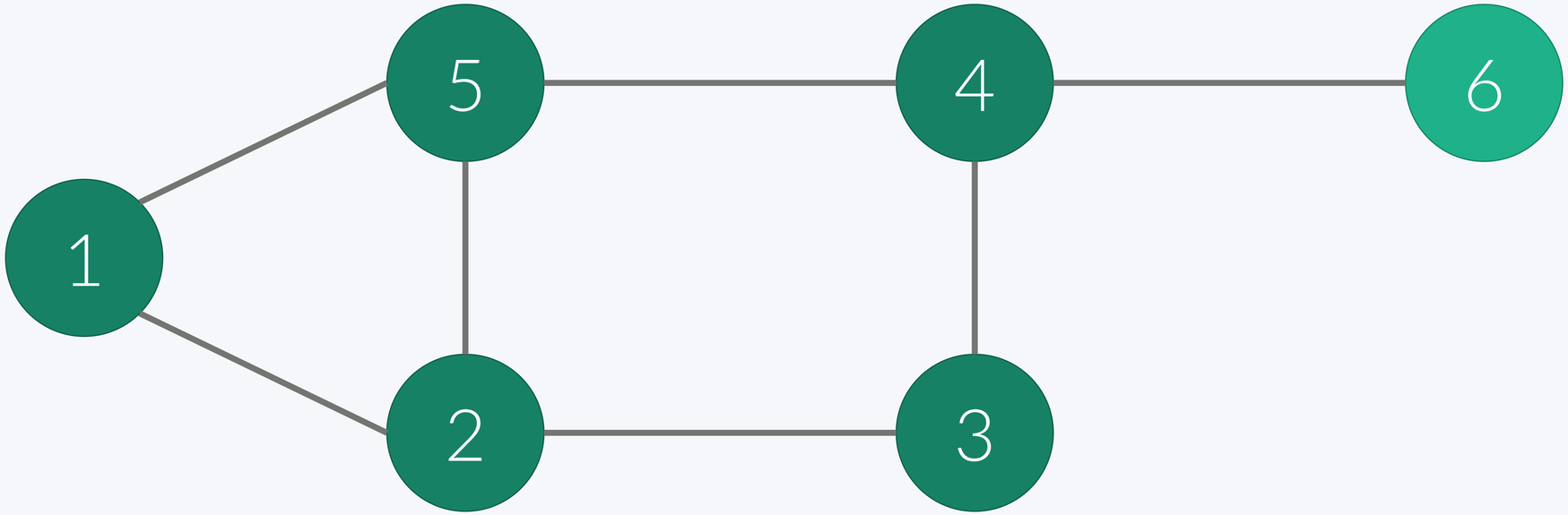


깊이 우선 탐색

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

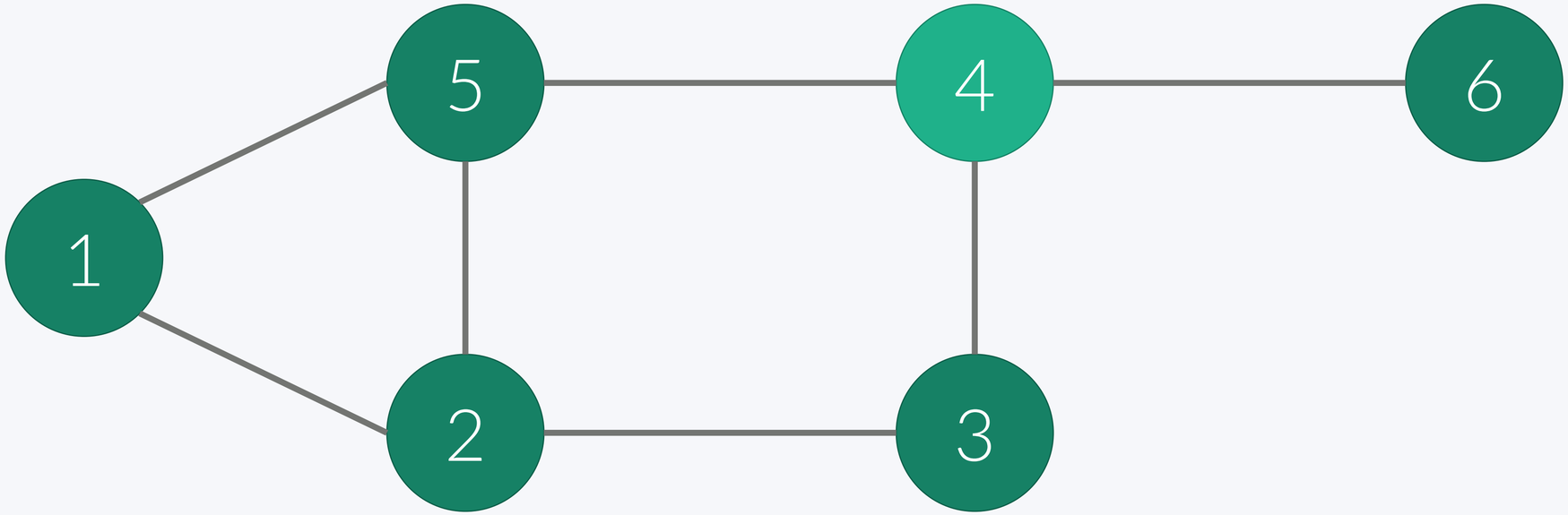


깊이 우선 탐색

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

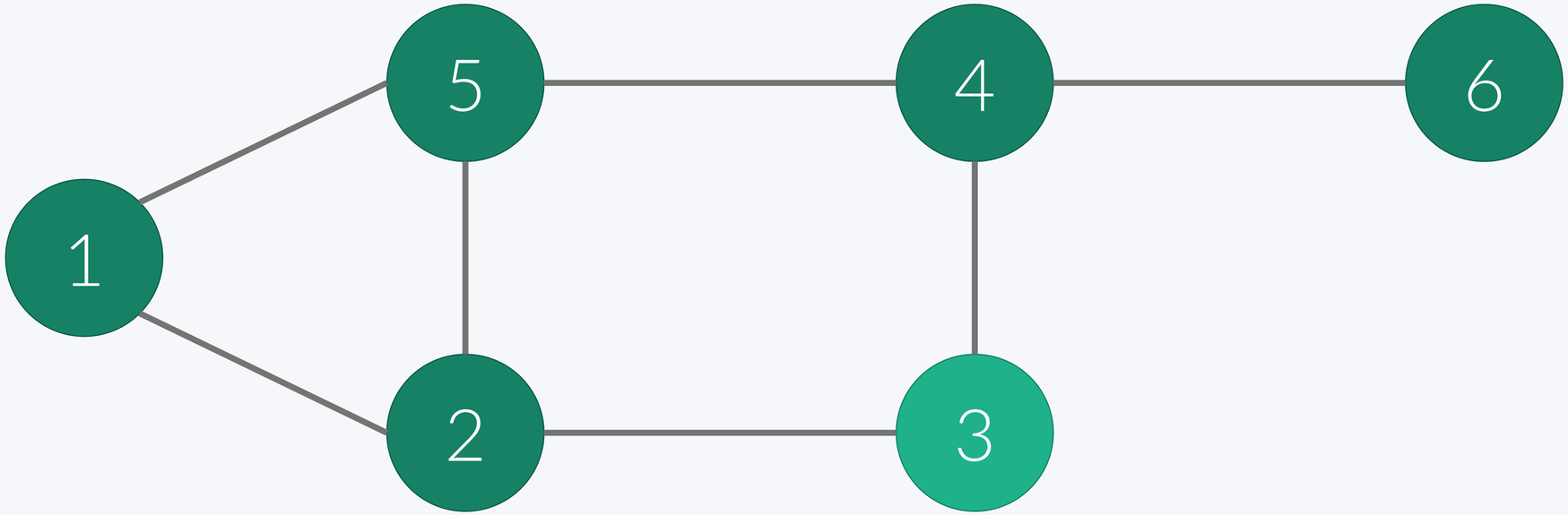


깊이 우선 탐색

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

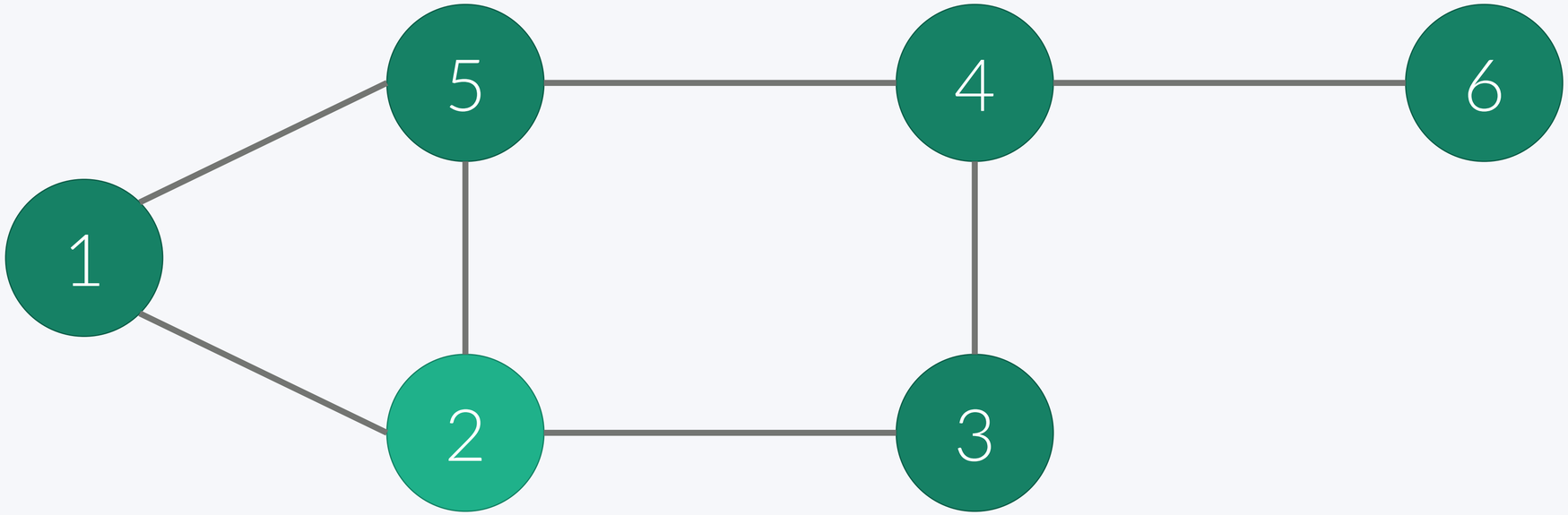


깊이 우선 탐색

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

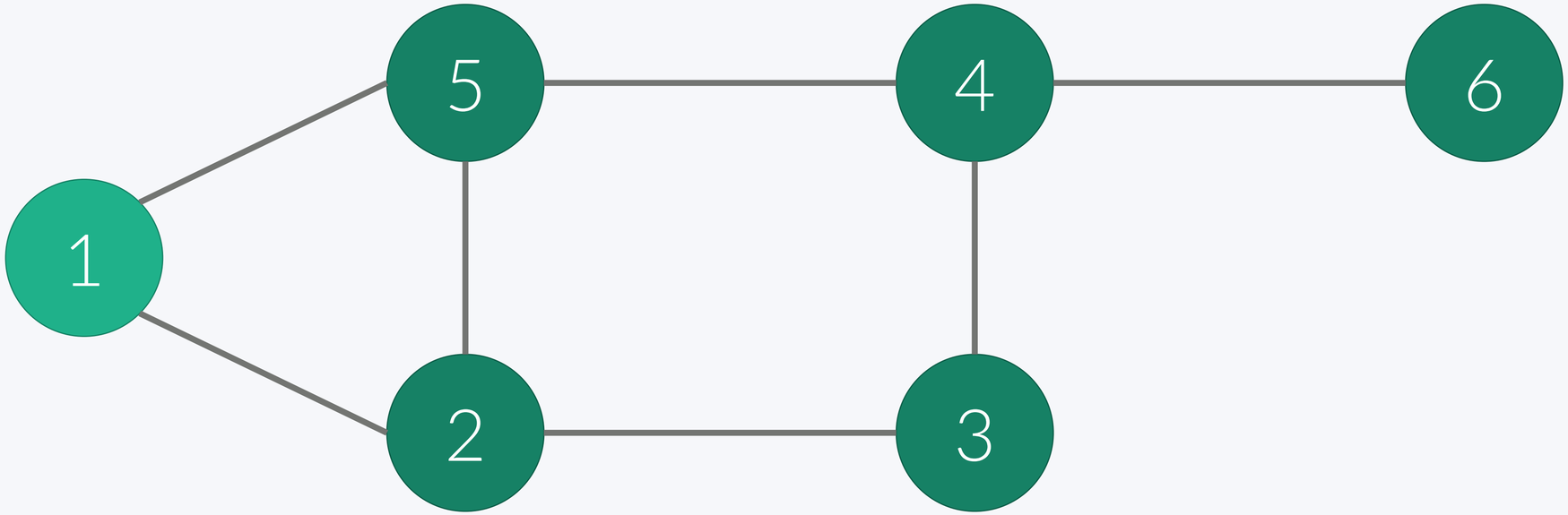


깊이 우선 탐색

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

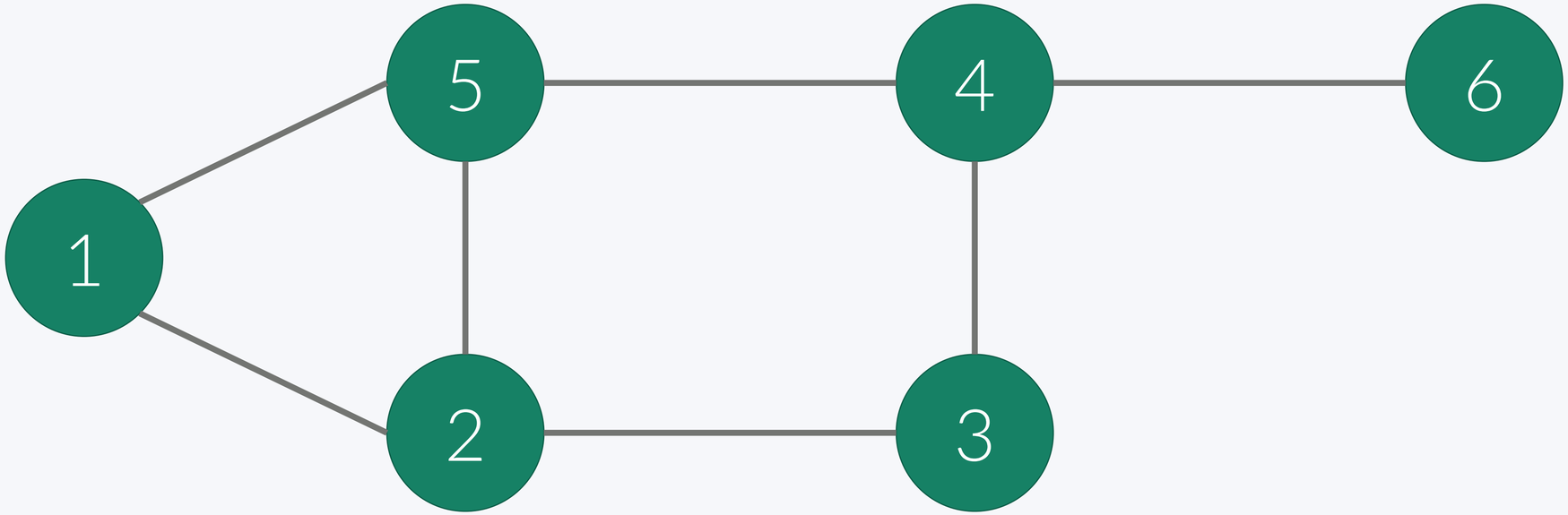


깊이 우선 탐색

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

시간복잡도 : $O(V^2)$

$V \times O(V)$

- 재귀 호출을 이용해서 구현할 수 있다.

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

x를 방문

(x) → (1)

간선 있음, 방문 x

- 인접 행렬을 이용한 구현

깊이 우선 탐색

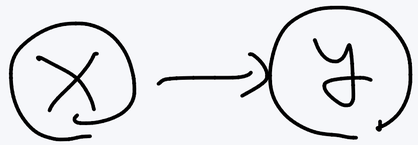
Depth First Search

시간복잡도 O(V+E)

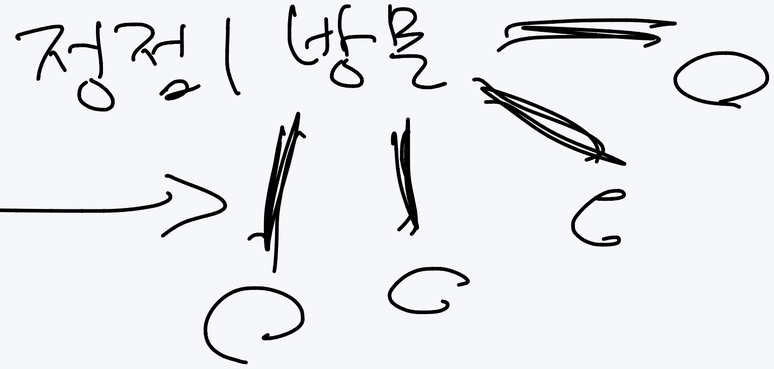
함수 V번

- 재귀 호출을 이용해서 구현할 수 있다.

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



모든 정점 1번 방문



- 인접 리스트를 이용한 구현

너비 우선 탐색

Breadth First Search

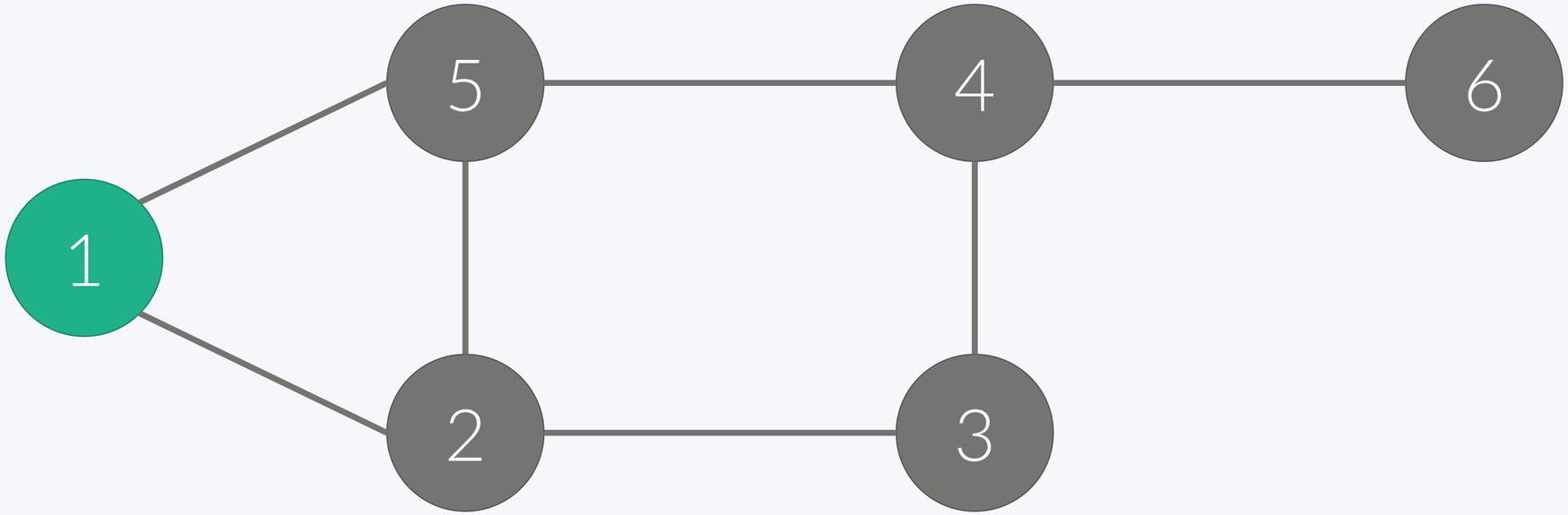
- 큐를 이용해서 지금 위치에서 갈 수 있는 것을 모두 큐에 넣는 방식
- 큐에 넣을 때 방문했다고 체크해야 한다.

너비 우선 탐색

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

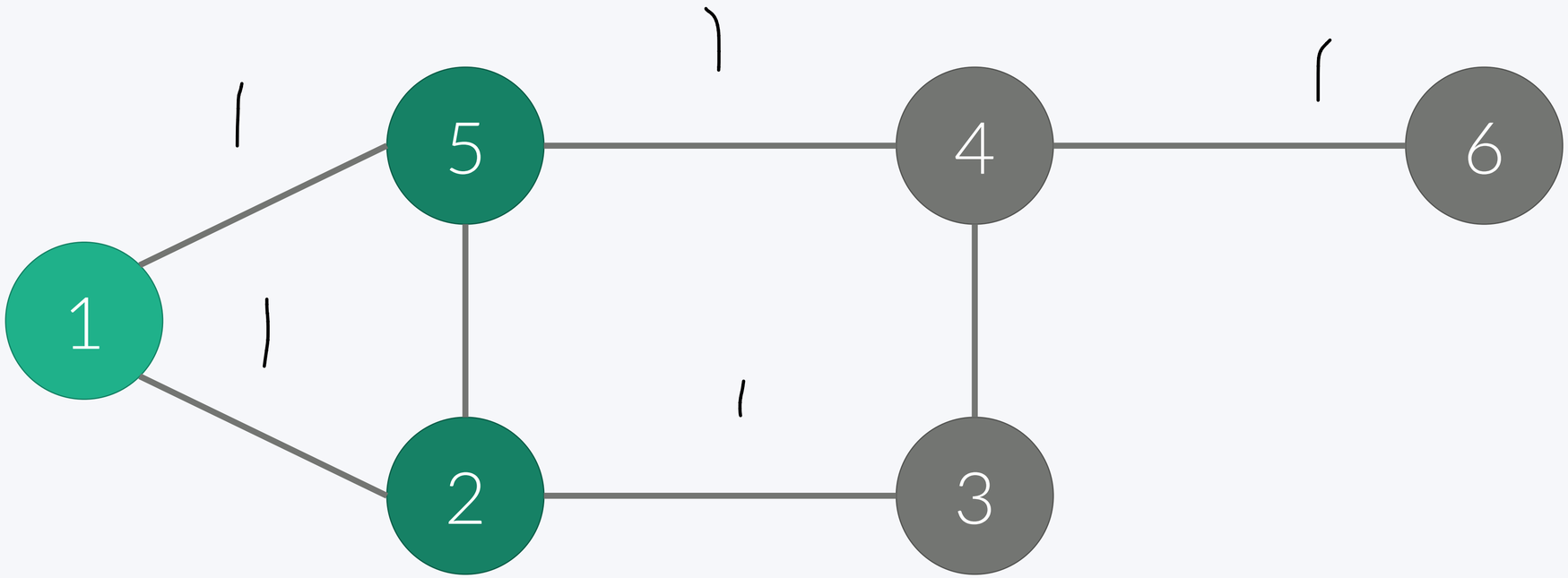


너비 우선 탐색

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

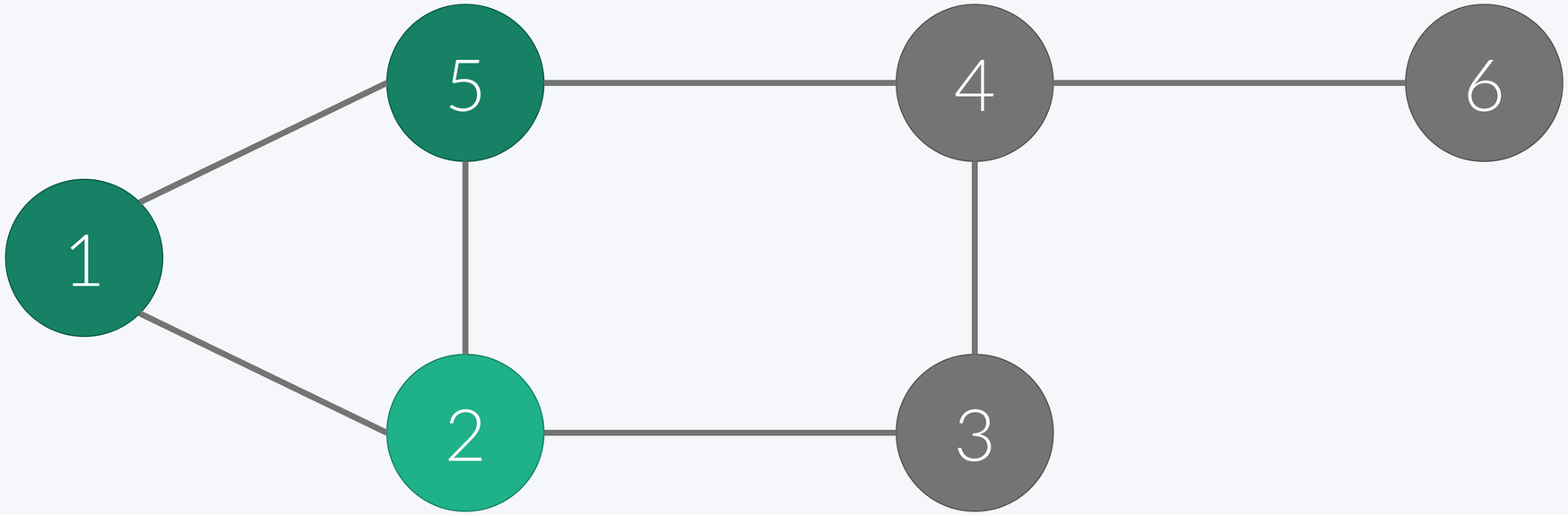


너비 우선 탐색

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

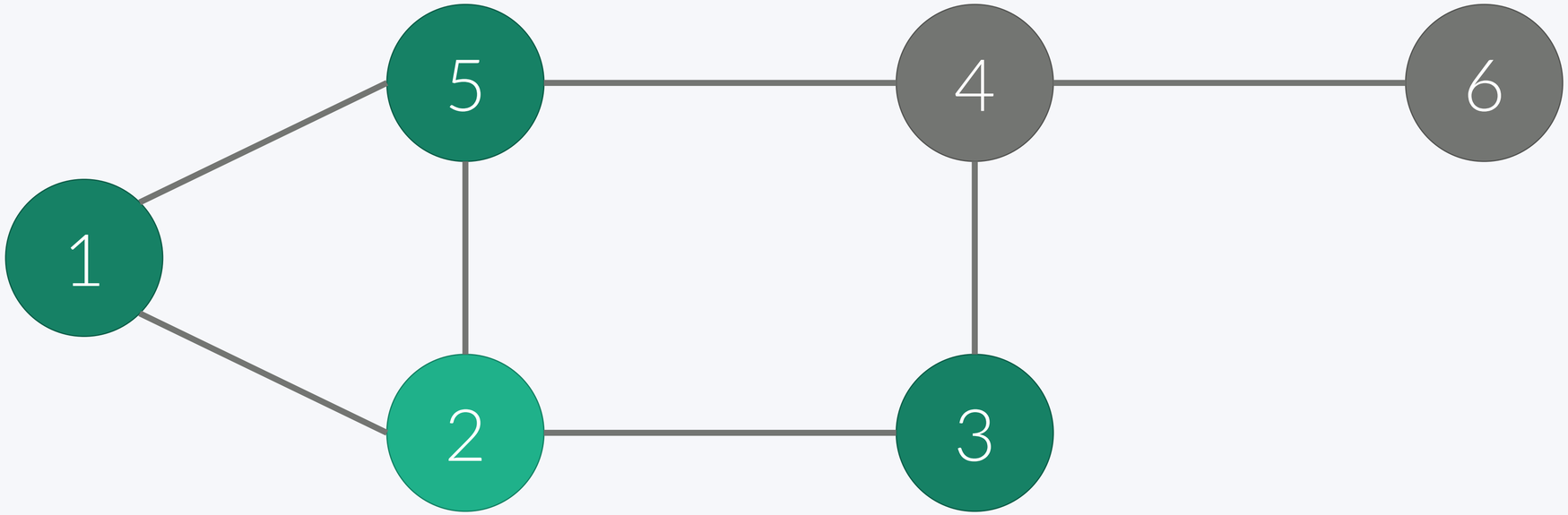


너비 우선 탐색

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

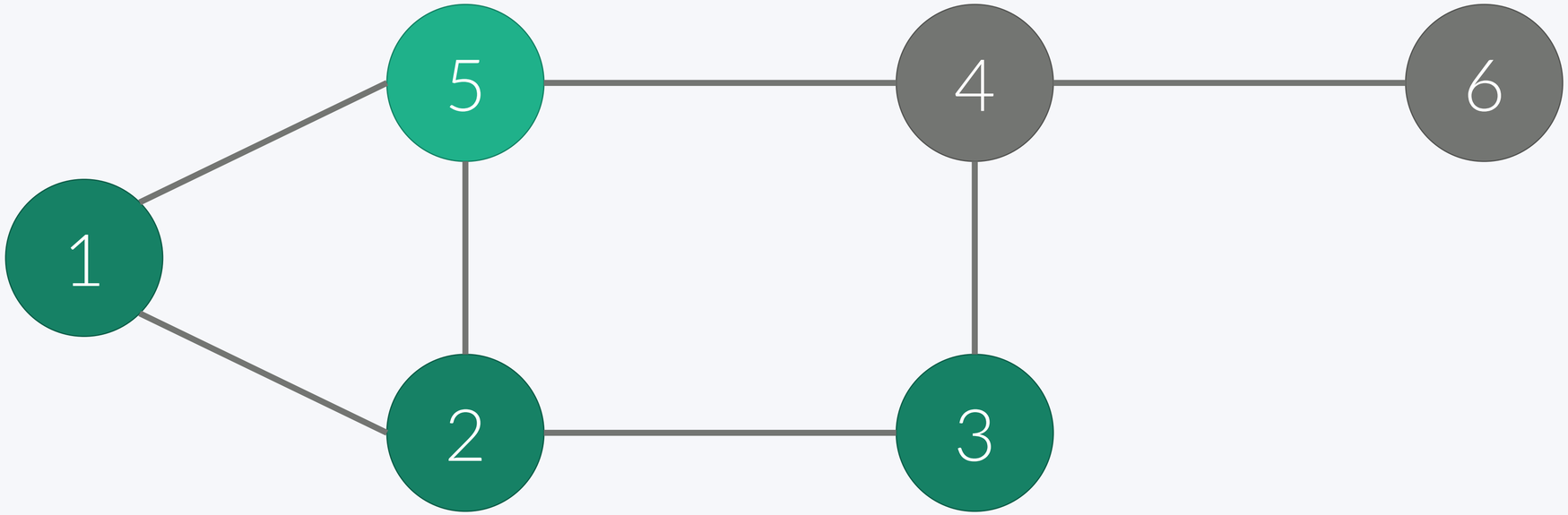


너비 우선 탐색

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

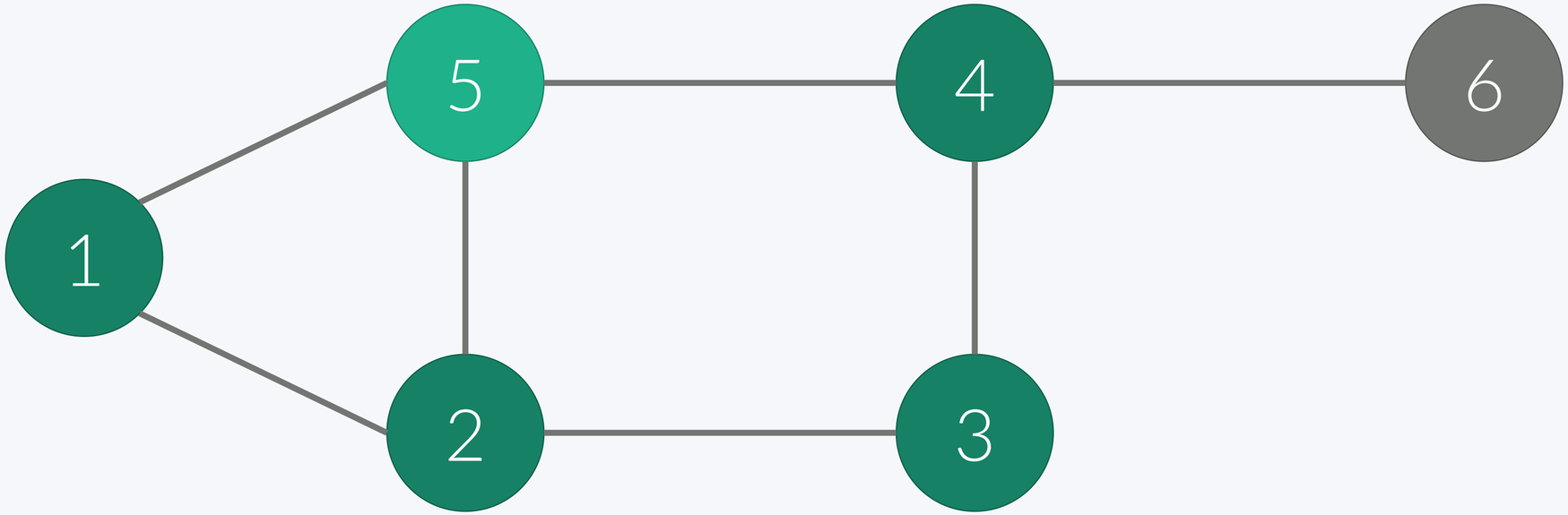


너비 우선 탐색

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

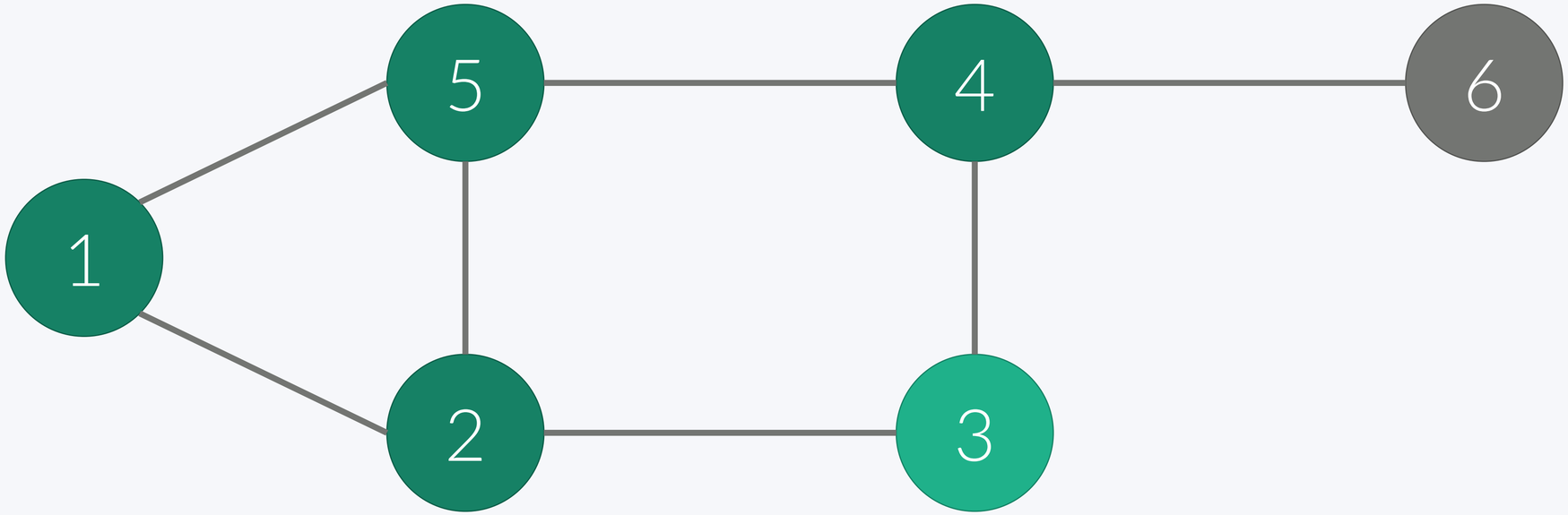


너비 우선 탐색

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

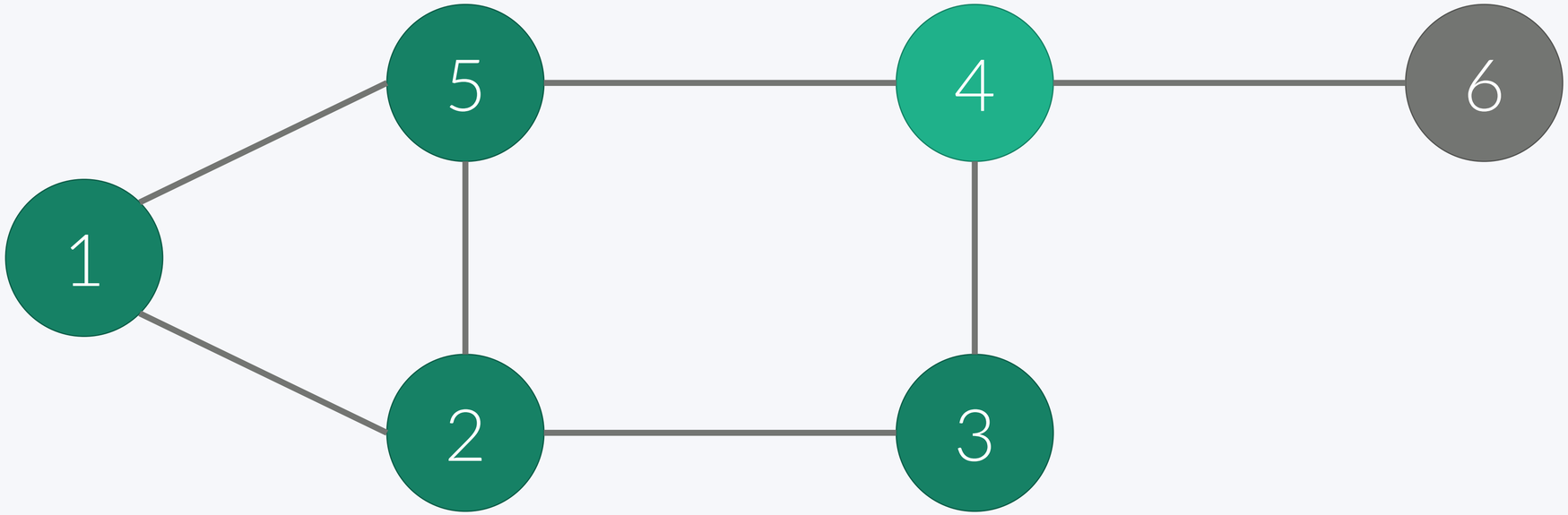


너비 우선 탐색

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

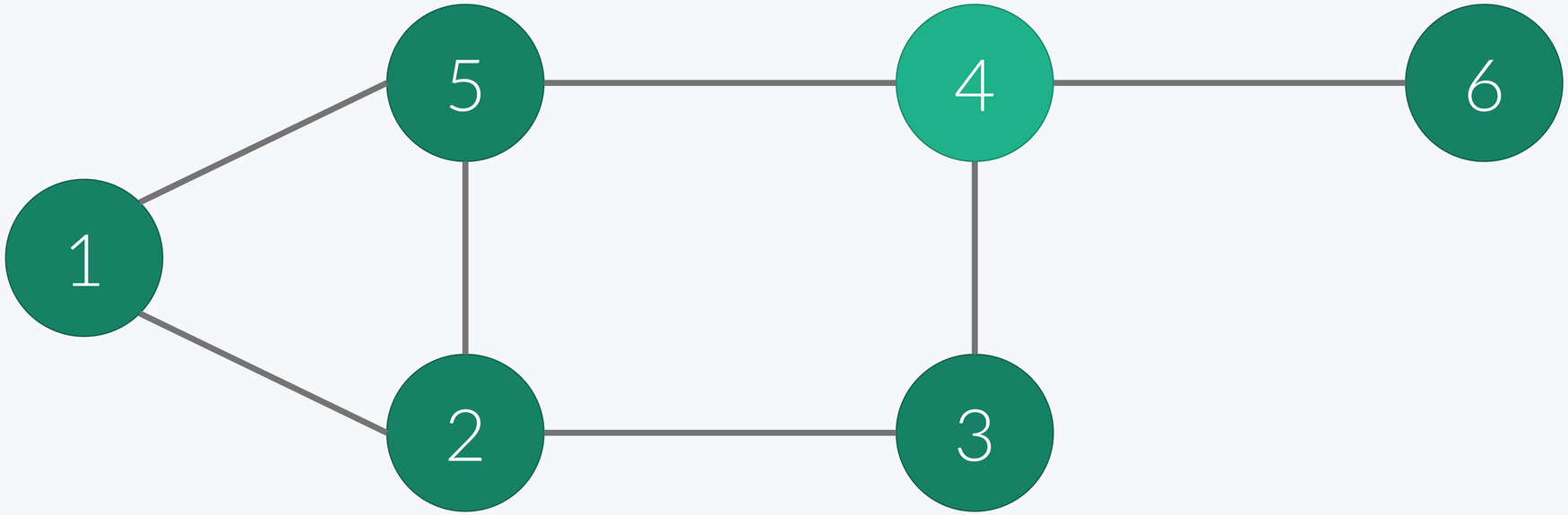


너비 우선 탐색

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

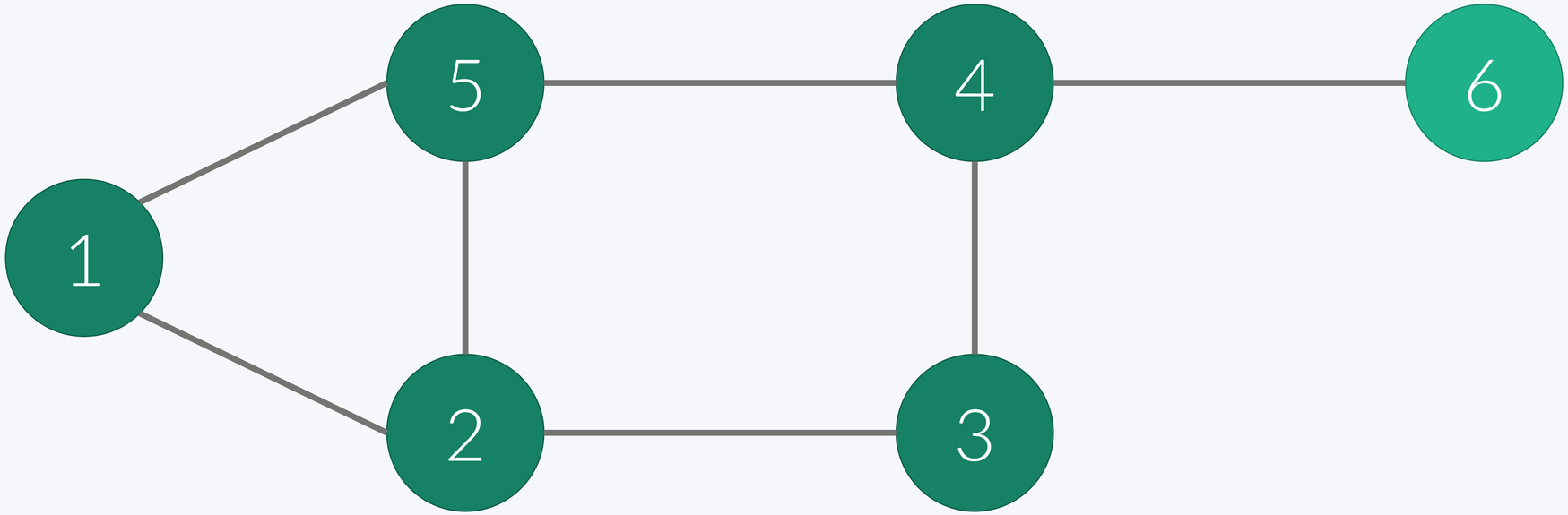


너비 우선 탐색

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

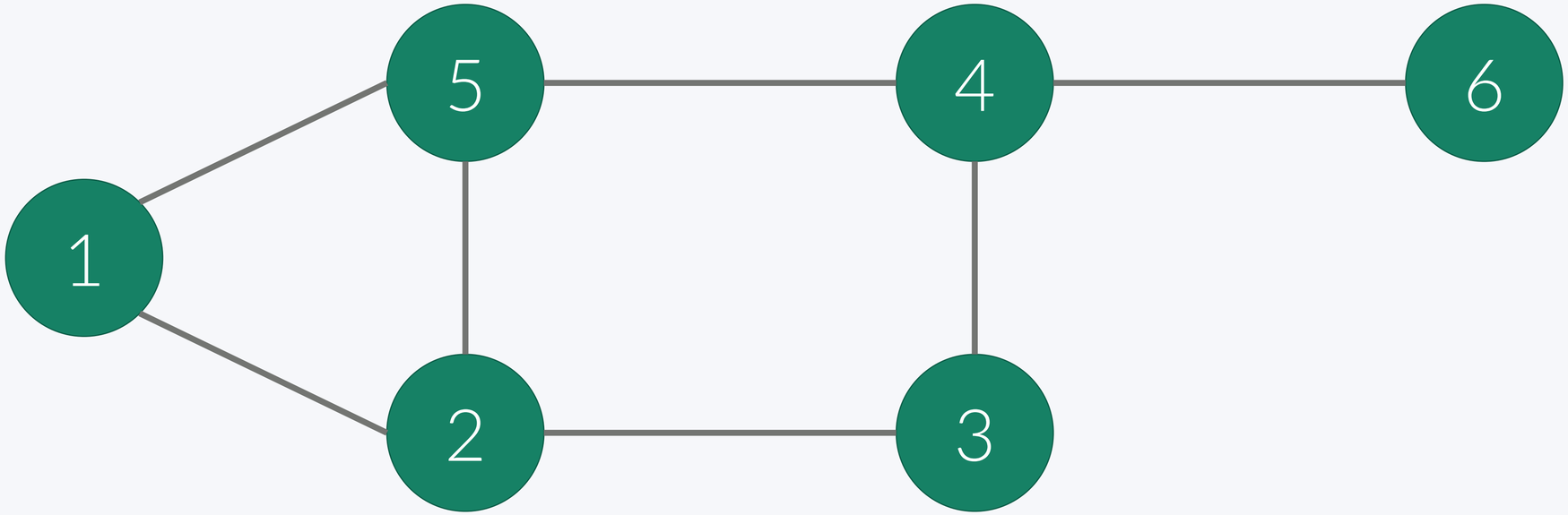


너비 우선 탐색

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 Queue를 이용해서 할 수 있다. (인접 행렬)

```
queue<int> q;
check[1] = true; q.push(1);
while (!q.empty()) {
    int x = q.front(); q.pop();
    printf("%d ",x);
    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

O(V+E)

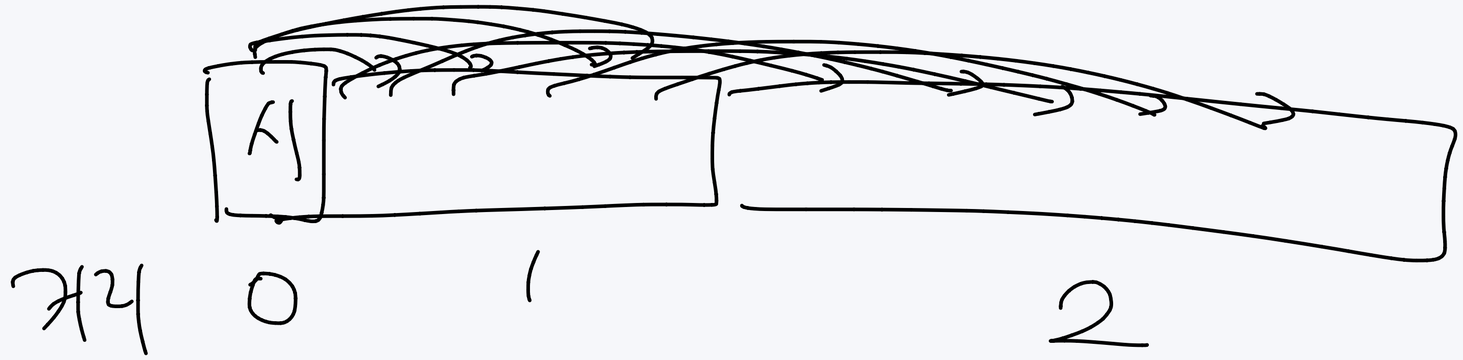
- BFS의 구현은 Queue를 이용해서 할 수 있다. (인접 리스트)

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

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://boj.kr/8cfb38af1fec4c02b45df8aa1ddc3d35
- 간선 리스트 소스: http://boj.kr/75117d4eb75046e595751ea07360d15c
- 비재귀 구현 소스: http://boj.kr/3dd5be9093ba4cc69ad0ce10d2ecb318

연결 그래프
Connected Graph

연결 요소

연결 요소

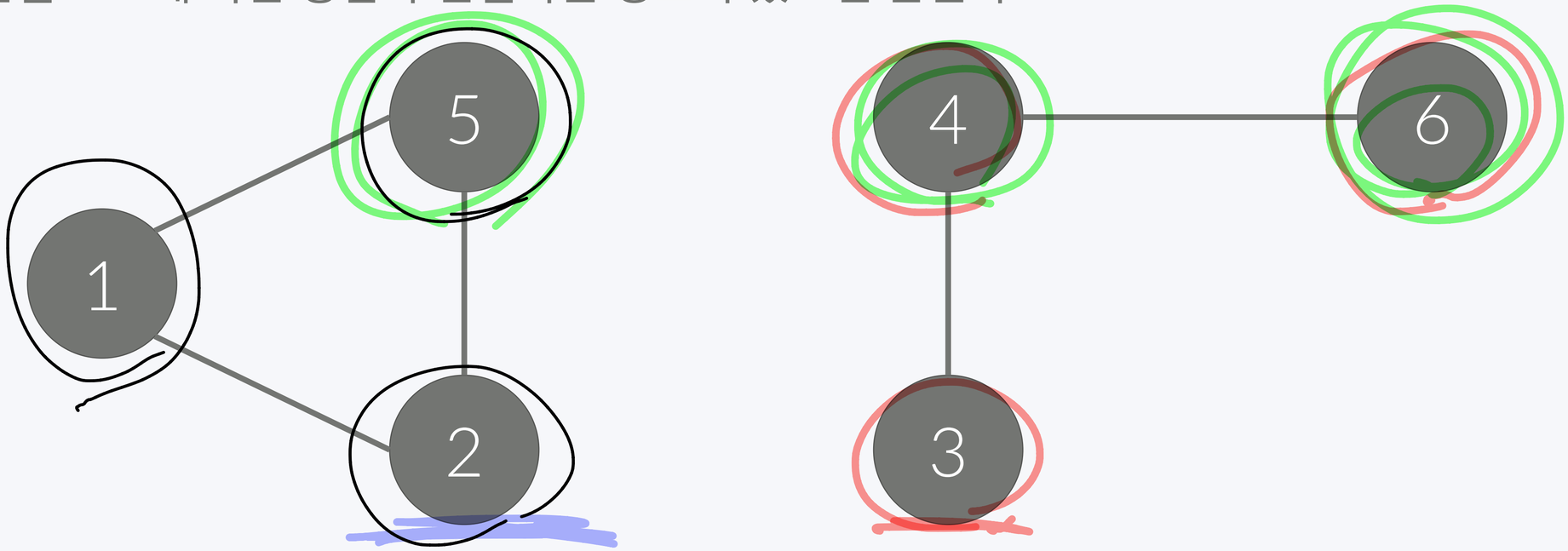
Connected Component

DFS/BFS

그래프 1개
연결요소 2개

- 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

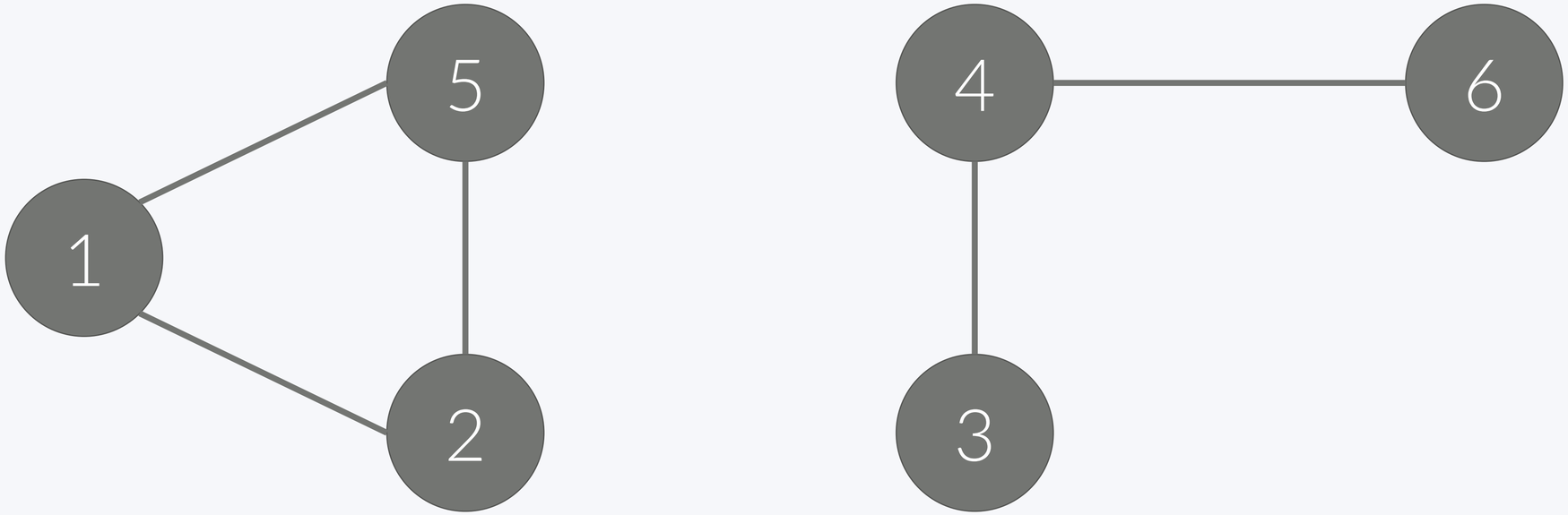
2



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

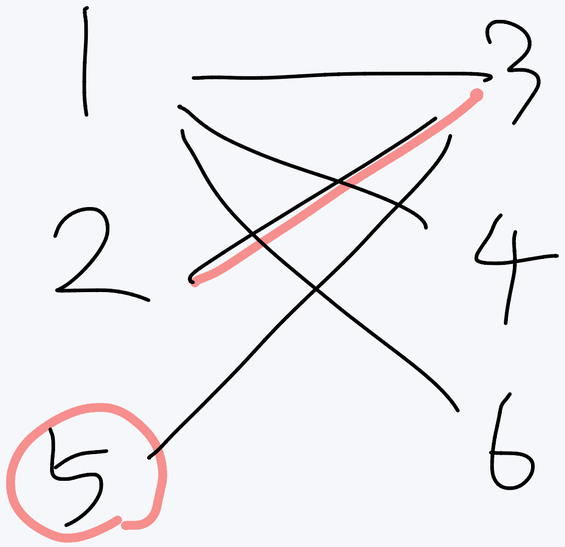
- 소스: http://boj.kr/834e40c947c446e3b1062db14a12a346

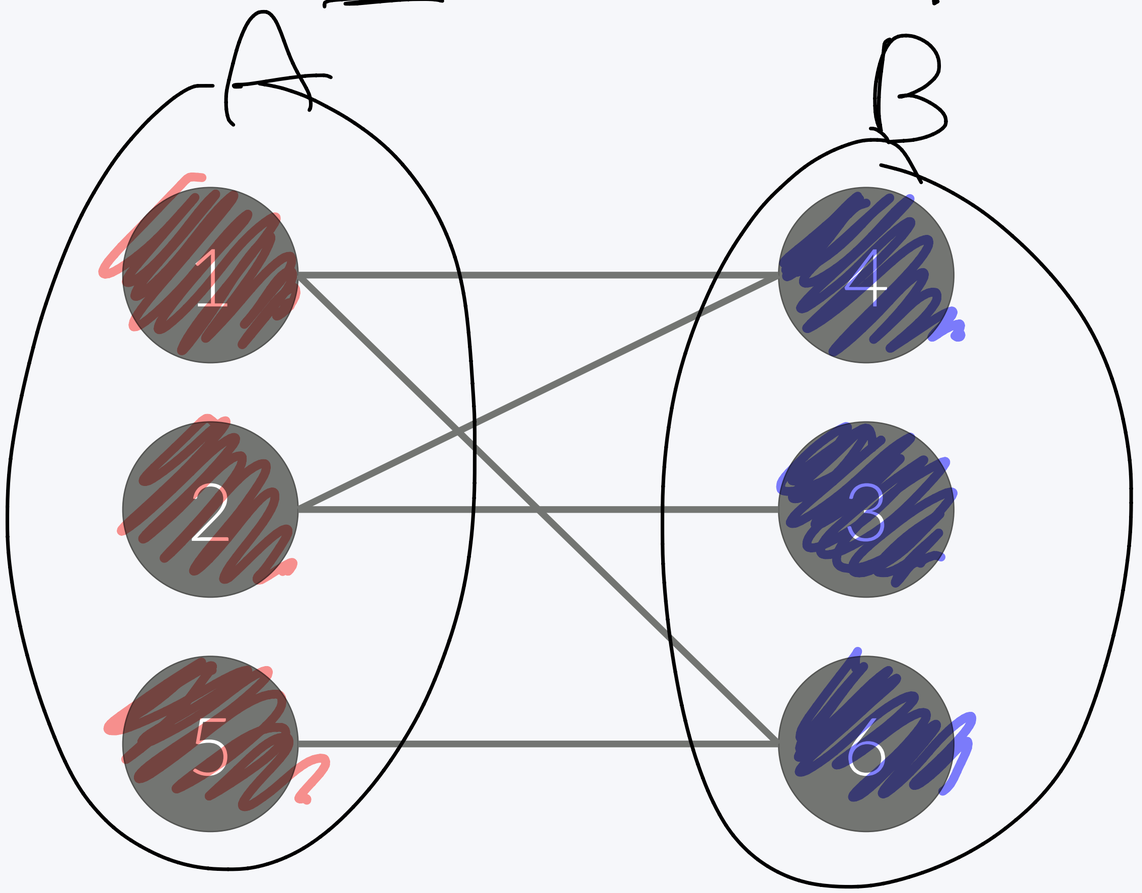
이분 그래프

이분 그래프

Bipartite Graph

- 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

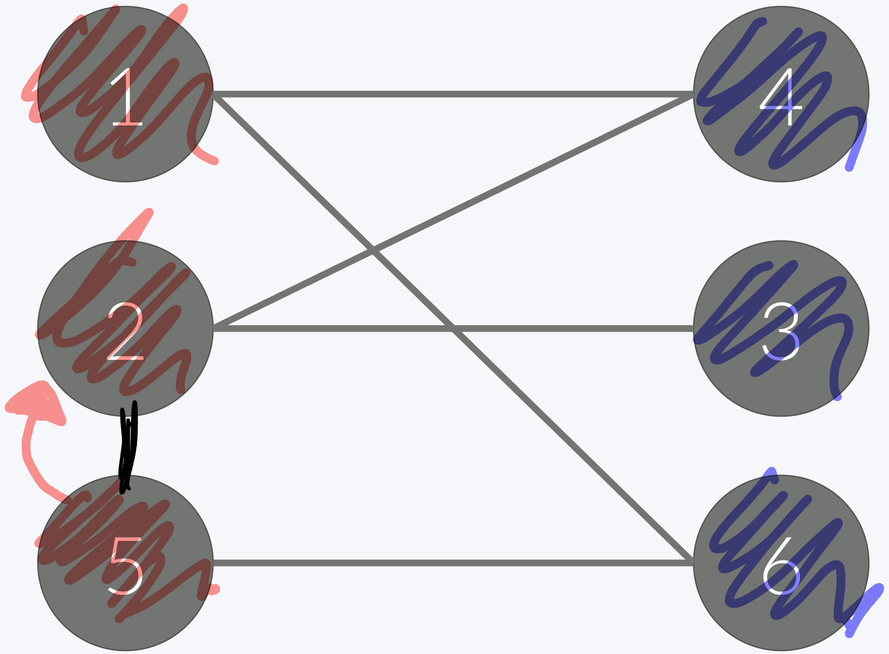




이분 그래프

Bipartite Graph

- 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://boj.kr/f4c6e3659f664a1db4c0499b4fdb29f4

플러드 필

플러드 필

Flood Fill

- 어떤 위치와 연결된 모든 위치를 찾는 알고리즘

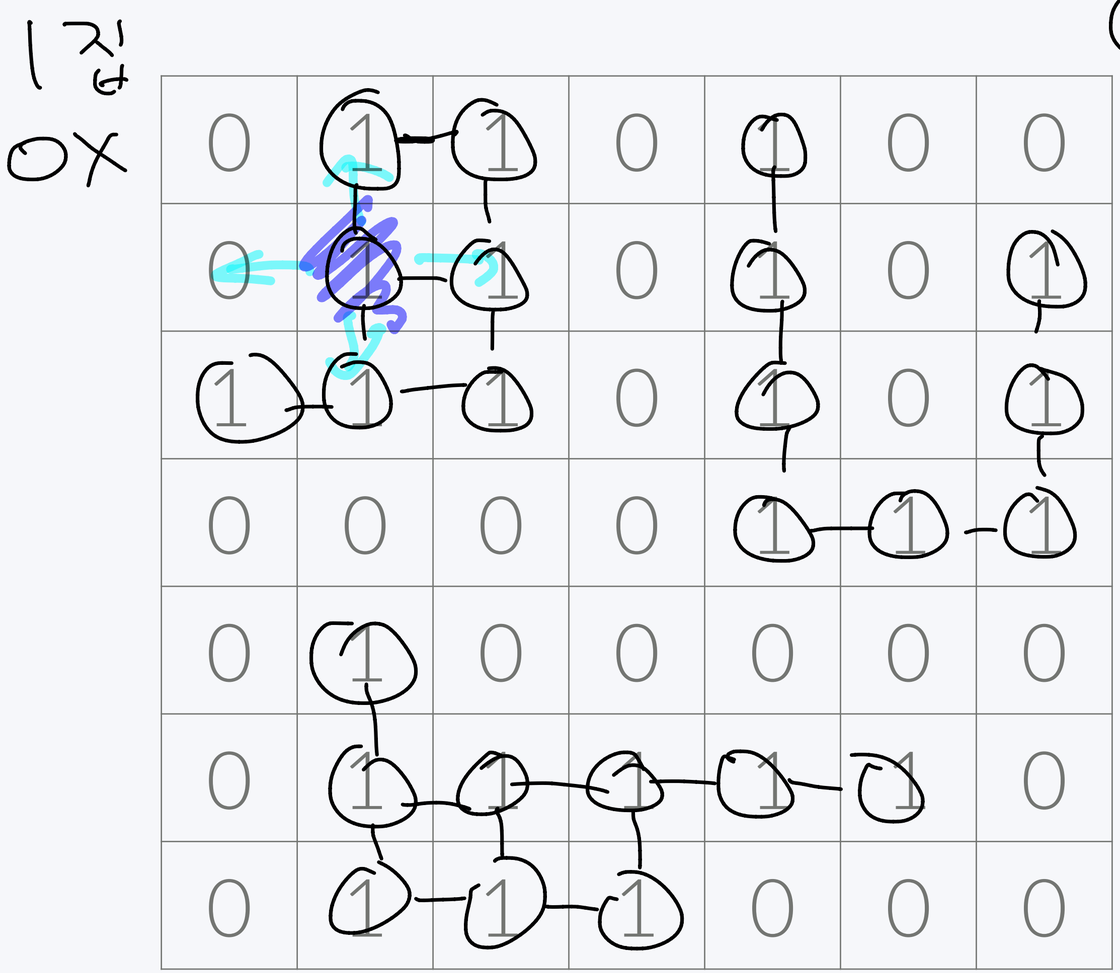
단지번호붙이기

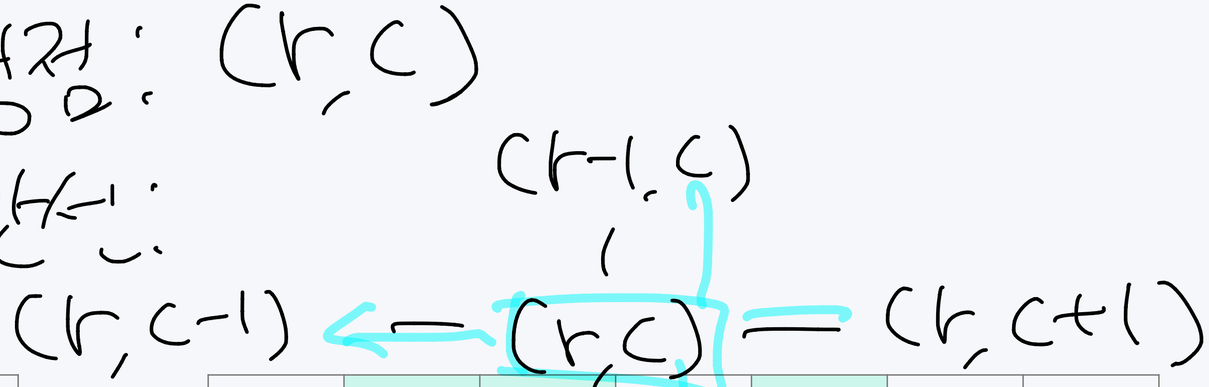
https://www.acmicpc.net/problem/2667

- 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





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

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

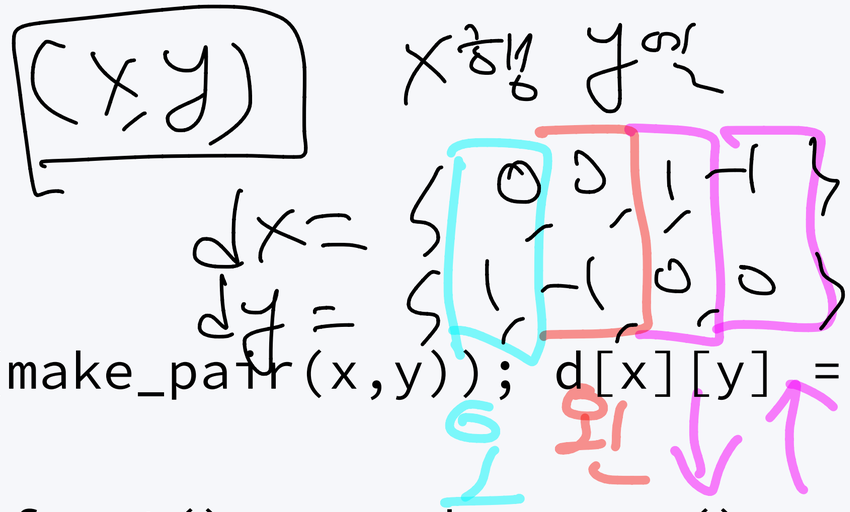
단지번호붙이기

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




```
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()) {
        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;
                }
            }
        }
    }
}
```

(x,y) → (nx,ny)

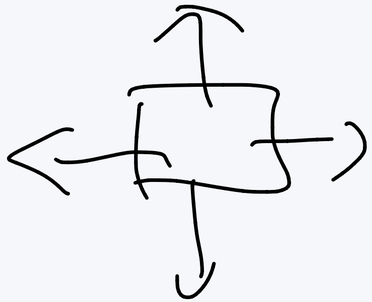
단지번호붙이기

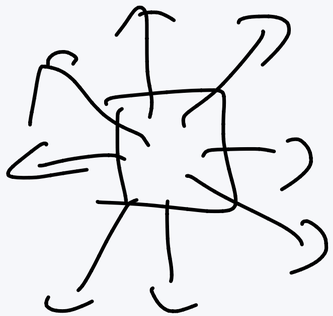
https://www.acmicpc.net/problem/2667

- BFS 소스: http://boj.kr/494137bf87e9463e9b8e43aebaf4b788
- DFS 소스: http://boj.kr/4007c833ec354fea85f1fe9ccd4d3b0d

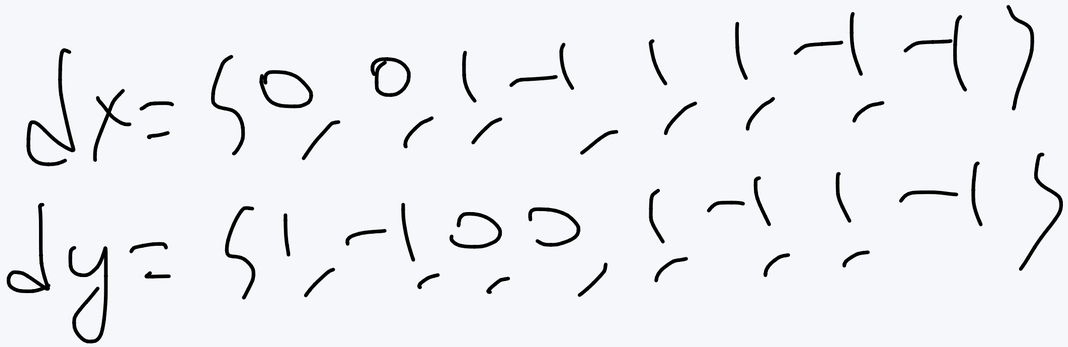
섬의 개수

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





- 소스: http://boj.kr/9219ebb6c3a84919849c93995005908f



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

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					

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

- (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://boj.kr/62c8486599a34ff48627c6dab5158802

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

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://boj.kr/5b73908ef3484bfbb5b8b93d0840a53b

다리 만들기

https://www.acmicpc.net/problem/2146

- 여러 섬으로 이루어진 나라에서
- 두 섬을 연결하는 가장 짧은 다리를 찾는 문제

다리 만들기

https://www.acmicpc.net/problem/2146

- 단지번호붙이기 + 토마토 문제
- 먼저, 섬을 그룹을 나눈다
- g[i][j] = (i,j)의 그룹 번호
- 그 다음 각각의 그룹에 대해서 다른 섬까지 거리를 계산한다
- 이 방법은 각각이 그룹에 대해서 BFS 알고리즘을 수행해야 하기 때문에 느리다

다리 만들기

https://www.acmicpc.net/problem/2146

- 소스: http://boj.kr/09841dfadee647789357dcc88a33056f

다리 만들기

https://www.acmicpc.net/problem/2146

- 더 빠른 알고리즘으로 땅을 확장하는 방식을 생각해 볼 수 있다.

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

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

다리 만들기

https://www.acmicpc.net/problem/2146

- 왼쪽: 그룹, 오른쪽: 거리

1	1	1					2	2	2
1	1	1	1					2	2
1		1	1					2	2
		1	1	1					2
			1						2
									2
				3	3				
				3	3	3			

0	0	0	-	-	-	-	0	0	0
0	0	0	0	-	-	-	-	0	0
0	-	0	0	-	-	-	-	0	0
-	-	0	0	0	-	-	-	-	0
-	-	-	0	-	-	-	-	-	0
-	-	-	-	-	-	-	-	-	0
-	-	-	-	-	-	-	-	-	-
-	-	-	-	0	0	-	-	-	-
-	-	-	-	0	0	0	-	-	-
-	-	-	-	-	-	-	-	-	-

다리 만들기

https://www.acmicpc.net/problem/2146

- 왼쪽: 그룹, 오른쪽: 거리

1	1	1	1			2	2	2	2
1	1	1	1	1			2	2	2
1	1	1	1	1			2	2	2
1	1	1	1	1	1			2	2
		1	1	1				2	2
			1					2	2
				3	3				2
			3	3	3	3			
			3	3	3	3	3		
				3	3	3			

0	0	0	1	-	-	1	0	0	0
0	0	0	0	1	-	-	1	0	0
0	1	0	0	1	-	-	1	0	0
1	1	0	0	0	1	-	-	1	0
-	-	1	0	1	-	-	-	1	0
-	-	-	1	-	-	-	-	1	0
-	-	-	-	1	1	-	-	-	1
-	-	-	1	0	0	1	-	-	-
-	-	-	1	0	0	0	1	-	-
-	-	-	-	1	1	1	-	-	-

다리 만들기

https://www.acmicpc.net/problem/2146

- 왼쪽: 그룹, 오른쪽: 거리

1	1	1	1	1	2	2	2	2	2
1	1	1	1	1	1	2	2	2	2
1	1	1	1	1	1	2	2	2	2
1	1	1	1	1	1	1	2	2	2
1	1	1	1	1	1		2	2	2
		1	1	1	3		2	2	2
			1	3	3	3		2	2
		3	3	3	3	3	3		2
		3	3	3	3	3	3	3	
			3	3	3	3	3		

0	0	0	1	2	2	1	0	0	0
0	0	0	0	1	2	2	1	0	0
0	1	0	0	1	2	2	1	0	0
1	1	0	0	0	1	2	2	1	0
2	2	1	0	1	2	-	2	1	0
-	-	2	1	2	2	-	2	1	0
-	-	-	2	1	1	2	-	2	1
-	-	2	1	0	0	1	2	-	2
-	-	2	1	0	0	0	1	2	-
-	-	-	2	1	1	1	2	-	-

다리 만들기

https://www.acmicpc.net/problem/2146

- 왼쪽: 그룹, 오른쪽: 거리

1	1	1	1	1	2	2	2	2	2
1	1	1	1	1	1	2	2	2	2
1	1	1	1	1	1	2	2	2	2
1	1	1	1	1	1	1	2	2	2
1	1	1	1	1	1	1	2	2	2
1	1	1	1	1	3	2	2	2	2
		1	1	3	3	3	2	2	2
	3	3	3	3	3	3	3	2	2
	3	3	3	3	3	3	3	3	2
		3	3	3	3	3	3	3	

0	0	0	1	2	2	1	0	0	0
0	0	0	0	1	2	2	1	0	0
0	1	0	0	1	2	2	1	0	0
1	1	0	0	0	1	2	2	1	0
2	2	1	0	1	2	3	2	1	0
3	3	2	1	2	2	3	2	1	0
-	-	3	2	1	1	2	3	2	1
-	3	2	1	0	0	1	2	3	2
-	3	2	1	0	0	0	1	2	3
-	-	3	2	1	1	1	2	3	-

다리 만들기

https://www.acmicpc.net/problem/2146

- 왼쪽: 그룹, 오른쪽: 거리

1	1	1	1	1	2	2	2	2	2
1	1	1	1	1	1	2	2	2	2
1	1	1	1	1	1	2	2	2	2
1	1	1	1	1	1	1	2	2	2
1	1	1	1	1	1	1	2	2	2
1	1	1	1	1	3	2	2	2	2
1	1	1	1	3	3	3	2	2	2
3	3	3	3	3	3	3	3	2	2
3	3	3	3	3	3	3	3	3	2
	3	3	3	3	3	3	3	3	2

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

다리 만들기

https://www.acmicpc.net/problem/2146

- 왼쪽: 그룹, 오른쪽: 거리

1	1	1	1	1	2	2	2	2	2
1	1	1	1	1	1	2	2	2	2
1	1	1	1	1	1	2	2	2	2
1	1	1	1	1	1	1	2	2	2
1	1	1	1	1	1	1	2	2	2
1	1	1	1	1	3	2	2	2	2
1	1	1	1	3	3	3	2	2	2
3	3	3	3	3	3	3	3	2	2
3	3	3	3	3	3	3	3	3	2
3	3	3	3	3	3	3	3	3	2

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

다리 만들기

https://www.acmicpc.net/problem/2146

- 각 칸과 인접한 칸의 그룹 번호가 다르면 다리를 만들 수 있다

1	1	1	1	1	2	2	2	2	2
1	1	1	1	1	1	2	2	2	2
1	1	1	1	1	1	2	2	2	2
1	1	1	1	1	1	1	2	2	2
1	1	1	1	1	1	1	2	2	2
1	1	1	1	1	3	2	2	2	2
1	1	1	1	3	3	3	2	2	2
3	3	3	3	3	3	3	3	2	2
3	3	3	3	3	3	3	3	3	2
3	3	3	3	3	3	3	3	3	2

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

다리 만들기

https://www.acmicpc.net/problem/2146

- 길이: 4

1	1	1	1	1	2	2	2	2	2
1	1	1	1	1	1	2	2	2	2
1	1	1	1	1	1	2	2	2	2
1	1	1	1	1	1	1	2	2	2
1	1	1	1	1	1	1	2	2	2
1	1	1	1	1	3	2	2	2	2
1	1	1	1	3	3	3	2	2	2
3	3	3	3	3	3	3	3	2	2
3	3	3	3	3	3	3	3	3	2
3	3	3	3	3	3	3	3	3	2

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

다리 만들기

https://www.acmicpc.net/problem/2146

- 길이: 2+1 = 3

1	1	1	1	1	2	2	2	2	2
1	1	1	1	1	1	2	2	2	2
1	1	1	1	1	1	2	2	2	2
1	1	1	1	1	1	1	2	2	2
1	1	1	1	1	1	1	2	2	2
1	1	1	1	1	3	2	2	2	2
1	1	1	1	3	3	3	2	2	2
3	3	3	3	3	3	3	3	2	2
3	3	3	3	3	3	3	3	3	2
3	3	3	3	3	3	3	3	3	2

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

다리 만들기

https://www.acmicpc.net/problem/2146

- 소스: http://boj.kr/c1fbe6fae9ae47ffa9020664581f8f3f