그래프와 BFS

최백준 choi@startlink.io

그래프

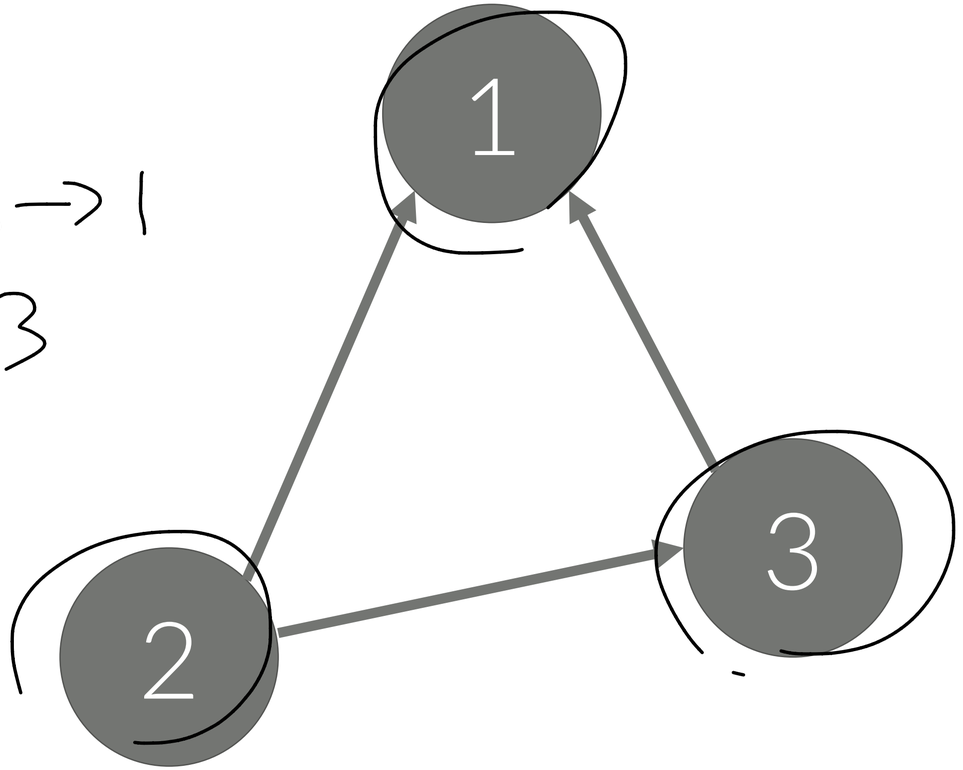
그래프

Graph

- 자료구조의 일종

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

2 → 1 3 → 1
2 → 3

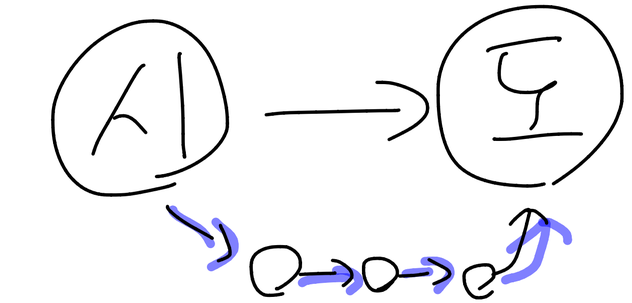


- G = (V, E)로 나타낸다.

경로

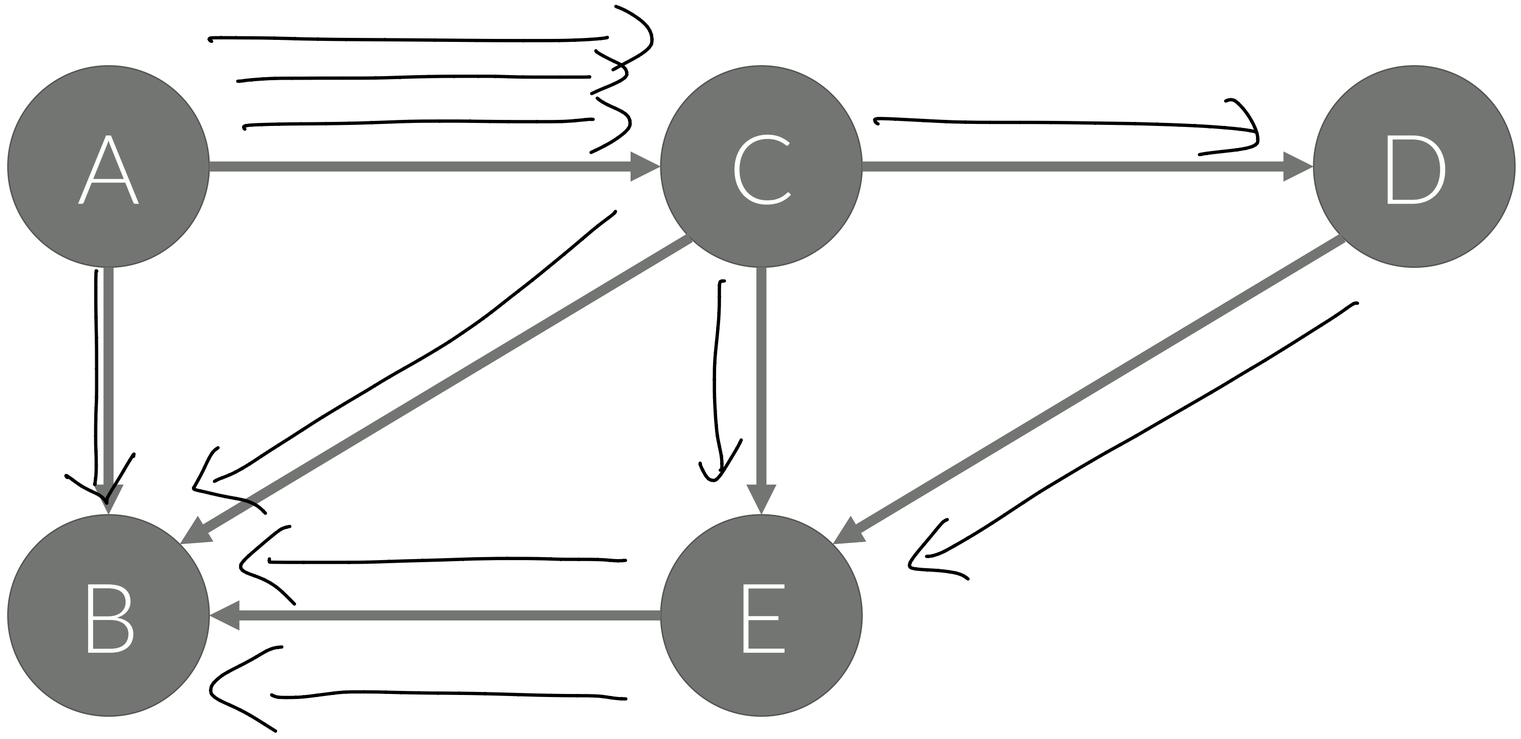
Path

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



최단경로

A → B



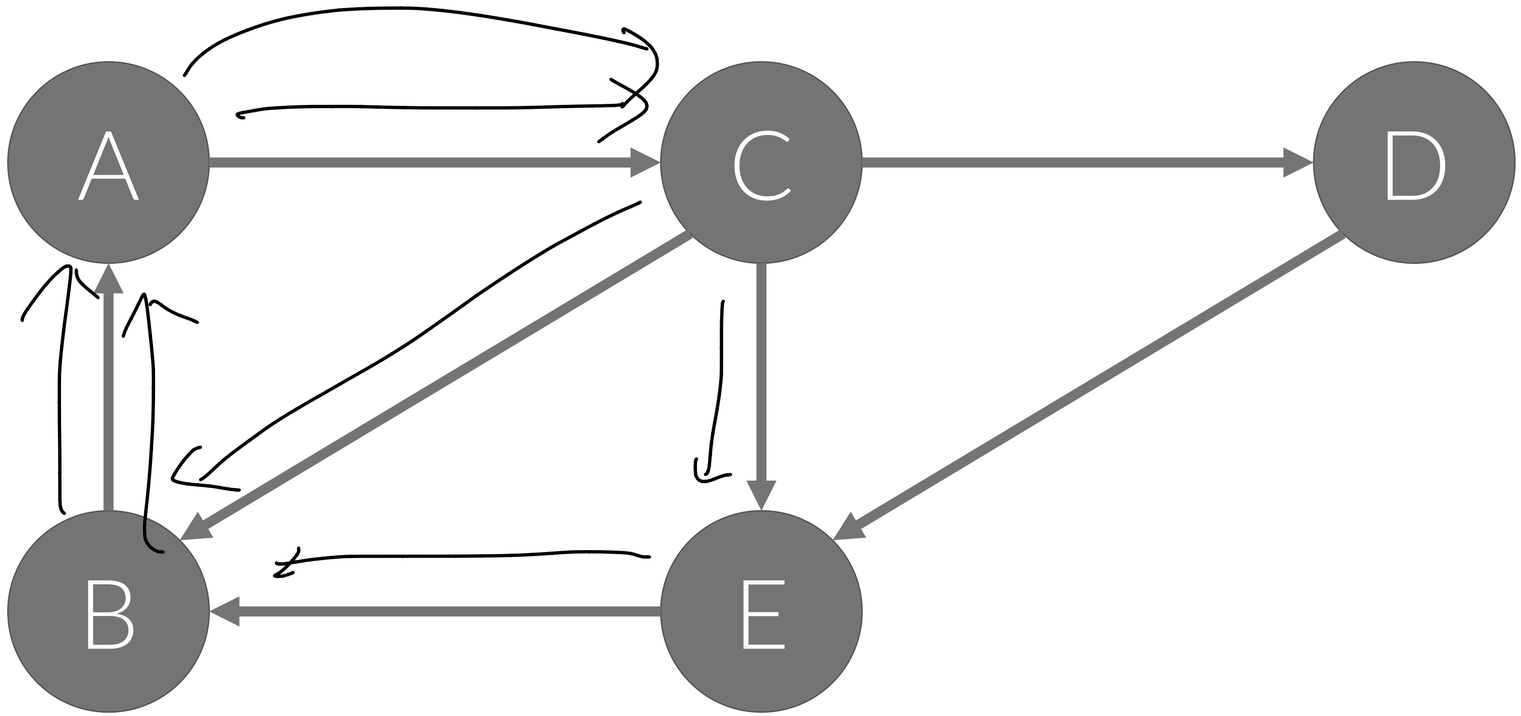
사이클

Path

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

경로 : A → A

A → A 같음



단순 경로와 단순 사이클

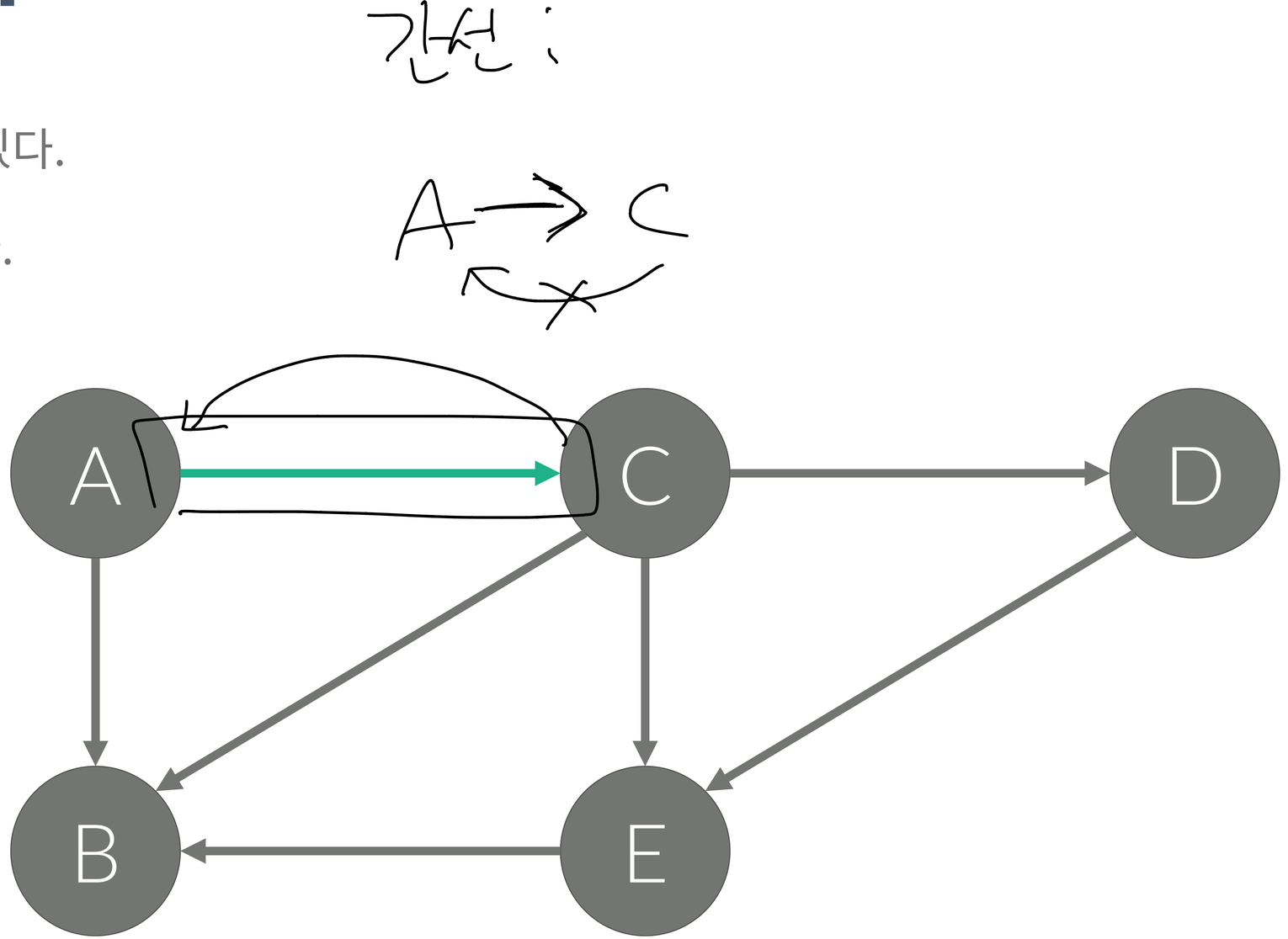
Simple Path and Simple Cycle

- 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

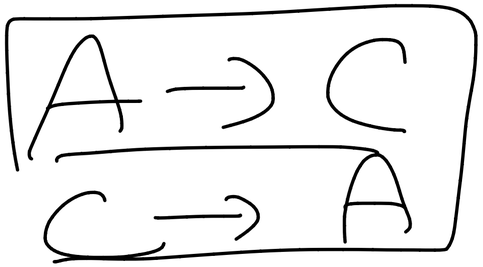
Directed Graph

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

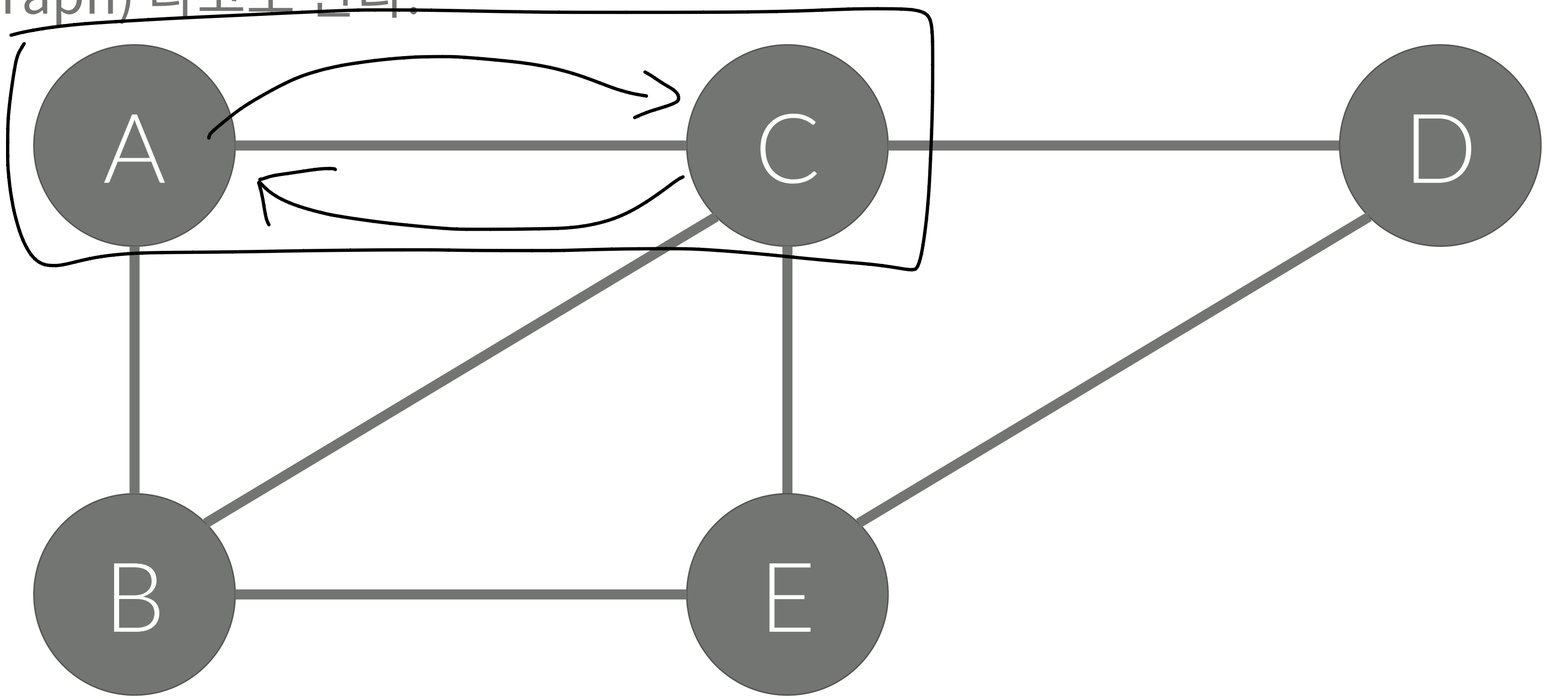


방향 없는 그래프

Undirected Graph



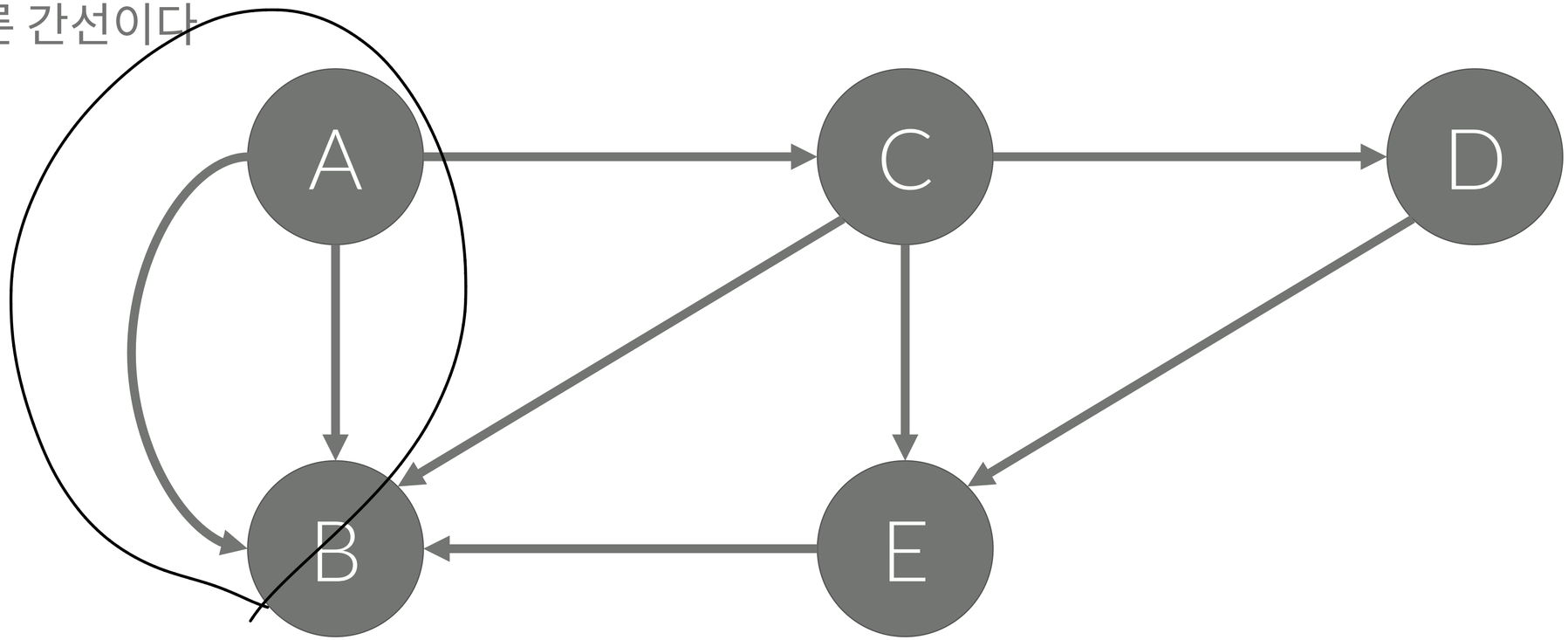
- A – C 와 같이 간선에 방향이 없다.
- A – C는 A → C와 C → A를 나타낸다.
- 양방향 그래프 (Bidirection Graph) 라고도 한다.



간선 여러개

Multiple Edge

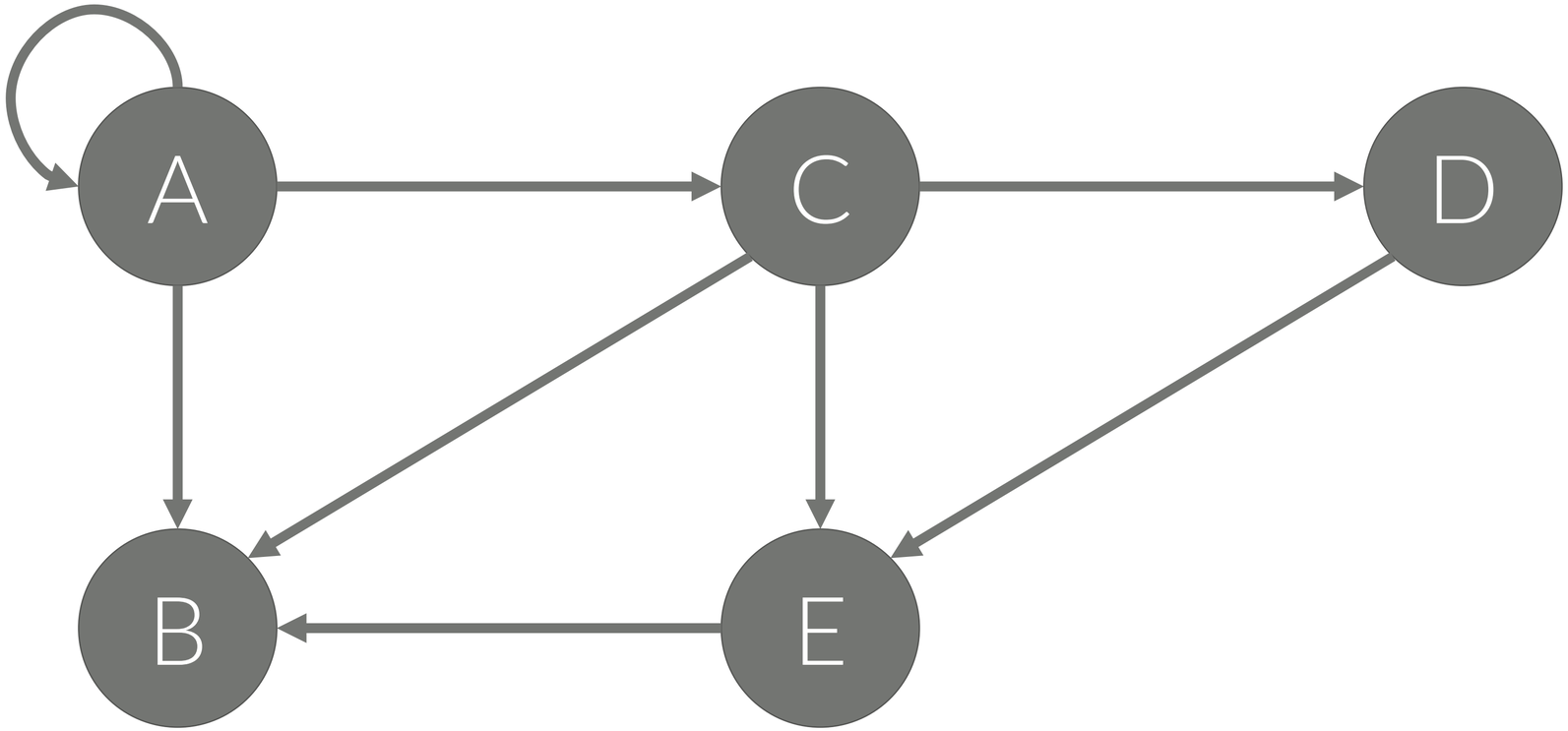
- 두 정점 사이에 간선이 여러 개일 수도 있다.
- 아래 그림의 A-B는 연결하는 간선이 2개이다.
- 두 간선은 서로 다른 간선이다



루프

Loop

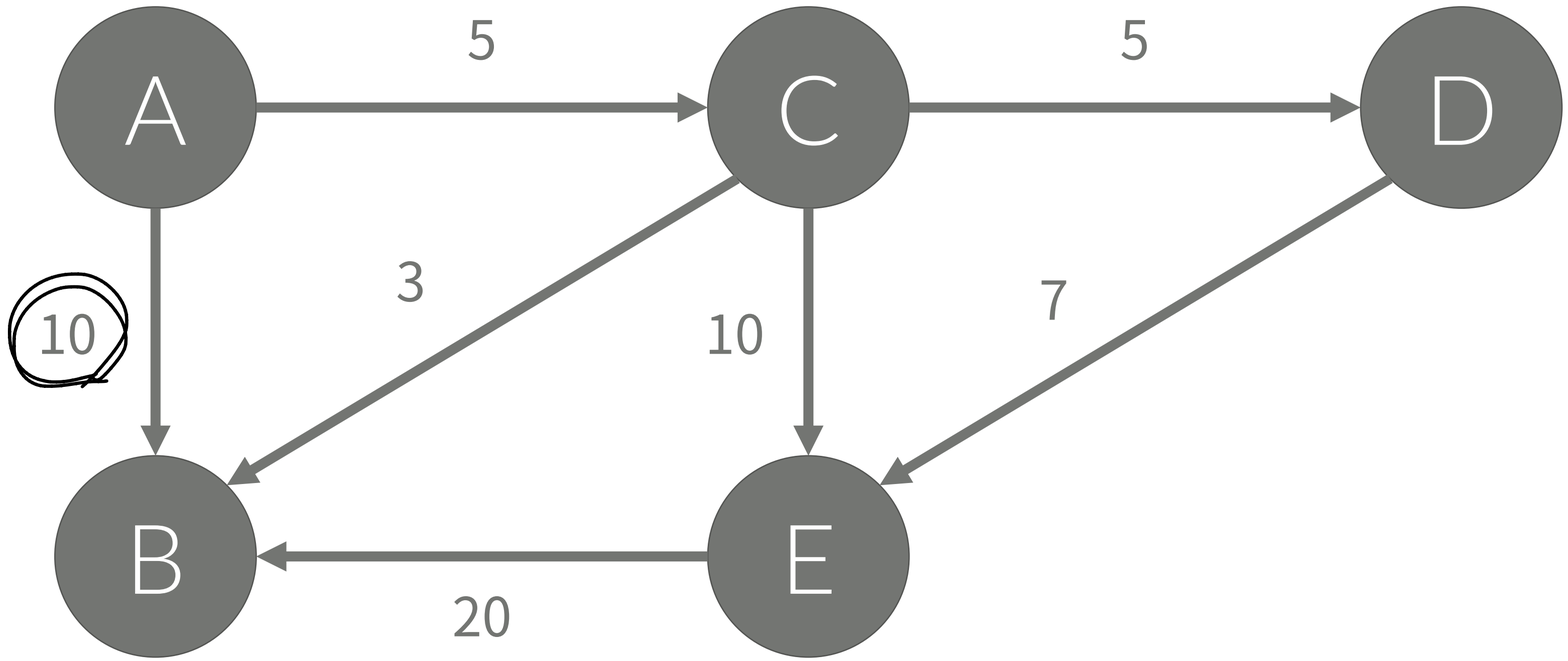
- 간선의 양 끝 점이 같은 경우가 있다.
- A → A



가중치

Weight

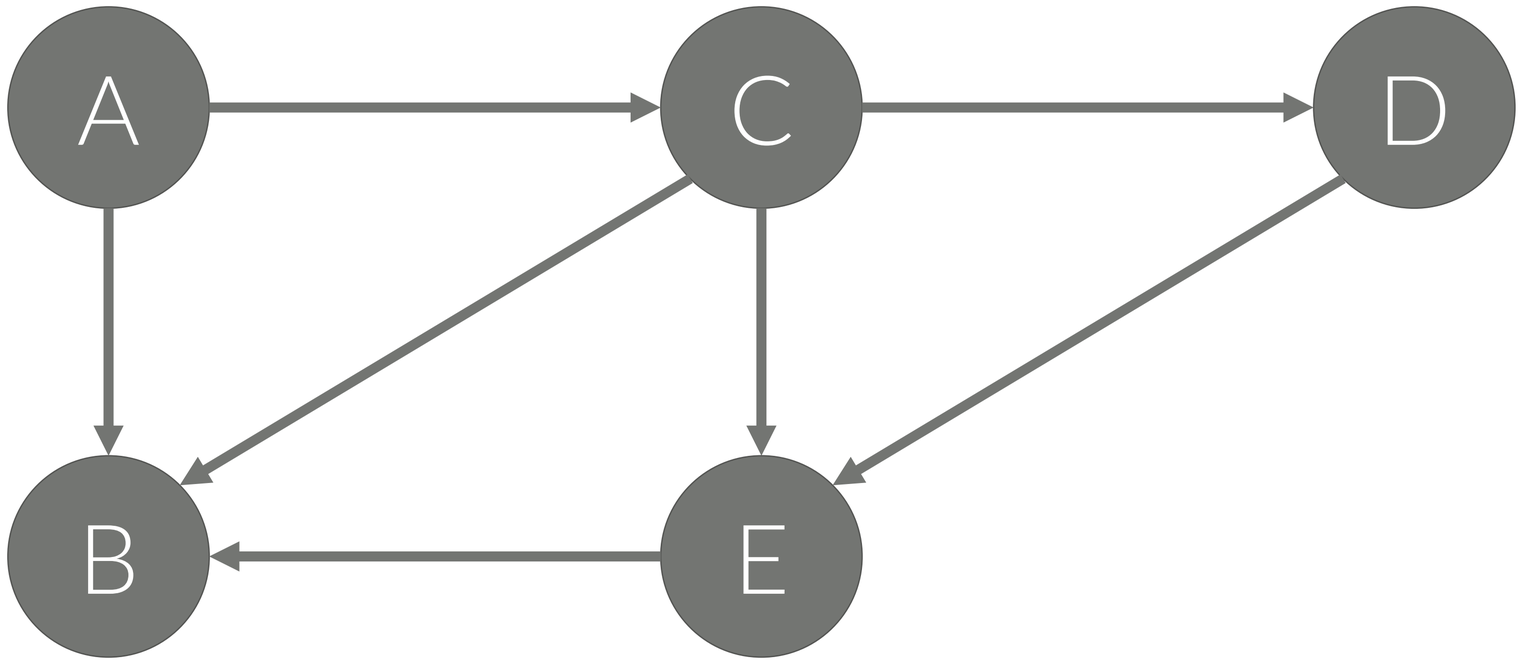
- 간선에 가중치가 있는 경우에는
- A에서 B로 이동하는 거리, 이동하는데 필요한 시간, 이동하는데 필요한 비용 등등등…



가중치

Weight

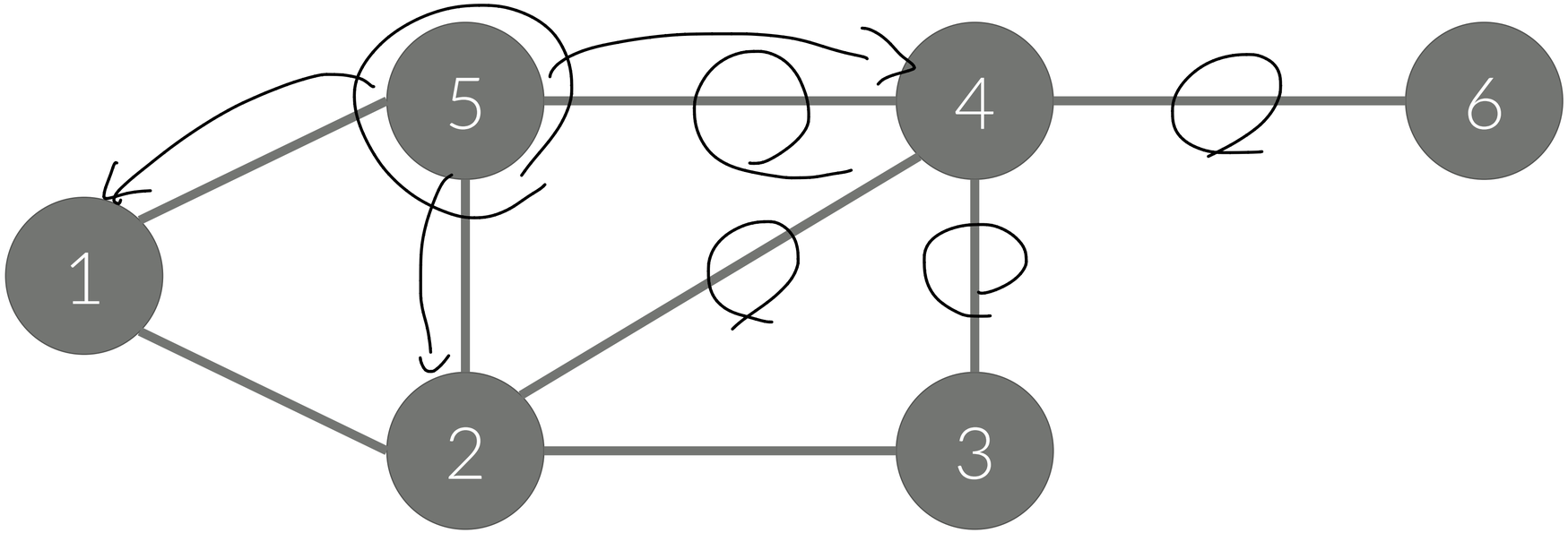
- 가중치가 없는 경우에는 1이라고 생각하면 된다



차수

Degree

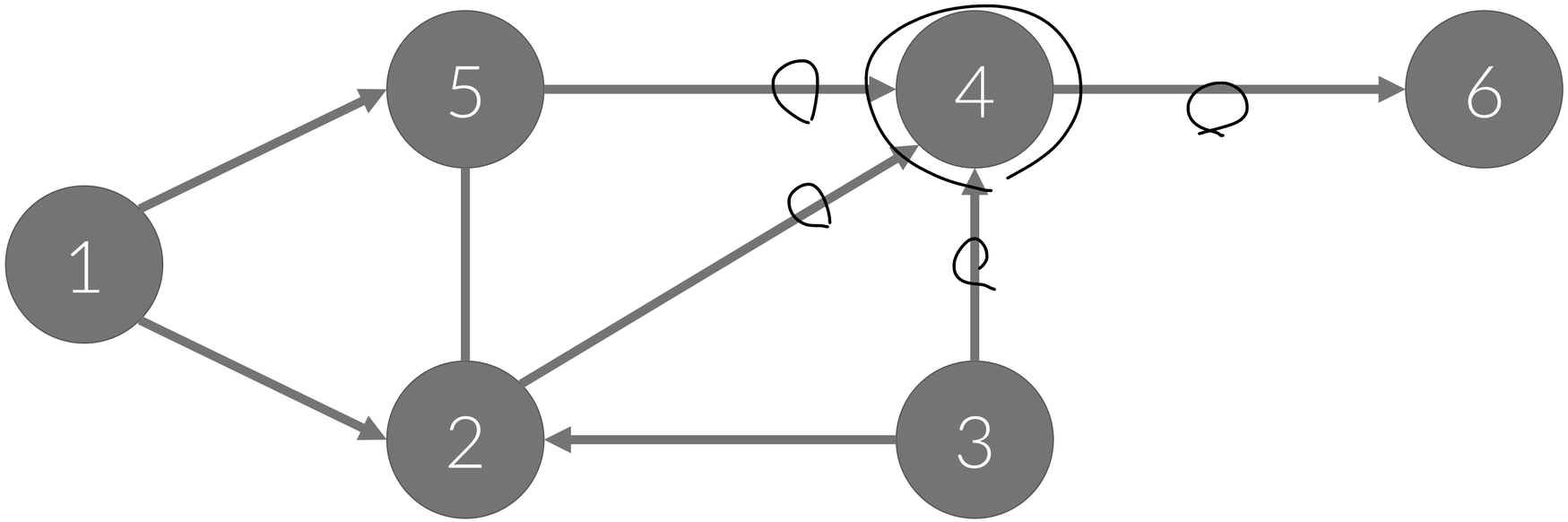
- 정점과 연결되어 있는 간선의 개수
- 5의 차수: 3
- 4이 차수: 4



차수

Degree

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

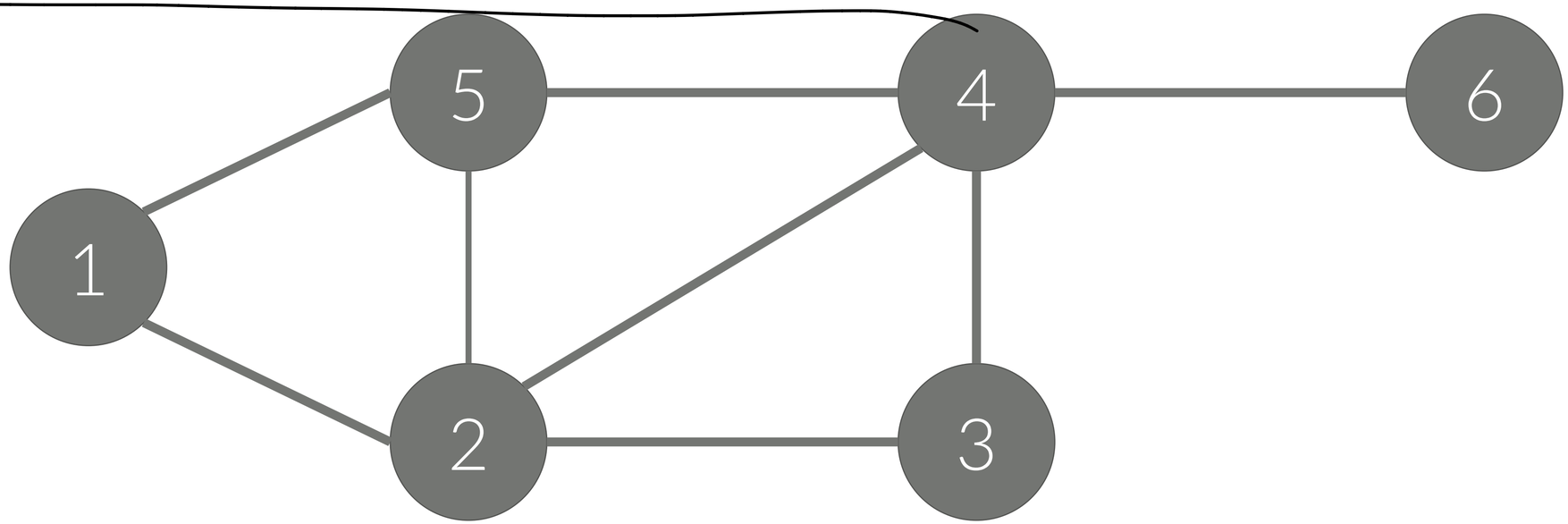


그래프의 표현

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)}

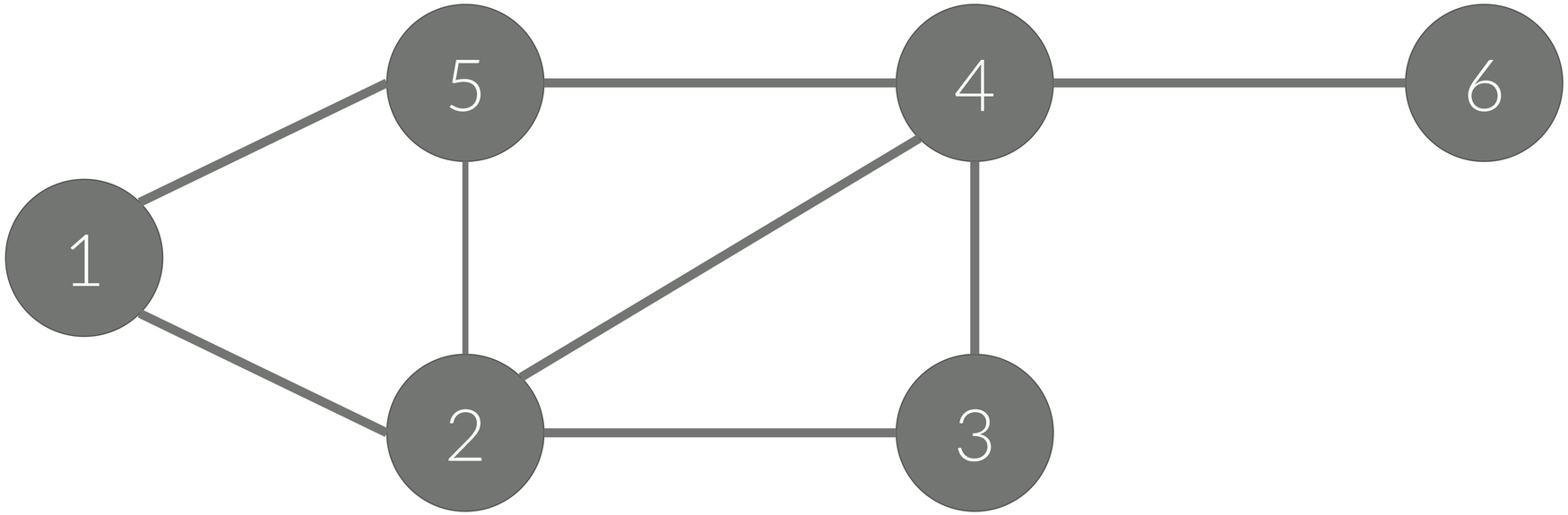


인접 행렬

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)

A[i][j] = 1 i -> j 있을 때
= 0 없을 때



인접 행렬

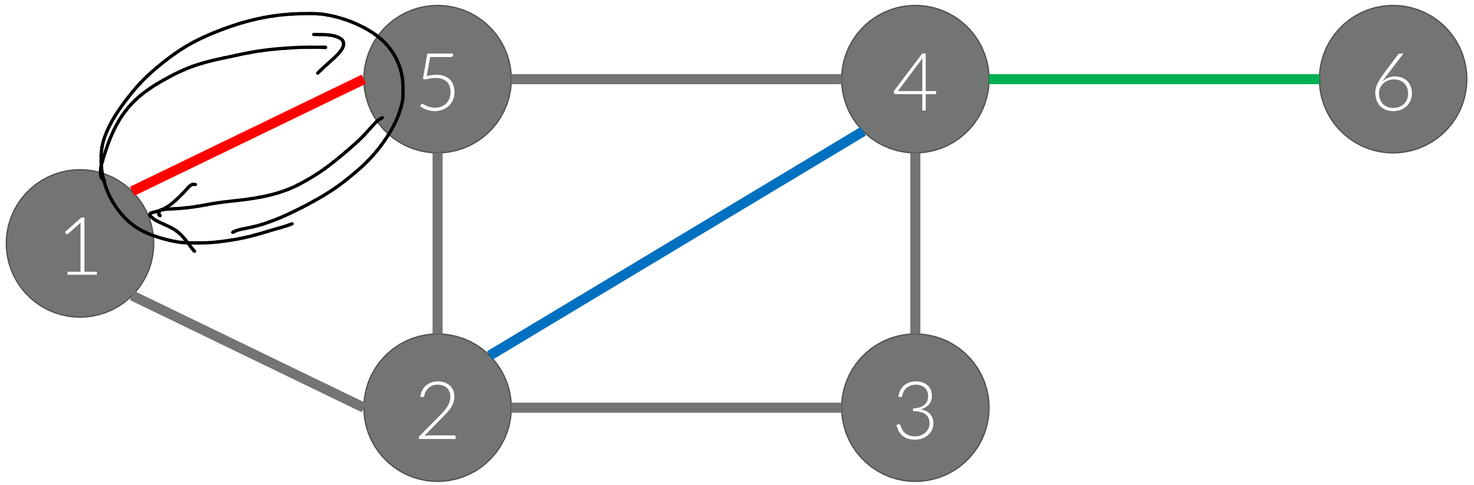
Adjacency-matrix

A

	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[1][5]

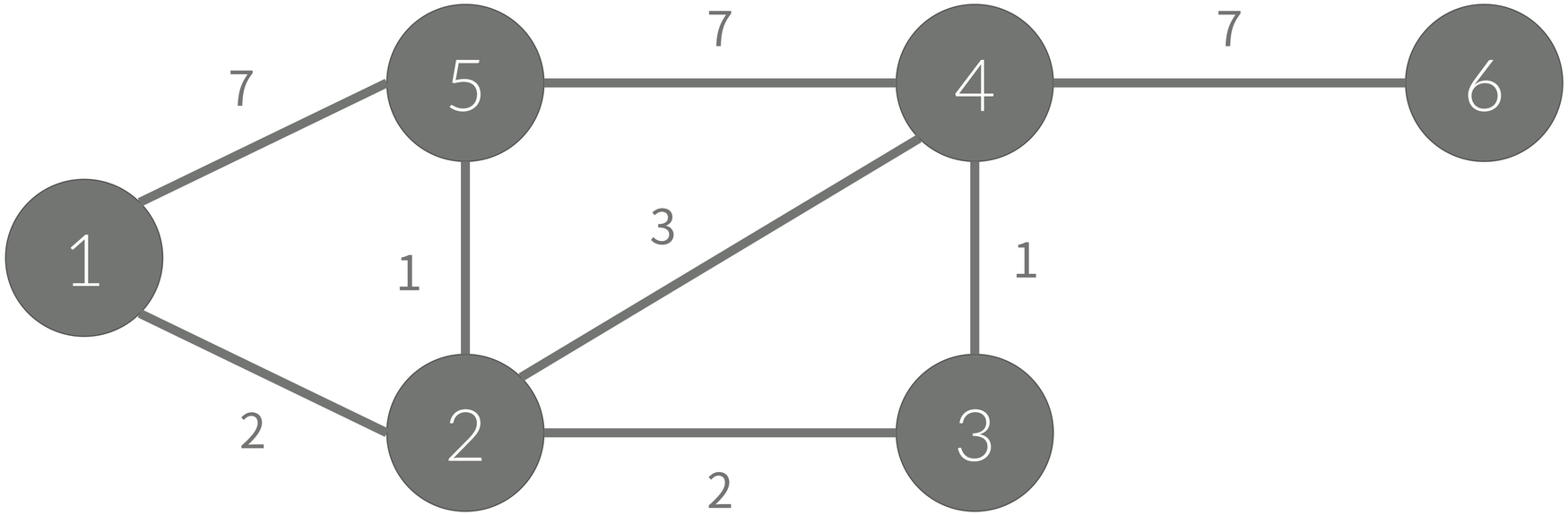
A[5][1]



인접 행렬

Adjacency-matrix

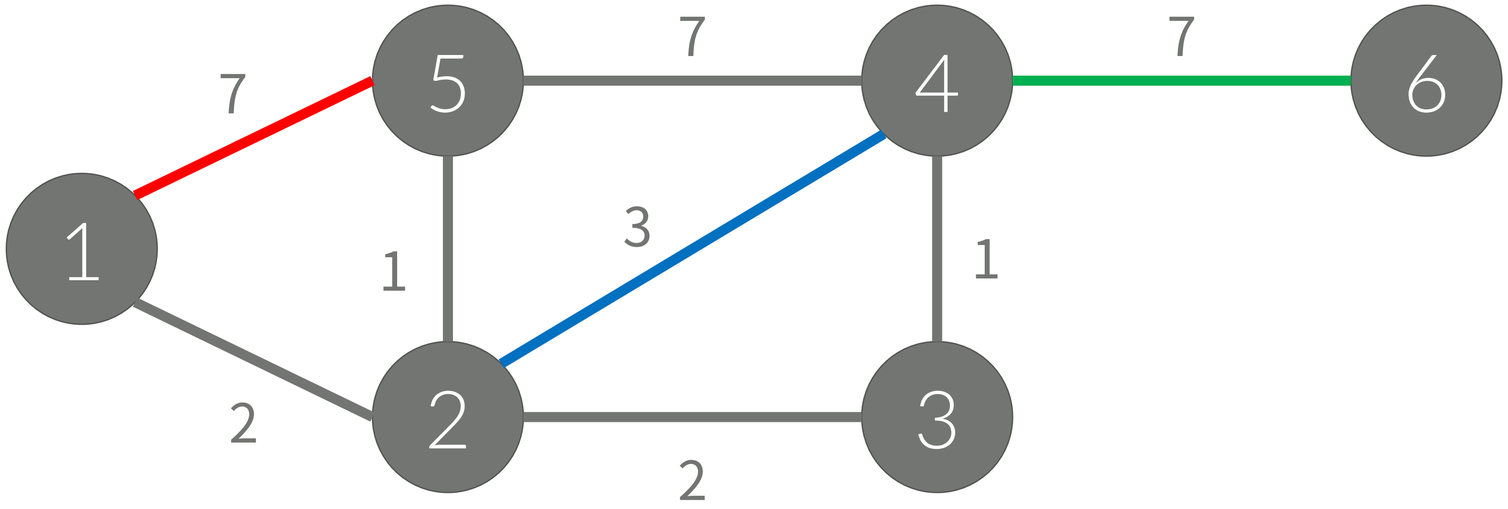
- 정점의 개수를 N이라고 했을 때
- N×N 크기의 이차원 배열을 이용한다
- A[i][j] = w (i -> j 간선이 있을 때, 그 가중치), 0 (없을 때)



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

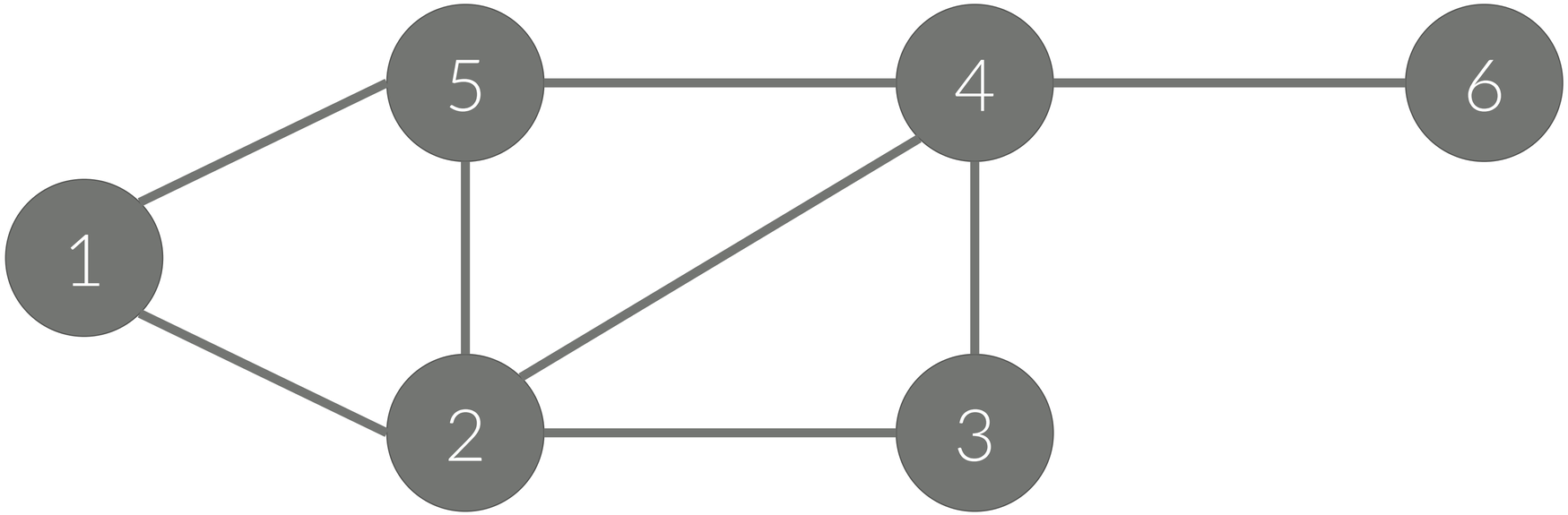


인접 리스트

Adjacency-list

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

간선



인접 리스트

Adjacency-list

A[1] 2 5

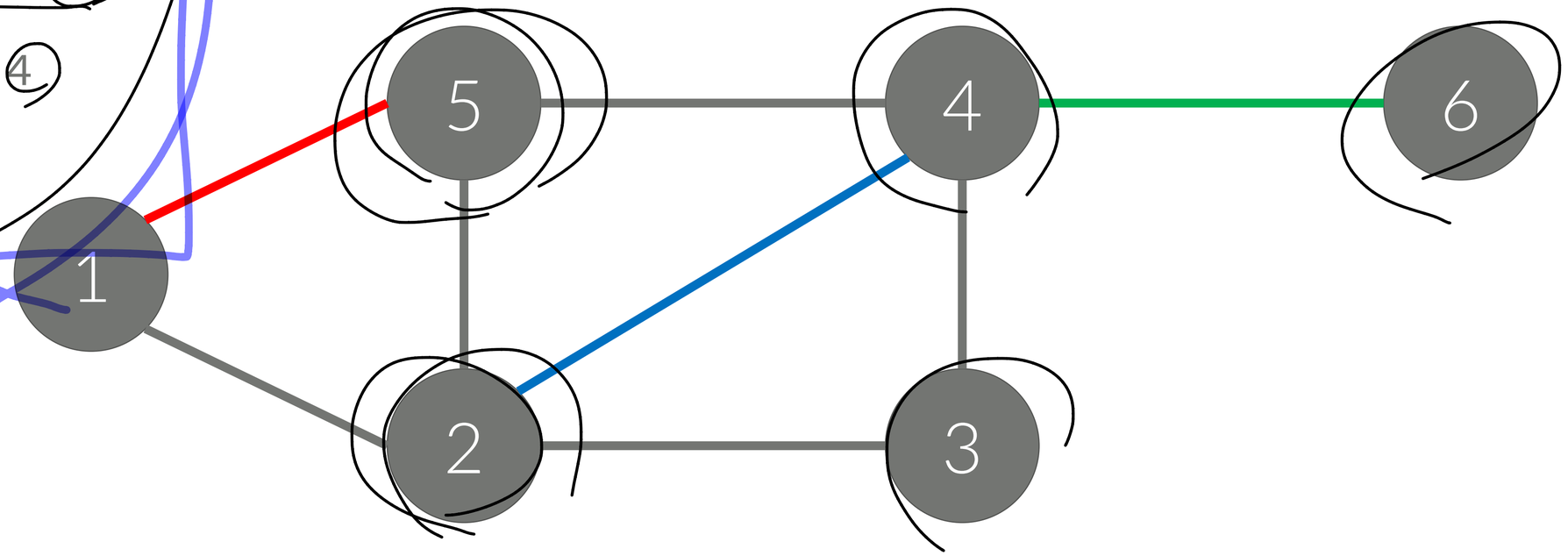
A[2] 1 3 4 5

A[3] 2 4

A[4] 3 5 2 6

A[5] 1 2 4

A[6] 4



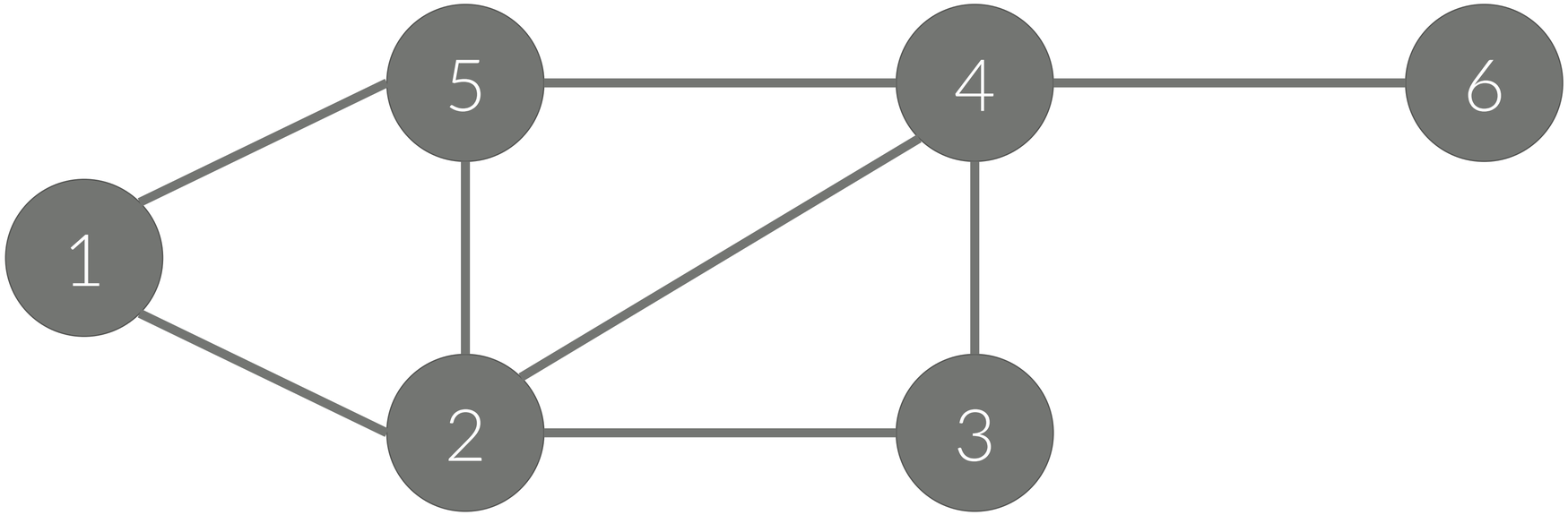
인접 리스트

Adjacency-list

vector
ArrayList

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

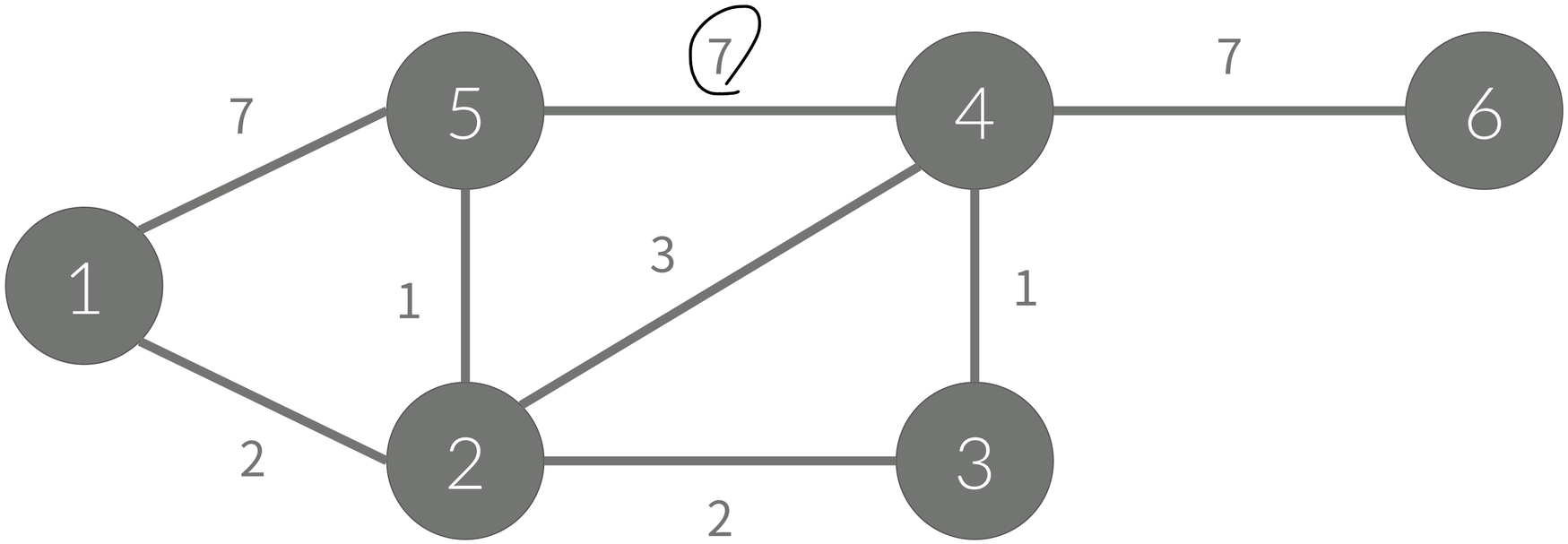
Python



인접 리스트

Adjacency-list

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



인접 리스트

Adjacency-list

X와 연결된 모든 간선 ←

인접행렬 O(V)

A[x][1] ~A[x][V]

리스트 : O(차수)

A[1] (2,2) (5,7)

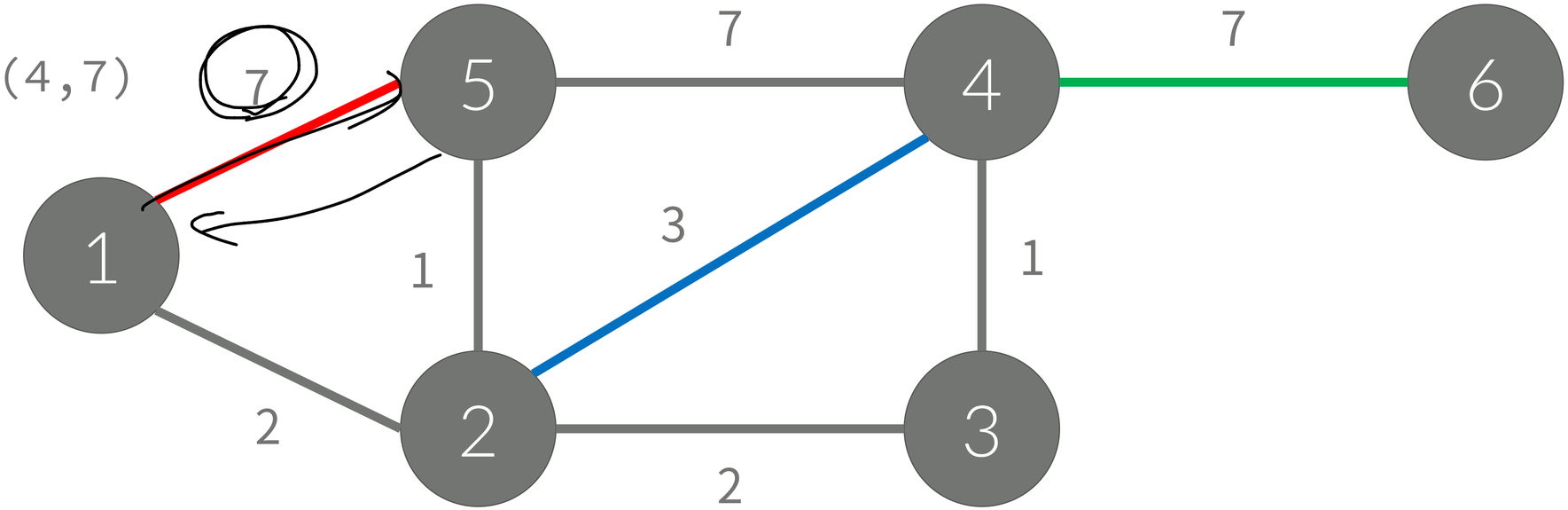
A[2] (1,2) (3,2) (4,3) (5,1)

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

- 인접 행렬: O(V^2)
- 인접 리스트: O(E)

완전 그래프

1 ≤ E ≤ 10만
1 ≤ V ≤ 1000

$E \ll V^2$

V ≤ 1000

$1000^2 = 12$
× 4

① u, v

u → v 존재

O(1)

A[u][v]

A[u]

O(차수)

② u, v , v, u

O(1)

A[v][u]

1 — 2
| X |
4 — 3

$E = \frac{V(V-1)}{2}$

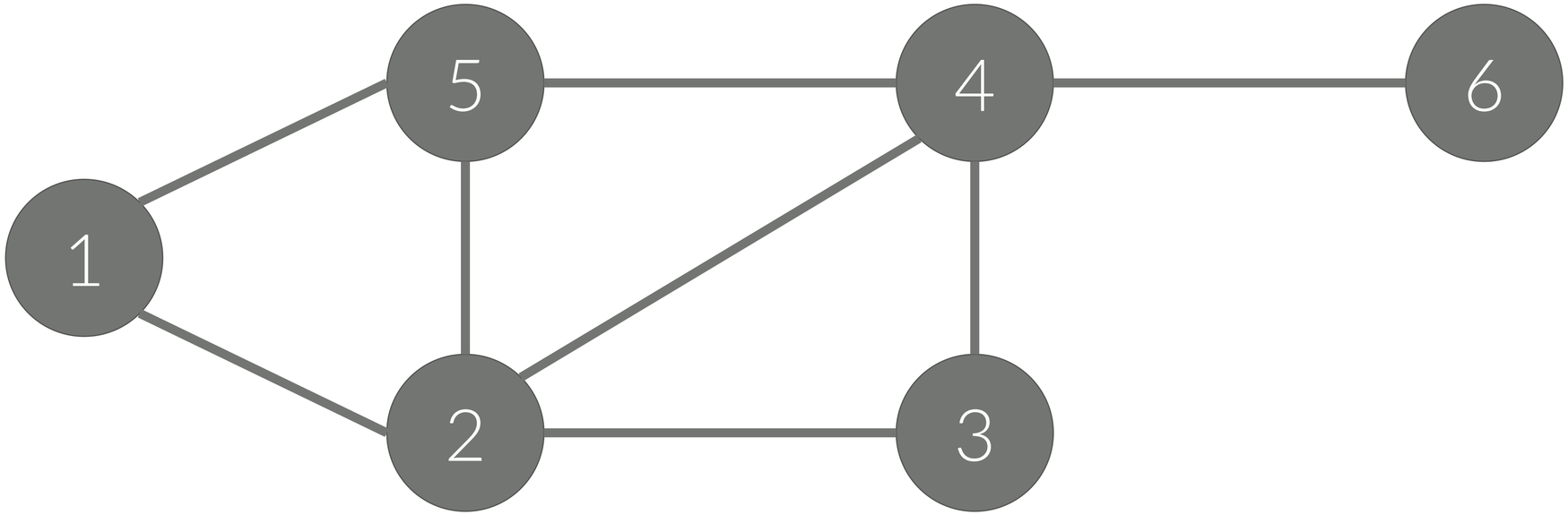
A[v]

간선 리스트

Edge-list

vector, ArrayList

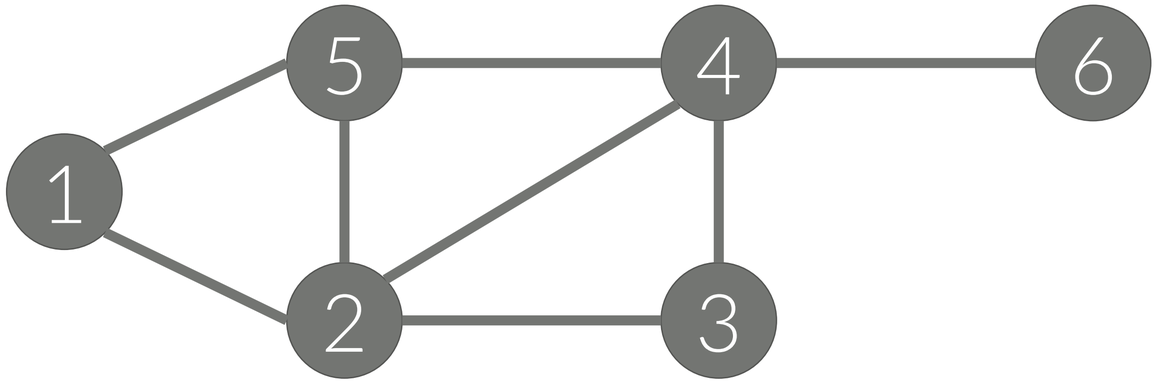
- 배열을 이용해서 구현한다.
- 간선을 모두 저장하고 있다.



간선 리스트

Edge List

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

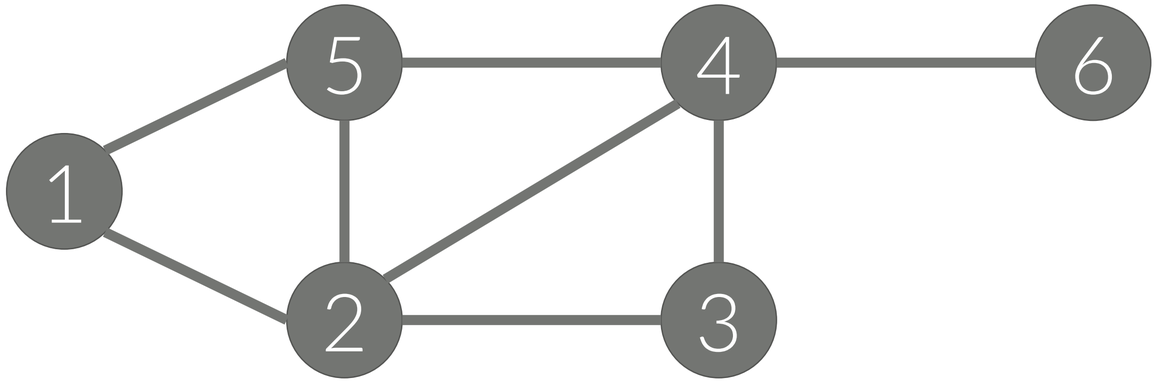



```
E[0] = 1 2
E[1] = 1 5
E[2] = 2 3
E[3] = 2 4
E[4] = 2 5
E[5] = 5 4
E[6] = 4 3
E[7] = 4 6
E[8] = 2 1
E[9] = 5 1
E[10] = 3 2
E[11] = 4 2
E[12] = 5 2
E[13] = 4 5
E[14] = 3 4
E[15] = 6 4
```

간선 리스트

Edge List

- 각 간선의 앞 정점을 기준으로 개수를 센다.



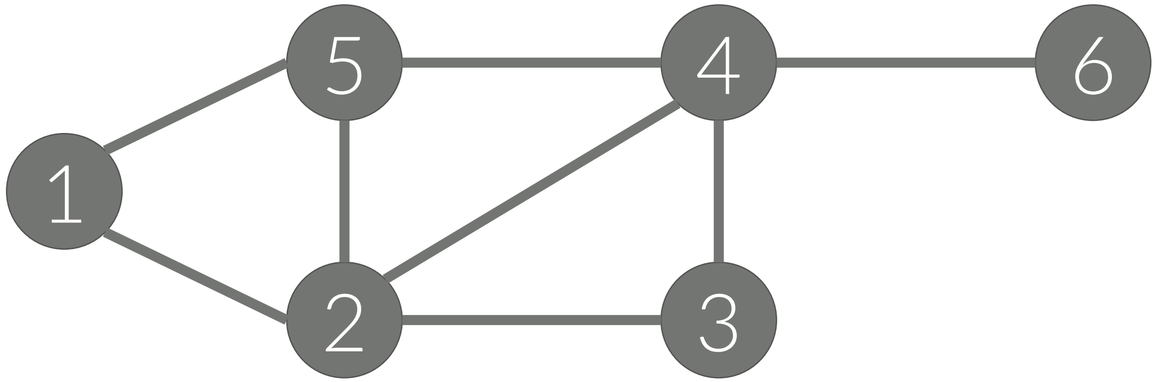
i	0	1	2	3	4	5	6
cnt[i]	0	2	4	2	4	3	1

```
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;
}
```



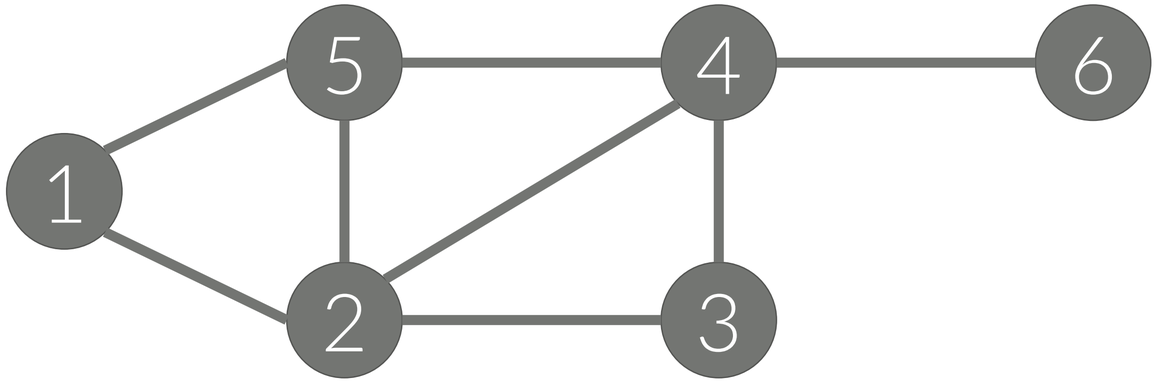
i	0	1	2	3	4	5	6
cnt[i]	0	2	4	2	4	3	1
	0	2	6	8	12	15	16

```
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

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



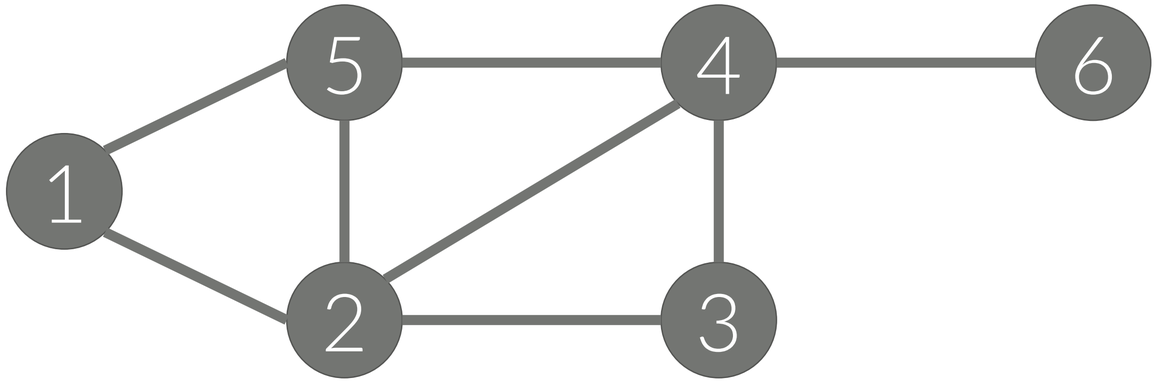
i	0	1	2	3	4	5	6
cnt[i]	0	2	6	8	12	15	16

```
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.



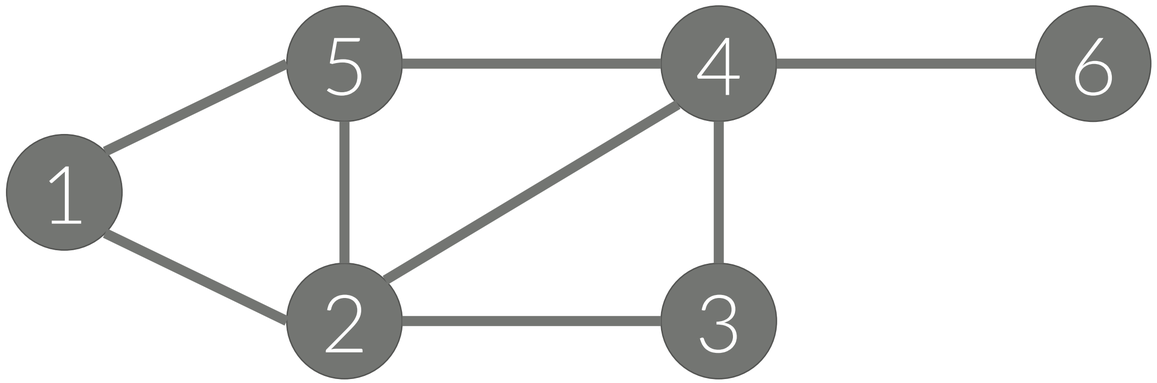
i	0	1	2	3	4	5	6
cnt[i]	0	2	6	8	12	15	16

```
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



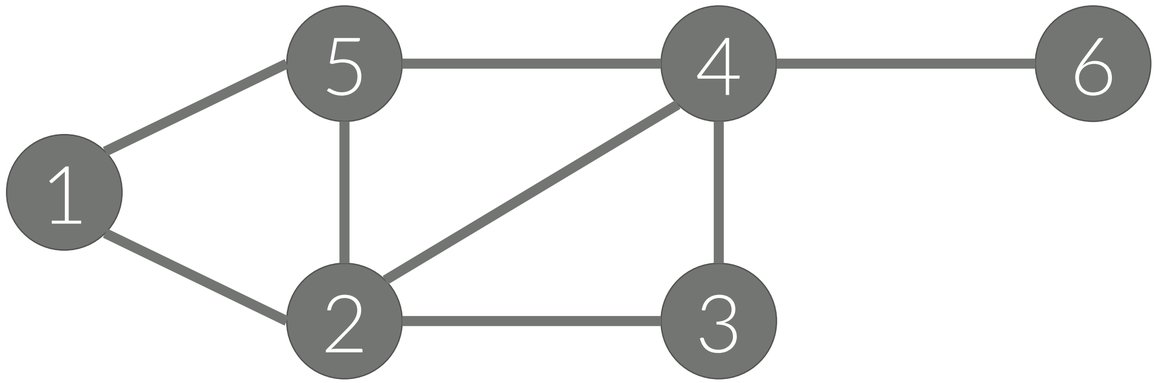
i	0	1	2	3	4	5	6
cnt[i]	0	2	6	8	12	15	16

```
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://codeplus.codes/565e0c8547224a5bae36876a3a0b678e

DFS / BFS

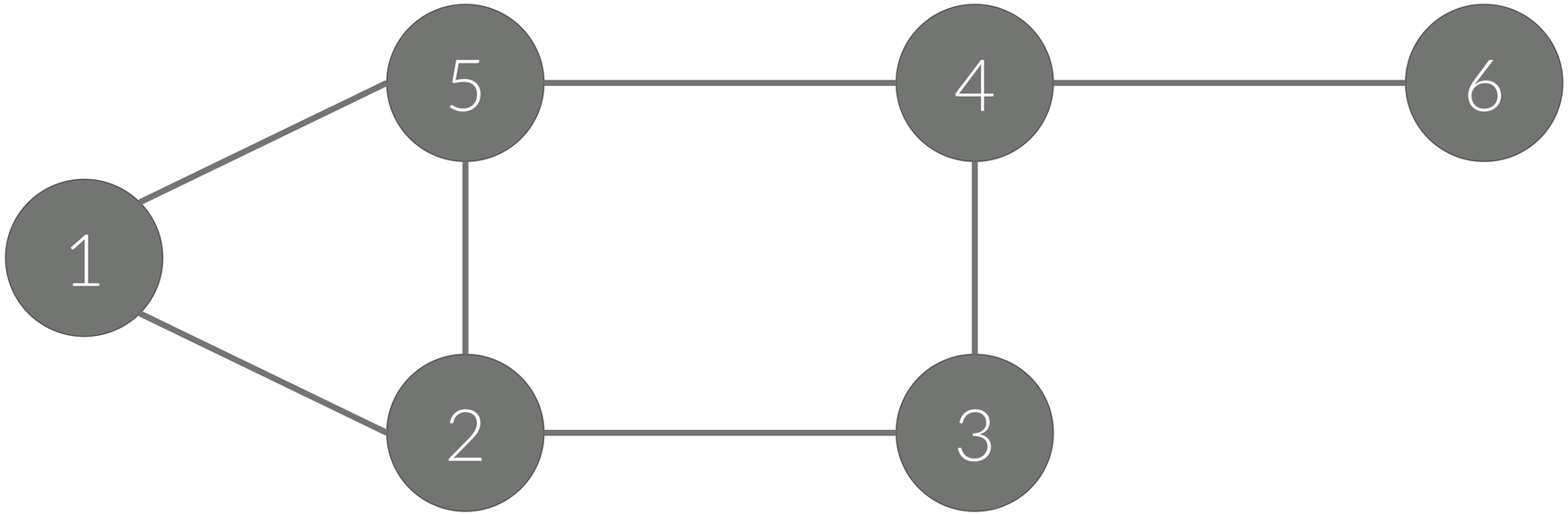
목적: 시작점 X 시작해서
모든 정점을 1번씩

그래프의 탐색

그래프의 탐색

DFS, BFS

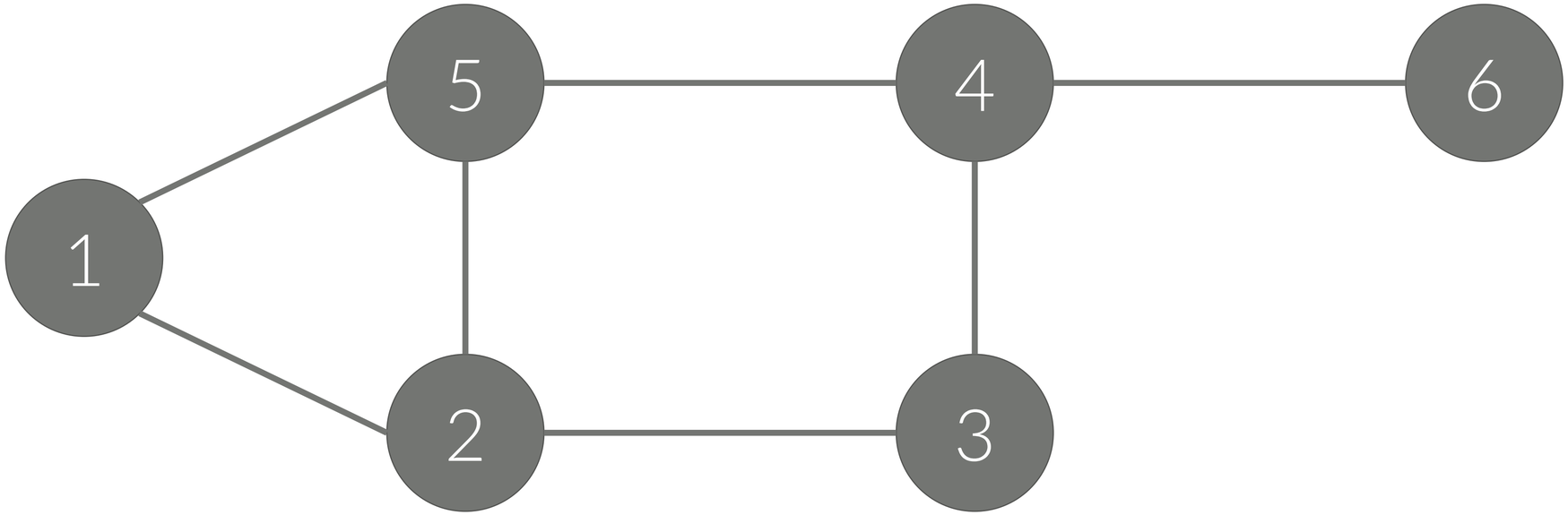
- DFS: 깊이 우선 탐색
- BFS: 너비 우선 탐색



깊이 우선 탐색

Depth First Search

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

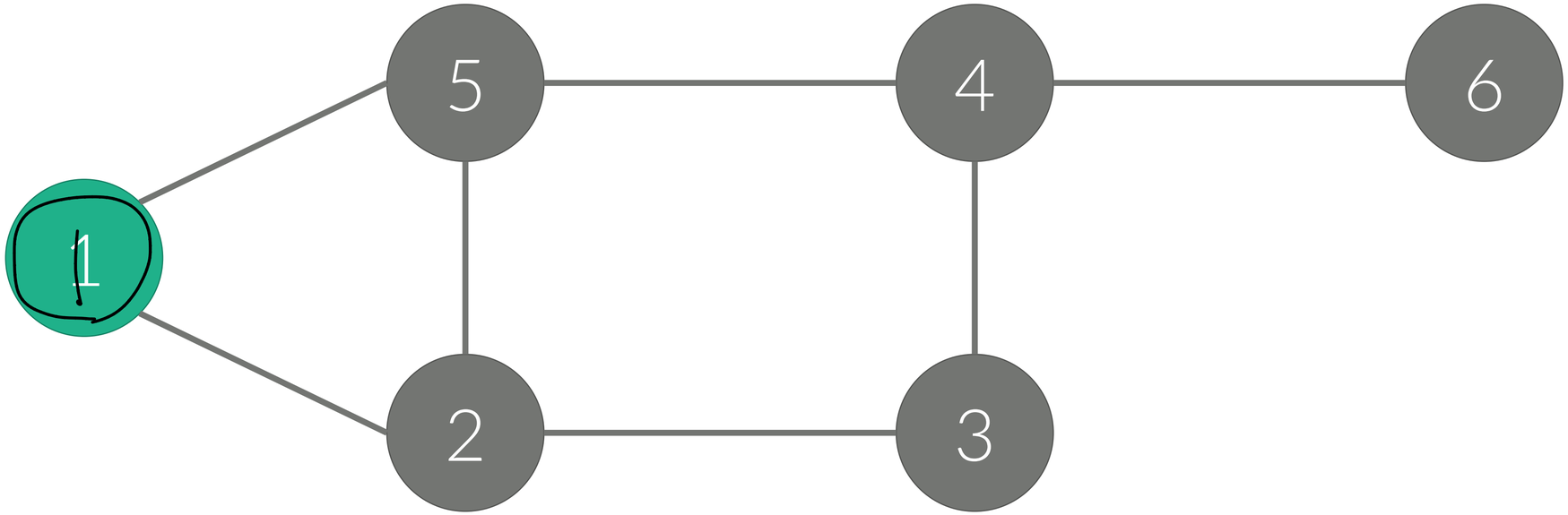


깊이 우선 탐색

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

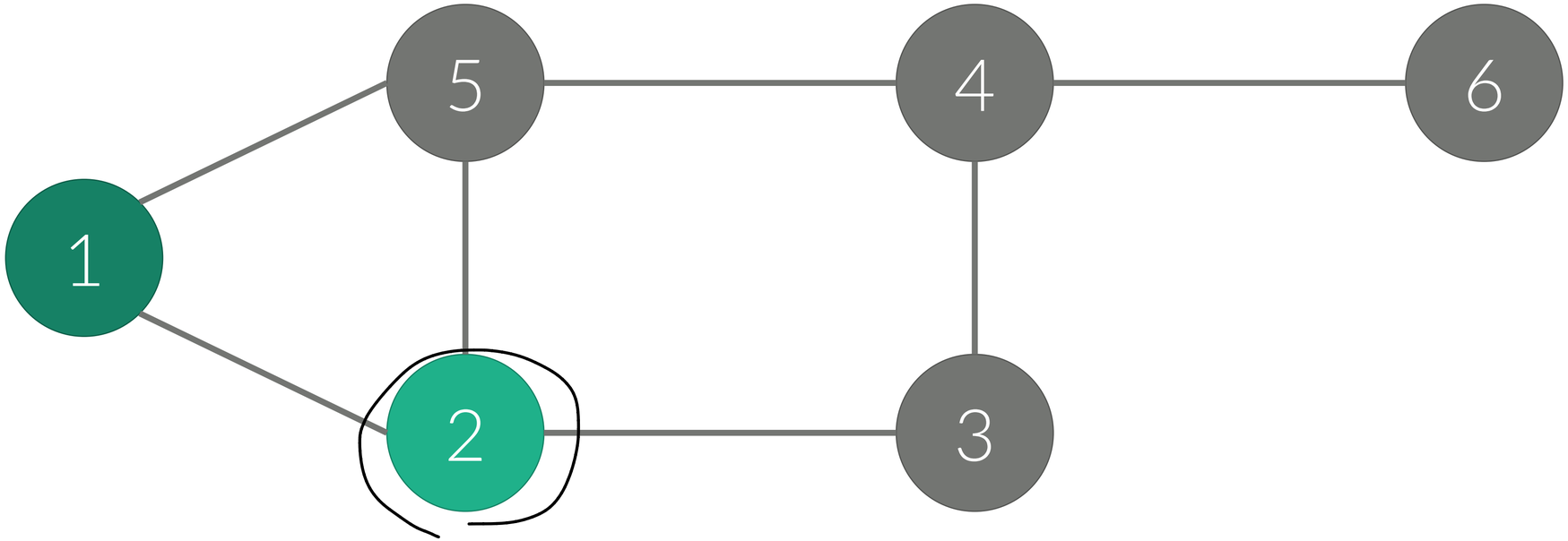


깊이 우선 탐색

Depth First Search

- 현재 정점: 2
- 순서: 1 2
- 스택: 1 2

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

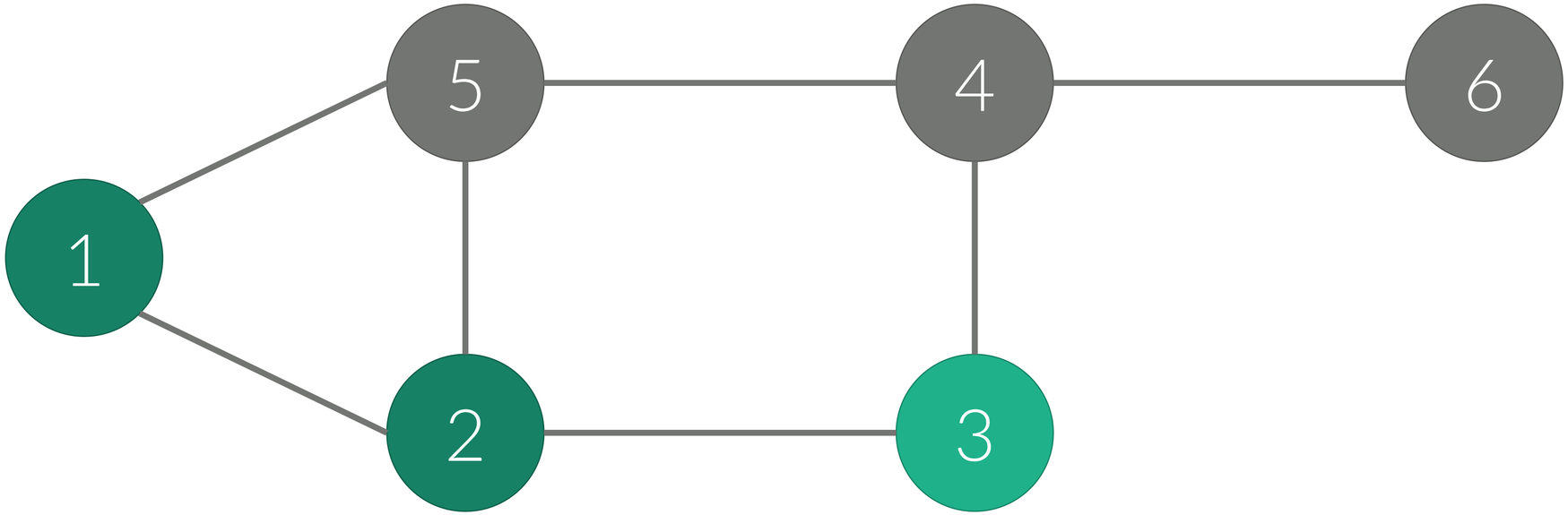


깊이 우선 탐색

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

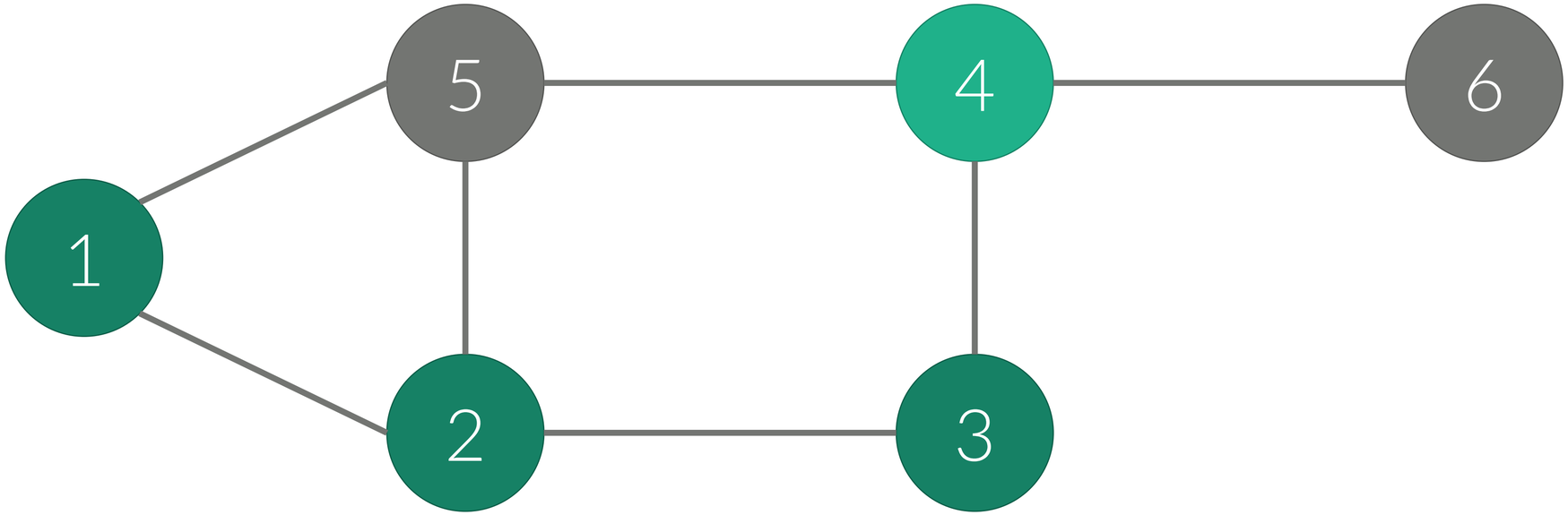


깊이 우선 탐색

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

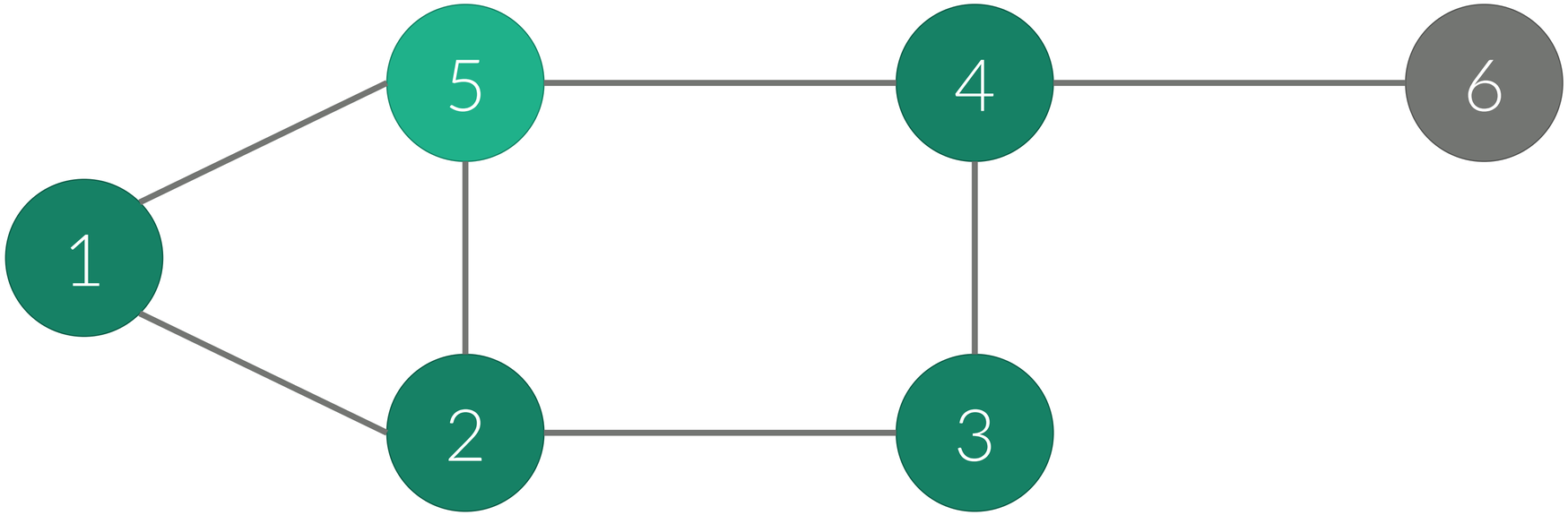


깊이 우선 탐색

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

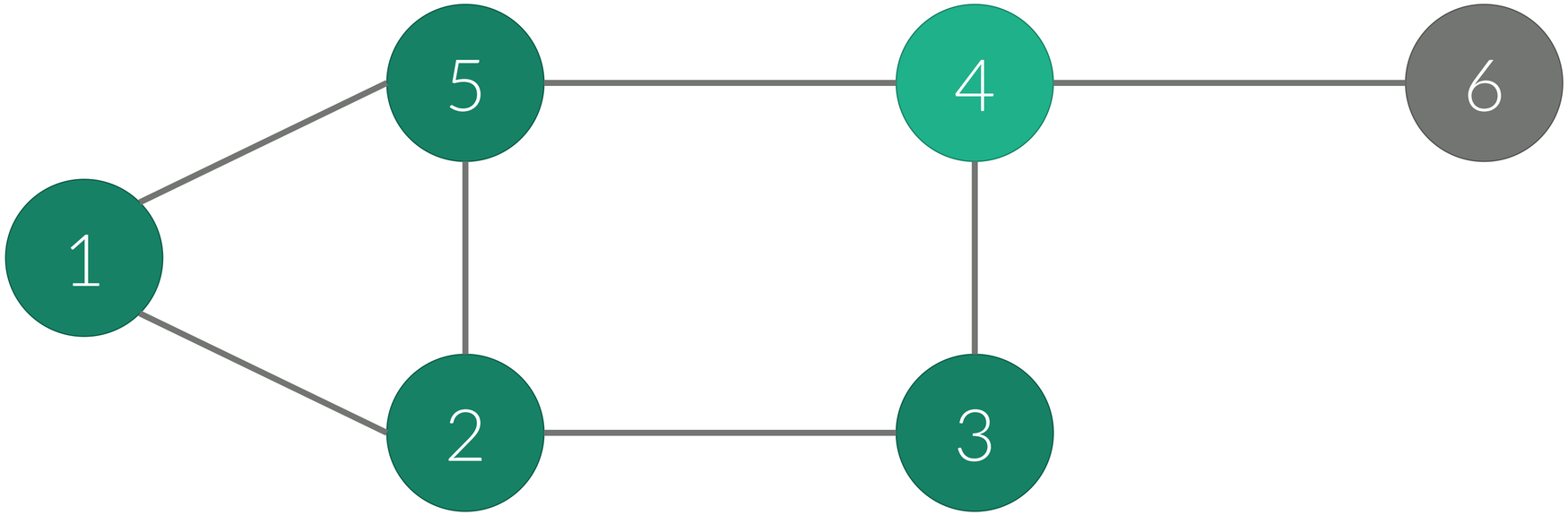


깊이 우선 탐색

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

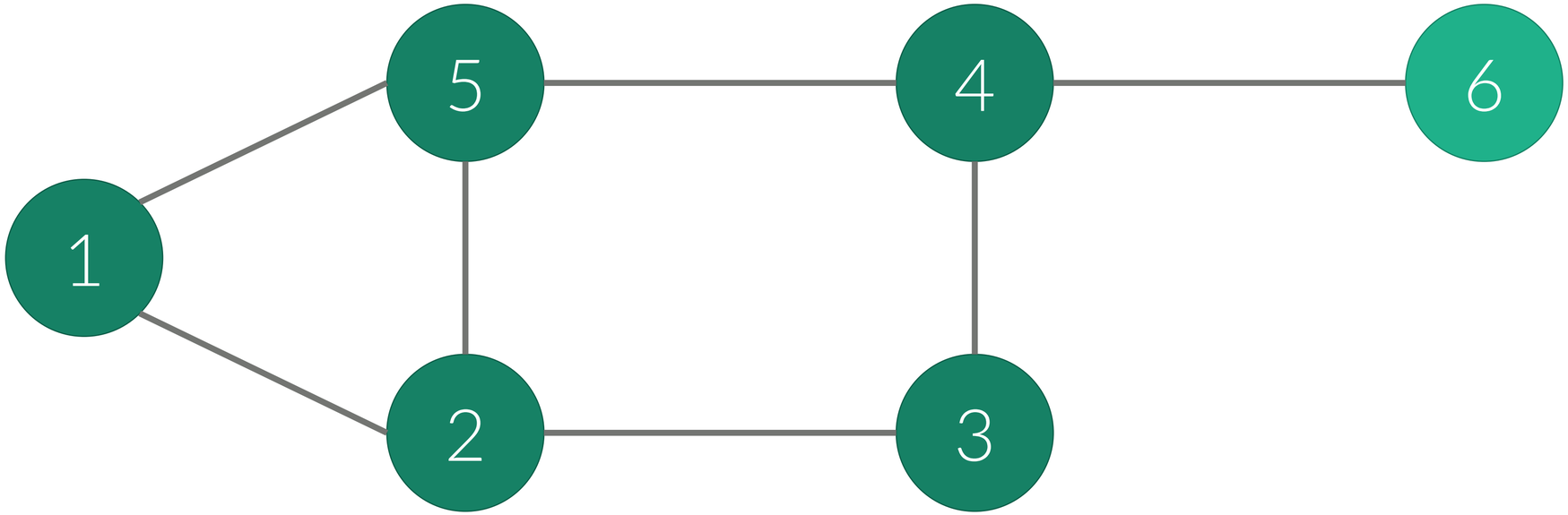


깊이 우선 탐색

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

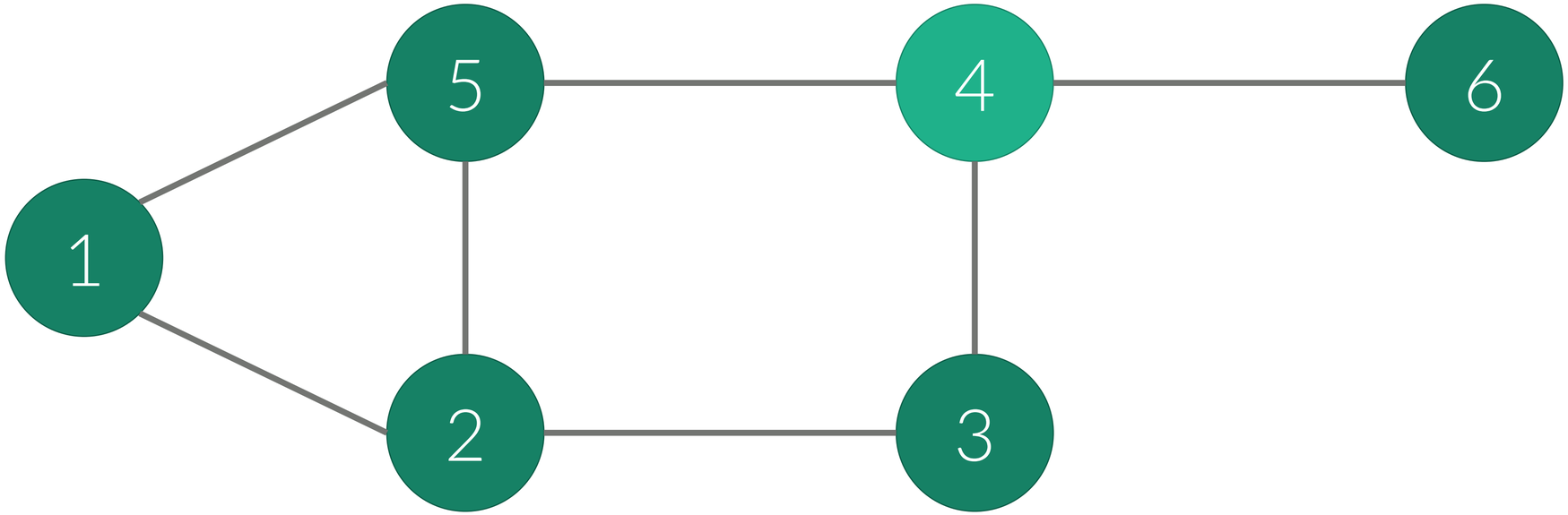


깊이 우선 탐색

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

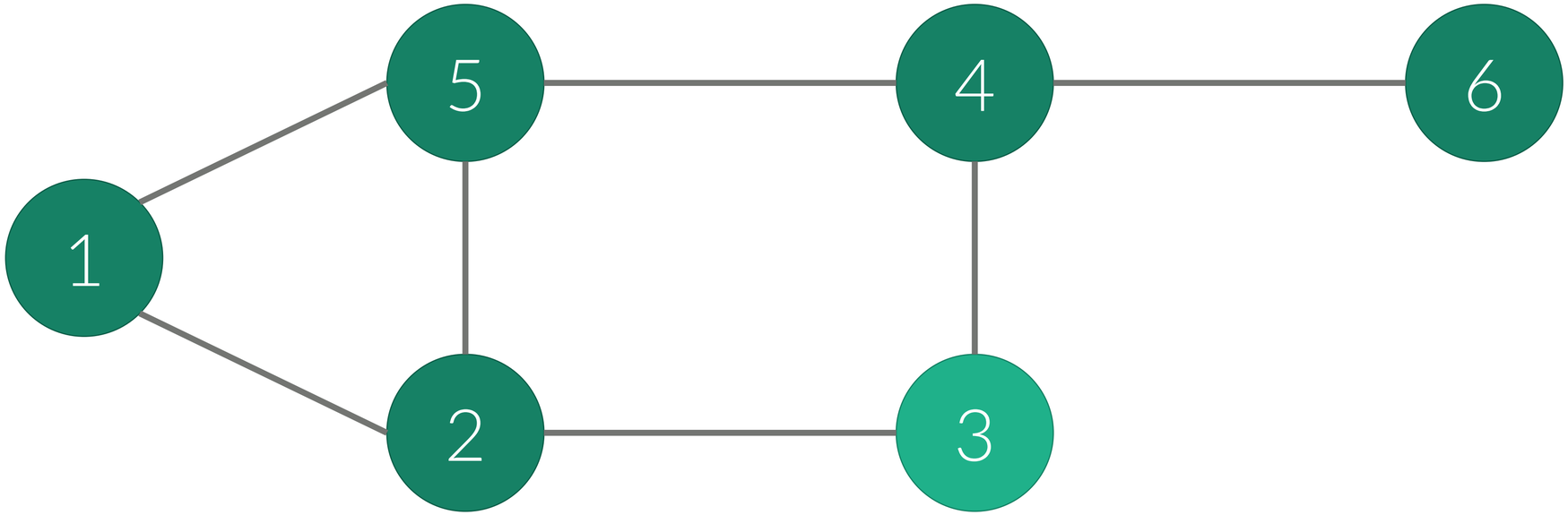


깊이 우선 탐색

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

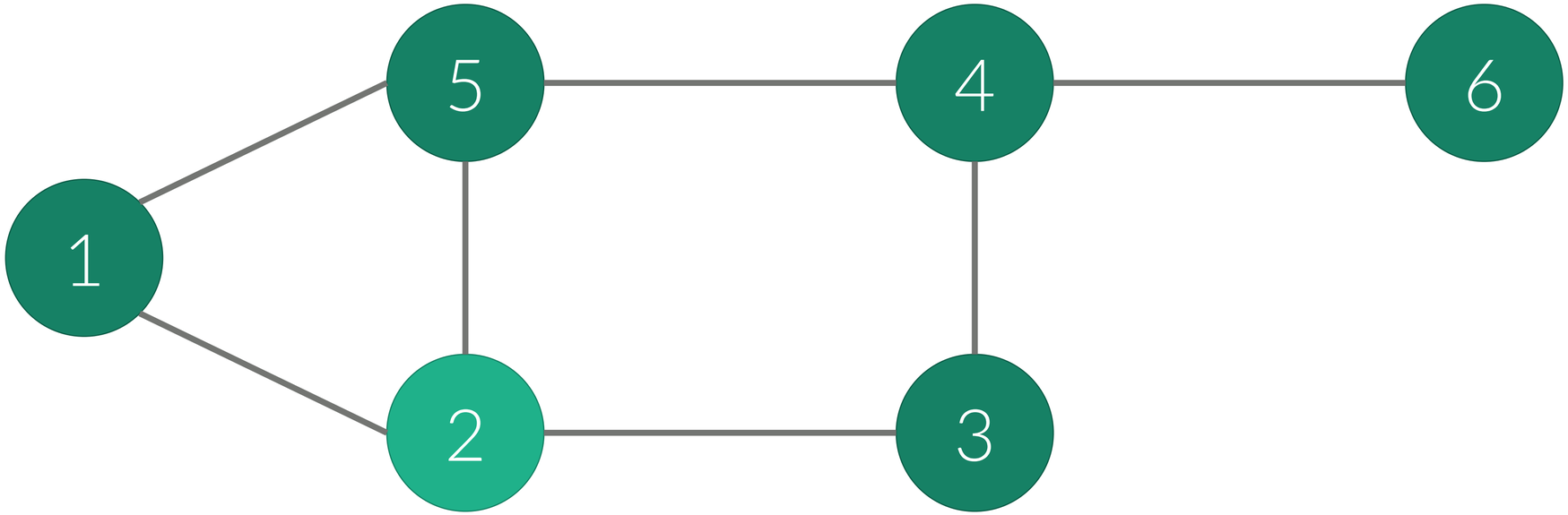


깊이 우선 탐색

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

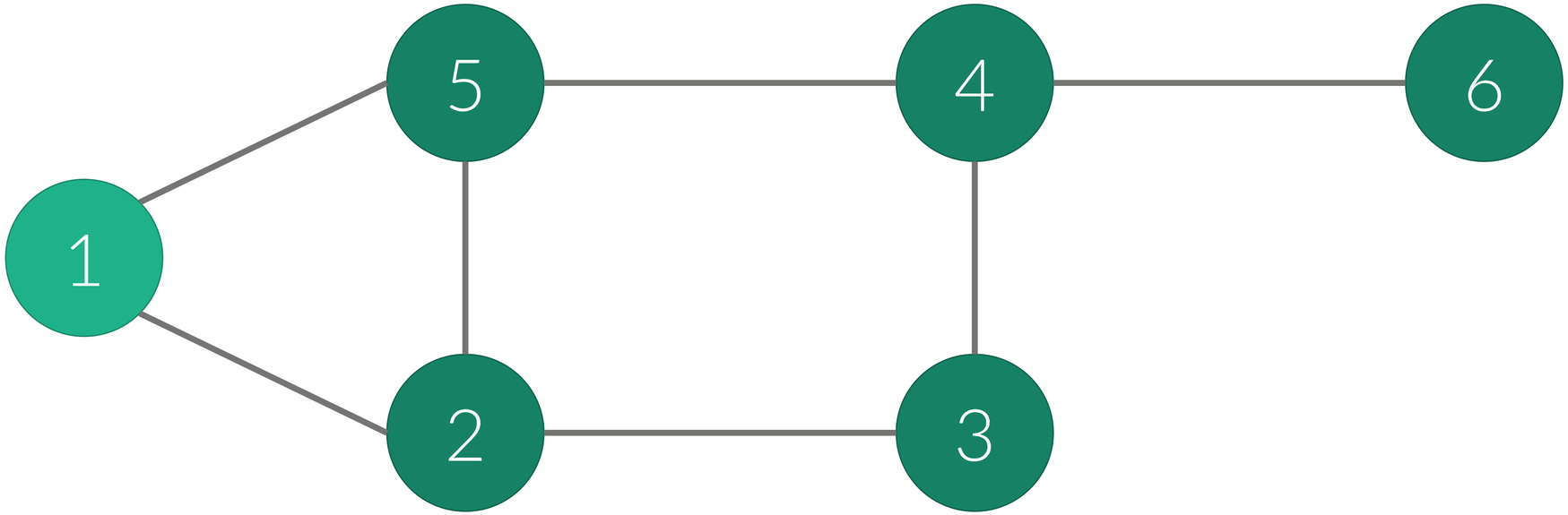


깊이 우선 탐색

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

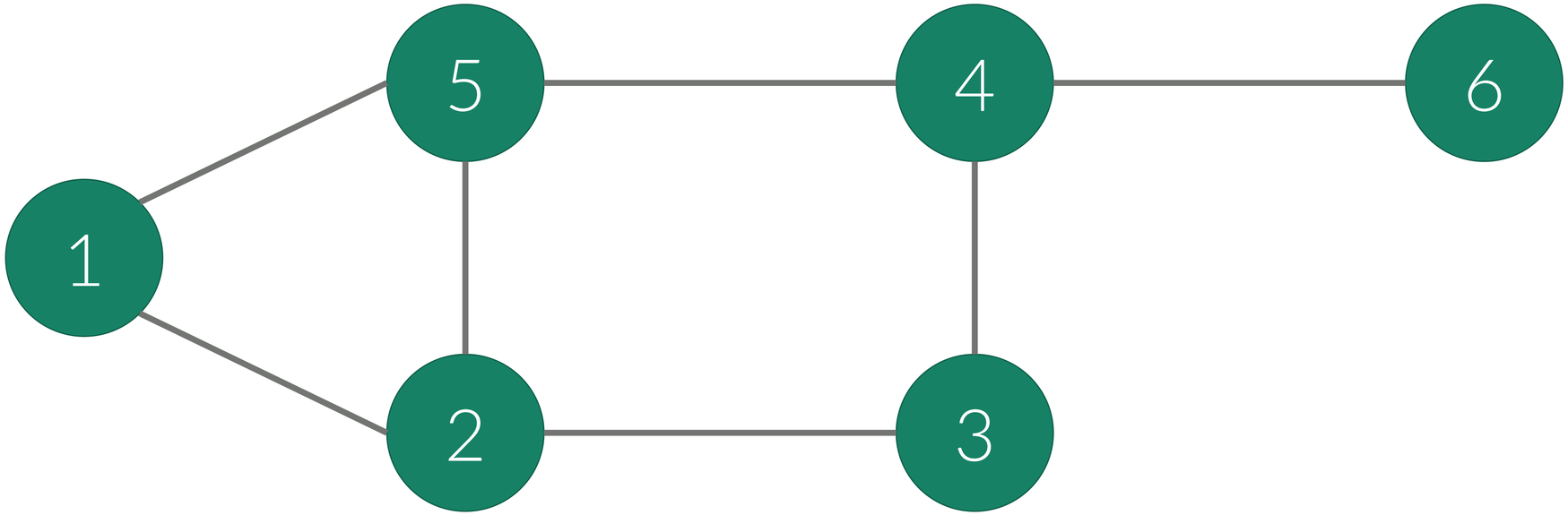


깊이 우선 탐색

Depth First Search

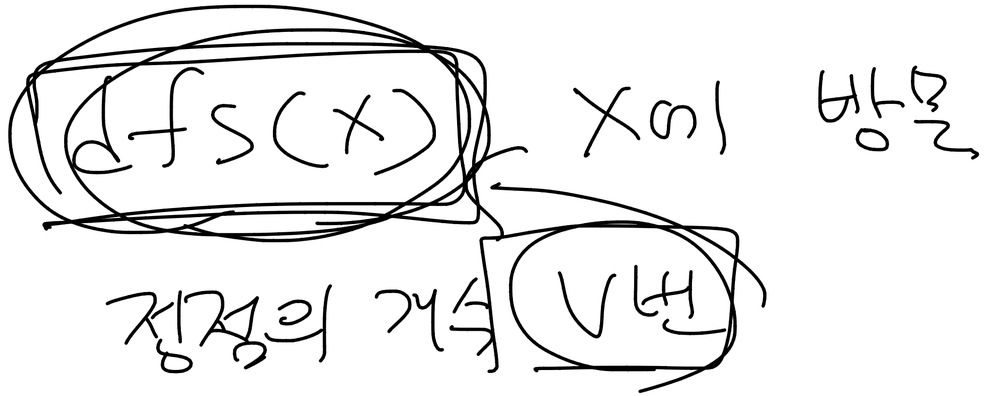
- 현재 정점:
- 순서: 1 2 3 4 5 6
- 스택:
- 탐색 종료

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



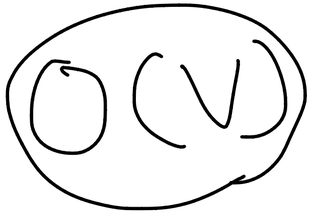
깊이 우선 탐색

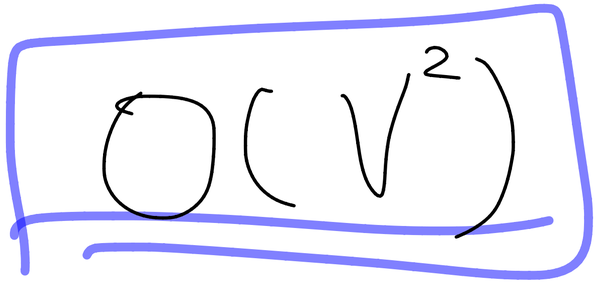
Depth First Search



- 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);
        }
    }
}
```





- 인접 행렬을 이용한 구현

깊이 우선 탐색

Depth First Search

- 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);
        }
    }
}
```

- 인접 리스트를 이용한 구현

dfs × V번

O(V+E)

너비 우선 탐색

Breadth First Search

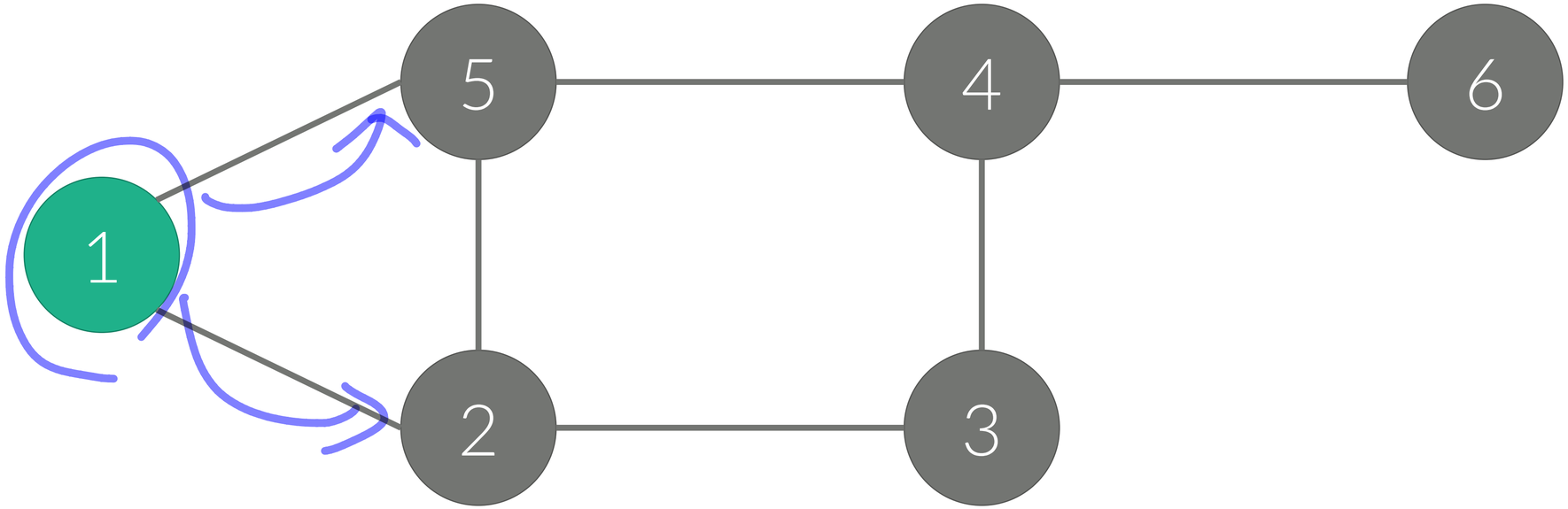
- 큐를 이용해서 지금 위치에서 갈 수 있는 것을 모두 큐에 넣는 방식
- 큐에 넣을 때 방문했다고 체크해야 한다.

너비 우선 탐색

Breadth First Search

- 현재 정점: 1
- 순서: 1
- 큐: 1 2 5

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

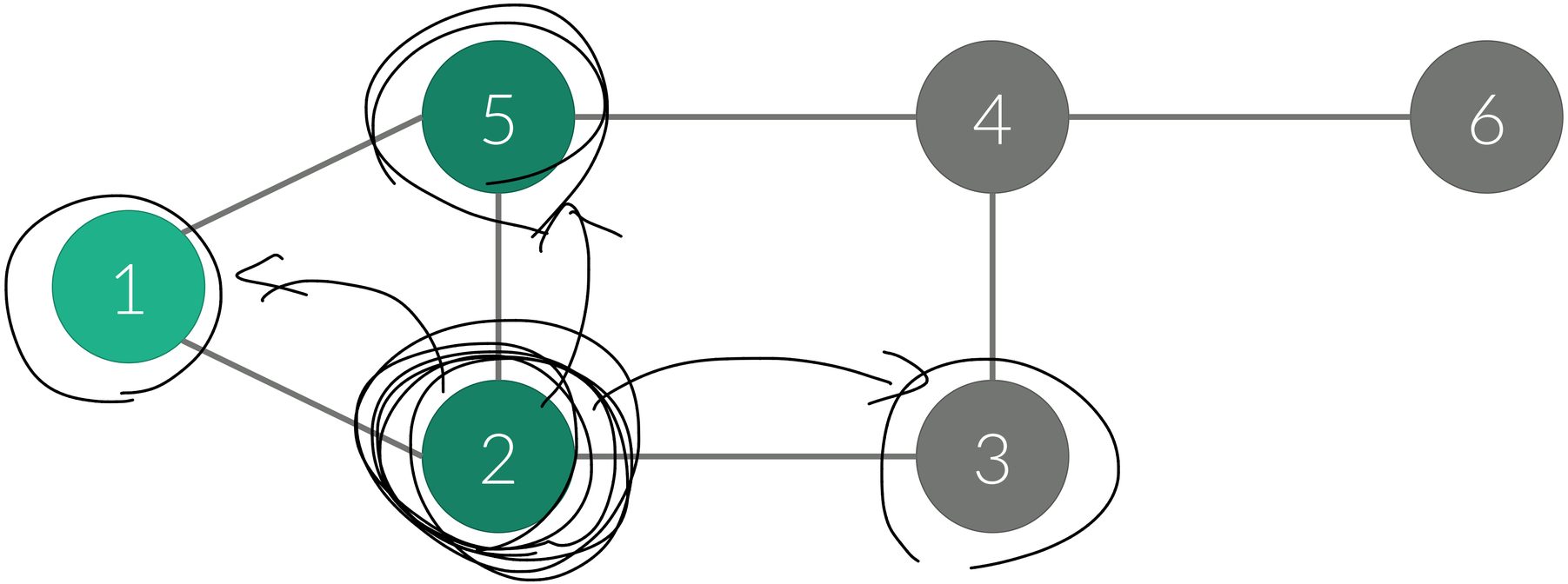


너비 우선 탐색

Breadth First Search

- 현재 정점: 1
- 순서: 1 2 5
- 큐: ~~1~~ 2 5 3 5

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

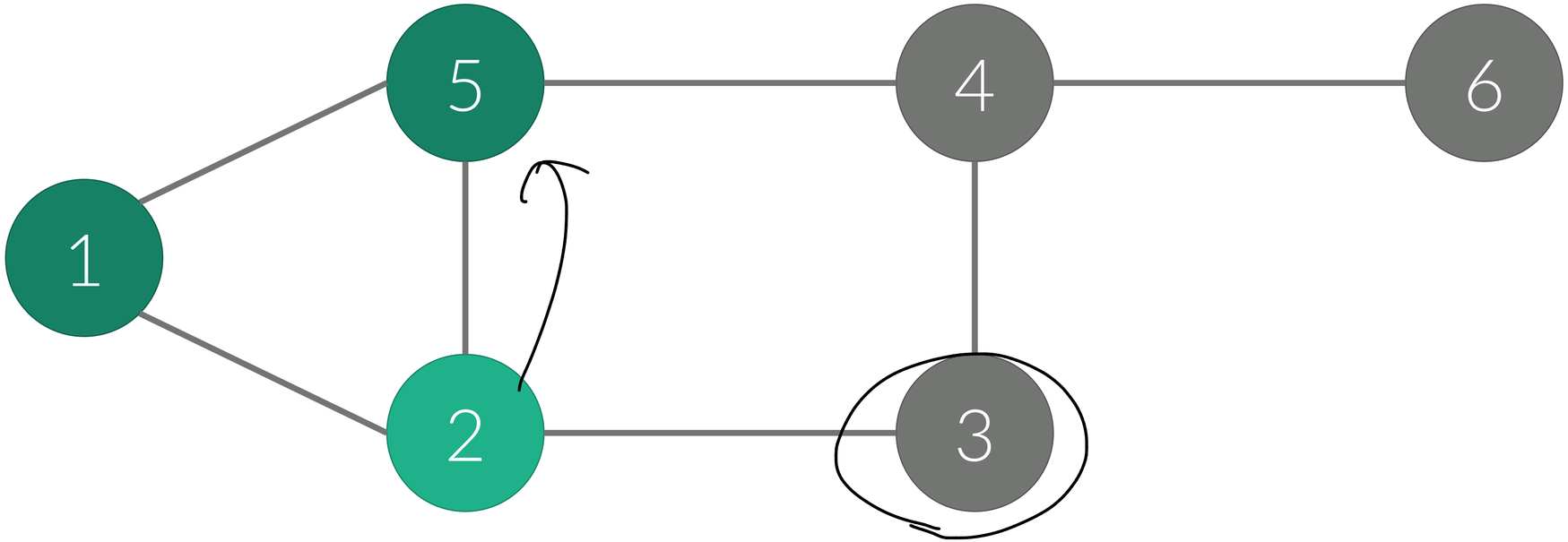


너비 우선 탐색

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	0	0	1	0

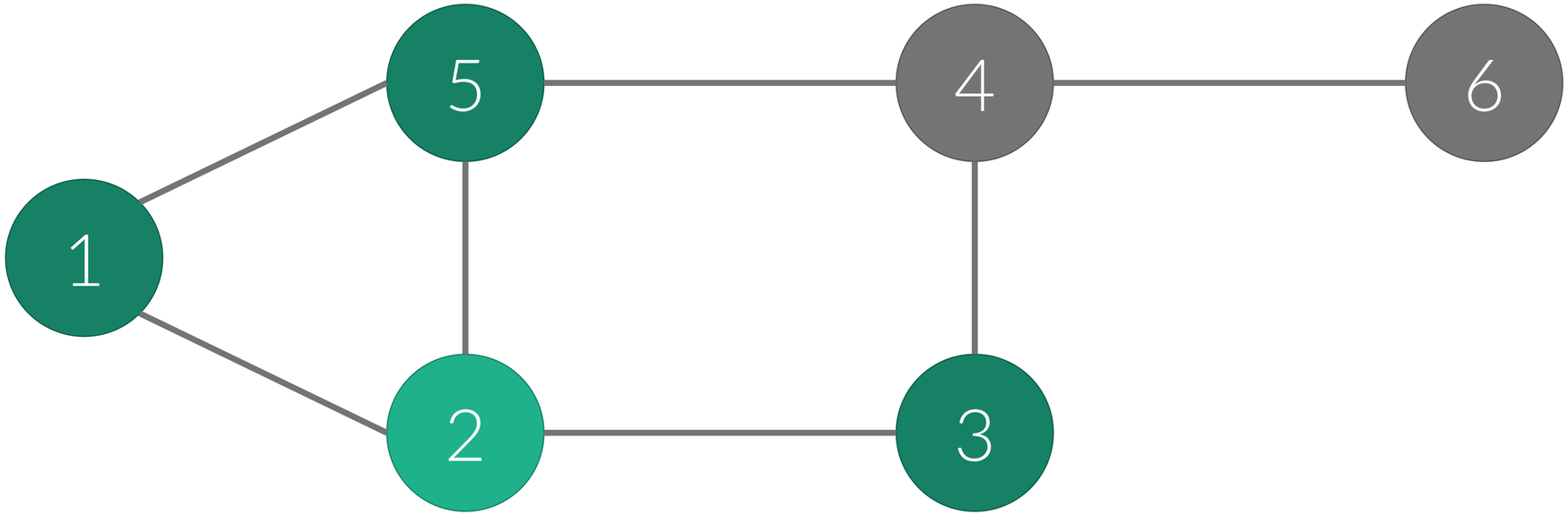


너비 우선 탐색

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

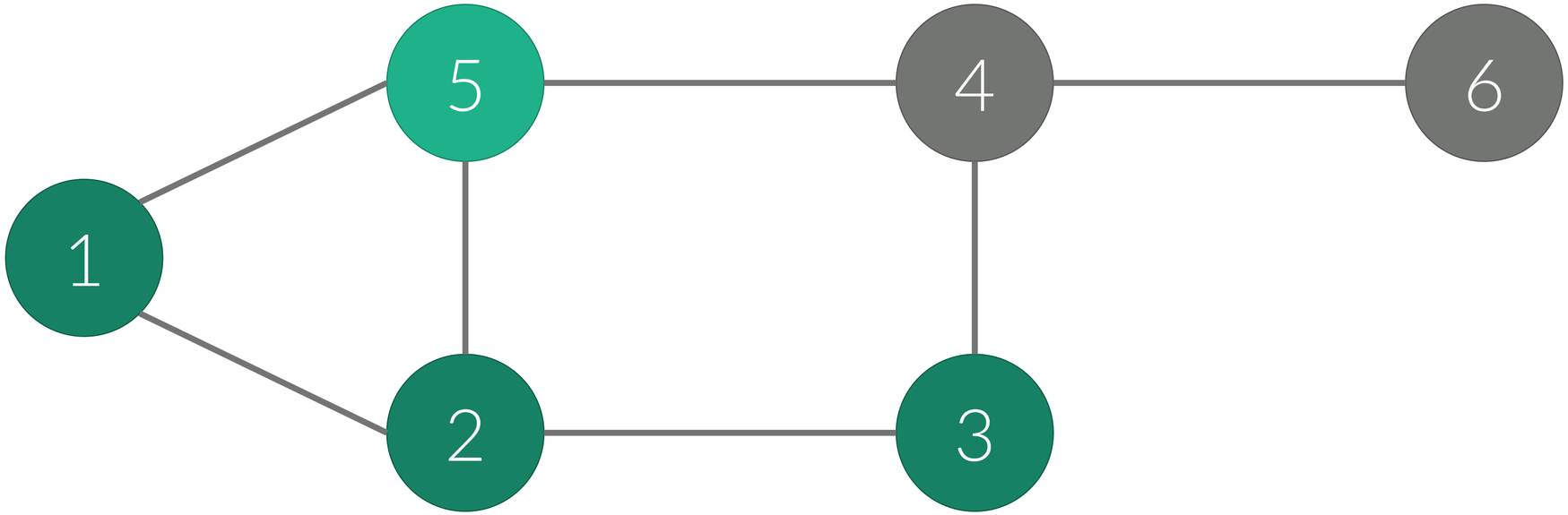


너비 우선 탐색

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

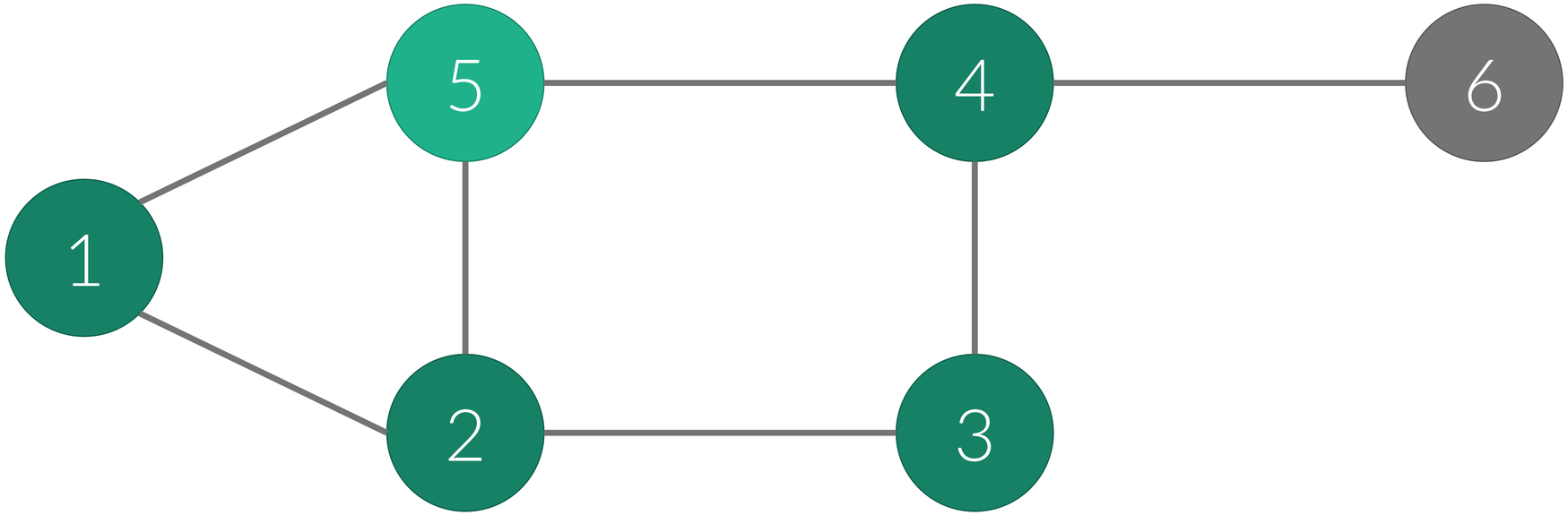


너비 우선 탐색

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

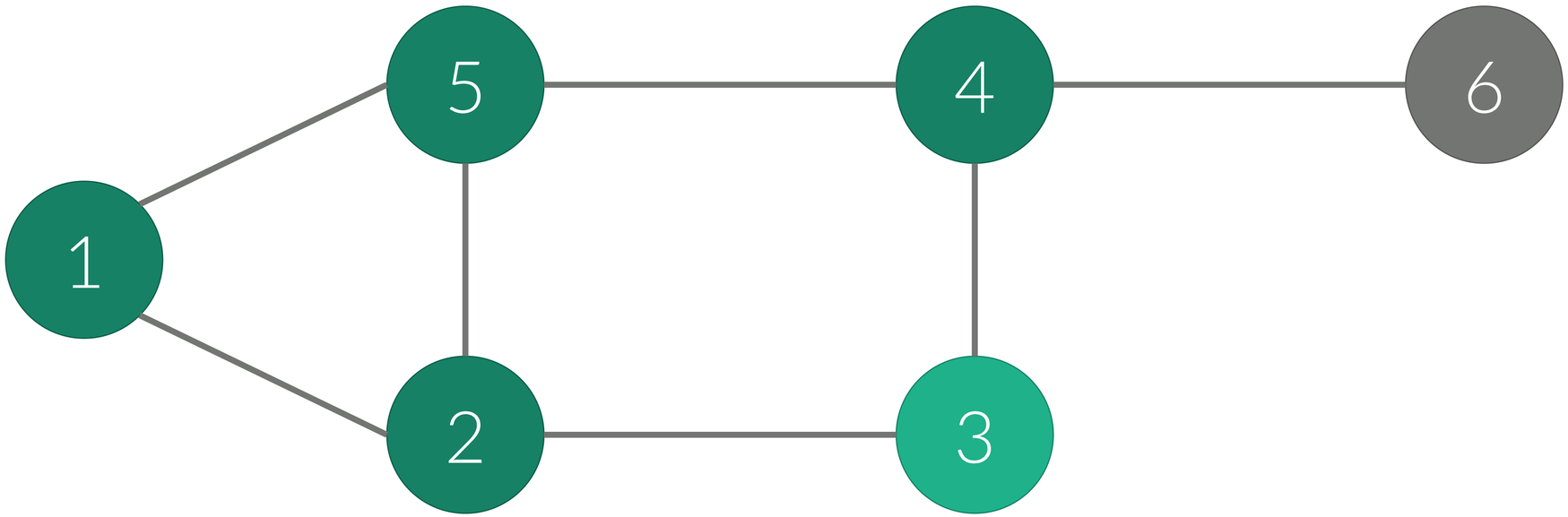


너비 우선 탐색

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

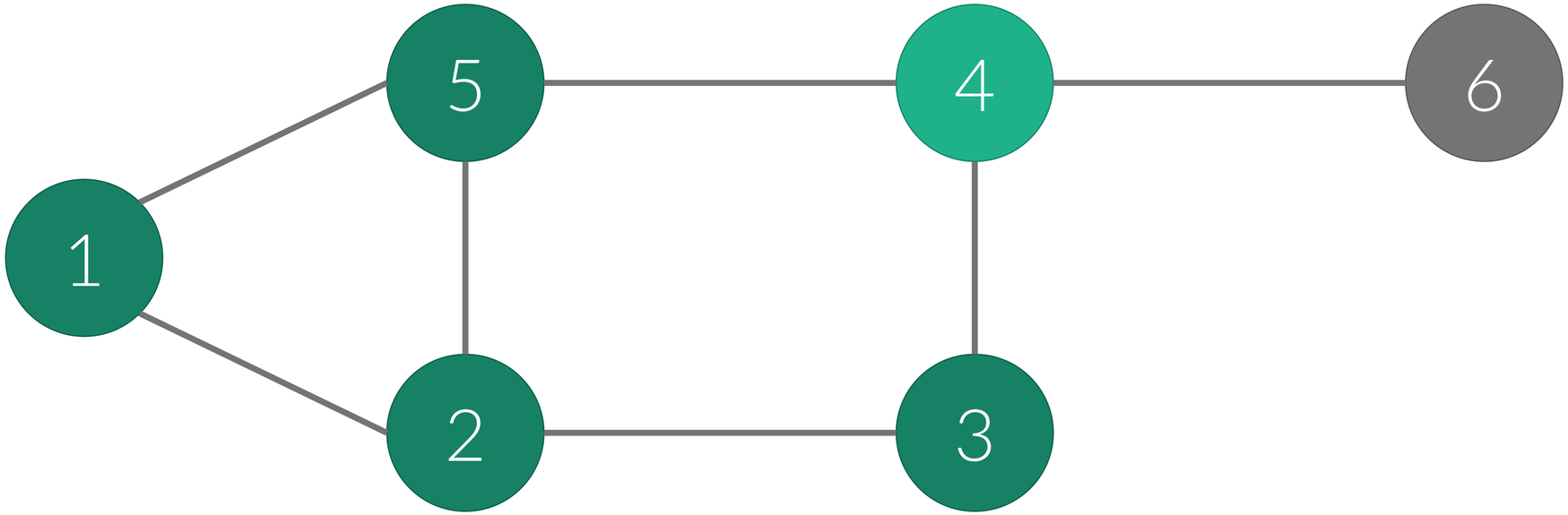


너비 우선 탐색

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

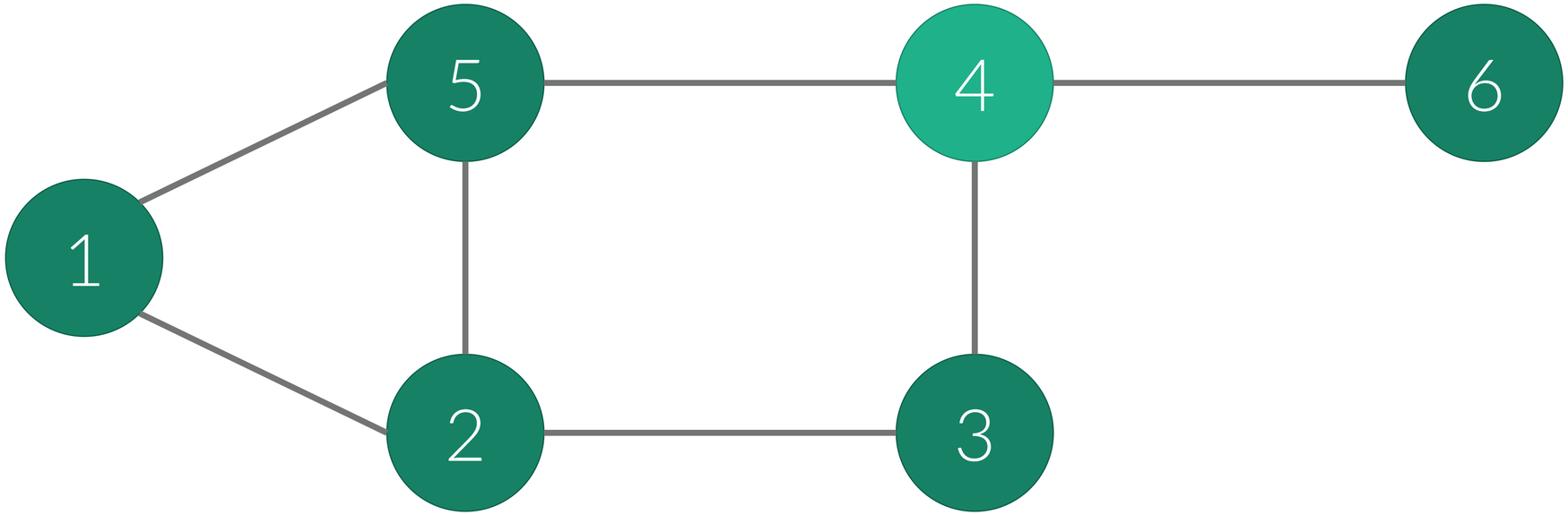


너비 우선 탐색

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

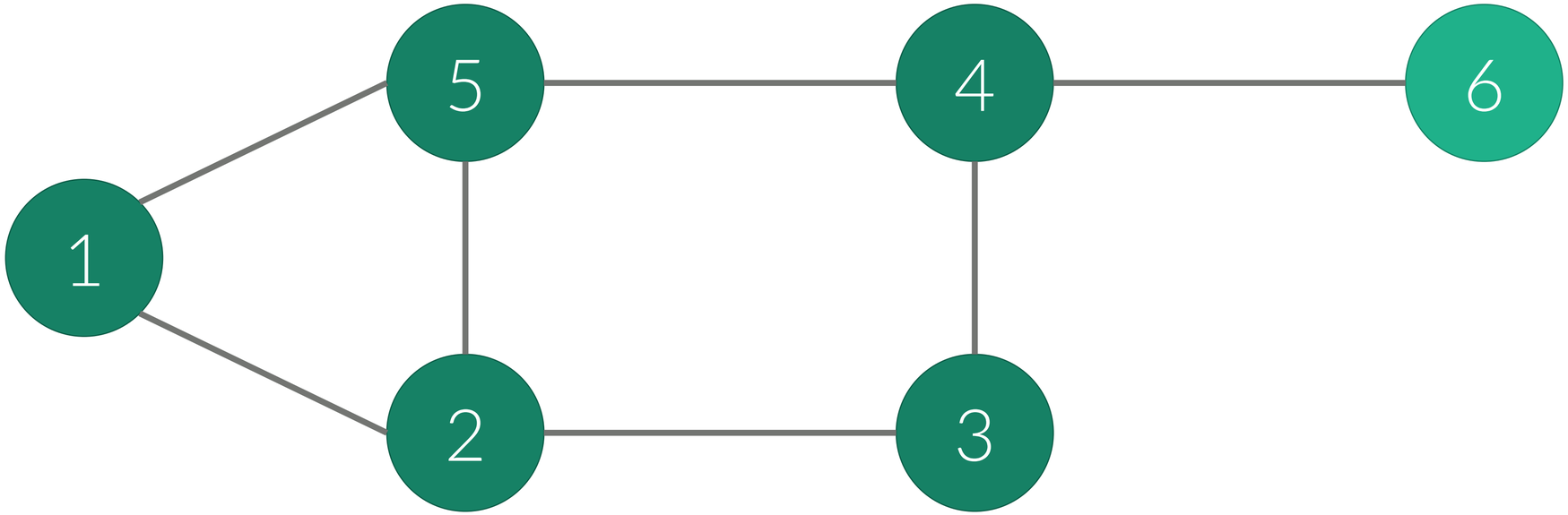


너비 우선 탐색

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

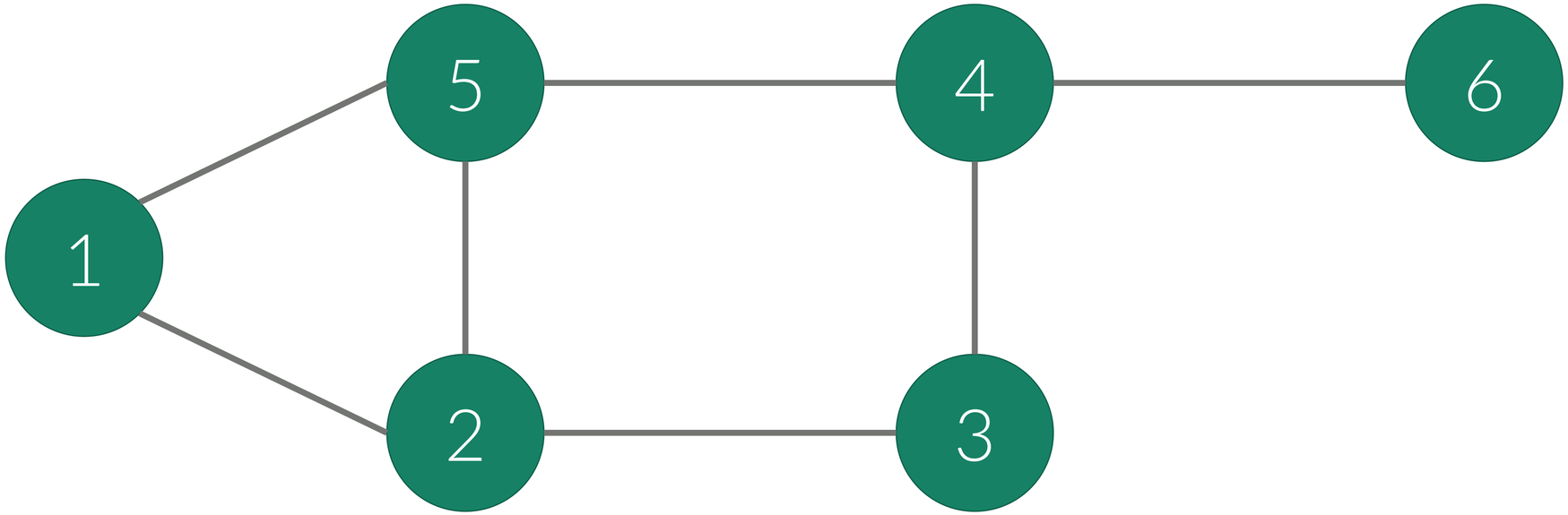


너비 우선 탐색

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

Queue<Integer>

- 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);
        }
    }
}
```

O(V)

너비 우선 탐색

Breadth First Search

- 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://codeplus.codes/67358d01aaeb434793de678961122486
- 간선 리스트 소스: http://codeplus.codes/bd786984224449e0b39b40d0d6d6e733
- 비재귀 구현 소스: http://codeplus.codes/362ddc2eb1394e478e9f623fd0ccc25e

연결 요소

연결 요소

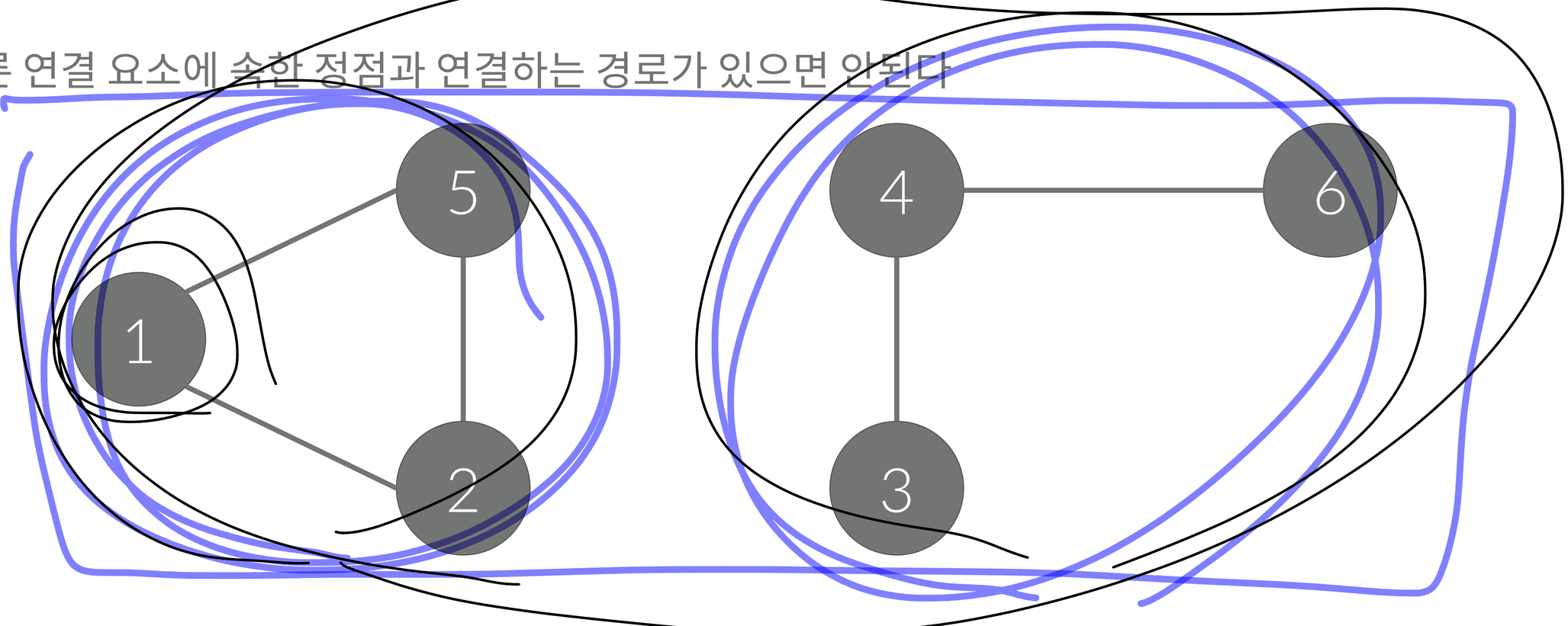
Connected Component

연결 그래프

1개 → 2개

2개

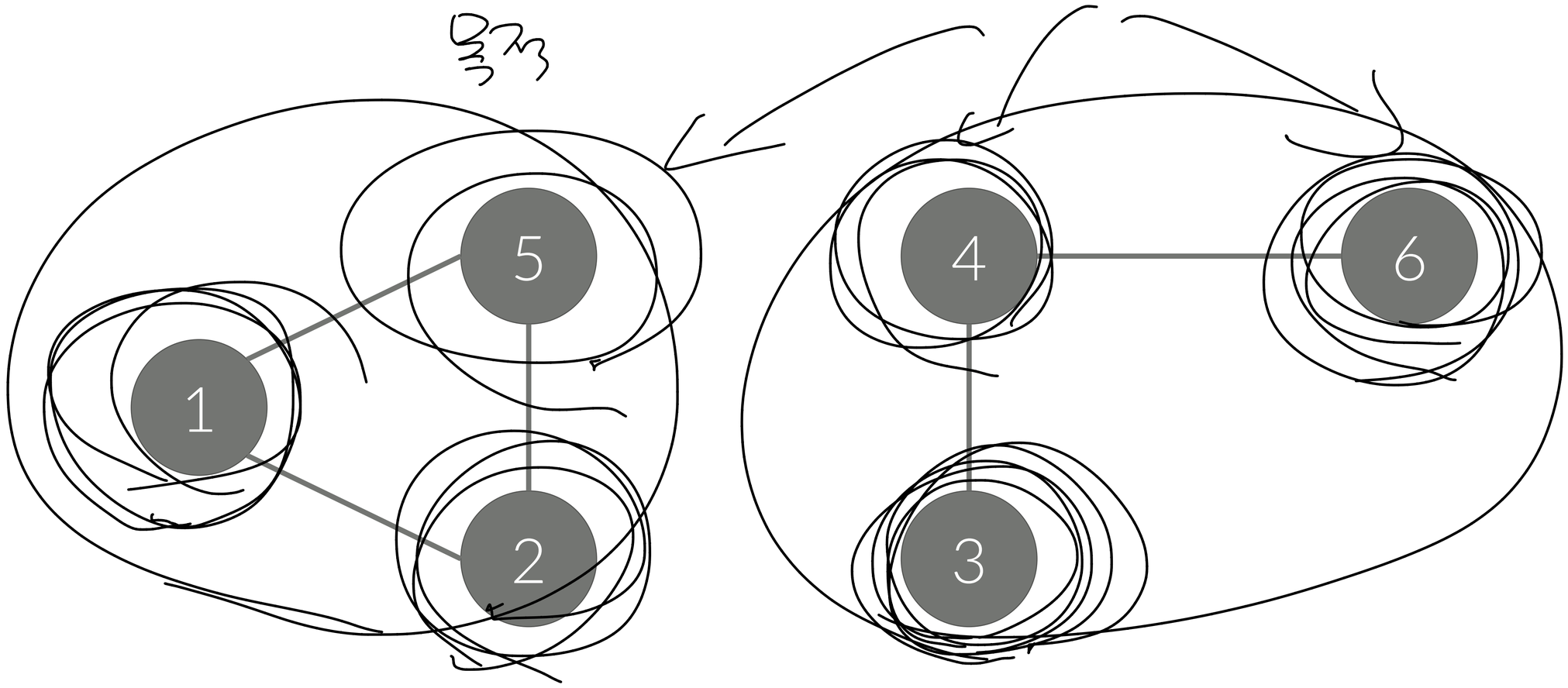
- 그래프가 아래 그림과 같이 나누어져 있지 않은 경우가 있을 수도 있다
- 이렇게 나누어진 각각의 그래프를 연결 요소라고 한다.
- 연결 요소에 속한 모든 정점을 연결하는 경로가 있어야 한다
- 또, 다른 연결 요소에 속한 정점과 연결하는 경로가 있으면 안된다



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/59a464c71a6a4957abd6497c43c1e4a8

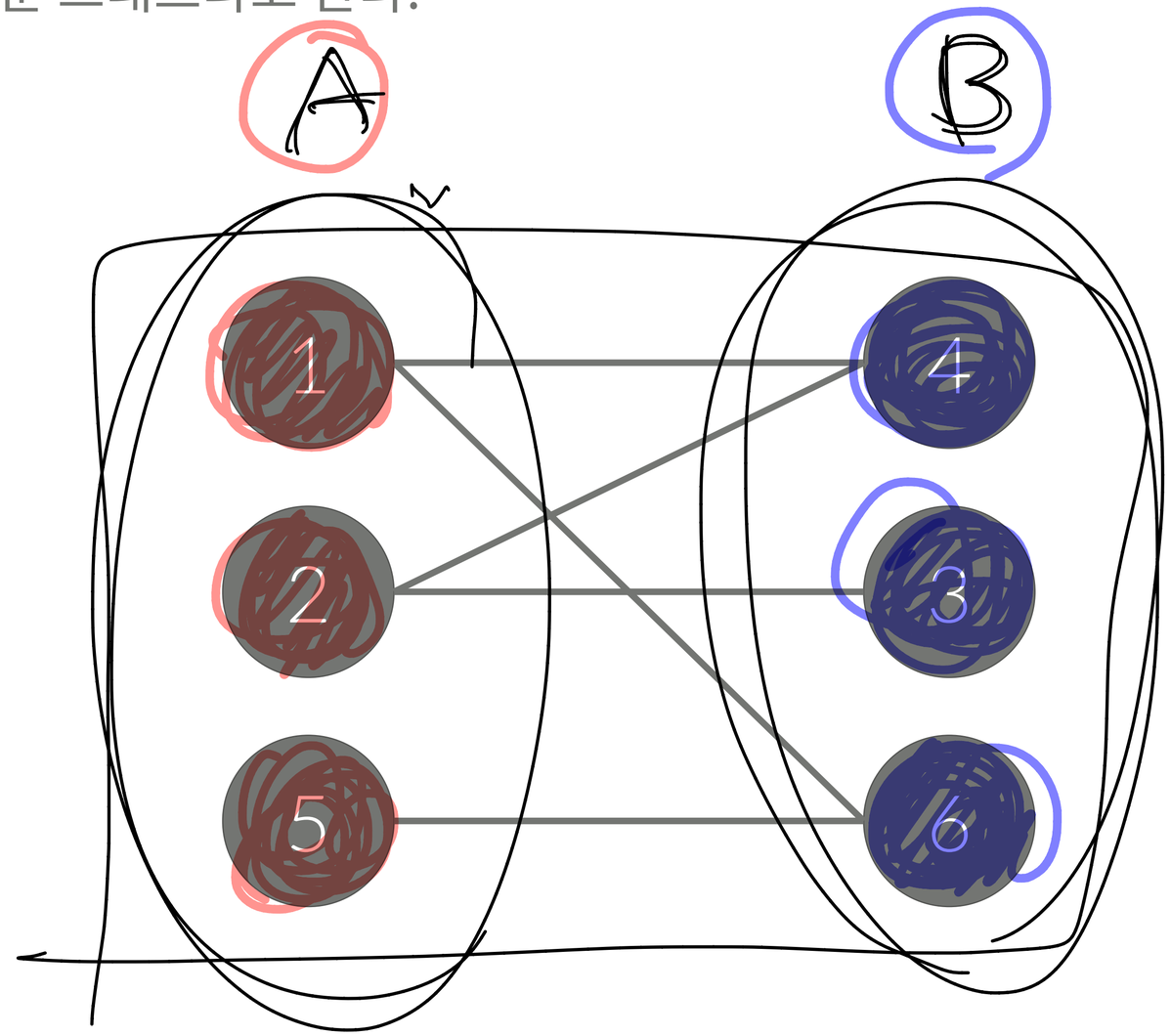
이분 그래프

이분 그래프

DFS, BFS

Bipartite Graph

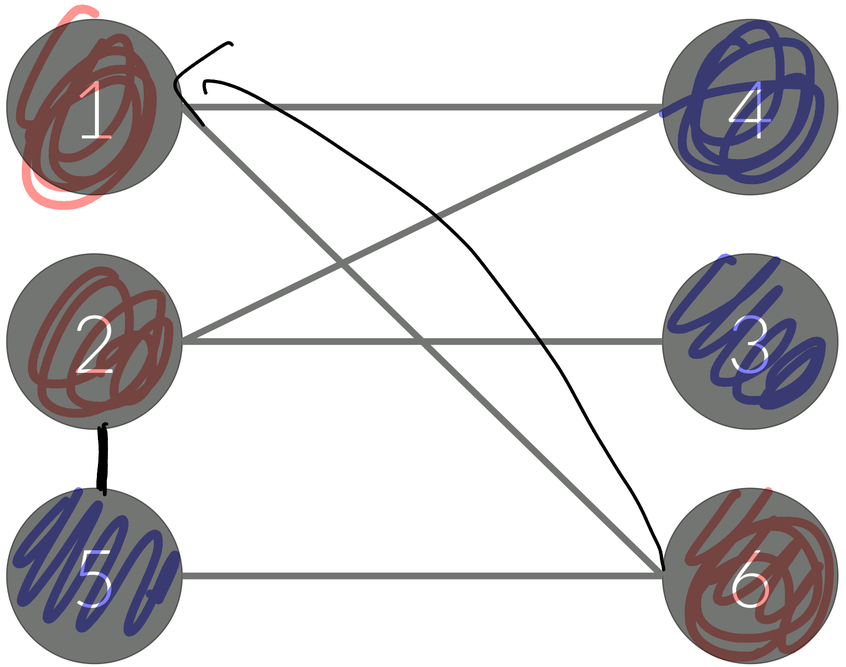
- 그래프를 다음과 같이 A와 B로 나눌 수 있으면 이분 그래프라고 한다.
- A에 포함되어 있는 정점끼리 연결된 간선이 없음
- B에 포함되어 있는 정점끼리 연결된 간선이 없음
- 모든 간선의 한 끝 점은 A에, 다른 끝 점은 B에



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

DFS

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

- 소스: http://codeplus.codes/e680a1b718534cb29185f7d8b4f0cd74

플러드 필

플러드 필

Flood Fill

- 어떤 위치와 연결된 모든 위치를 찾는 알고리즘

단지번호붙이기

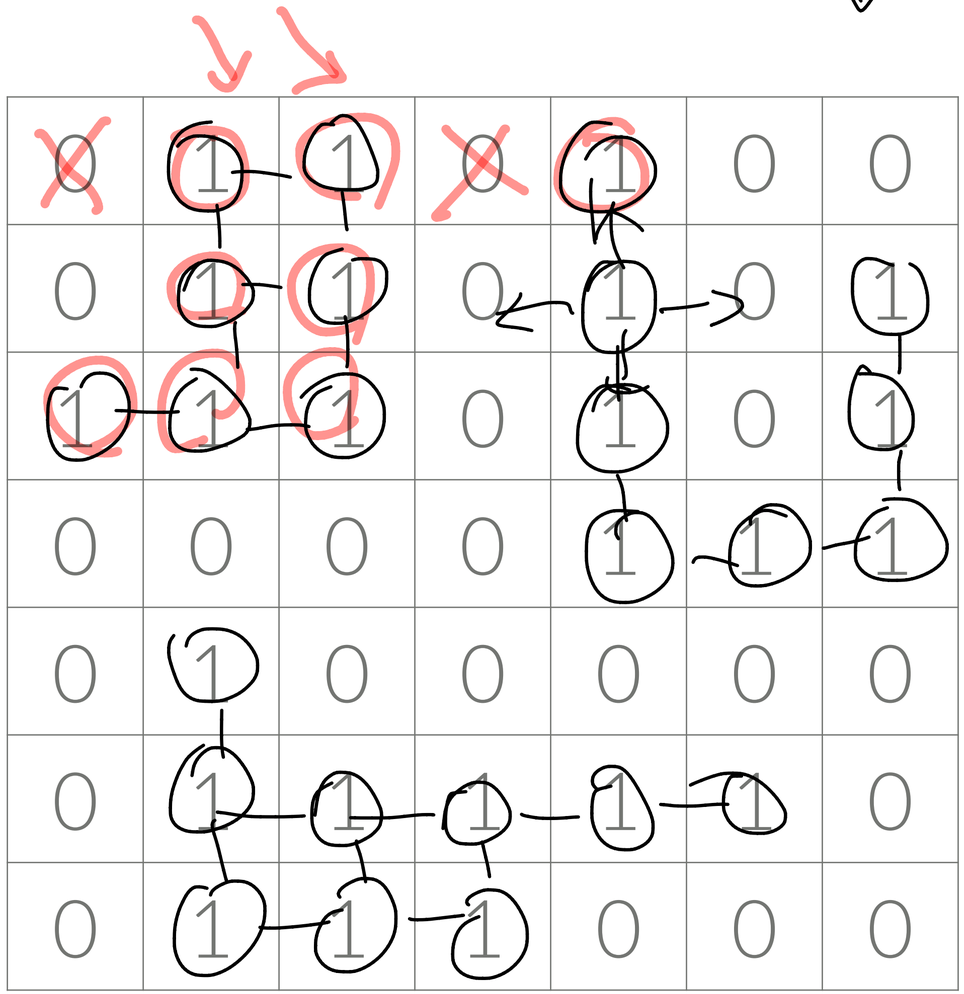
https://www.acmicpc.net/problem/2667

- 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

3개의



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

7

8

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

9

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

(x,y) → (nx,ny)

0 ≤ n-1

```
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;
                }
            }
        }
    }
}
```

dx[] = {1, -1, 0, 0}

dy[] = {0, 0, 1, -1}

↓ ↑ → ←

단지번호붙이기

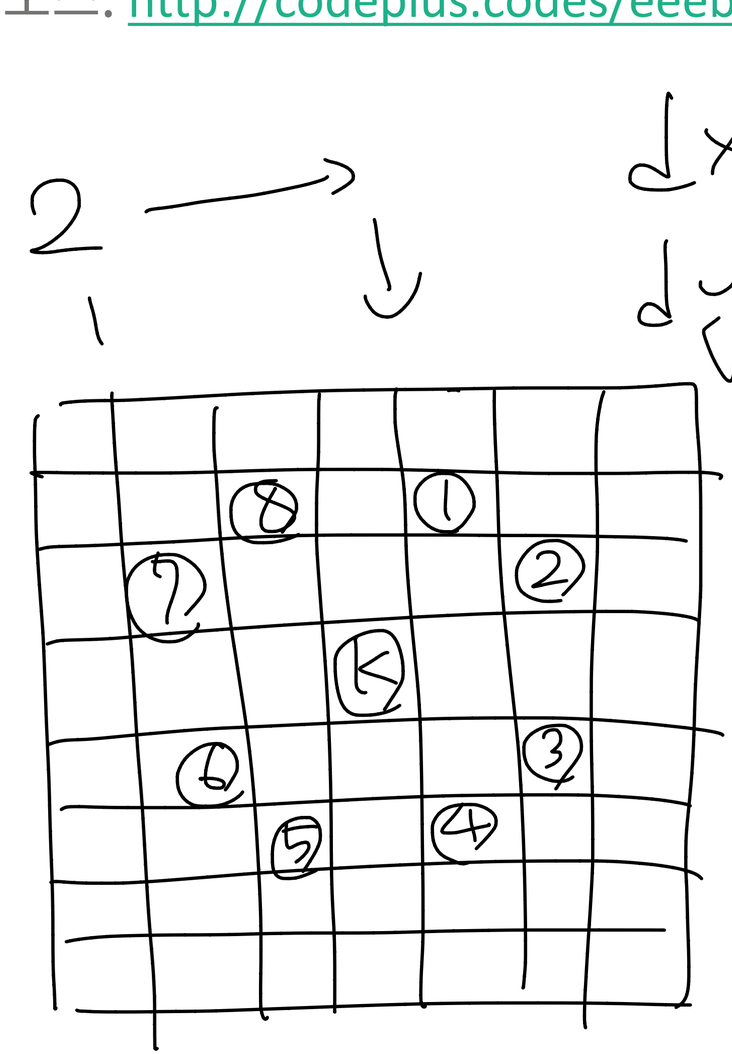
https://www.acmicpc.net/problem/2667

- BFS 소스: http://codeplus.codes/3e6999a82c774a51b2b70da44e90247f
- DFS 소스: http://codeplus.codes/23892898ed3045908a2316c632d55d08

섬의 개수

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

- 소스: http://codeplus.codes/eeebe2e472d44fc081f1ed68eb0b512d



dx[] = {1, -1, 0, 0, -1, -1, 1, 1}

dy[] = {0, 0, 1, -1, -1, 1, -1, 1}

dx[] = {-2, -1, 1, 2, 2, 1, -1, -2}

dy[] = {1, 2, 2, 1, -1, -2, -2, -1}

BFS

BFS

BFS

- BFS의 목적은 임의의 정점에서 시작해서, 모든 정점을 한 번씩 방문하는 것이다.

BFS

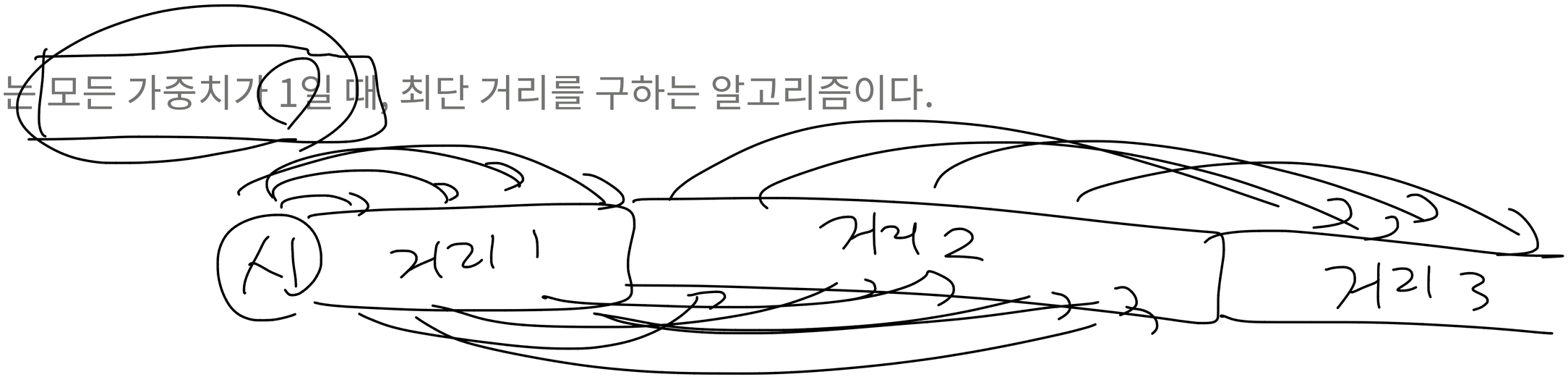
BFS

- BFS는 최단 거리를 구하는 알고리즘이다.

BFS

BFS

- BFS는 모든 가중치가 1일 때, 최단 거리를 구하는 알고리즘이다.



BFS

BFS

최단거리

- BFS를 이용해 해결할 수 있는 문제는 아래와 같은 조건을 만족해야 한다

1. 최소 비용 문제이어야 한다
2. 간선의 가중치가 1이어야 한다
3. 정점과 간선의 개수가 적어야 한다. (적다는 것은 문제의 조건에 맞춰서 해결할 수 있다는 것을 의미한다)

시간 제한
메모리 제한

BFS

BFS

- BFS를 이용해 해결할 수 있는 문제는 아래와 같은 조건을 만족해야 한다

1. **최소 비용** 문제이어야 한다
2. **간선의 가중치**가 1이어야 한다
3. 정점과 간선의 개수가 적어야 한다. (적다는 것은 문제의 조건에 맞춰서 해결할 수 있다는 것을 의미한다)

- 간선의 가중치가 문제에서 구하라고 하는 최소 비용과 의미가 일치해야 한다
- 즉, 거리의 최소값을 구하는 문제라면 가중치는 거리를 의미해야 하고, 시간의 최소값을 구하는 문제라면 가중치는 시간을 의미해야 한다

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

시작

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

도착

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://codeplus.codes/3c4ab88008f14d8aa05c1885170b99f4

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

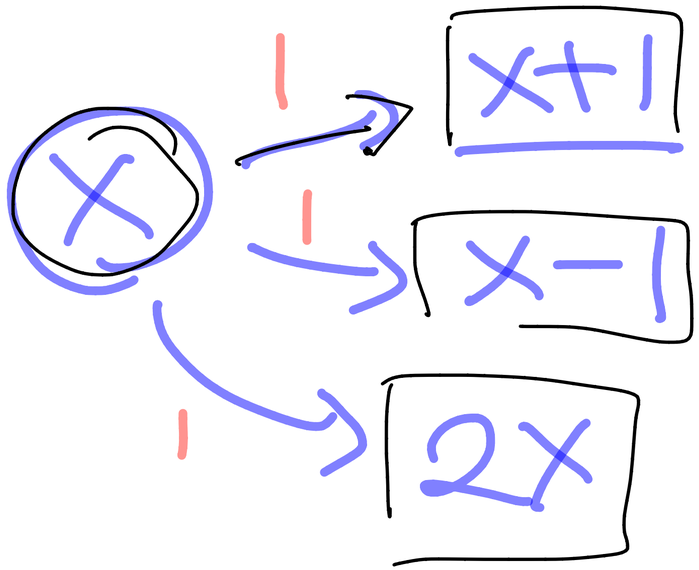
토마토

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

- 소스: http://codeplus.codes/77af5c9a751645f79b67466b533f9d77

숨바꼭질

https://www.acmicpc.net/problem/1697



- 수빈이의 위치: N
- 동생의 위치: K
- 동생을 찾는 가장 빠른 시간을 구하는 문제

- 수빈이가 할 수 있는 행동 (위치: X)

1. 걷기: X+1 또는 X-1로 이동 (1초)
2. 순간이동: 2*X로 이동 (1초)

숨바꼭질

https://www.acmicpc.net/problem/1697

- 수빈이의 위치: N
- 동생의 위치: K
- 동생을 찾는 **가장 빠른 시간**을 구하는 문제

- 수빈이가 할 수 있는 행동 (위치: X)

1. 걷기: X+1 또는 X-1로 이동 (**1초**)
2. 순간이동: 2*X로 이동 (**1초**)

숨바꼭질

https://www.acmicpc.net/problem/1697

- 수빈이의 위치: 5
- 동생의 위치: 17

- 5-10-9-18-17 로 4초만에 동생을 찾을 수 있다.

숨바꼭질

https://www.acmicpc.net/problem/1697

- 큐에 수빈이의 위치를 넣어가면서 이동시킨다
- 한 번 방문한 곳은 다시 방문하지 않는 것이 좋기 때문에, 따로 배열에 체크하면서 방문

숨바꼭질

https://www.acmicpc.net/problem/1697

- 가장 처음

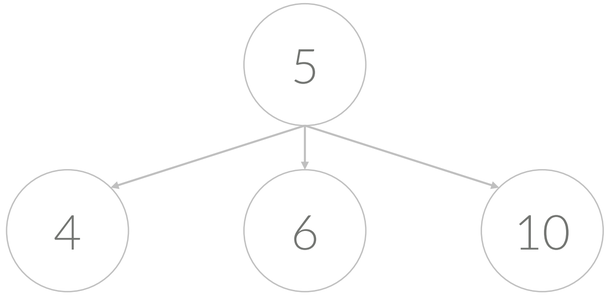
5

- Queue: 5

숨바꼭질

https://www.acmicpc.net/problem/1697

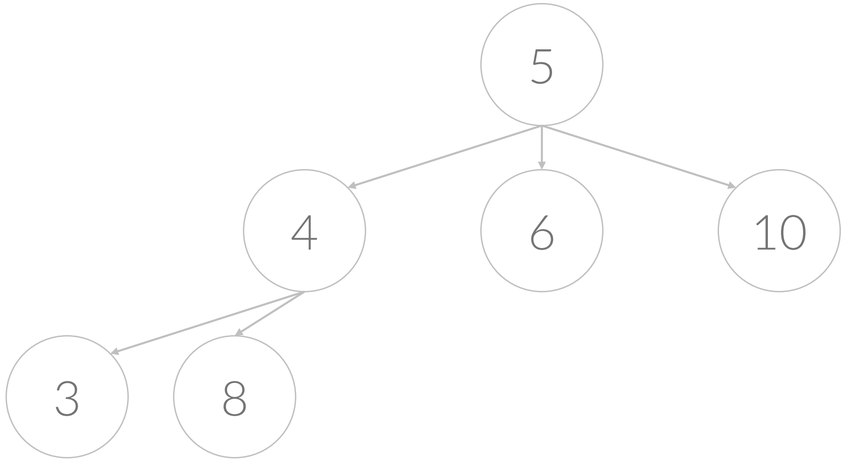
- 5에서 이동
- Queue: 5 4 6 10



숨바꼭질

https://www.acmicpc.net/problem/1697

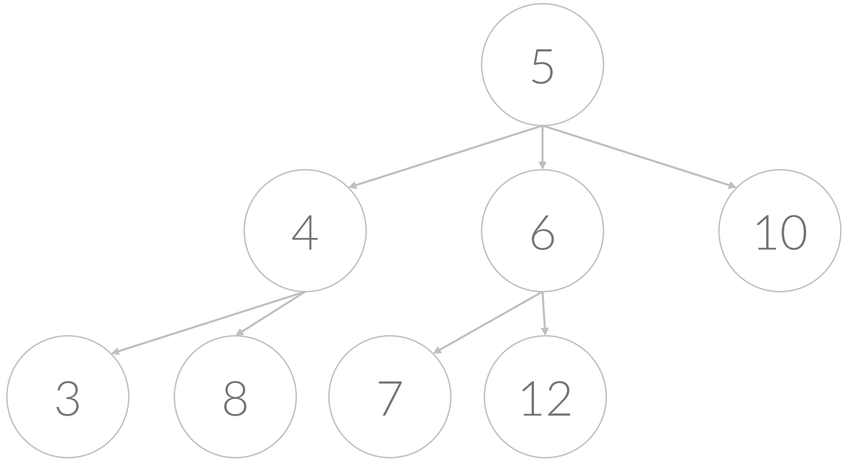
- 4에서 이동
- Queue: 5 4 6 10 3 8



숨바꼭질

https://www.acmicpc.net/problem/1697

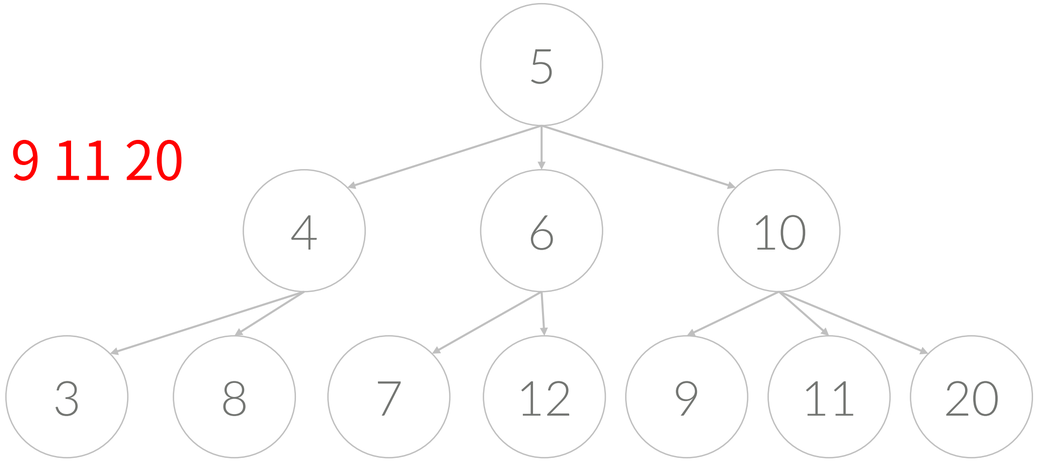
- 6에서 이동
- Queue: 5 4 6 10 3 8 7 12



숨바꼭질

https://www.acmicpc.net/problem/1697

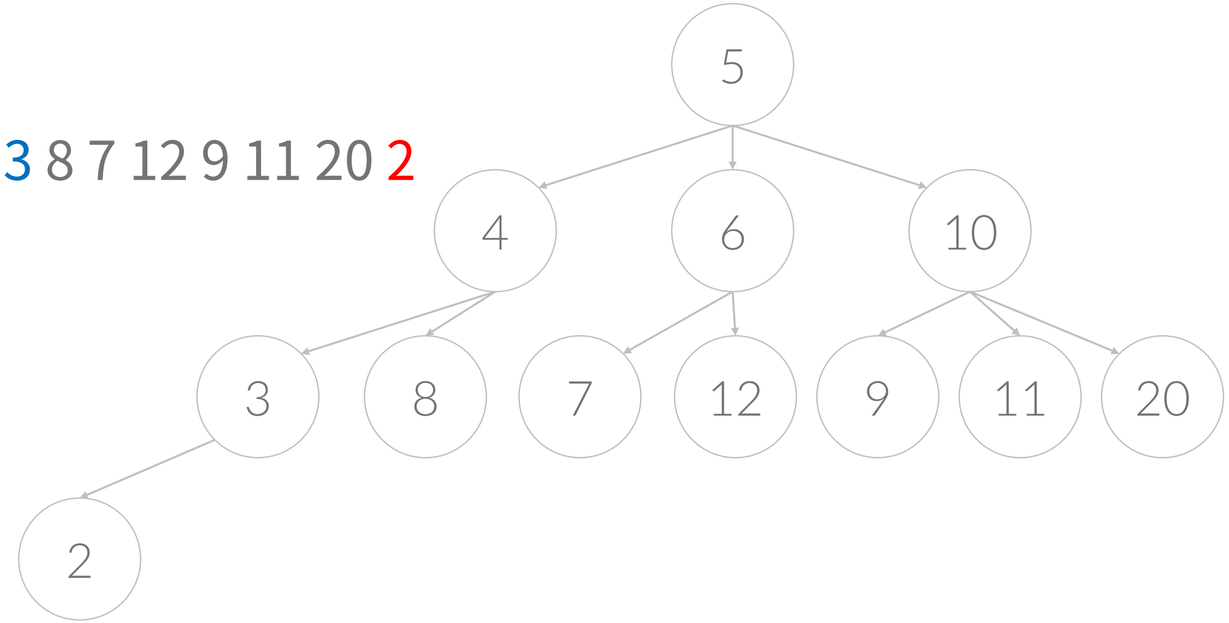
- 10에서 이동
- Queue: 5 4 6 10 3 8 7 12 9 11 20



숨바꼭질

https://www.acmicpc.net/problem/1697

- 3에서 이동
- Queue: 5 4 6 10 3 8 7 12 9 11 20 2



숨바꼭질

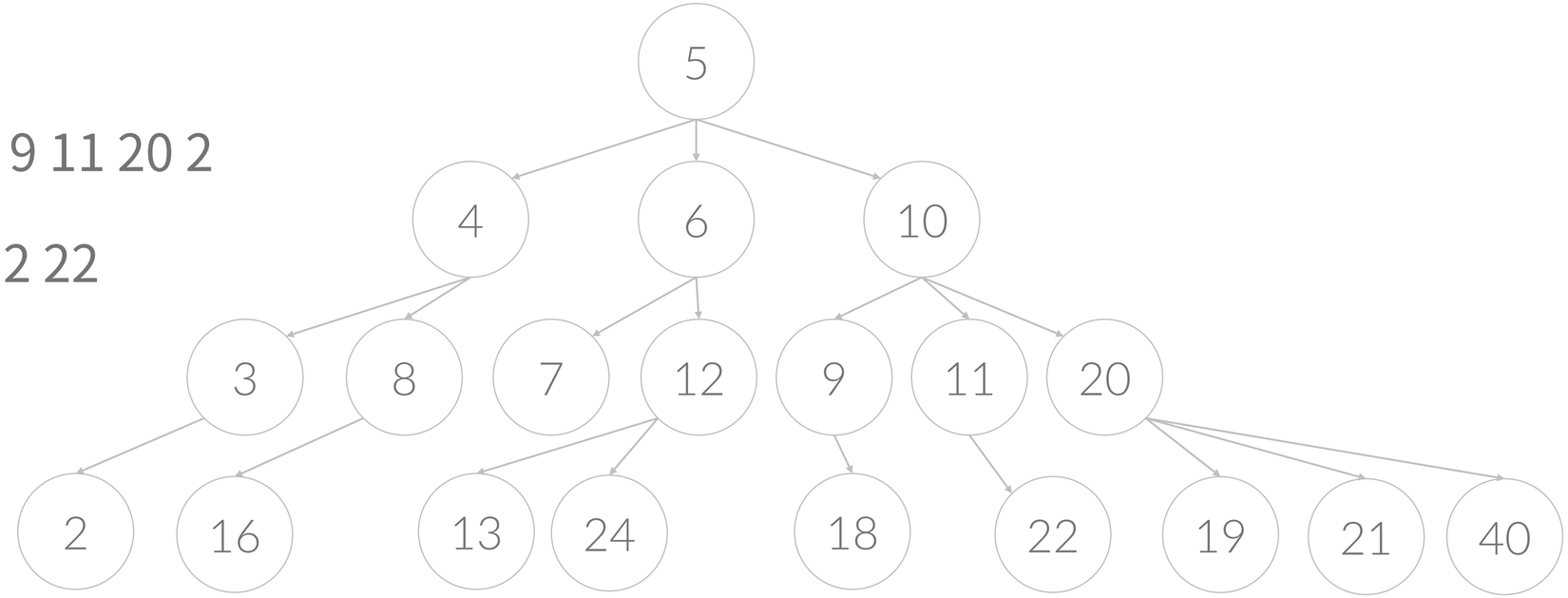
https://www.acmicpc.net/problem/1697

- 이런식으로…

숨바꼭질

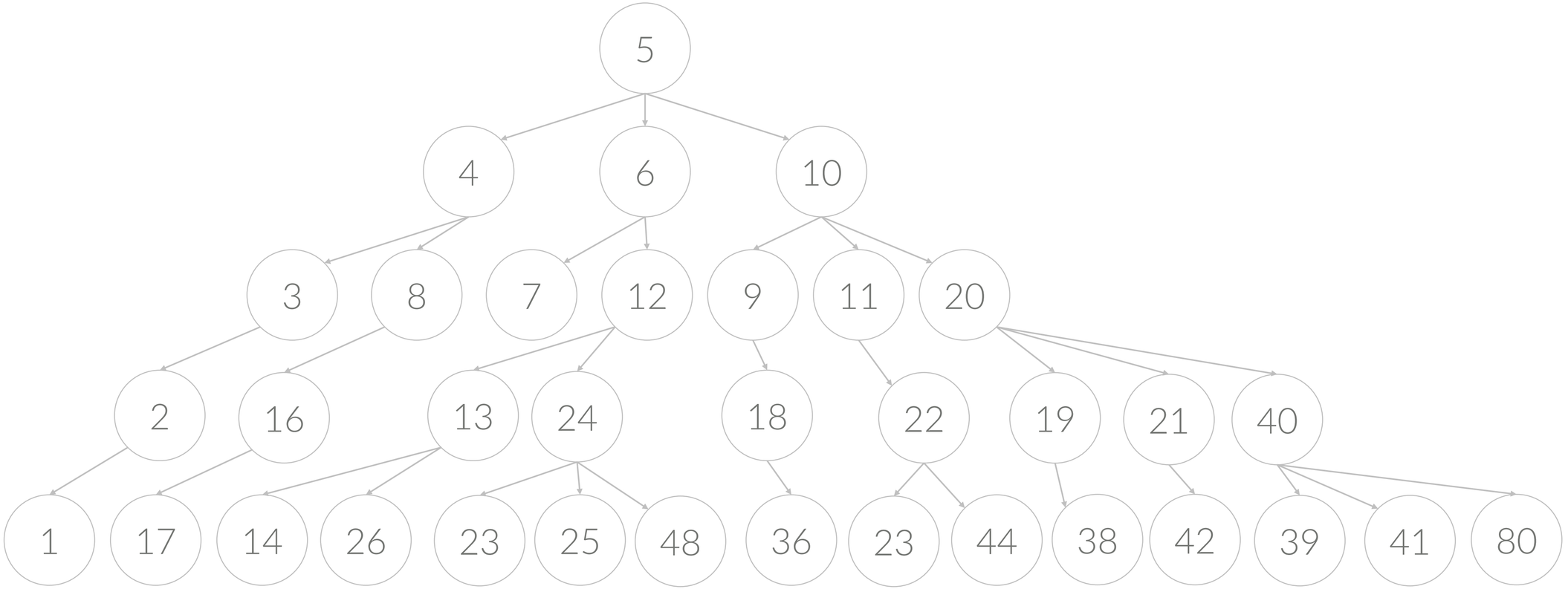
https://www.acmicpc.net/problem/1697

- Queue:
- 5 4 6 10 3 8 7 12 9 11 20 2
- 16 13 15 28 18 12 22
- 19 21 40



숨바꼭질

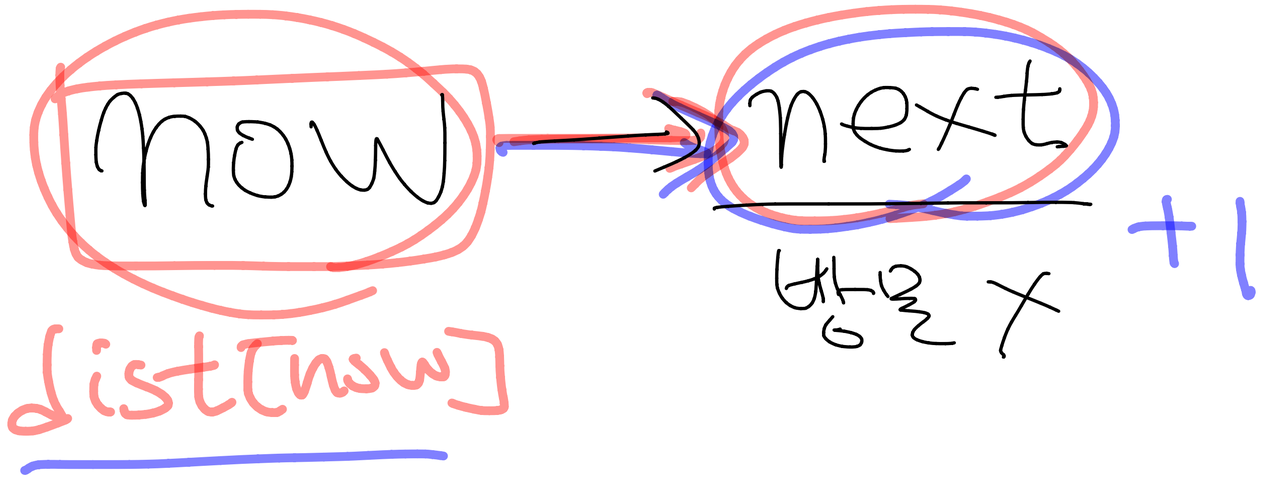
https://www.acmicpc.net/problem/1697



숨바꼭질

https://www.acmicpc.net/problem/1697

- check[i] = i를 방문했는지
- dist[i] = i를 몇 번만에 방문했는지

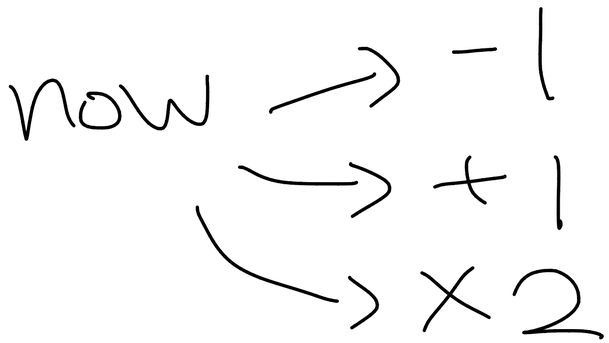


숨바꼭질

n → m

https://www.acmicpc.net/problem/1697

```
check[n] = true;
dist[n] = 0;
queue<int> q;
q.push(n);
while (!q.empty()) {
    int now = q.front();
    q.pop();
    if (now-1 >= 0) {
        if (check[now-1] == false) {
            q.push(now-1);
            check[now-1] = true;
            dist[now-1] = dist[now] + 1;
        }
    }
```




```
    if (now+1 < MAX) {
        if (check[now+1] == false) {
            q.push(now+1);
            check[now+1] = true;
            dist[now+1] = dist[now] + 1;
        }
    }
    if (now*2 < MAX) {
        if (check[now*2] == false) {
            q.push(now*2);
            check[now*2] = true;
            dist[now*2] = dist[now] + 1;
        }
    }
}
```

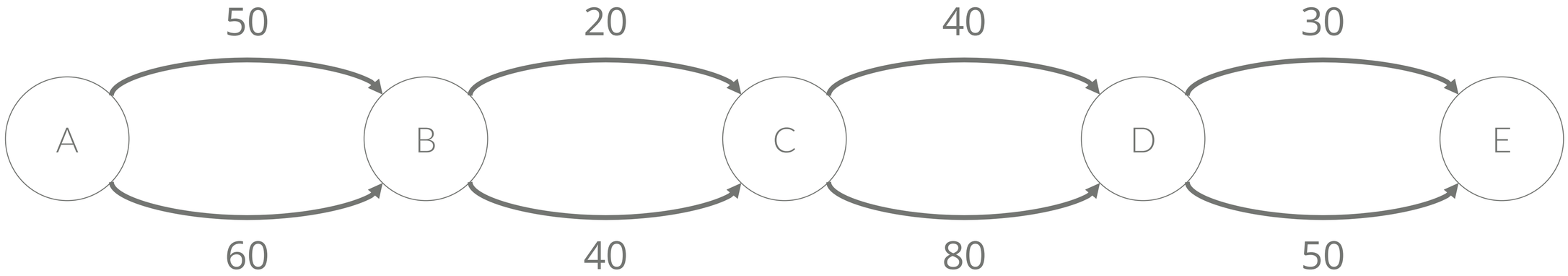
숨바꼭질

https://www.acmicpc.net/problem/1697

- 소스: http://codeplus.codes/9116e7f3d4634964a7c5b3f0f88bb332

BFS

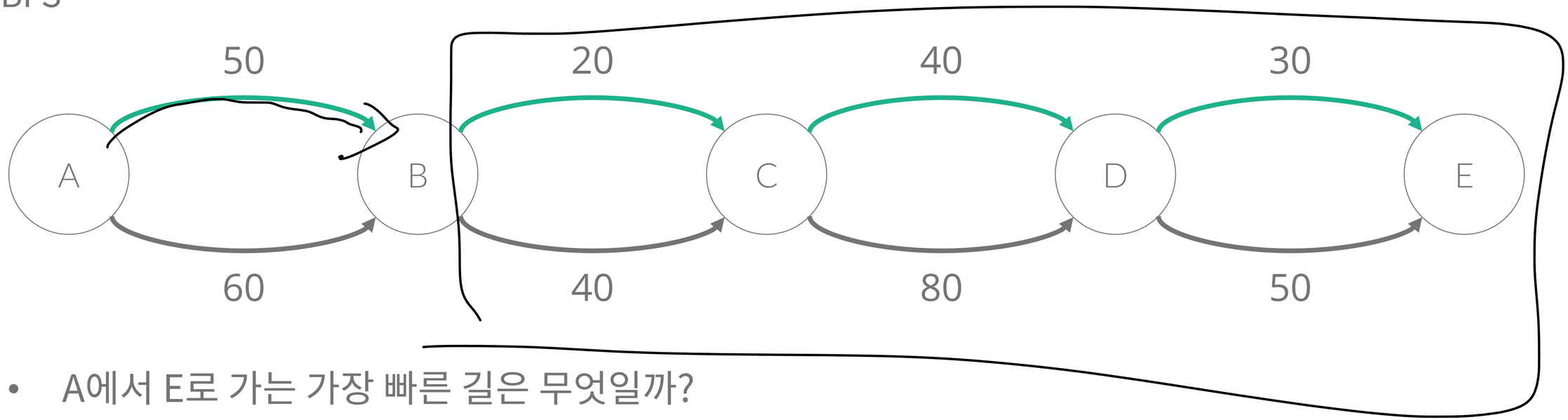
BFS



- A에서 E로 가는 가장 빠른 길은 무엇일까?

BFS

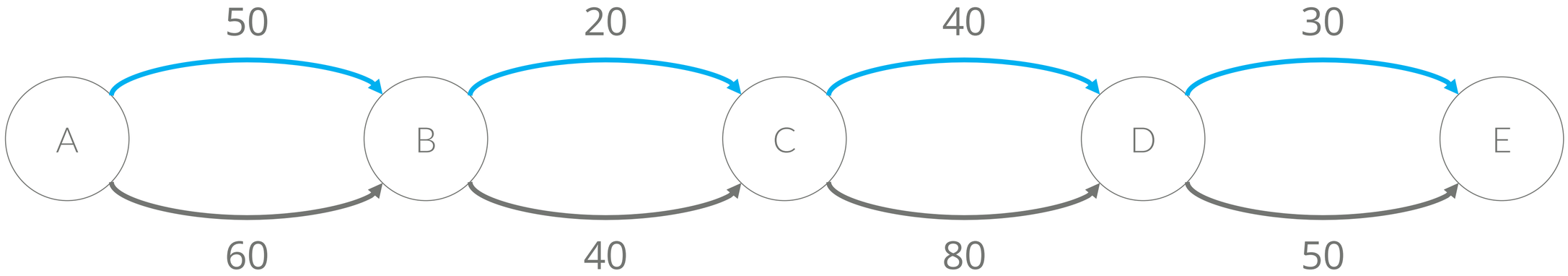
BFS



- A에서 E로 가는 가장 빠른 길은 무엇일까?
- A에서 B로 가는 가장 빠른 길 + B에서 E로 가는 가장 빠른 길

BFS

BFS

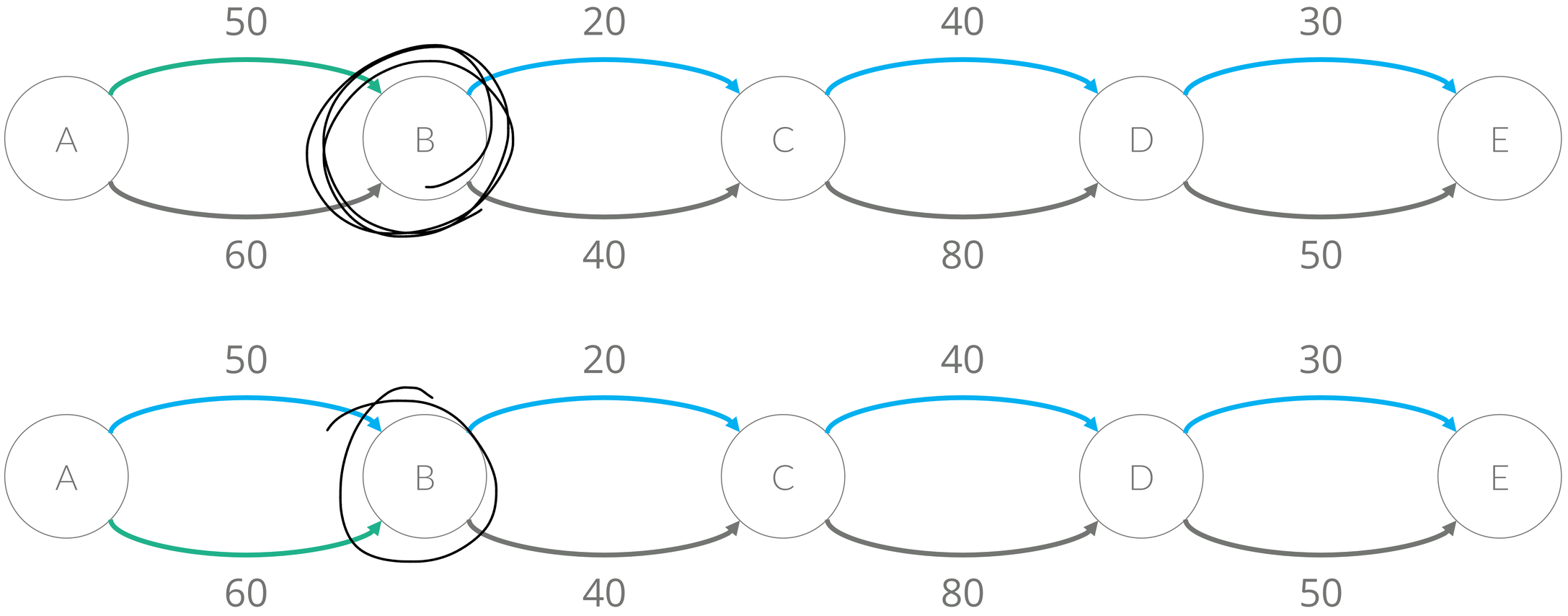


- A에서 E로 가는 가장 빠른 길은 무엇일까?
- 단, 파란 간선은 한 번만 사용할 수 있다.

BFS

BFS

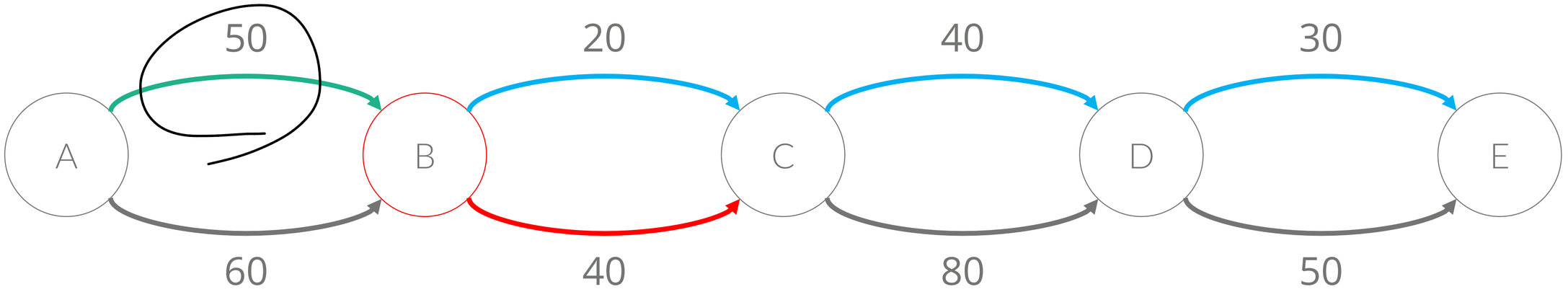
- 파란 간선을 한 번만 사용할 수 있다면, 위 B와 아래 B는 같은 정점이라고 할 수 없다

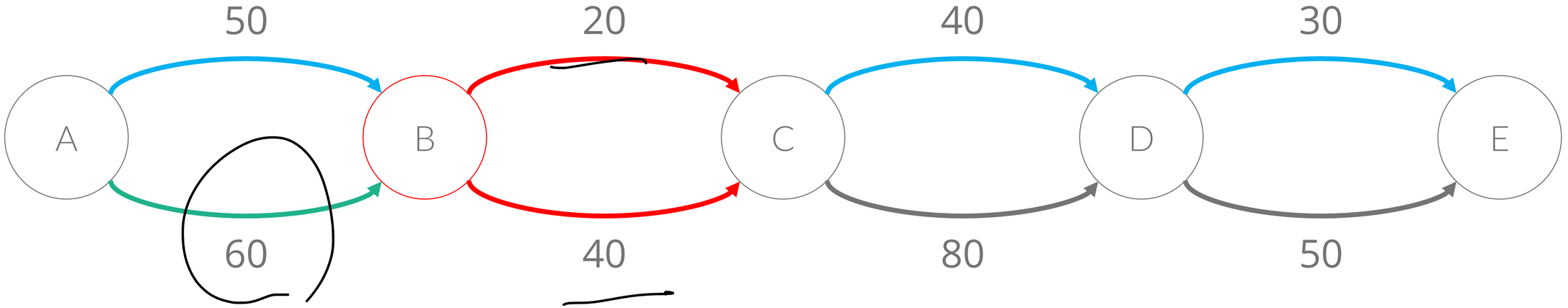


BFS

BFS

- 위 B와 아래 B에서 이동할 수 있는 방법이 다르기 때문에 같은 정점이 아니다





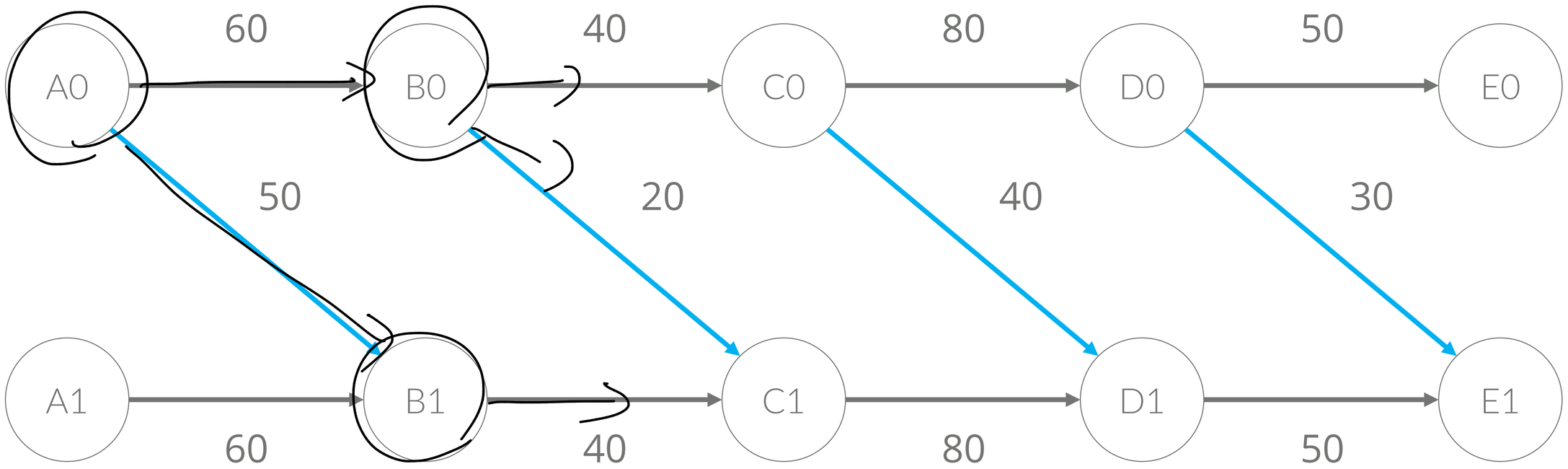
BFS

BFS

- 위 B와 아래 B를 다르다고 하는 기준은 파란 간선을 사용한 횟수이다
- 따라서, 정점을 파란 간선을 사용한 횟수를 기준으로 나눌 수 있다.

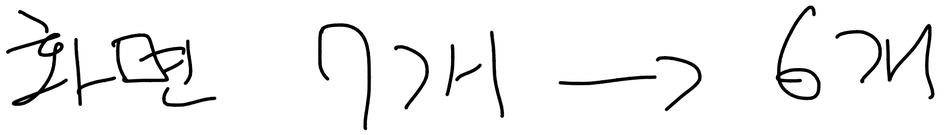
BFS

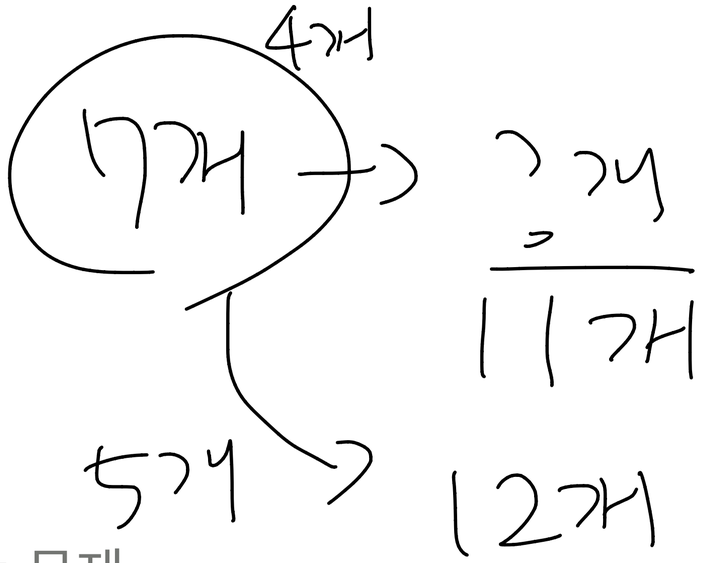
BFS



이모티콘

https://www.acmicpc.net/problem/14226





- 화면에 이모티콘은 1개다
- 할 수 있는 연산
  - 화면에 있는 이모티콘을 모두 복사해서 클립보드에 저장
  - 클립보드에 있는 모든 이모티콘을 화면에 붙여넣기
  - 화면에 있는 이모티콘 중 하나를 삭제
- S개의 이모티콘을 만드는데 걸리는 시간의 최소값을 구하는 문제

이모티콘

https://www.acmicpc.net/problem/14226

- BFS에서 하나의 정점이 서로 다른 두 개의 정보를 저장하고 있으면 안된다
- 화면에 있는 이모티콘의 개수가 5개인 경우
- 클립보드에 있는 이모티콘의 개수에 따라서, 클립보드에서 복사하기 연산의 결과가 다르다
- 즉, 화면에 이모티콘의 개수 s와 클립보드에 있는 이모티콘의 개수 c가 중요하다

이모티콘

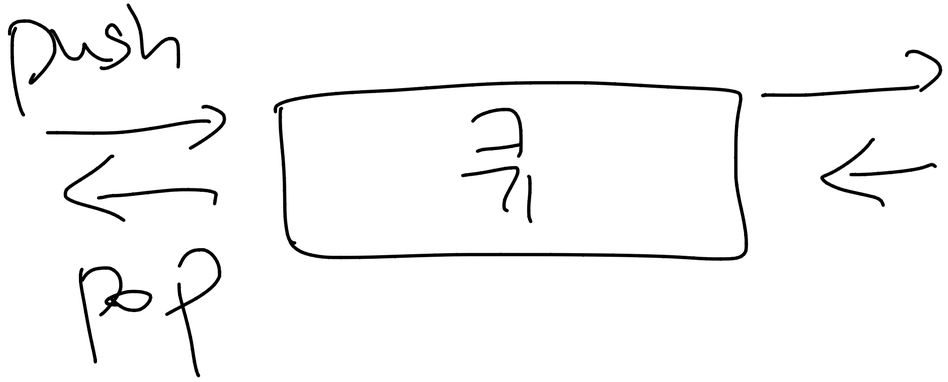
https://www.acmicpc.net/problem/14226

- 복사: (s, c) -> (s, s)
- 붙여넣기: (s, c) -> (s+c, c)
- 삭제: (s, c) -> (s-1, c)
- $2 \leq S \leq 1,000$ 이기 때문에 BFS 탐색으로 가능하다.

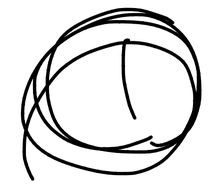
이모티콘

https://www.acmicpc.net/problem/14226

- 소스: http://codeplus.codes/88648734b074475494ad8253b121cc68



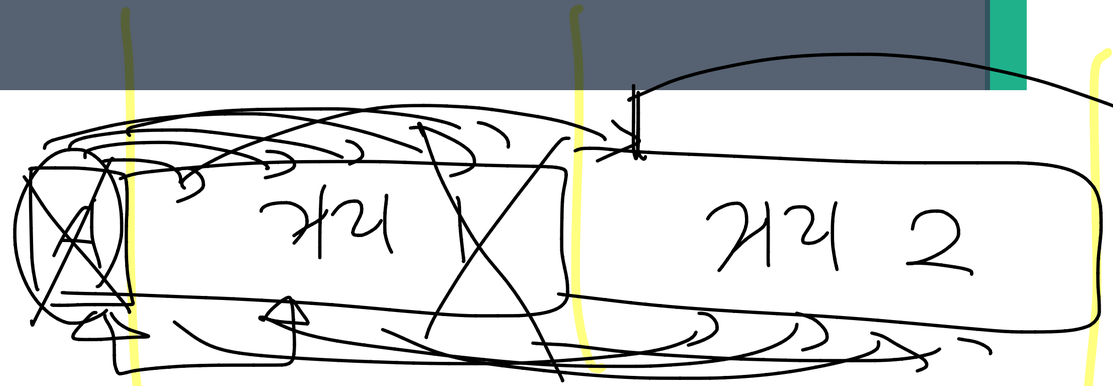
BFS



Deque

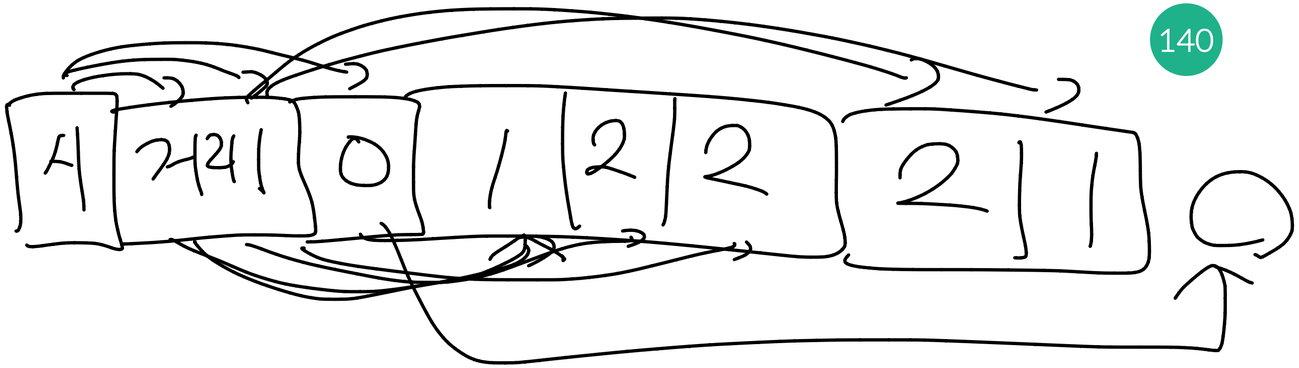
Double ended Queue

덱 사용하기



숨바꼭질 3

https://www.acmicpc.net/problem/13549



- 수빈이의 위치: N
- 동생의 위치: K
- 동생을 찾는 가장 빠른 시간을 구하는 문제

- 수빈이가 할 수 있는 행동 (위치: X)

1. 걷기: X+1 또는 X-1로 이동 (1초)
2. 순간이동: 2*X로 이동 (0초)

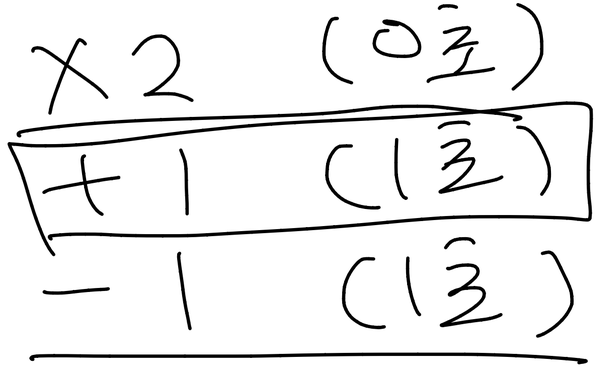
숨바꼭질 3

https://www.acmicpc.net/problem/13549

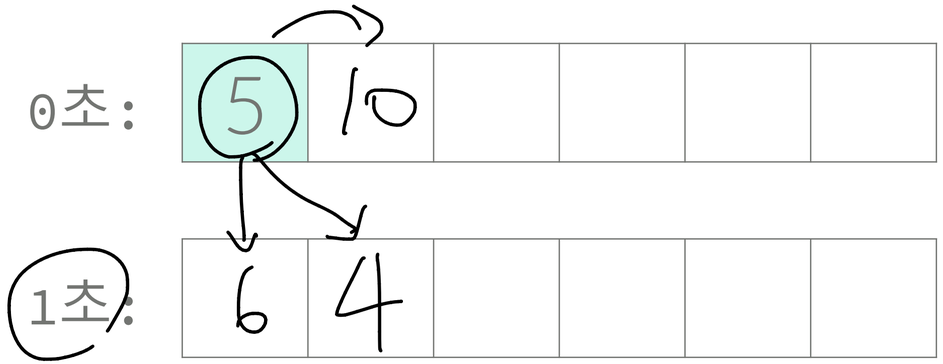
- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정

숨바꼭질 3

https://www.acmicpc.net/problem/13549



- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정



숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정

- 0초:

5	10				

- 1초:

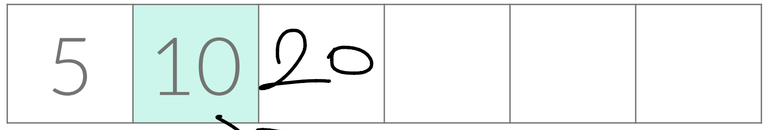
4	6				

숨바꼭질 3

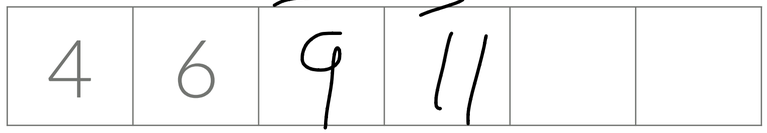
https://www.acmicpc.net/problem/13549

- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정

- 0초:



- 1초:



숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정

- 0초:

5	10	20			

- 1초:

4	6	9	11		

숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정

- 0초:

5	10	20			

- 1초:

4	6	9	11	19	

숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정

- 0초:

5	10	20			

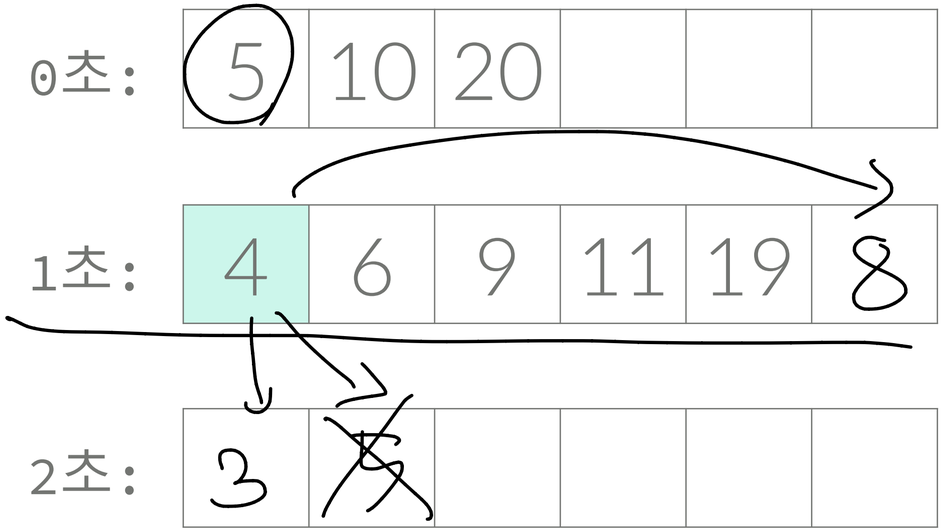
- 1초:

4	6	9	11	19	

숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정



숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정

- 0초:

5	10	20			

- 1초:

4	6	9	11	19	8

- 2초:

3					

숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정

- 0초:

5	10	20			

- 1초:

4	6	9	11	19	8						

- 2초:

3					

숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정

- 0초:

5	10	20			

- 1초:

4	6	9	11	19	8	12					

- 2초:

3	7				

숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정

- 0초:

5	10	20			

- 1초:

4	6	9	11	19	8	12					

- 2초:

3	7				

숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정

- 0초:

5	10	20			

- 1초:

4	6	9	11	19	8	12	18				

- 2초:

3	7	8			

숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정

- 0초:

5	10	20			

- 1초:

4	6	9	11	19	8	12	18				

- 2초:

3	7	8			

숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정

- 0초:

5	10	20			

- 1초:

4	6	9	11	19	8	12	18				

- 2초:

3	7	8			

숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정

- 0초:

5	10	20			

- 1초:

4	6	9	11	19	8	12	18				

- 2초:

3	7	8			

숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 5에서 17을 가는 경우 0 ~ 20까지만 위치가 있다고 가정

- 0초:

5	10	20			

- 1초:

4	6	9	11	19	8	12	18	16			

- 2초:

3	7	8			

숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 이런식으로 BFS를 진행한다.

숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 덱을 사용해 순간 이동은 덱의 앞에, 걷기는 덱의 뒤에 넣는 방법도 생각해 볼 수 있다.

숨바꼭질 3

https://www.acmicpc.net/problem/13549

- 큐 소스: http://codeplus.codes/337f9cfd8fa1431d8f1db309d996d70e
- 덱 소스: http://codeplus.codes/969bf7744a9b4e2d86231e97e243e5fc

알고스팟

https://www.acmicpc.net/problem/1261

- 미로는 N*M크기이고, 총 1*1크기의 방으로 이루어져 있다
- 빈 방은 자유롭게 다닐 수 있지만, 벽은 부수지 않으면 이동할 수 없다
- (x, y)에 있을 때, 이동할 수 있는 방은 (x+1, y), (x-1, y), (x, y+1), (x, y-1) 이다
- (1, 1)에서 (N, M)으로 이동하려면 벽을 최소 몇 개 부수어야 하는지 구하는 문제

알고스팟

https://www.acmicpc.net/problem/1261

- 처음 상태

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

알고스팟

https://www.acmicpc.net/problem/1261

- 벽을 부수지 않고 이동할 수 있는 곳

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

0	0				
0					
0	0				

알고스팟

https://www.acmicpc.net/problem/1261

- 벽을 1개 부수고 이동할 수 있는 곳

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

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

알고스팟

https://www.acmicpc.net/problem/1261

- 벽을 2개 부수고 이동할 수 있는 곳

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

0	0	1	2	2	2
0	1	1	1	1	1
0	0	1	2	2	2
1	1	1	1	1	2
1	2	1	1	2	2
2	1	1	1	2	2

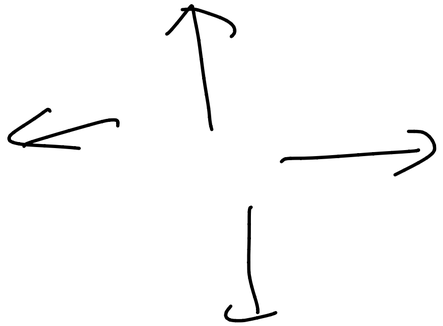
알고스팟

https://www.acmicpc.net/problem/1261

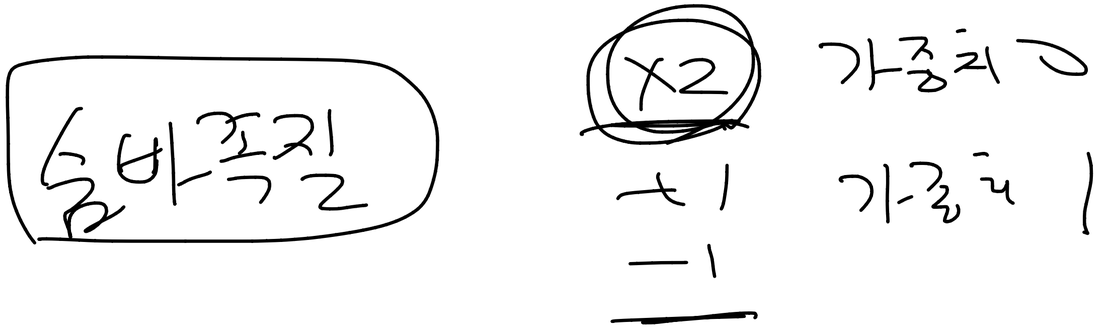
- BFS탐색을 벽을 부순 횟수에 따라서 나누어서 수행해야 한다.
- 소스: http://codeplus.codes/39b009ac78f14b3d844d2e5129c728b3

알고스팟

https://www.acmicpc.net/problem/1261



- 어차피 벽을 뚫는다와 안 뚫는다로 나누어지기 때문에, 덱을 사용한다
- 벽을 뚫는 경우에는 뒤에, 안 뚫는 경우에는 앞에 추가한다.
- 소스: http://codeplus.codes/06807df347e5477aa21bb52572e0c246

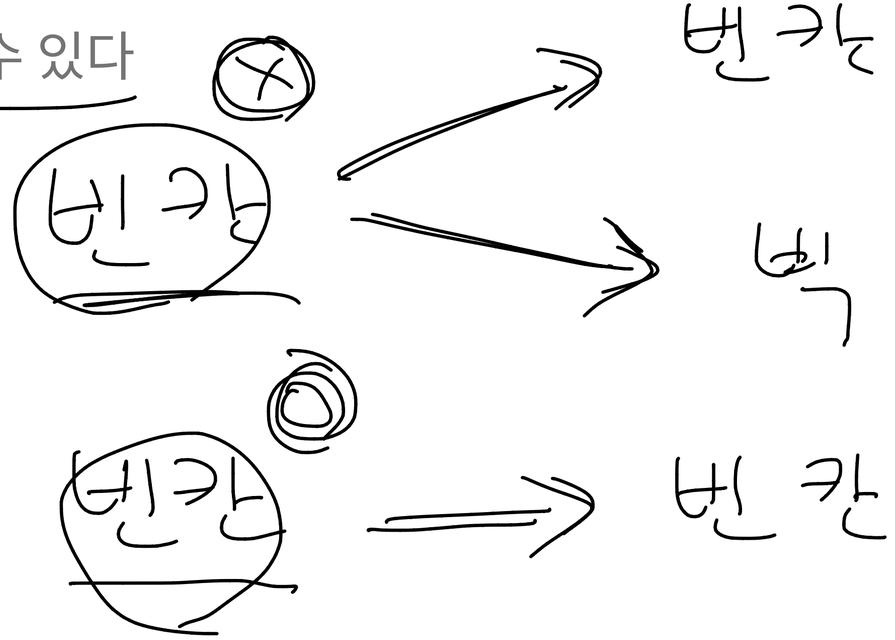


BFS

벽 부수고 이동하기

https://www.acmicpc.net/problem/2206

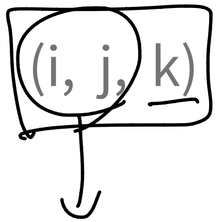
- NxM의 행렬로 나타내는 지도에서 (1, 1)에서 (N,M)으로 최단 거리로 이동하는 문제
- 0은 빈 칸, 1은 벽
- 단, 벽은 한 번 부수고 지나갈 수 있다



벽 부수고 이동하기

https://www.acmicpc.net/problem/2206

- 벽을 부순다는 조건이 없으면 일반적인 미로 탐색 문제이다
- 어떤 칸에 방문했을 때, 벽을 부순 적이 있는 경우와 아직 부순 적이 없는 경우는 다른 경우 이기 때문에
- 상태 (i, j) 대신에 (i, j, k) (k == 0이면 벽을 부순 적이 없음, 1이면 있음) 으로 BFS 탐색을 진행한다.



벽 부수고 이동하기

https://www.acmicpc.net/problem/2206

- 소스: http://codeplus.codes/3199c842c7d14ea99aab9ddf29150ce5

탈출

https://www.acmicpc.net/problem/3055

- 지도는 R행 C열이다
- 비어있는 곳은 '.'
- 물이 차있는 지역은 '*'
- 돌은 'X'
- 비버의 굴은 'D'
- 고슴도치의 위치는 'S'

탈출

https://www.acmicpc.net/problem/3055

- 먼저, 물이 언제 차는지 미리 구해놓은 다음에
- 고슴도치를 그 다음에 이동시킨다

탈출

https://www.acmicpc.net/problem/3055

- 지도 상태

.	D	.	*
.	.	.	.
	.	X	.
S	.	*	.
.	.	.	.

- 물이 차는 시간

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

- 고슴도치의 이동

0			

탈출

https://www.acmicpc.net/problem/3055

- 지도 상태

.	D	.	*
.	.	.	.
.	.	X	.
S	.	*	.
.	.	.	.

- 물이 차는 시간

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

- 고슴도치의 이동

1			
0			
1			

탈출

https://www.acmicpc.net/problem/3055

- 지도 상태

.	D	.	*
.	.	.	.
.	.	X	.
S	.	*	.
.	.	.	.

- 물이 차는 시간

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

- 고슴도치의 이동

2			
1			
0			
1			

탈출

https://www.acmicpc.net/problem/3055

- 지도 상태

.	D	.	*
.	.	.	.
.	.	X	.
S	.	*	.
.	.	.	.

- 물이 차는 시간

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

- 고슴도치의 이동

3			
2			
1			
0			
1			

탈출

https://www.acmicpc.net/problem/3055

- 지도 상태

.	D	.	*
.	.	.	.
.	.	X	.
S	.	*	.
.	.	.	.

- 물이 차는 시간

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

- 고슴도치의 이동

3	4		
2			
1			
0			
1			

탈출

https://www.acmicpc.net/problem/3055

- 소스: http://codeplus.codes/63dbd8fc3bdd44a78e5cce72da82fa8d

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.