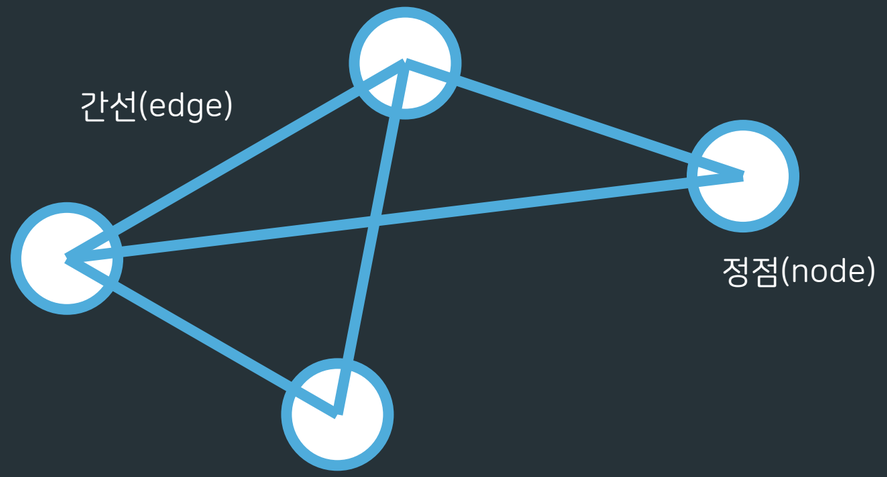
알튜비튜

DFS & BFS

오늘은 그래프 탐색 알고리즘인 깊이 우선 탐색(DFS)과 너비 우선 탐색(BFS)를 배웁니다.
앞으로 배울 그래프 알고리즘의 시작이자 코딩테스트에 높은 확률로 한 문제 이상 나오는 알고리즘이죠.

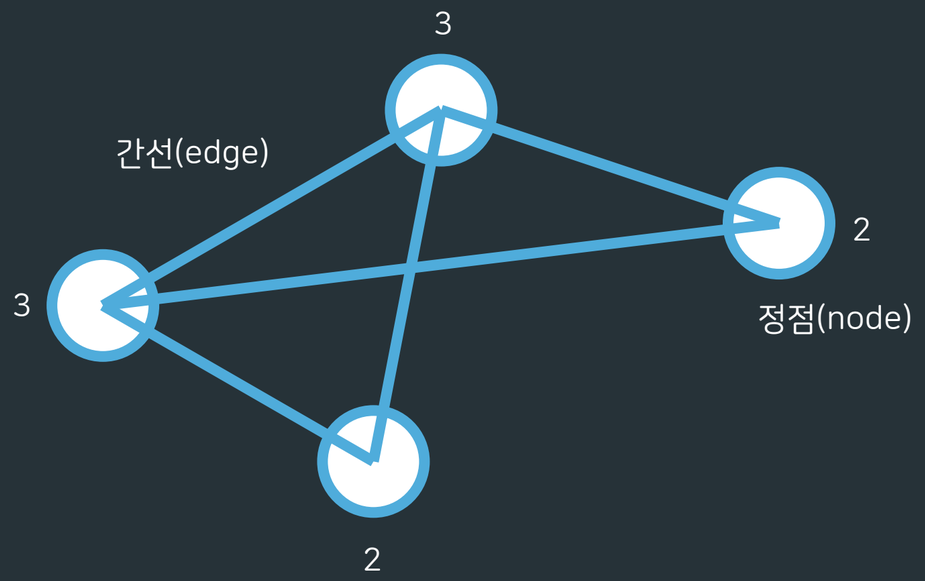
이거 먼저 봅시다!



그래프

- 정점(node)과 그 정점을 연결하는 간선(edge)으로 이루어진 자료구조
- 간선의 방향은 단방향(방향 그래프)이나 양방향(무방향 그래프)으로 나뉨
- 간선에 가중치가 있을 수 있음

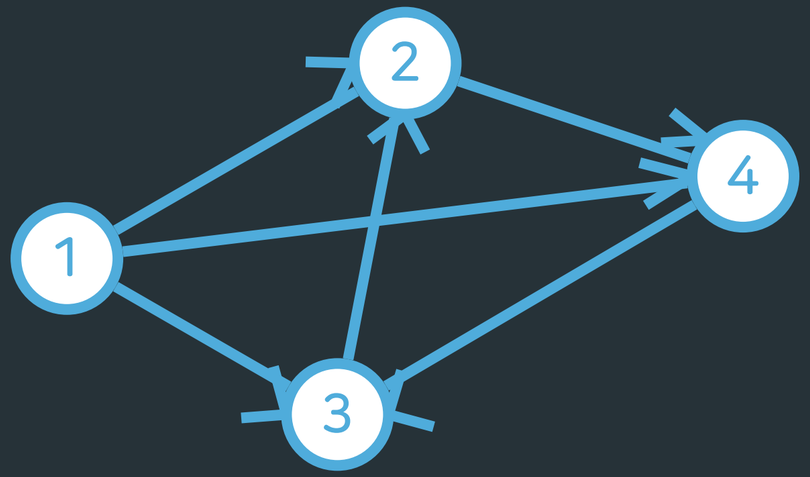
그래프 용어



그래프

- degree(차수): 무방향 그래프에서 정점에 연결된 간선의 수
  - 무방향 그래프의 총 차수(degree)의 합 = 간선의 수 x 2
- indegree: 방향 그래프에서 해당 정점으로 들어오는 간선의 수
- outdegree: 방향 그래프에서 해당 정점에서 나가는 간선의 수
  - 방향 그래프의 총 차수(indegree, outdegree)의 합 = 간선의 수

그래프 경로

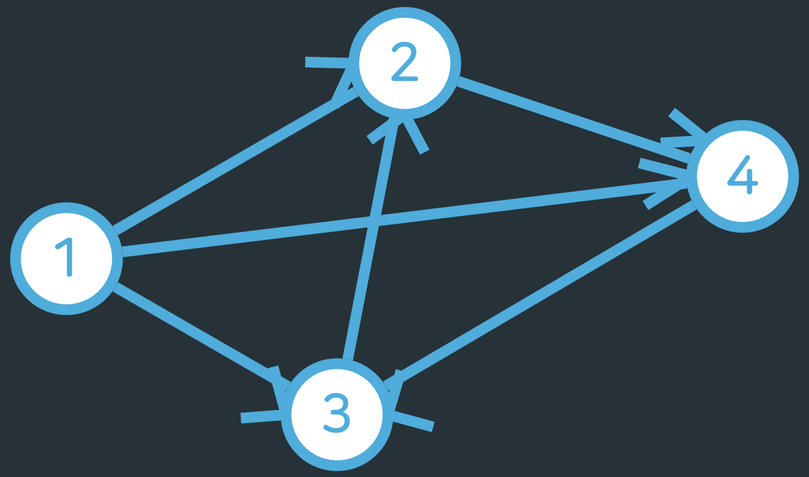


- 정점 1 → 4 경로
  - 1 → 4
  - 1 → 2 → 4
  - 1 → 3 → 2 → 4

경로(path)

- 경로는 여러 개일 수 있다
- 사이클(cycle): 한 정점에서 다시 동일한 정점으로 돌아오는 경로 (ex. 3 → 2 → 4)

그래프 구현



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

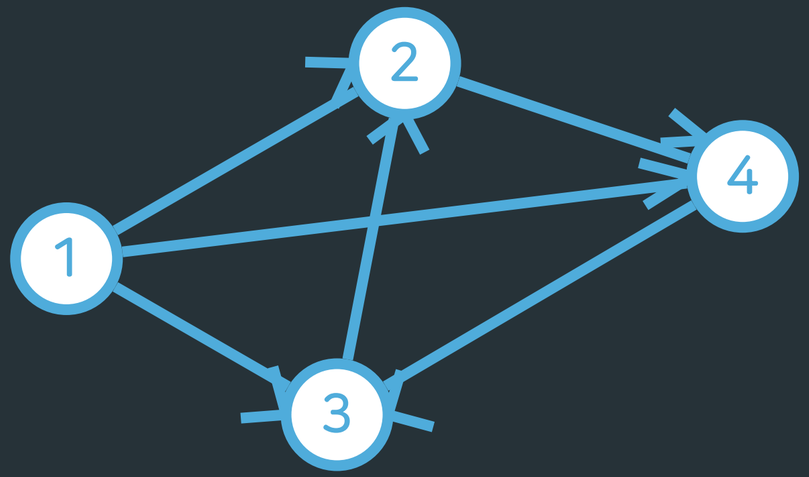
* 만약 가중치가 주어졌다면 가중치 값 저장

인접 행렬 (Adjacency Matrix)

- 두 정점 간의 연결 관계를 N * N 크기의 행렬(2차원 배열)로 나타냄
- 두 정점 i - j 간의 연결 관계 확인: O(1)
- 특정 노드와 연결되어 있는 모든 노드 확인: O(N)
- 공간 복잡도: $O(N^2)$

→ 정점이 많아지면 사용 불가 (메모리 초과)

그래프 구현



1	2	3	4
2	4		
3	2		
4	3		

인접 리스트 (Adjacency List)

- 각 정점에 연결된 정점들을 리스트(1차원 배열)에 담아 보관
- 두 정점 i - j 간의 연결 관계 확인: O(min(degree(i),degree(j)))
- 특정 노드(i)와 연결되어 있는 모든 노드 확인: O(degree(i))
- 공간 복잡도: O(N+E)

인접 행렬 vs 인접 리스트

#알튜비튜

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

인접 행렬

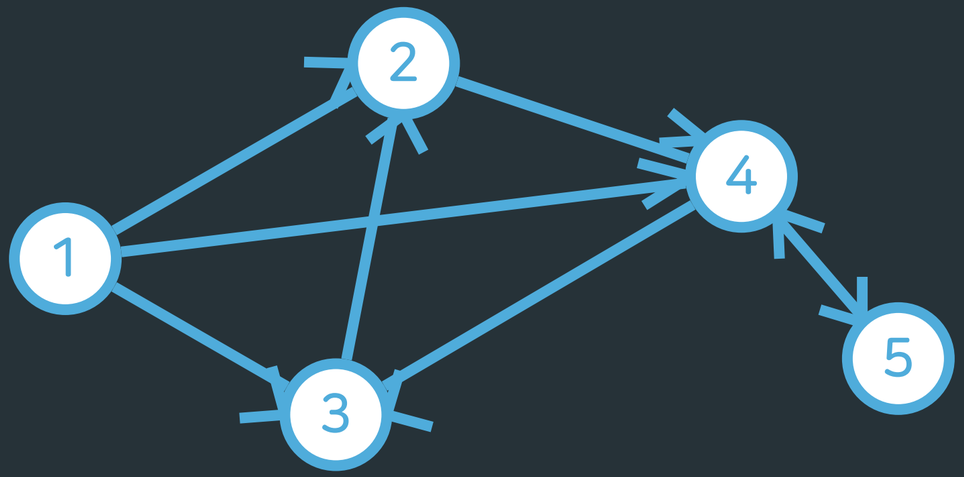
- 정점 사이의 연결 관계(간선)가 많을 경우
- 특정 정점과 정점의 연결 관계를 많이 확인해야 하는 경우

1	2	3	4
2	4		
3	2		
4	3		

인접 리스트

- 정점 사이의 연결관계(간선)가 적은 경우
- 연결된 정점들을 탐색해야 하는 경우

그래프 탐색



그래프 탐색

- 그래프의 모든 노드를 탐색하기 위해 간선을 따라 순회하는 것
- 탐색 방법에 따라 BFS(너비 우선 탐색), DFS(깊이 우선 탐색)로 나뉨
- 탐색 시, 방문 체크 꼭 필요
- 그래프는 가장 폭 넓고 깊은 주제 중 하나인데, 탐색이 그 중 기본이라 할 수 있음
- 시간 복잡도: 인접 행렬로 구현 시 $O(N^2)$, 인접 리스트로 구현 시 O(N+E)

그래프 탐색

#알튜비튜

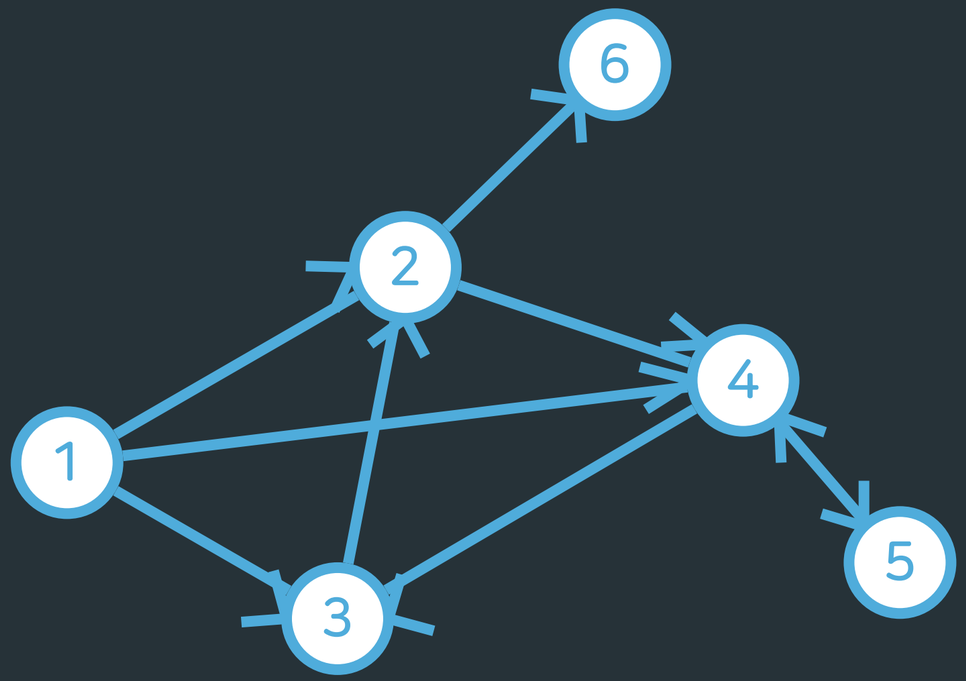
DFS (깊이 우선 탐색)

- 최대한 깊게 탐색 후 빠져 나옴
- 한 정점을 깊게 탐색해서 빠져 나왔다면, 나머지 정점 계속 동일하게 탐색
- 스택(stack), 재귀함수로 구현

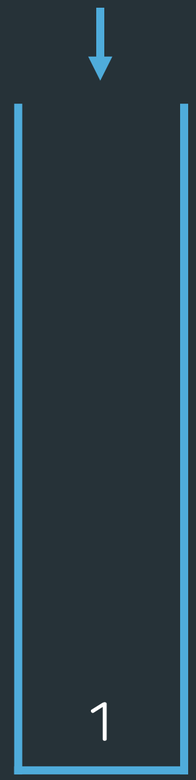
BFS (너비 우선 탐색)

- 자신의 자식들부터 순차적으로 탐색
- 순차 탐색 이후, 다른 정점의 자식들 탐색
- 큐(queue)로 구현

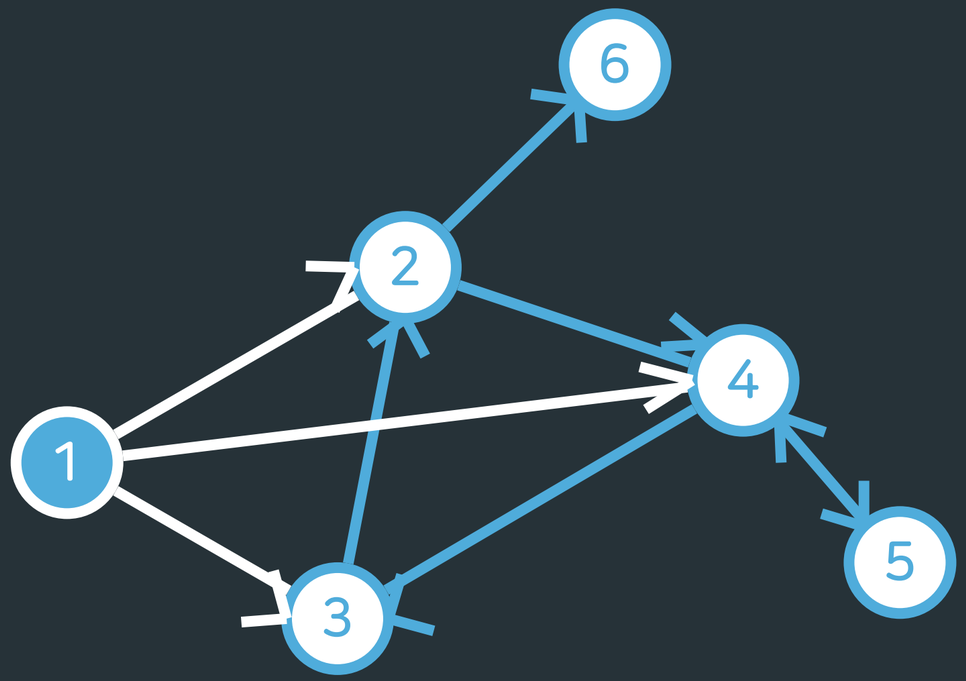
DFS

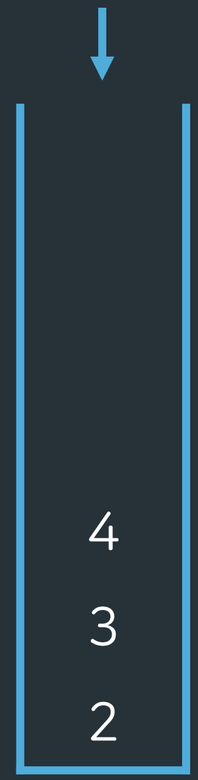


* 탐색은 어디에서든 시작 가능. 시작 노드는 주로 문제에서 주어줌



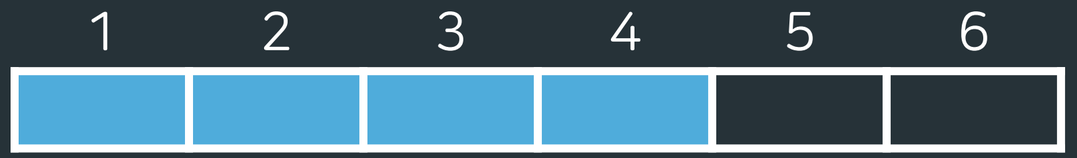
DFS



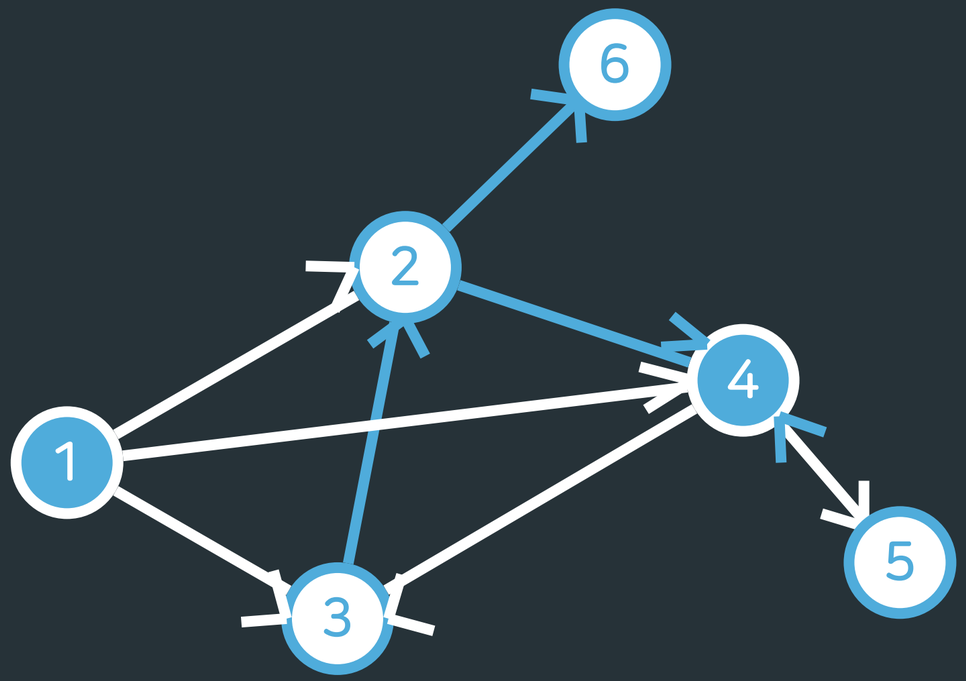


- 탐색 순서: 1

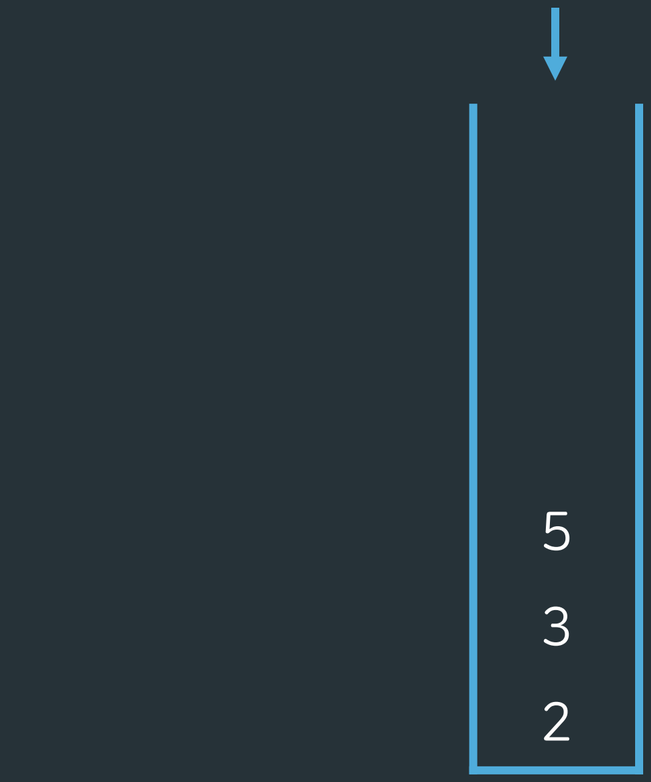
스택에 넣을 때 방문 체크! →

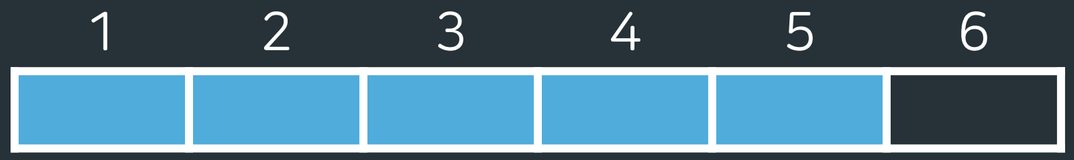


DFS

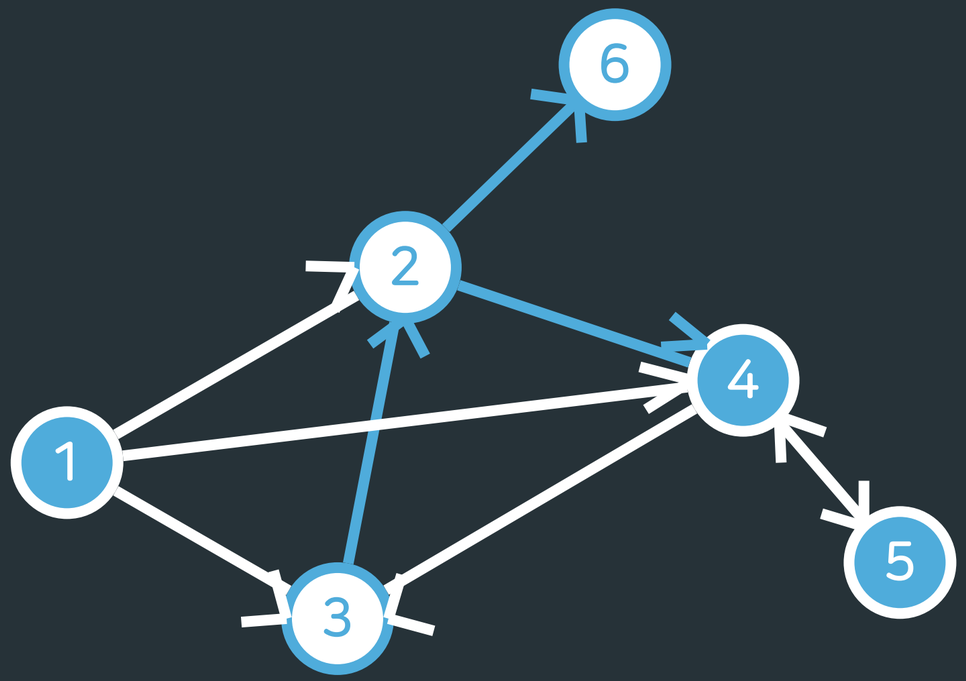


- 탐색 순서: 1 → 4

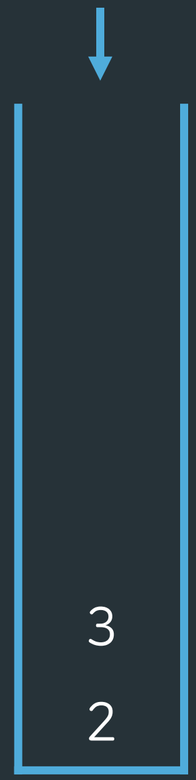


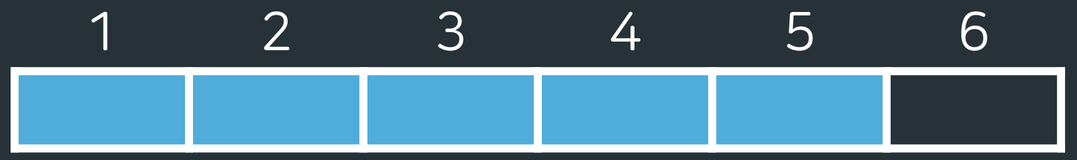


DFS

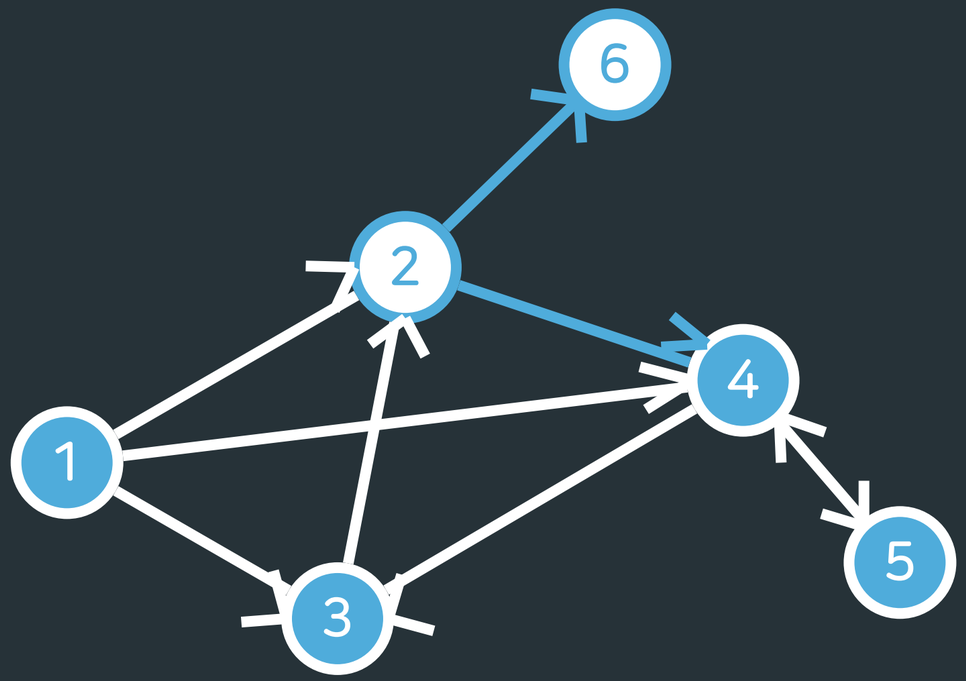


- 탐색 순서: 1 → 4 → 5

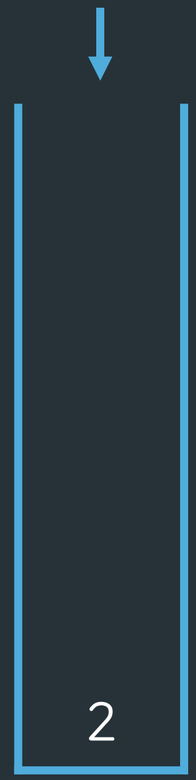


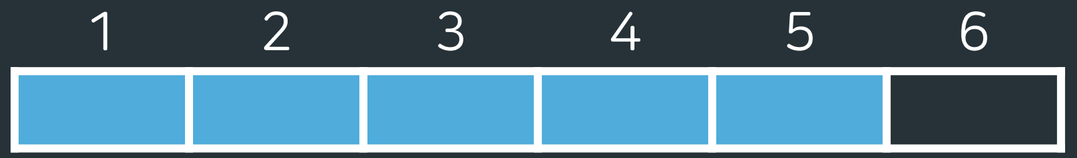


DFS

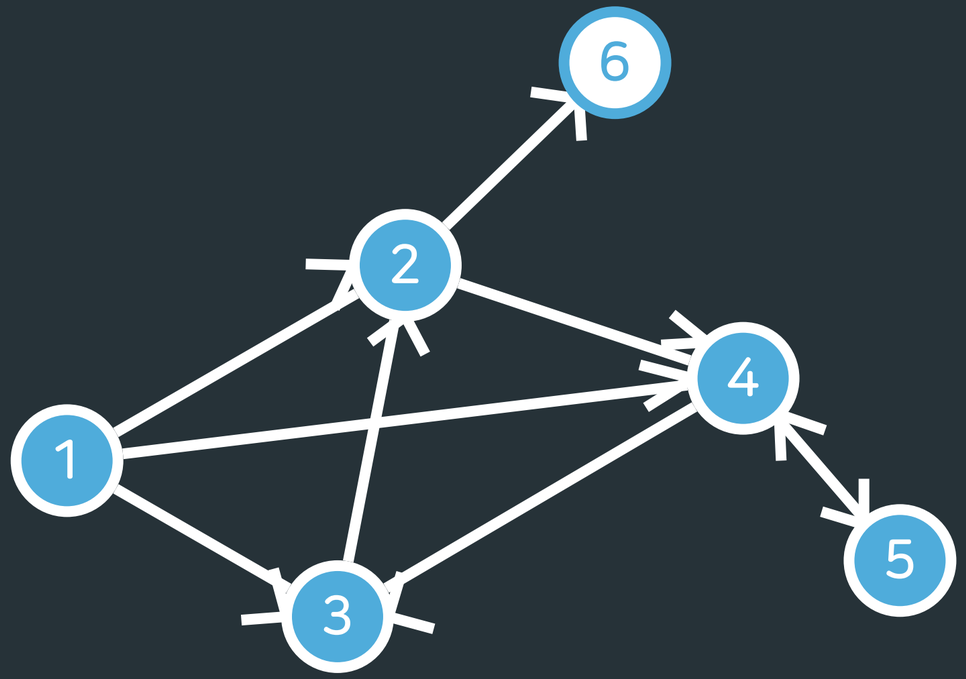


- 탐색 순서: 1 → 4 → 5 → 3

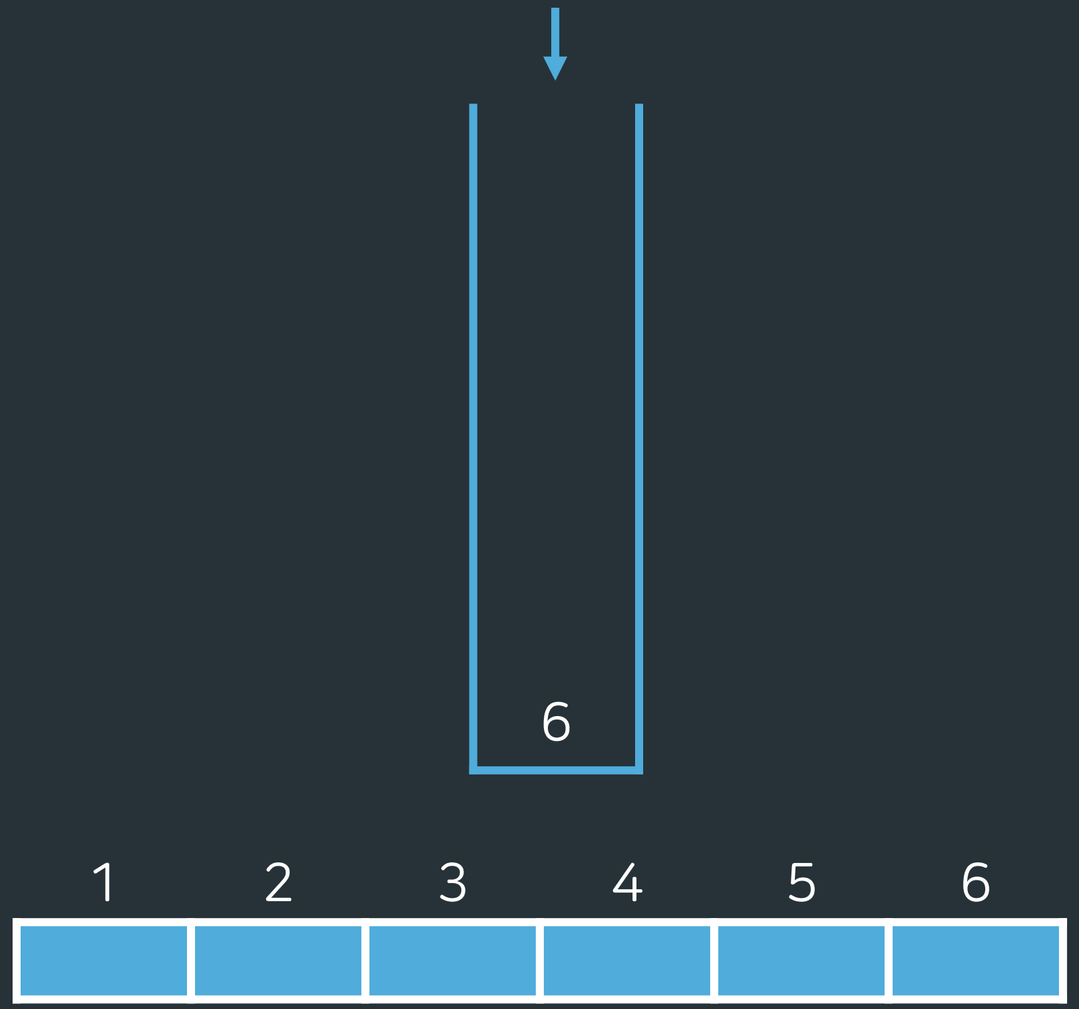




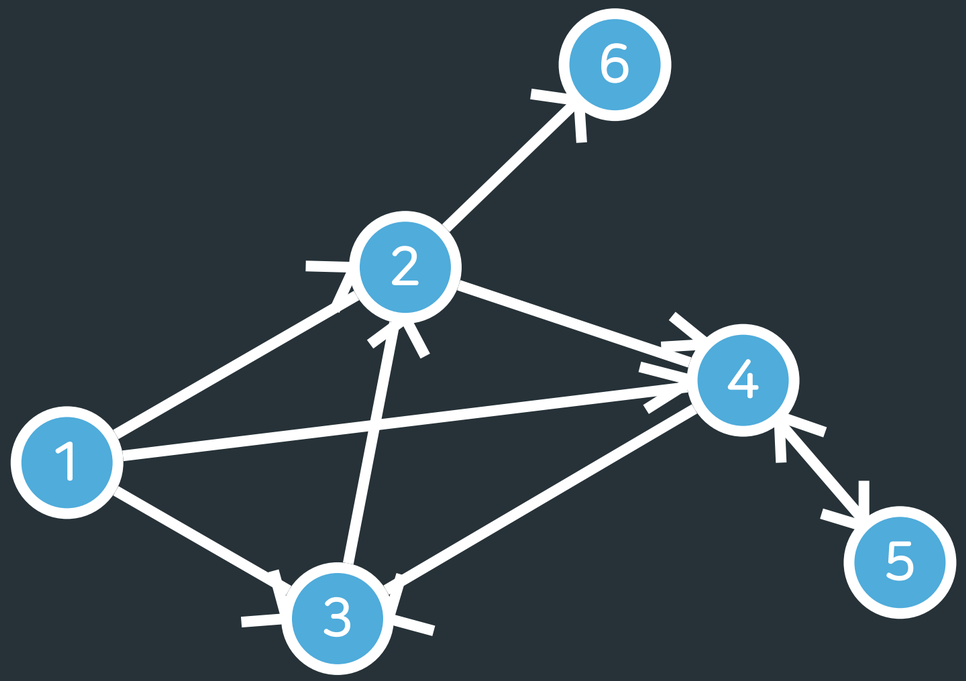
DFS



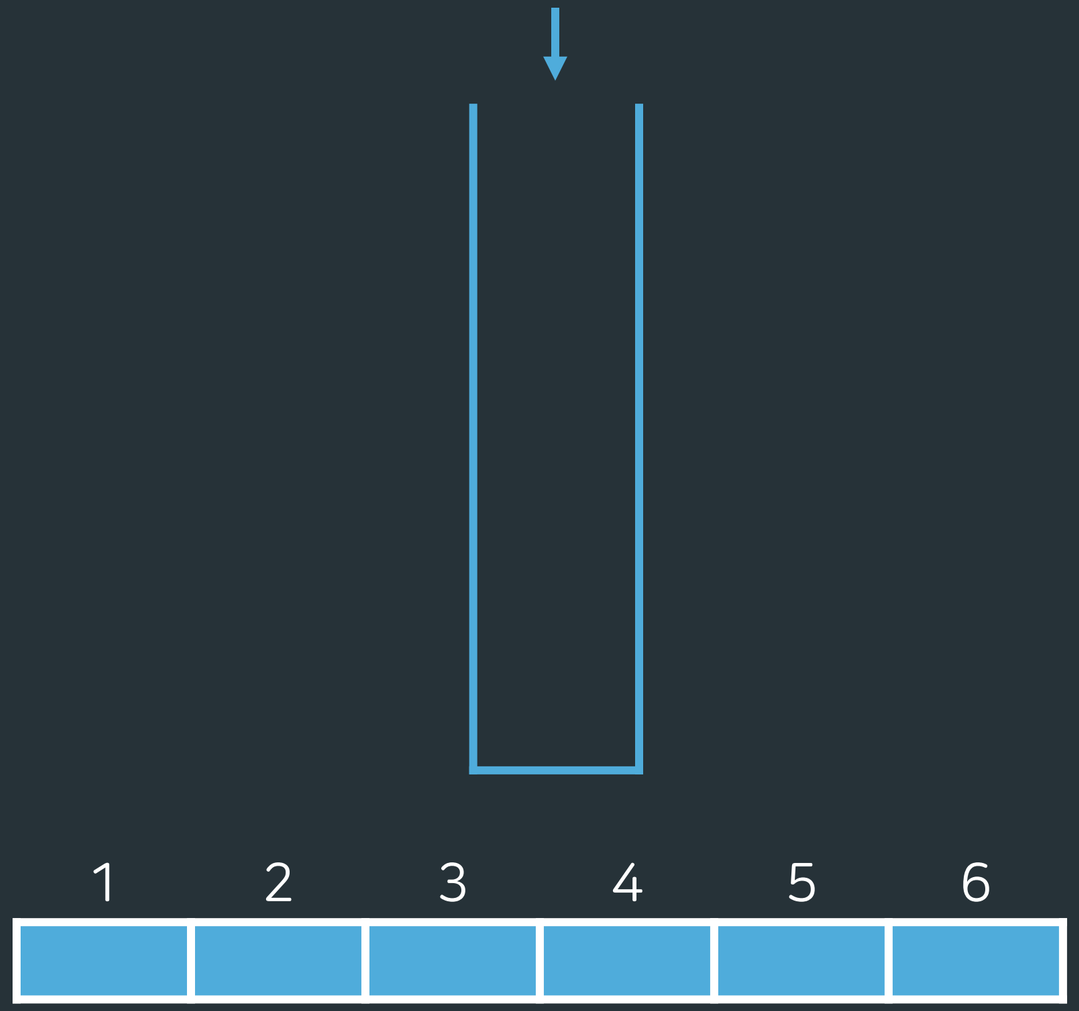
- 탐색 순서: 1 → 4 → 5 → 3 → 2



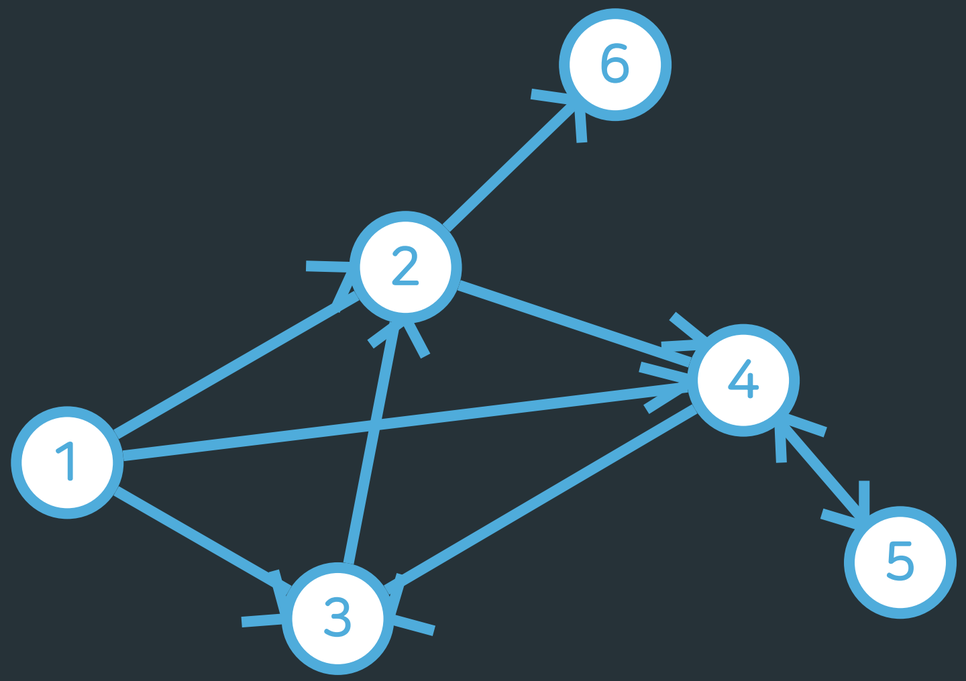
DFS

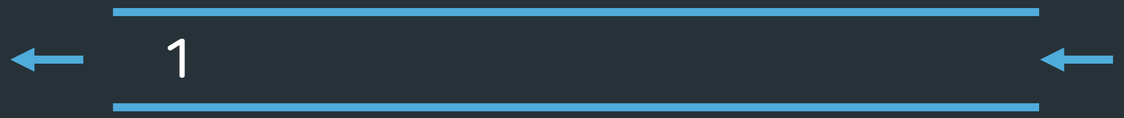


- 탐색 순서: 1 → 4 → 5 → 3 → 2 → 6



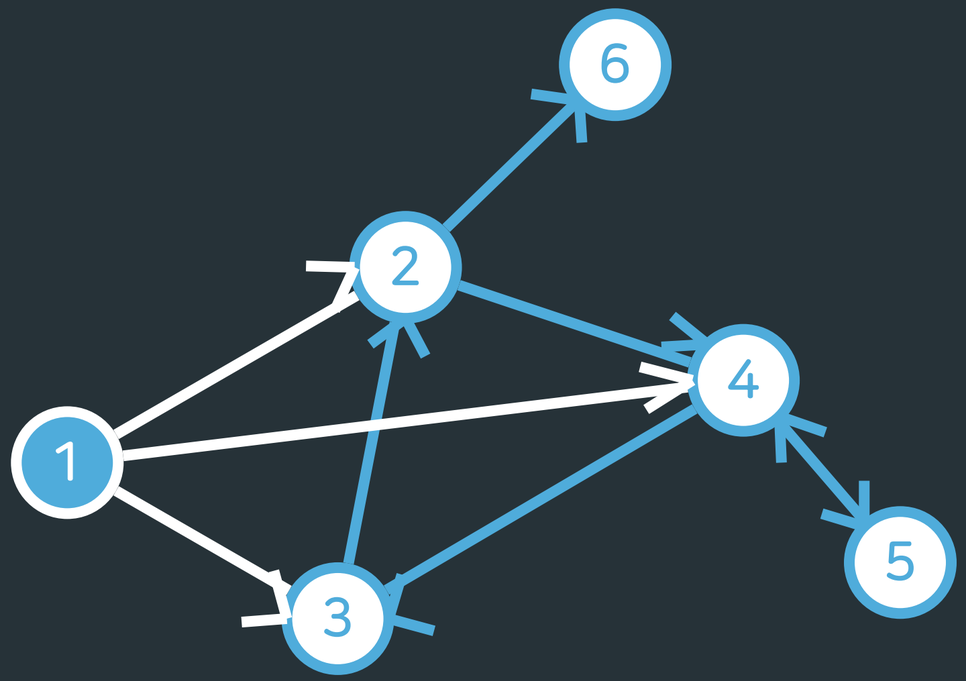
BFS

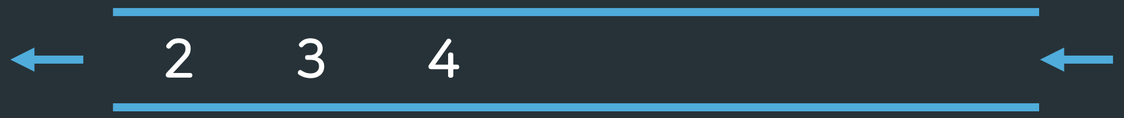




* 탐색은 어디에서든 시작 가능. 시작 노드는 주로 문제에서 주어줌

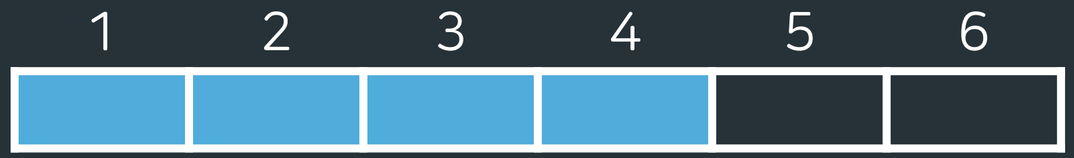
BFS



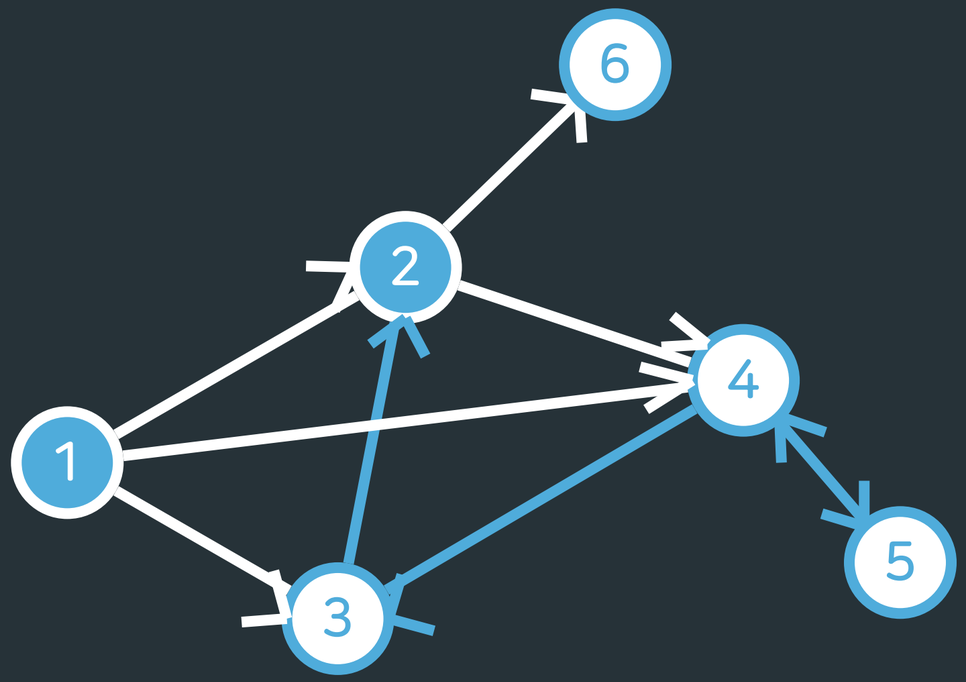


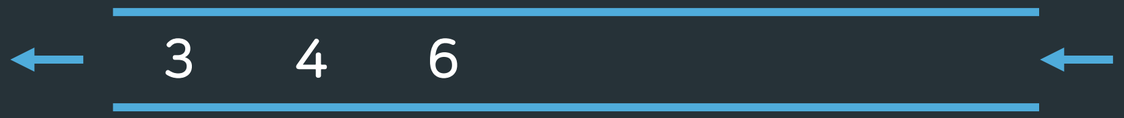
- 탐색 순서: 1

큐에 넣을 때 방문 체크! →

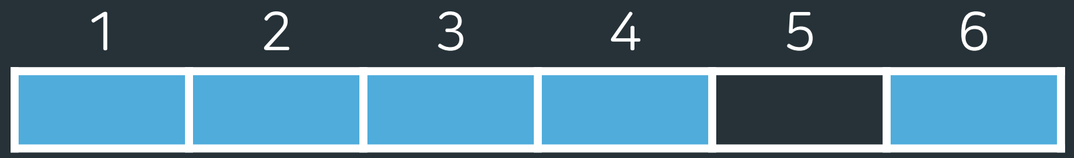


BFS

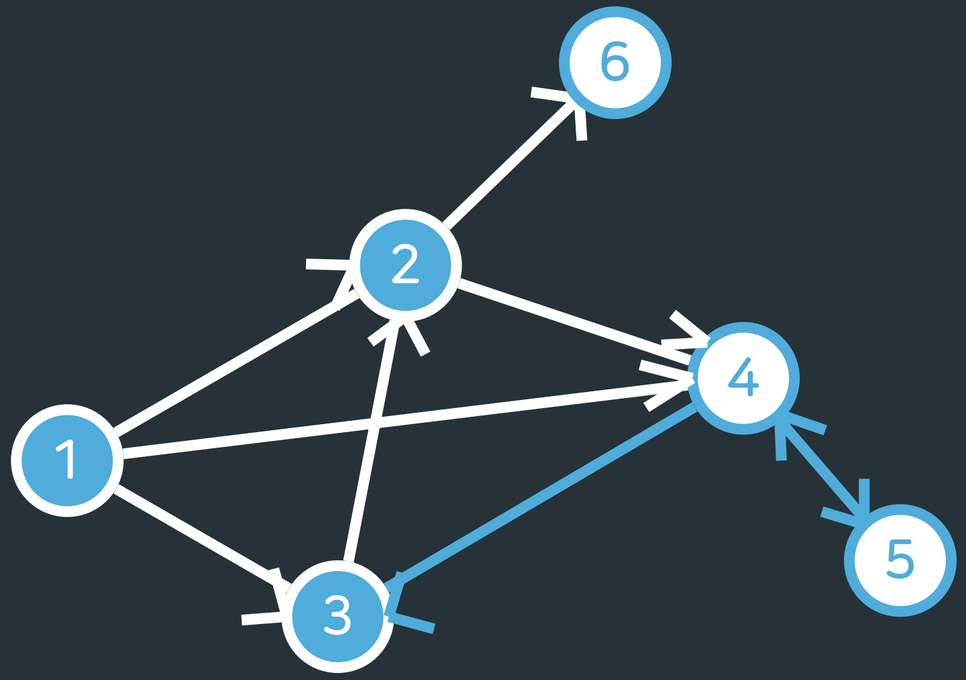


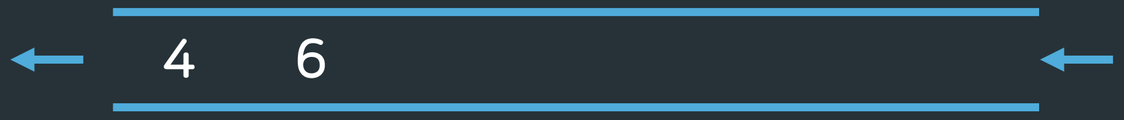


- 탐색 순서: 1 → 2

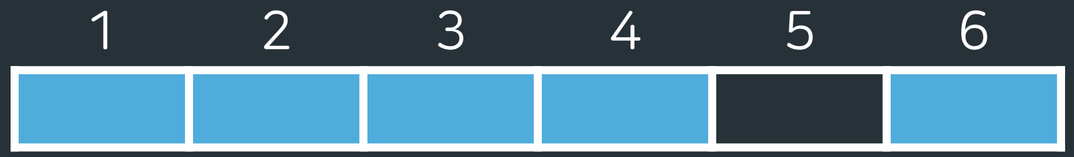


BFS

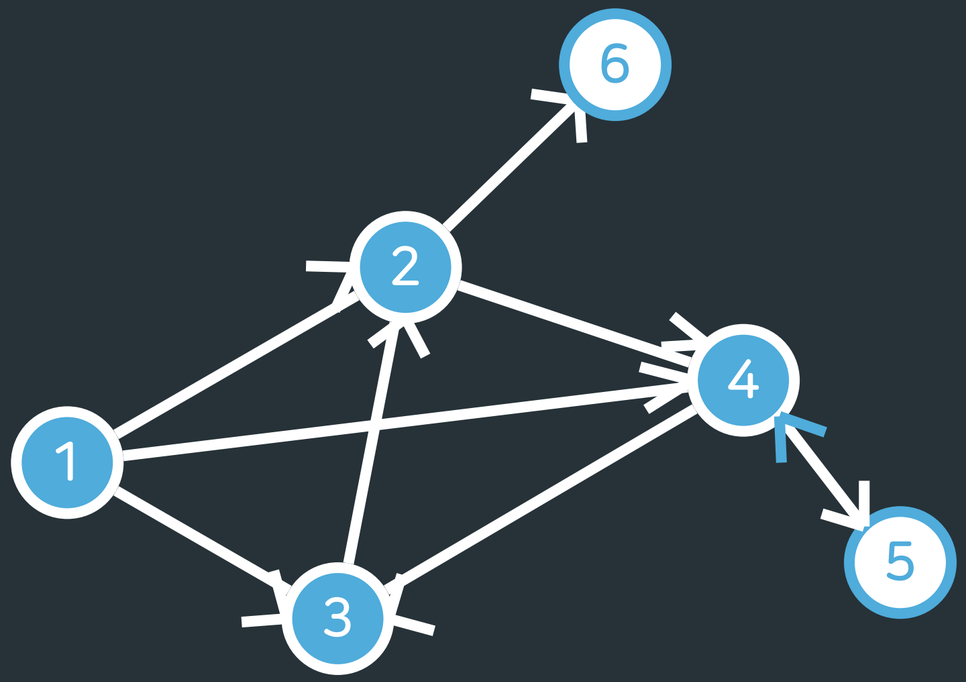


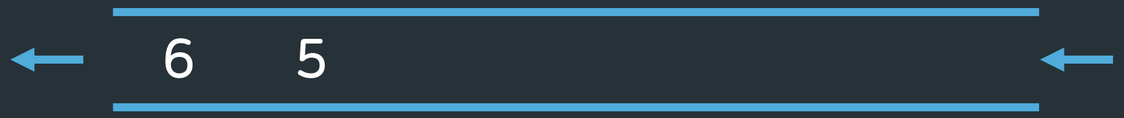


- 탐색 순서: 1 → 2 → 3

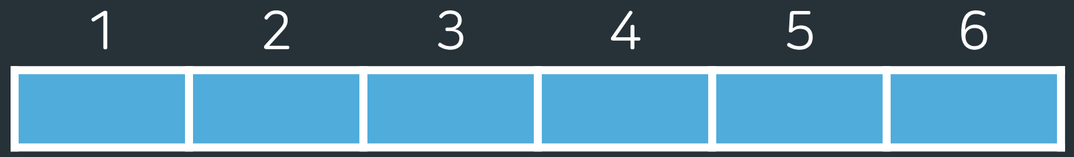


BFS

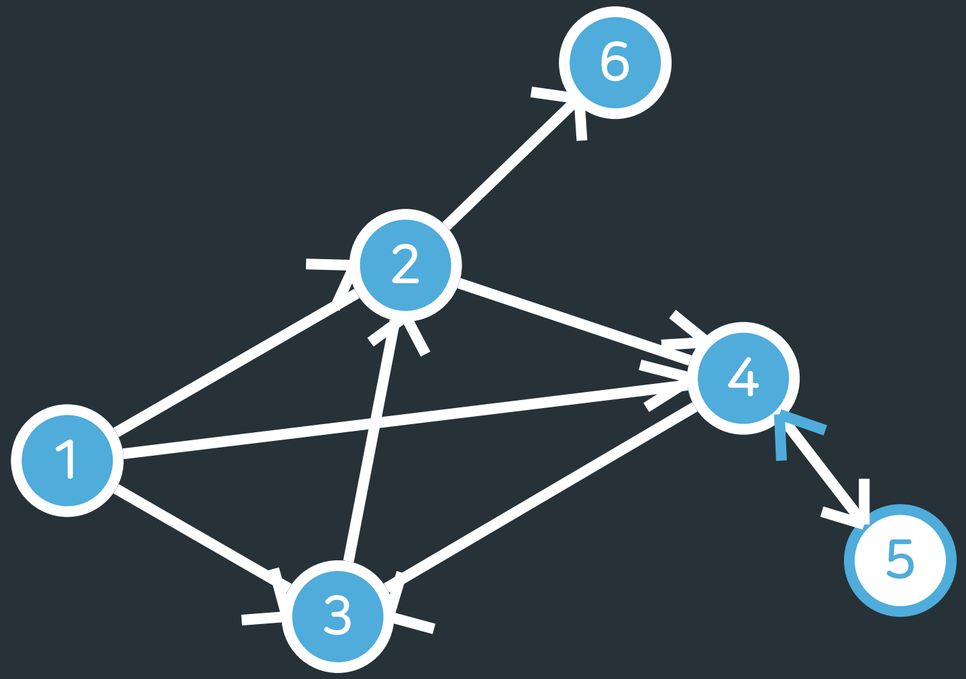


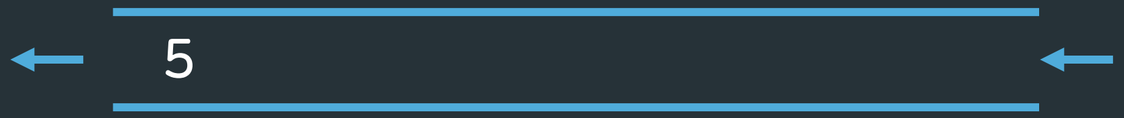


- 탐색 순서: 1 → 2 → 3 → 4

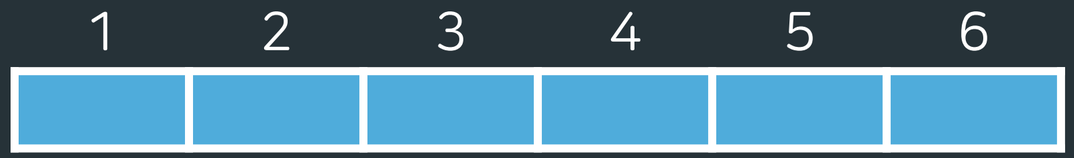


BFS

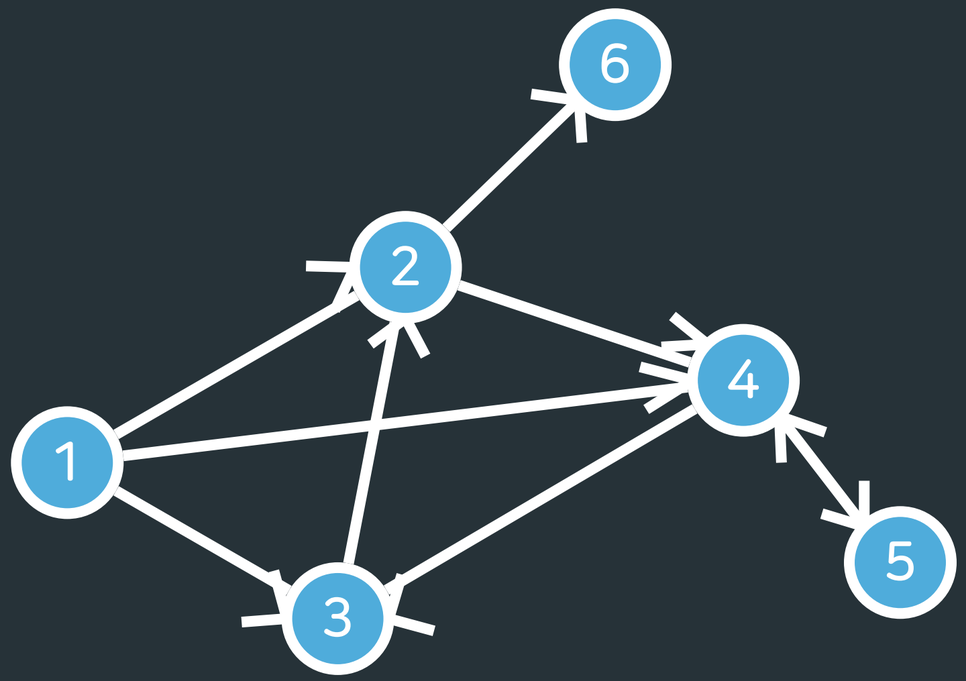


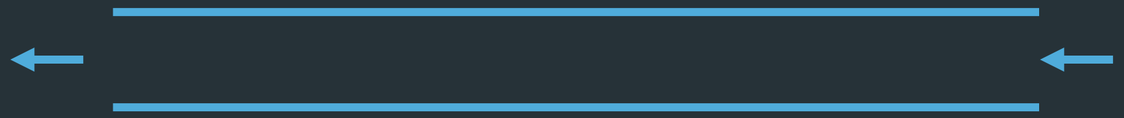


- 탐색 순서: 1 → 2 → 3 → 4 → 6

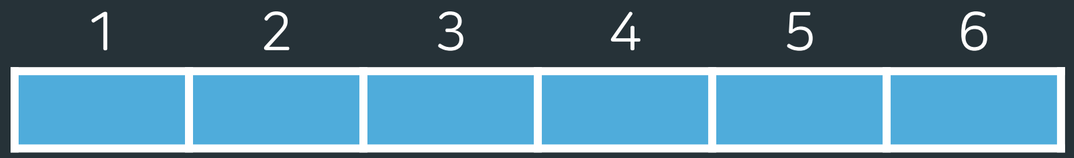


BFS

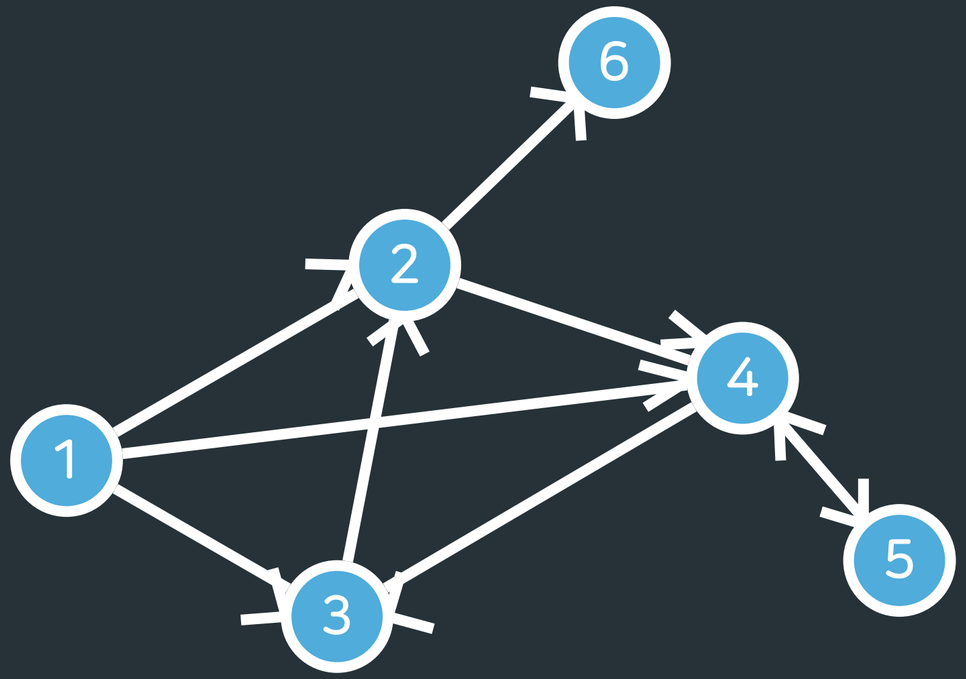




- 탐색 순서: 1 → 2 → 3 → 4 → 6 → 5



DFS vs BFS



DFS

- 탐색 순서: 1 → 4 → 5 → 3 → 2 → 6

BFS

- 탐색 순서: 1 → 2 → 3 → 4 → 6 → 5

#알튜비튜

1260번 : DFS와 BFS - Silver 2

문제

- 그래프를 DFS로 탐색한 결과와 BFS로 탐색한 결과를 출력하는 문제
- 단, 방문할 수 있는 정점이 여러 개인 경우, 정점 번호가 작은 것부터 방문

제한 사항

- 정점 개수 N의 범위 1 <= N <= 1,000
- 간선 개수 M의 범위 1 <= M <= 10,000

예제

#알튜비튜

예제 입력

```
4 5 1
1 2
1 3
1 4
2 4
3 4
```

예제 출력

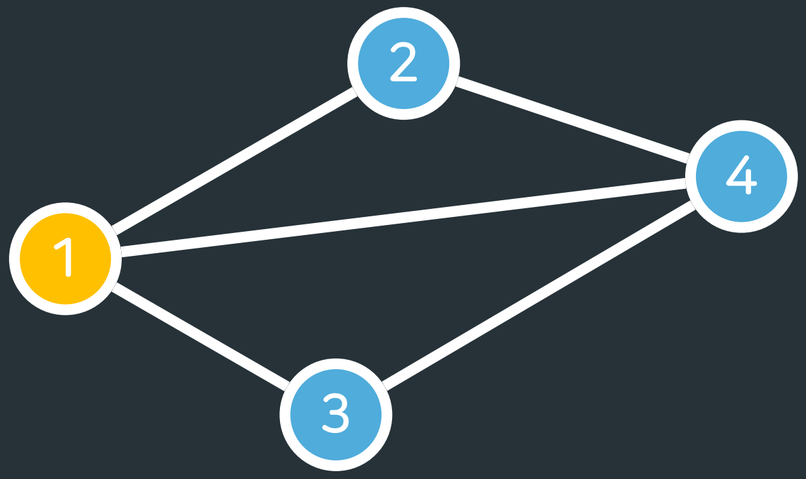
```
1 2 4 3
1 2 3 4
```

← DFS

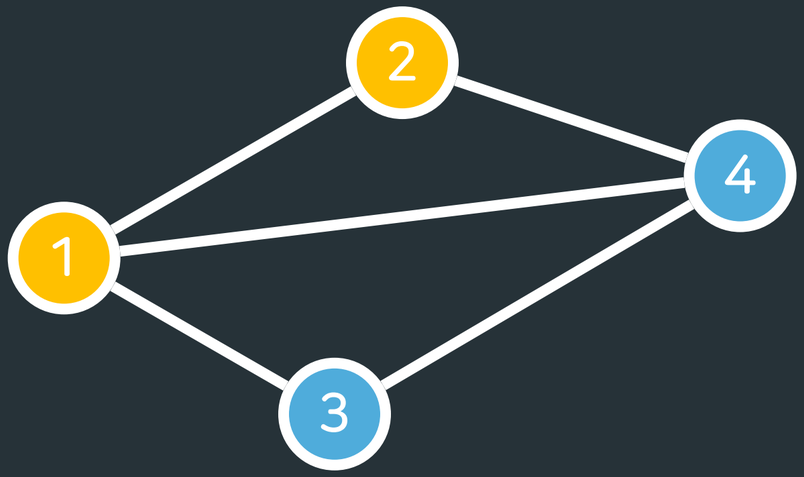
← BFS

예제 - dfs

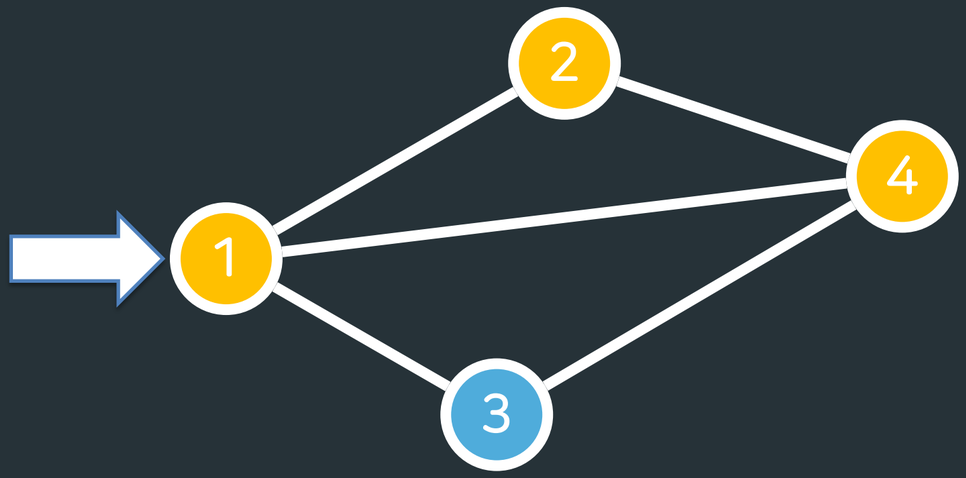
```
4 5 1
1 2
1 3
1 4
2 4
3 4
```



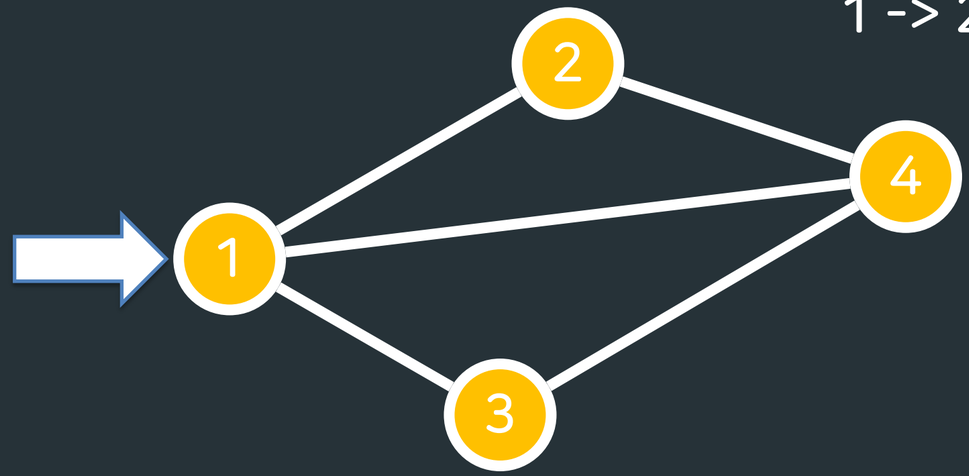
1



1 -> 2



1 -> 2 -> 4



1 -> 2 -> 4 -> 3

예제 - dfs

#알튜비튜

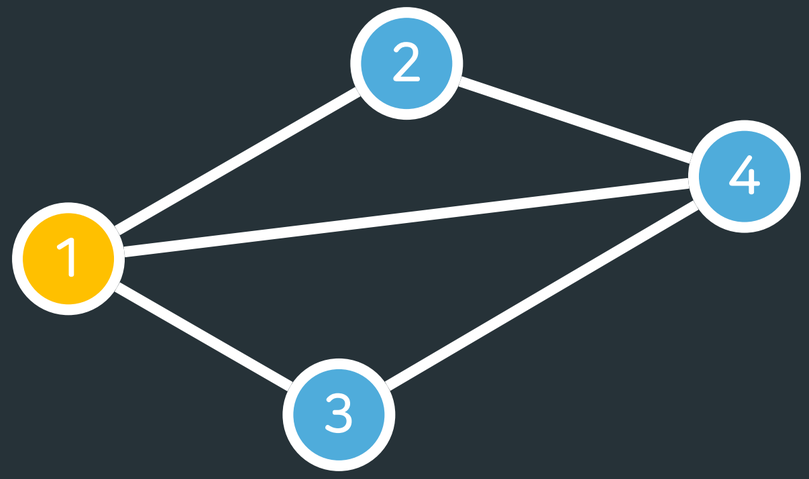
```cpp
vector<bool> visited_dfs;

void dfs(int index, vector<int>& ans_dfs) {
    visited_dfs[index] = true;
    ans_dfs.push_back(index);

    for (int i = 1; i <= n; i++) {
        if (li[index][i] == 1 && !visited_dfs[i]) {
            dfs(i, ans_dfs);
        }
    }
}
```

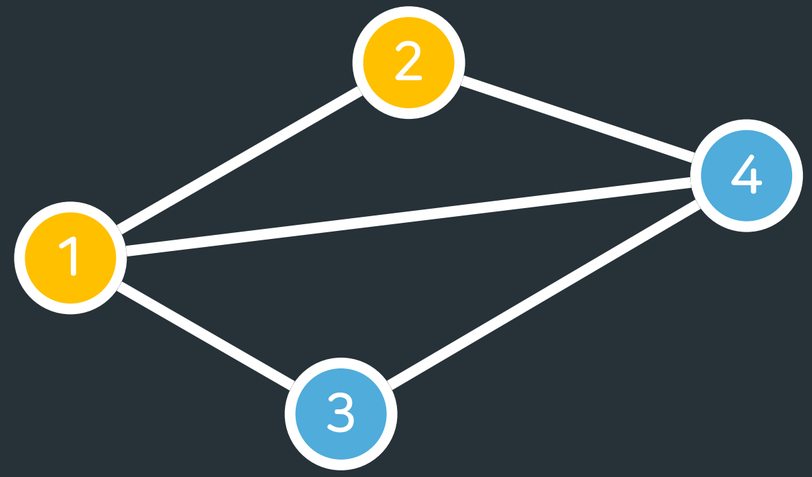
예제 - bfs

```
4 5 1
1 2
1 3
1 4
2 4
3 4
```



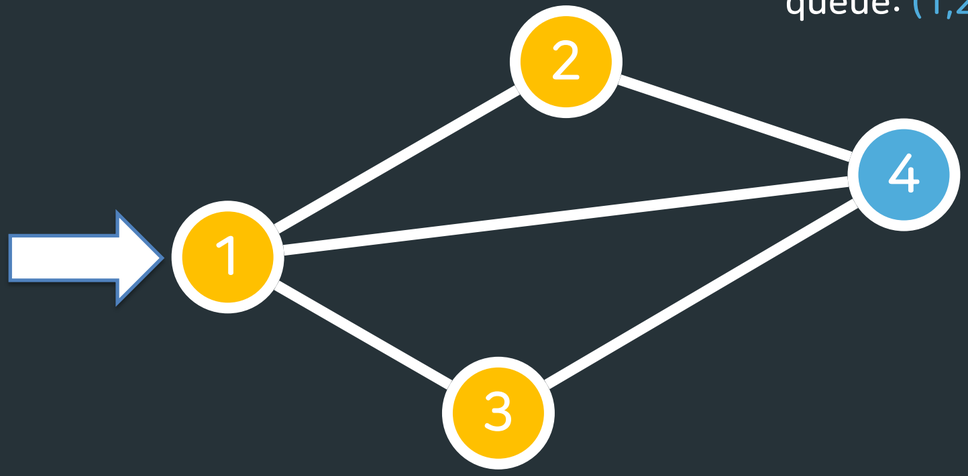
1

queue: (1,2), (1,3), (1,4)



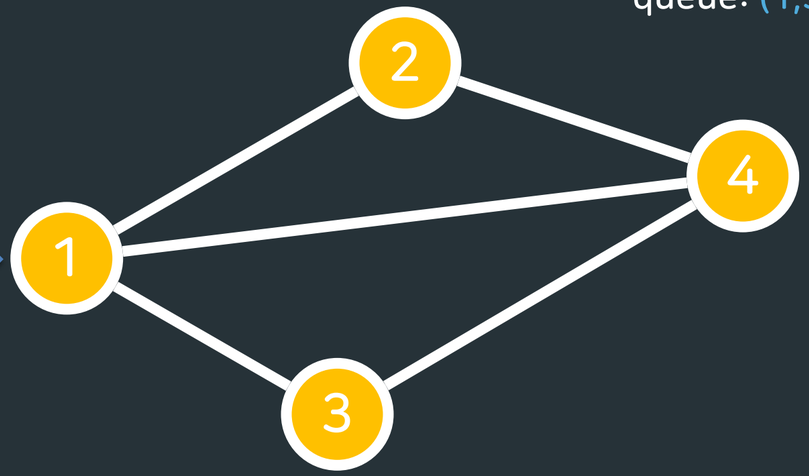
1 -> 2

queue: (1,3), (1,4), (2,4)



1 -> 2 -> 3

queue: (1,4), (2,4), (3,4)



1 -> 2 -> 3 -> 4

예제 - bfs

#알튜비튜

```
vector<bool> visited_bfs;

void bfs(vector<int>& ans_bfs) {
    queue<int> q;
    q.push(v);
    visited_bfs[v] = true;

    while (!q.empty()) {
        int current_v = q.front();
        q.pop();
        ans_bfs.push_back(current_v);

        for (int i = 1; i <= n; i++) {
            if (li[current_v][i] == 1 && !visited_bfs[i]) {
                visited_bfs[i] = true;
                q.push(i);
            }
        }
    }
}
```

2636번 : 치즈

문제

- 정사각형 칸들로 이루어진 사각형 모양 판이 있고, 그 위에 얇은 치즈가 있음
- 치즈는 공기와 접촉하면 녹아 없어짐
- 치즈가 녹아 없어지는데 걸리는 시간, 모두 녹기 한 시간 전 치즈 조각이 놓여있는 칸의 개수 출력
- 치즈가 녹는 과정에서 여러 조각으로 나누어 질 수도 있음

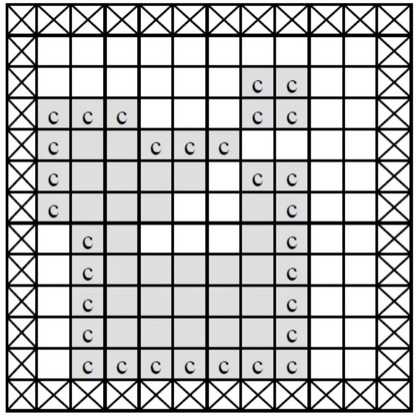
제한 사항

- 세로, 가로 길이는 최대 100

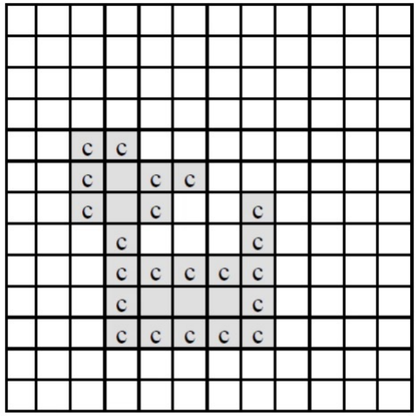
예제

예제 출력

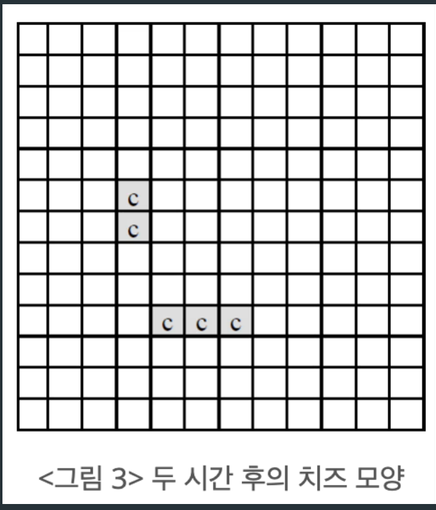
```
3
5
```



<그림 1> 원래 치즈 모양



<그림 2> 한 시간 후의 치즈 모양



<그림 3> 두 시간 후의 치즈 모양

풀이 방법

- 1. 출력 해야 하는 것 : 치즈가 녹는데 걸리는 시간, 치즈가 모두 녹기 한 시간 전 남아있는 치즈 개수
- 2. 치즈 : 이번 턴에 녹는 치즈와 녹을 수 없는 치즈 2종류

  (녹는 치즈 : 공기와 직접적으로 맞닿아 있는 치즈)
- 3. 녹는 치즈에 대해 다음 턴에 이를 공기로 취급 → 다시 이번 턴에 녹는 치즈가 있는지 탐색

코드

#알튜비튜

```
while (!q.empty()) {
        cheese_cnt = q.size();

        while(!q.empty()) {
            int x = q.front().first;
            int y = q.front().second;
            q.pop();//지금 검사하는 요소 제거
            for (int i = 0; i < 4; i++) { //동서남북으로 검사
                int new_x = x + dx[i];
                int new_y = y + dy[i];

                if (new_x >= 0 && new_x < n && new_y >= 0 && new_y < m) {
                    //인덱스 내에 있는지 검사
                    if (!visited[new_x][new_y] && board[new_x][new_y] == 1) {
                        //방문한 적 없고,녹을 수 있는 치즈
                        tmp.push(make_pair(new_x,new_y));
                    }
                    else if(!visited[new_x][new_y] && board[new_x][new_y] == 0) {
                        //방문한 적 없는 공기 -> bfs로 동서남북에
                        //이번 턴에 녹을 수 있는 치즈가 있는지 검사
                        q.push(make_pair(new_x,new_y));
                    }
                    visited[new_x][new_y] = true;
                }
            }
        } // 안에 검사한 while 문 -> 이번 턴에 녹일 수 있는 치즈에 대한 검사
        while (!tmp.empty()) {
            q.push(tmp.front());
            tmp.pop();
        }
        total_time++;
    }
```

DFS & BFS 특징

#알튜비튜

특징

- 모든 노드를 방문하고자 하는 경우엔 DFS, BFS 상관없음
- 가중치가 주어지거나 특정 경로를 찾아야 할 때 DFS
- 두 노드 사이의 최단거리를 찾을 때는 BFS (ex. 미로찾기)

→ BFS는 현재 노드에서 가장 가까운 곳(자식 노드)부터 탐색하기 때문

- 그래프가 단방향으로 주어지는지 양방향으로 주어지는지 잘 살펴보자!

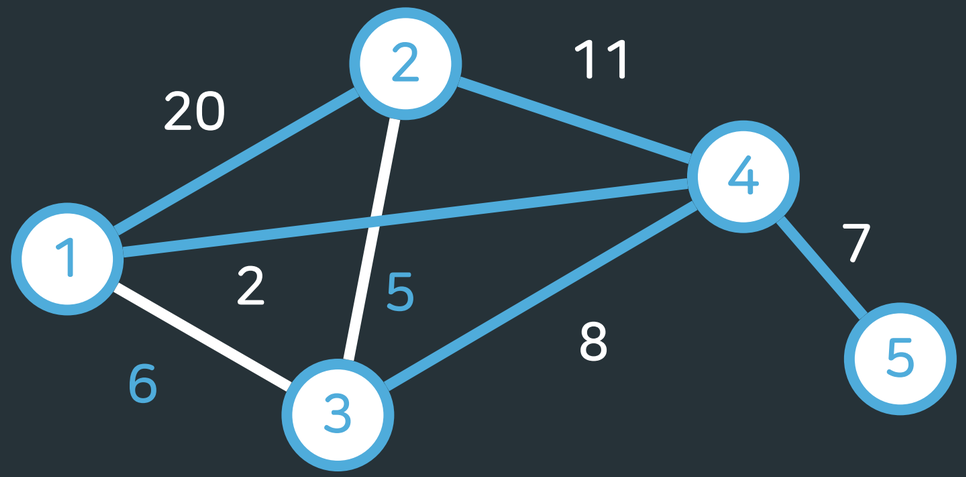
그래프 미리보기

#알튜비튜

그래프 알고리즘

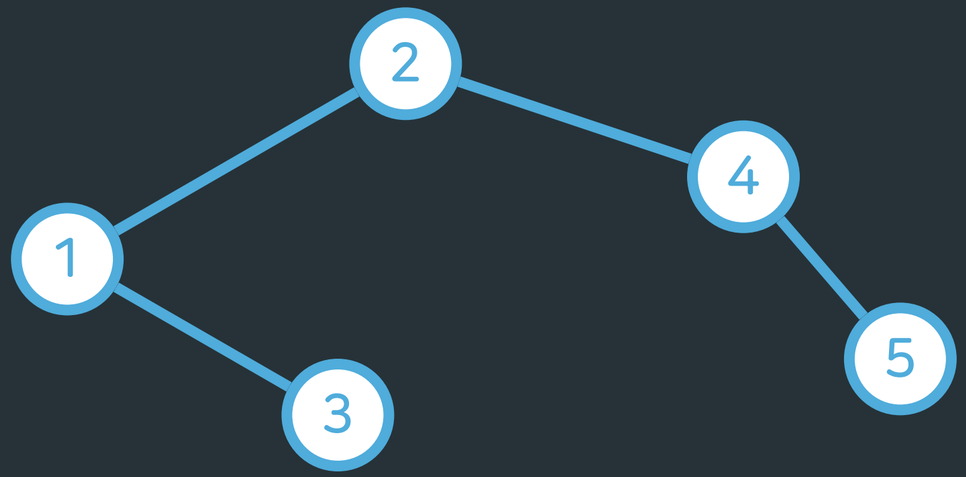
- 정점 사이의 최단 거리를 구해야 할 때 → 최단 경로 알고리즘
- 사이클이 없는 그래프를 다룰 때 → 트리
- 가중치의 총 합이 가장 작은 트리를 만들어야 할 때 → 최소 비용 신장 트리
- 방향 그래프에서 선후관계를 기반으로 정점을 나열해야 할 때 → 위상 정렬
- 그래프에서 정점 사이의 관계를 서로소로 정의해 집합으로 나누어야 할 때 → 유니온 파인드

그래프 미리보기



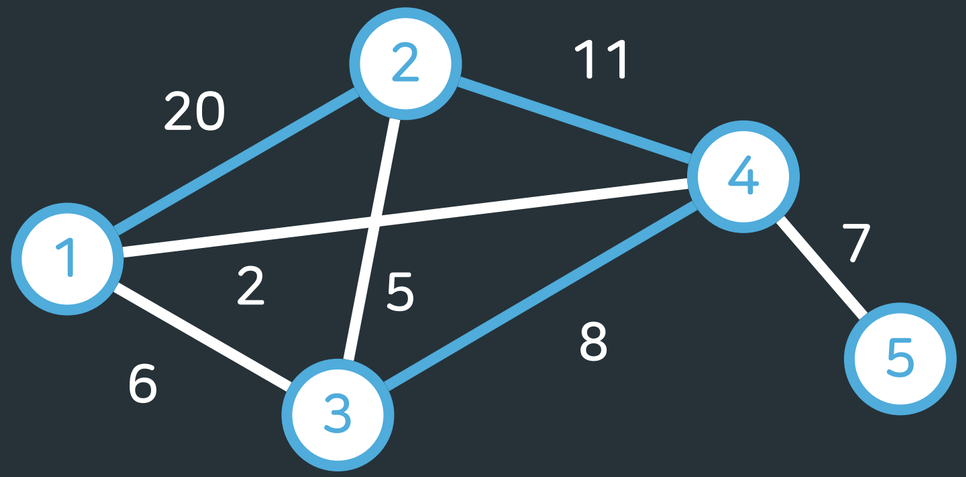
정점 사이의 최단 거리를 구해야 할 때 → 최단 경로 알고리즘

그래프 미리보기



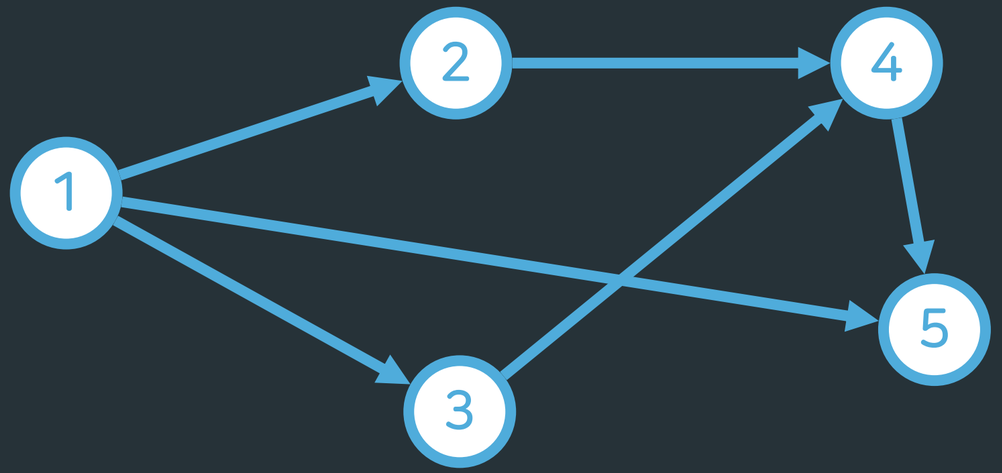
사이클이 없는 그래프를 다룰 때 → 트리

그래프 미리보기



가중치의 총 합이 가장 작은 트리를 만들어야 할 때 → 최소 비용 신장 트리

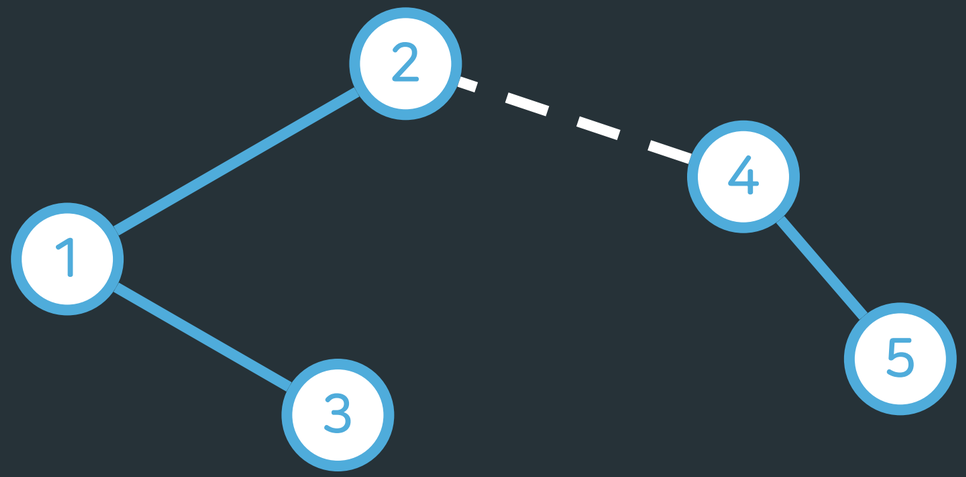
그래프 미리보기



1 → 2 → 3 → 4 → 5

방향 그래프에서 선후 관계를 기반으로 정점을 나열해야 할 때 → 위상 정렬

그래프 미리보기



그래프에서 정점 사이의 관계를 서로소로 정의해 집합으로 나누어야 할 때 → 유니온 파인드

마무리

#알튜비튜

정리

- 그래프는 인접 행렬과 인접 리스트로 구현 가능
- 깊이 우선 탐색이라 불리는 DFS와 너비 우선 탐색이라 불리는 BFS가 대표적인 탐색 알고리즘
- 해당 노드를 끝까지 파고들어야 한다면 DFS, 해당 노드 주변을 먼저 탐색한다면 BFS
- DFS는 stack, 재귀 호출을 통해 구현, BFS는 queue를 통해 구현
- DFS와 BFS 모두 방문 체크 꼭 필요

이것도 알아보세요!

- 방문 체크를 하는 방법이나 위치는 문제에 따라 달라질 수 있어요. 코드가 어떻게 돌아가는지 직접 디버깅을 많이 해봅시다
- DFS는 재귀 호출로 구현한다는 점에서 백트래킹과 비슷한 점이 많아요. 백트래킹과 DFS의 차이를 알아봅시다

과제

#알튜비튜

필수

- 2644번 : 촌수 계산 – Silver 2
- 2606번 : 바이러스 – Silver 3
- 2615번 : 오목 – Silver 1

도전

- 14500번 : 테트로미노 – Gold 4
- 14502번 : 연구소 – Gold 4

과제 마감일

#알튜비튜

과제제출 마감 ~ 4월 9일 화요일 18:59

추가제출 마감 ~ 4월 11일 목요일 23:59