그래프 탐색

자주 사용되는 DFS/BFS 능숙하게 사용하자!

이것이 코딩테스트다

개념정리 편

탐색이란?

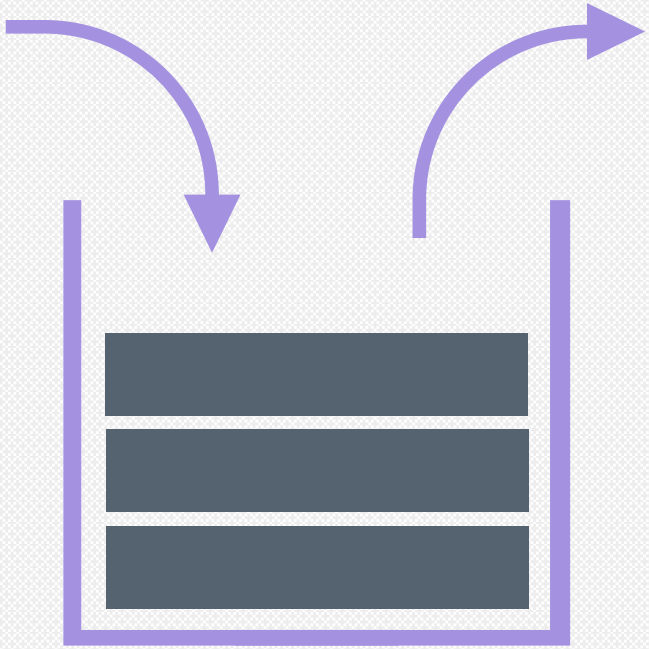
많은 양의 데이터 중에서 원하는 데이터를 찾는 과정

들어가기 전에 알아야 하는 자료구조

큐

스택 **재귀**

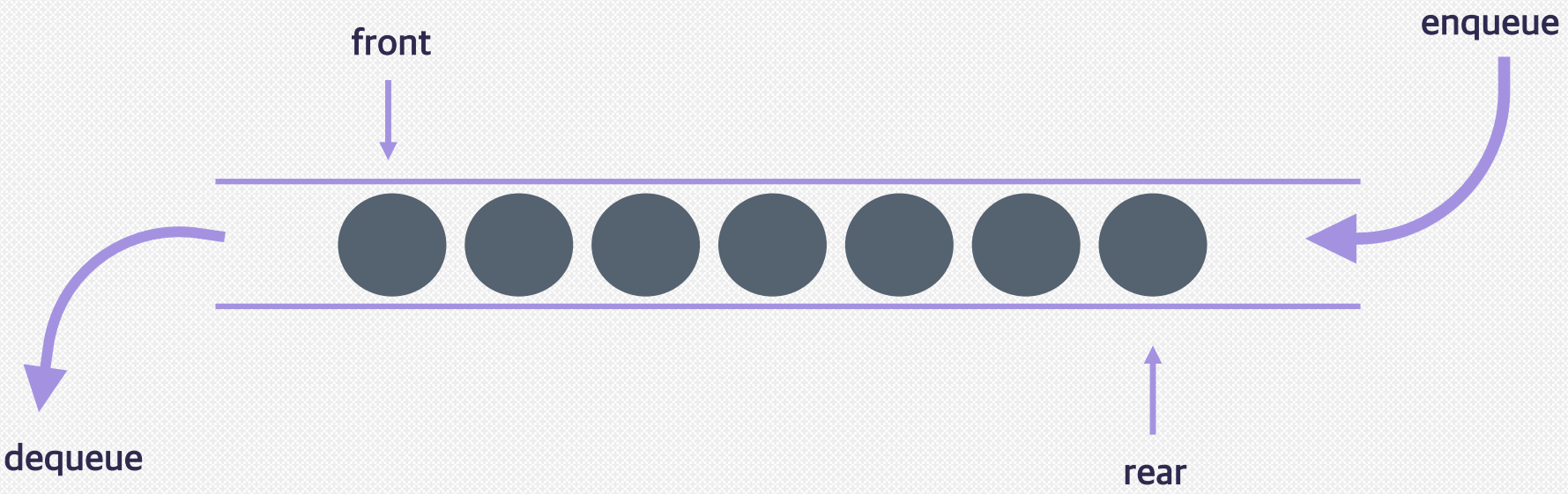
스택



Stack

- FILO(First In, Last Out) 구조를 띄고 있는 자료구조로, 삽입과 삭제 연산이 동일한 한군데에서 발생함
- 삽입/삭제 연산에 있어 시간 복잡도가 O(1)
- 이전에 활용한 데이터를 역으로 추적하거나, 처음 들어온 데이터보다 나중에 들어온 데이터가 빨리 나가야 할 때 사용

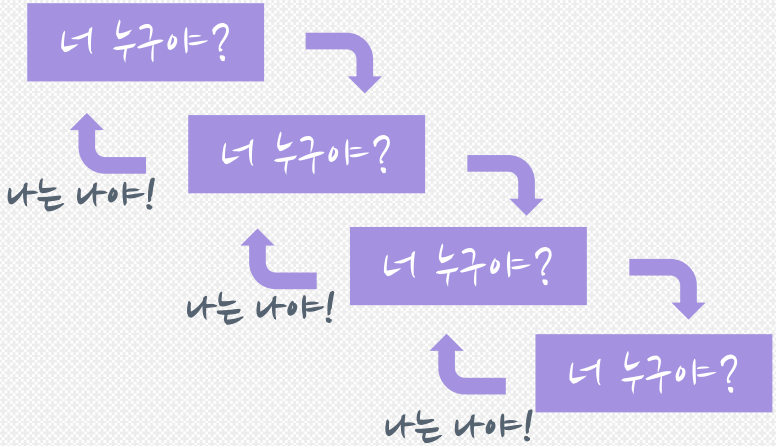
큐



Queue

- FIFO(First In, First Out) 구조를 띄고 있는 자료구조로, 삽입과 삭제 연산이 서로 다른 군데에서 발생함
- 삽입/삭제 연산에 있어 시간 복잡도가 O(1)
- 주로 순차적으로 진행하는 스케줄링에서 사용

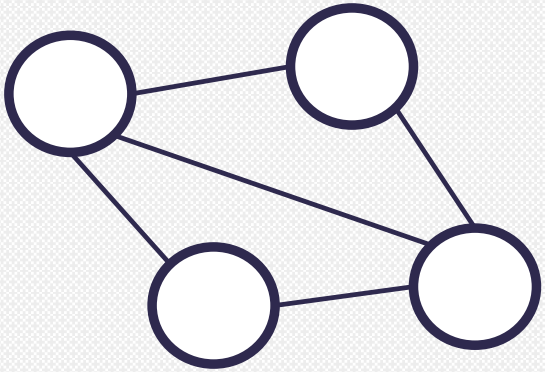
재귀



Recursion

- 어떤 함수에서 자신을 다시 호출하여 작업을 수행하는 방식
- 메모리를 많이 차지하여 반복문에 비해 성능이 느림
- 기저 조건은 반드시!!

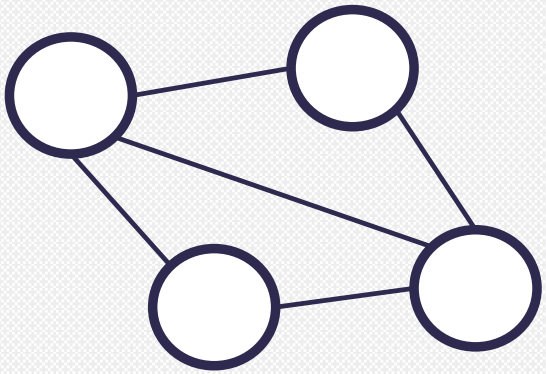
그래프



Graph

- 노드와 그 노드를 잇는 간선을 하나로 모아 놓은 자료구조

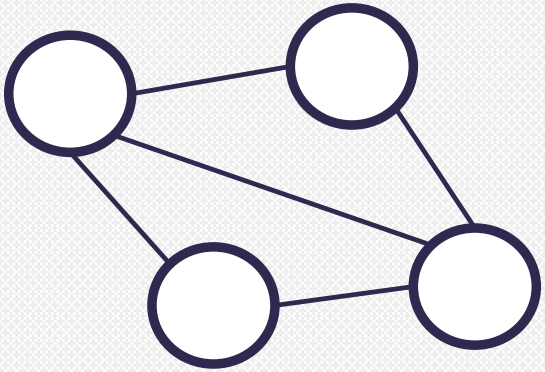
그래프



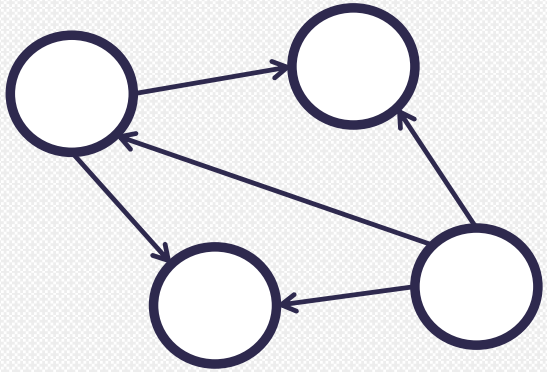
- 간선(Edge)
  노드와의 관계를 표현
- 정점(Vertex, Node)
  데이터가 담기는 지점
- 차수(Degree)
  하나의 정점에 인접한 정점의 수

- 노드와 그 노드를 잇는 간선을 하나로 모아 놓은 자료구조

그래프



- 무방향 그래프
  간선의 방향이 없음



- 방향 그래프
  간선의 방향이 있음

많은 양의 데이터 중에서 원하는 데이터를 찾는 과정

Graph Search

그래프의 모든 노드를 탐색하기 위해 간선을 따라 순회

DFS

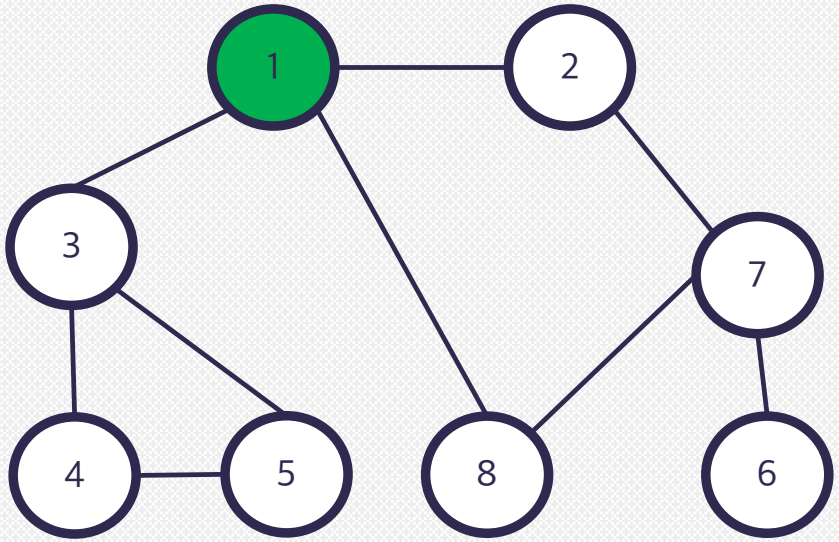
DFS

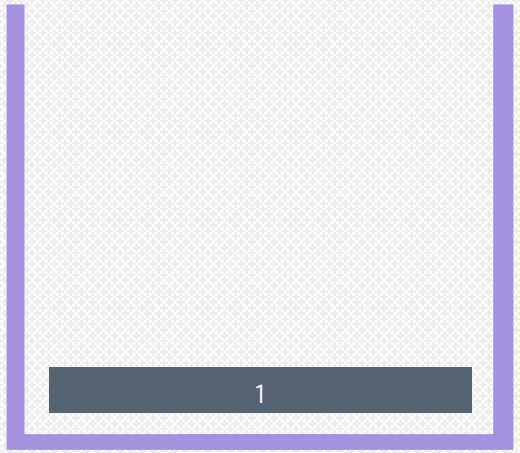
- 깊이 우선 탐색이라고 부르며 그래프에서 깊이 부분을 우선적으로 탐색하는 알고리즘
- 스택 자료구조(혹은 재귀 함수)를 이용

1. 탐색 시작 노드를 스택에 삽입하고 방문 처리
2. 스택에 최상단 노드에 방문하지 않은 인접한 노드가 하나라도 있으면, 그 노드를 스택에 넣고 방문 처리
   방문하지 않은 인접 노드가 없으면 스택에서 최상단 노드를 꺼냄
3. 더 이상 2번의 과정을 수행 할 수 없을 때까지 반복

```
function dfs(pos) {
     visit pos
     for children of pos
        if each child has not been visited
            dfs(pos)
}
```

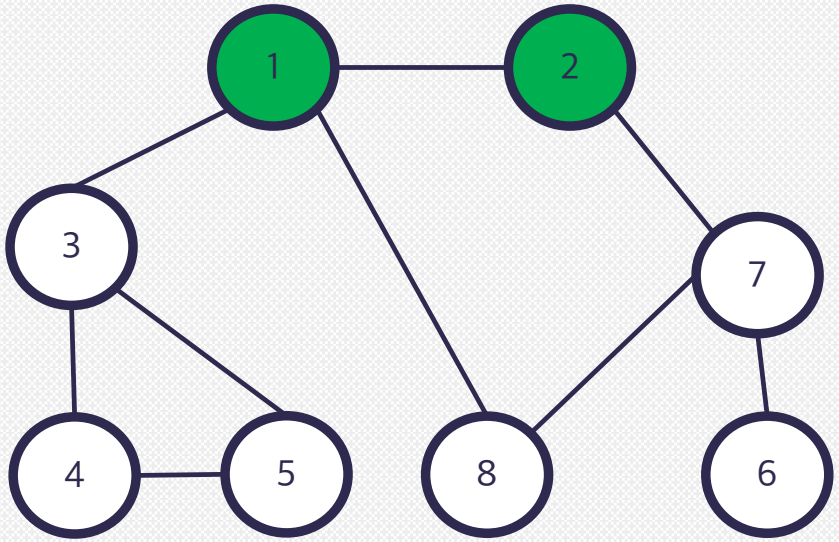
DFS

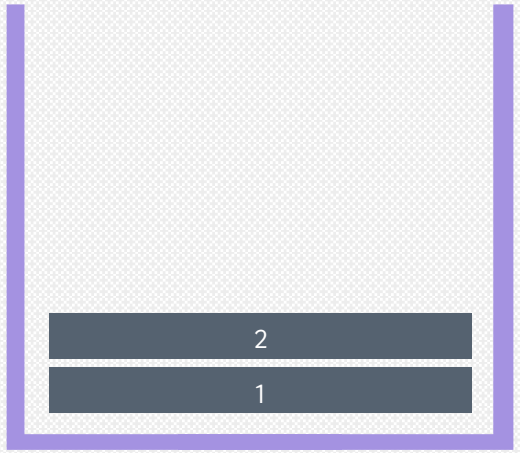




탐색 순서 1

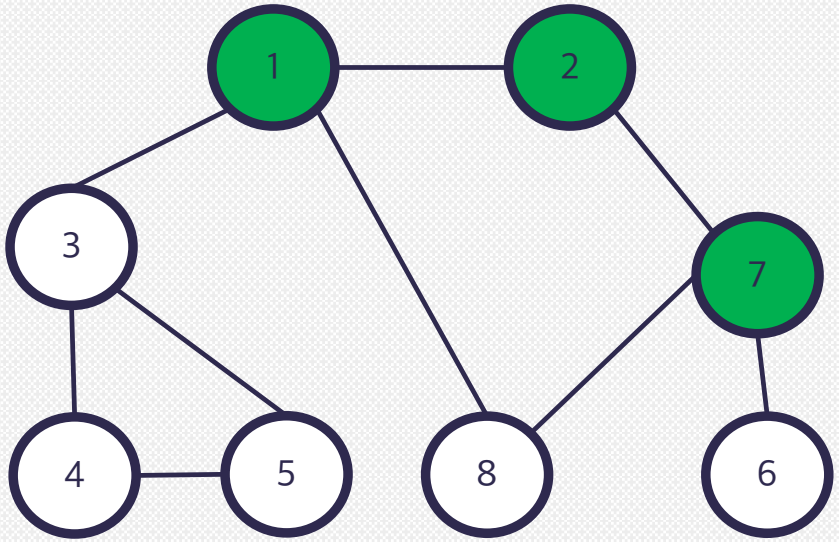
DFS

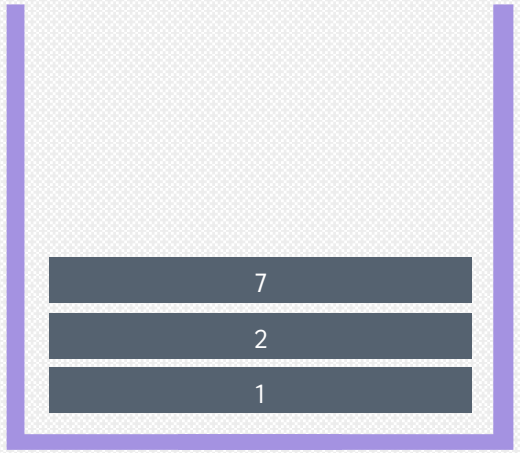




탐색 순서 1 - 2

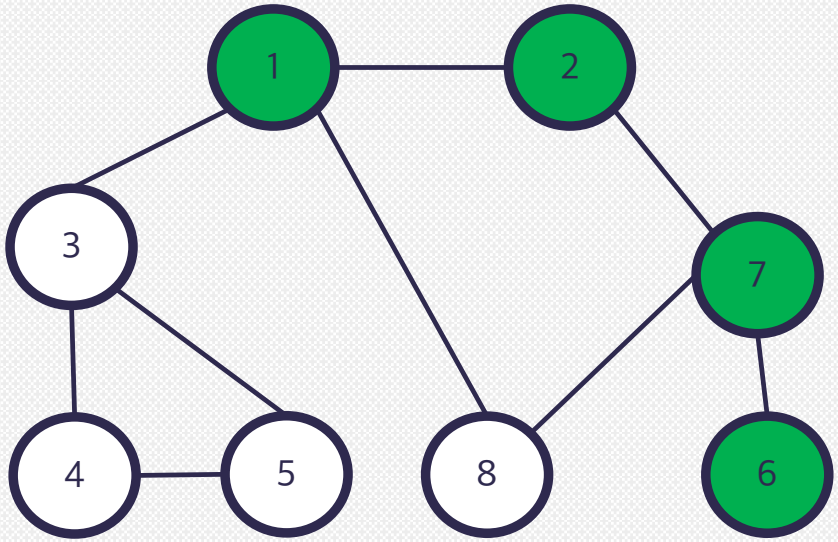
DFS

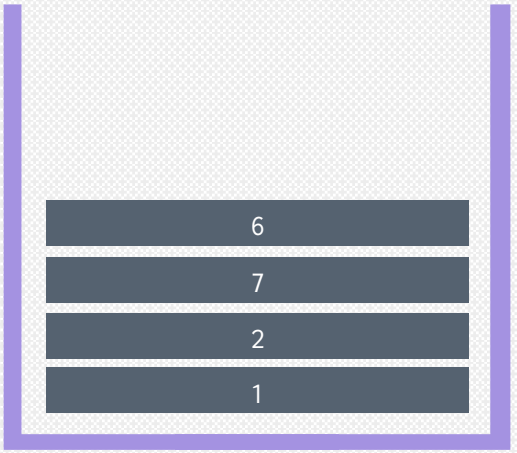




탐색 순서 1 - 2 - 7

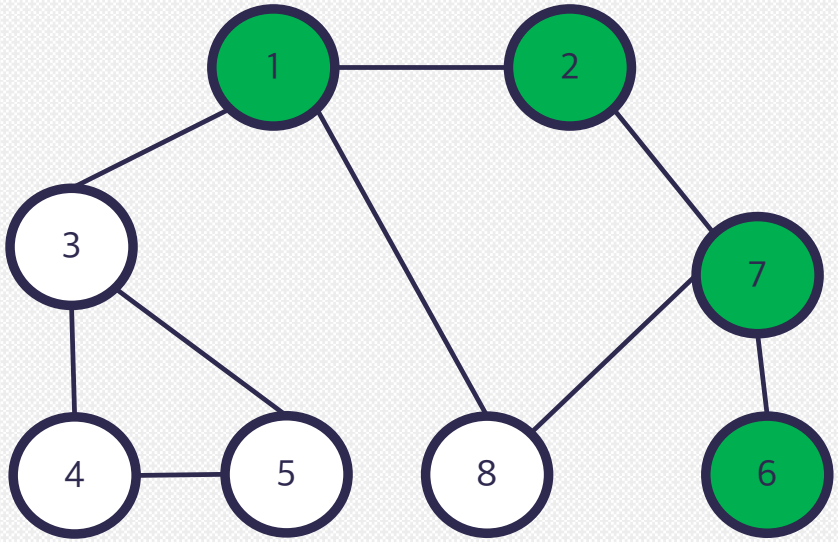
DFS

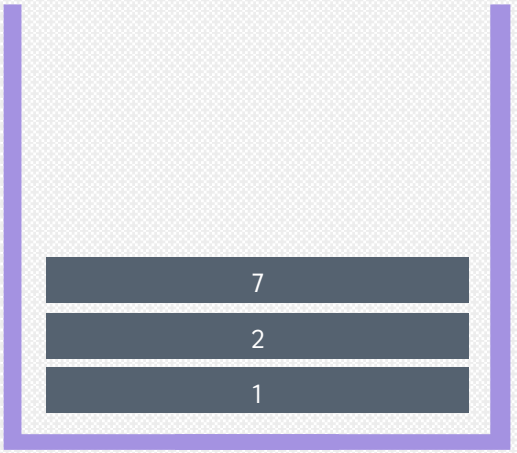




탐색 순서 1 - 2 - 7 - 6

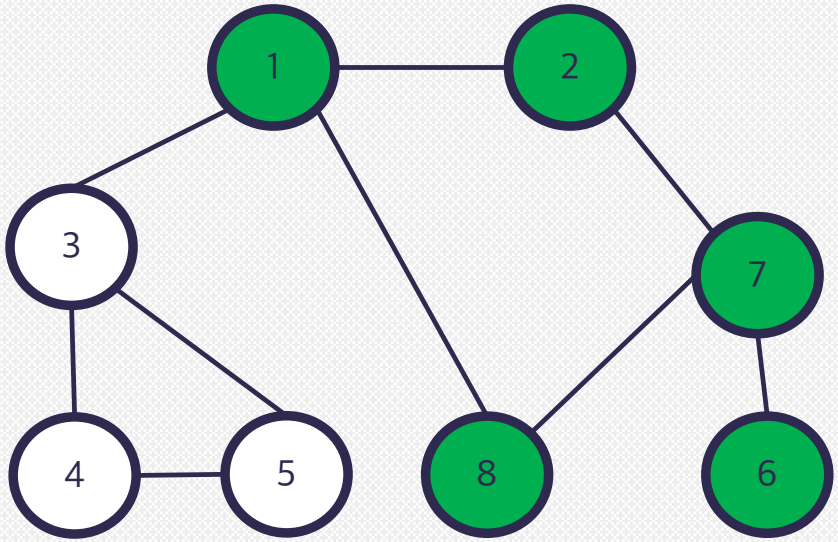
DFS

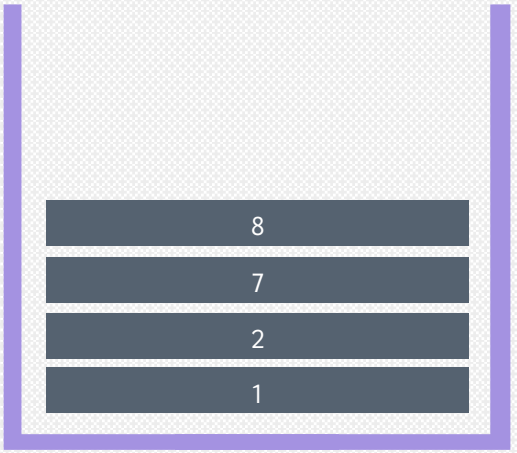




탐색 순서 1 - 2 - 7 - 6

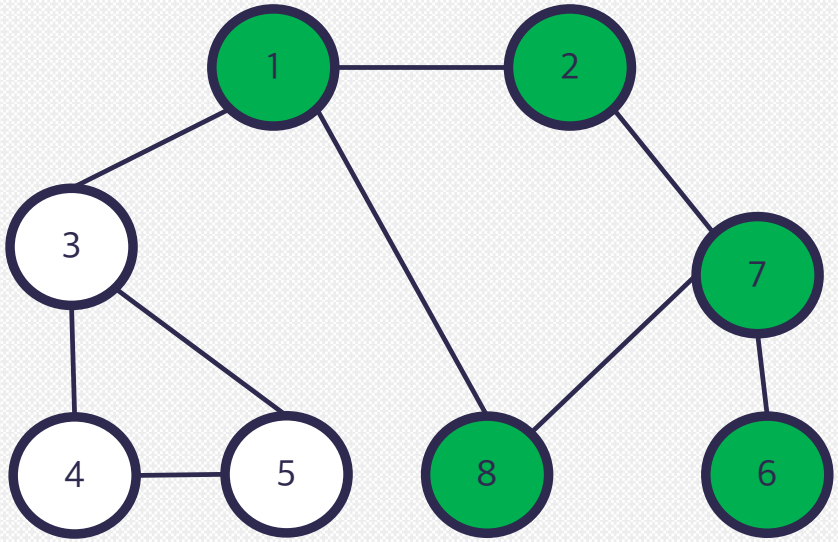
DFS

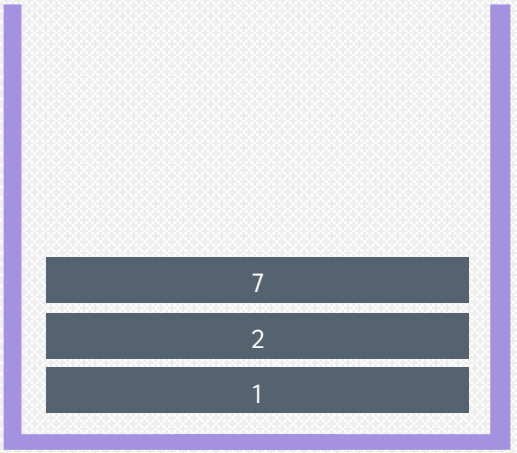




탐색 순서 1 - 2 - 7 - 6 - 8

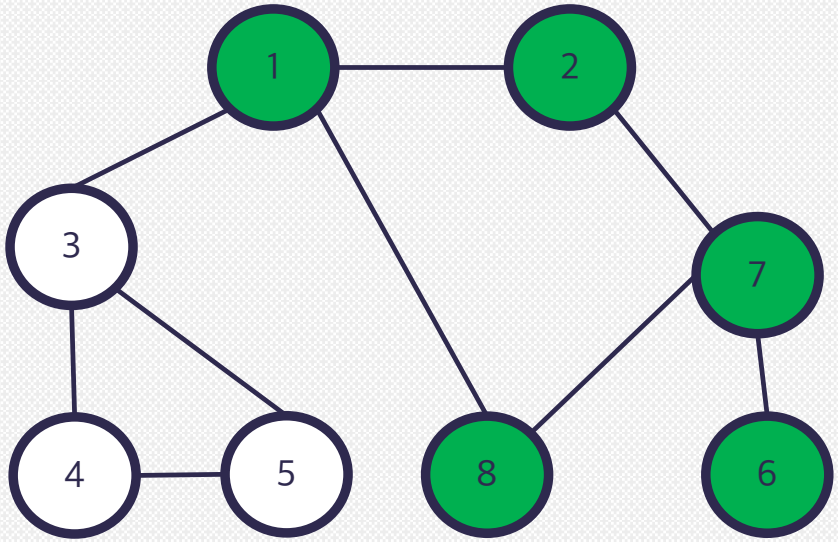
DFS

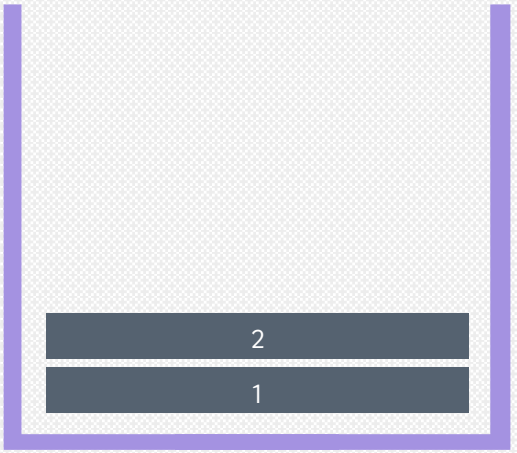




탐색 순서 1 - 2 - 7 - 6 - 8

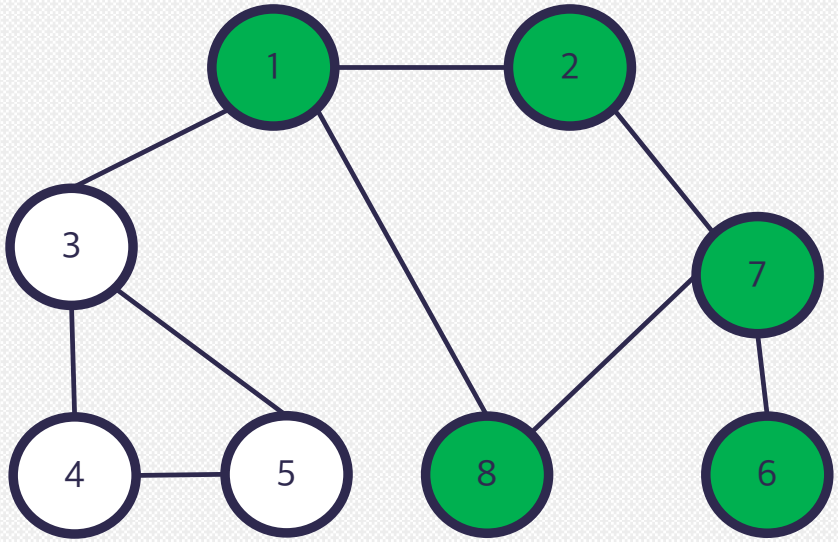
DFS

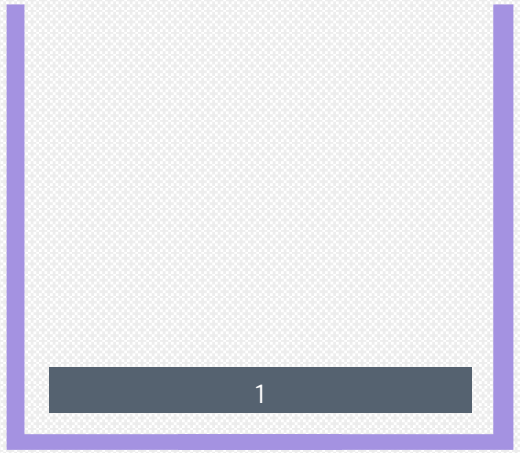




탐색 순서 1 - 2 - 7 - 6 - 8

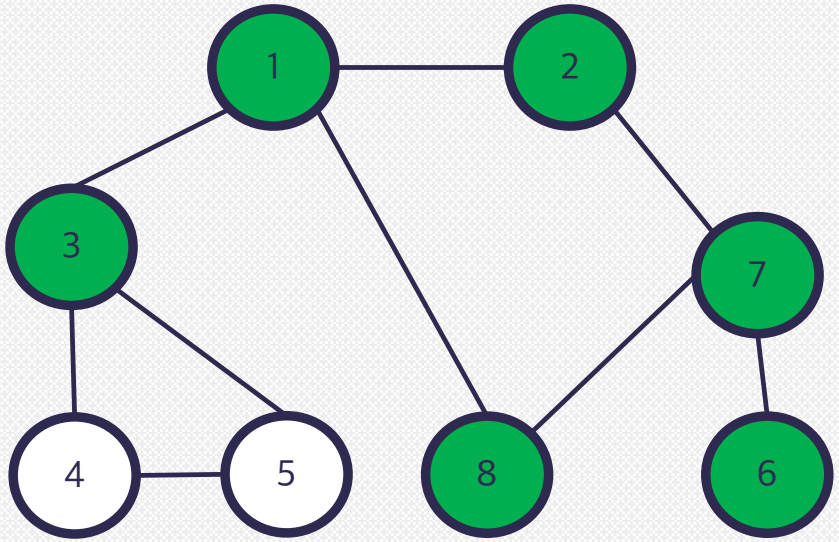
DFS

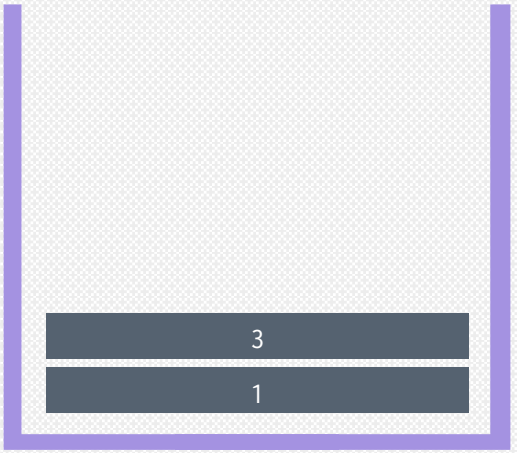




탐색 순서 1 - 2 - 7 - 6 - 8

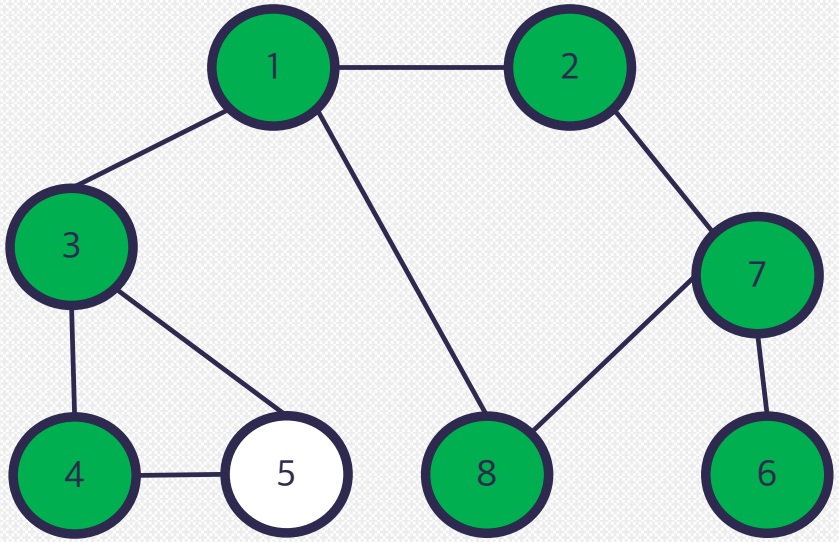
DFS

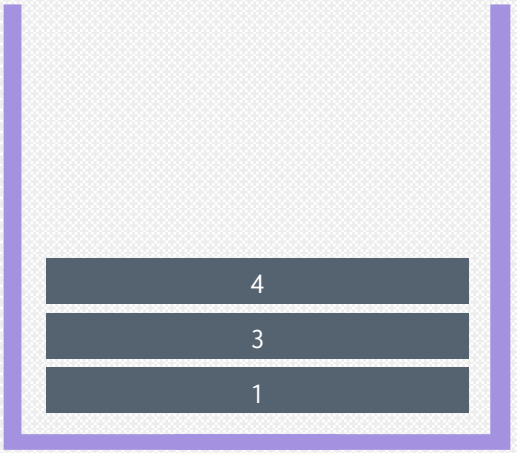




탐색 순서 1 - 2 - 7 - 6 - 8 - 3

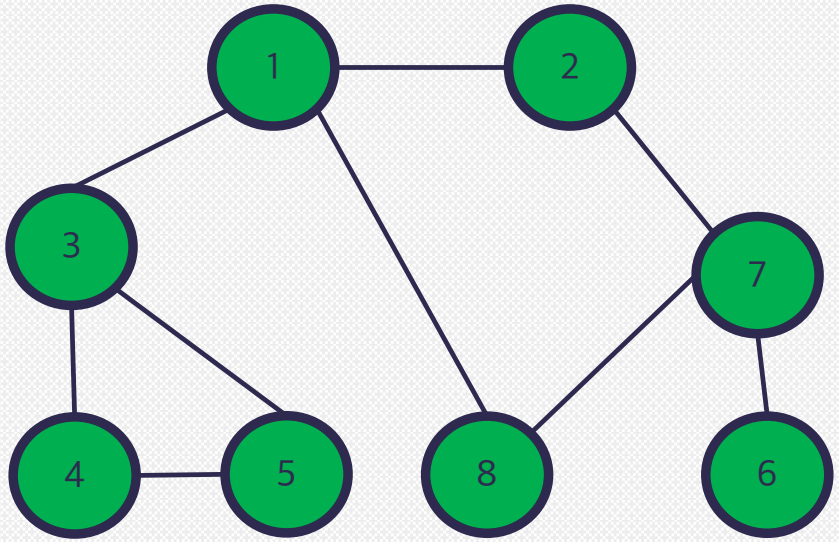
DFS

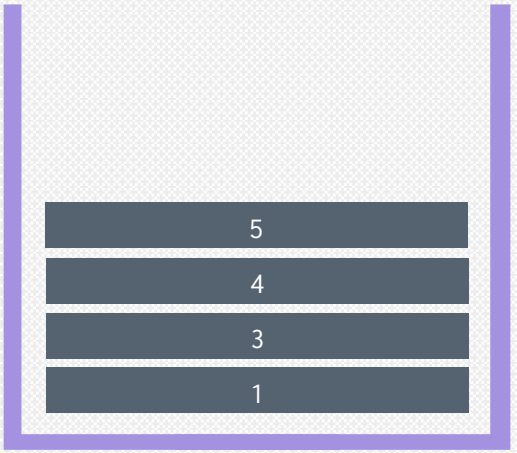




탐색 순서 1 - 2 - 7 - 6 - 8 - 3 - 4

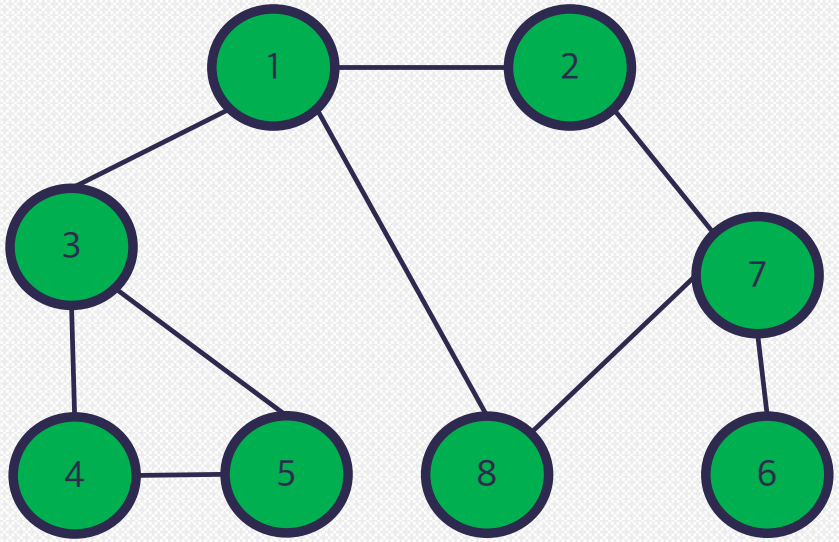
DFS

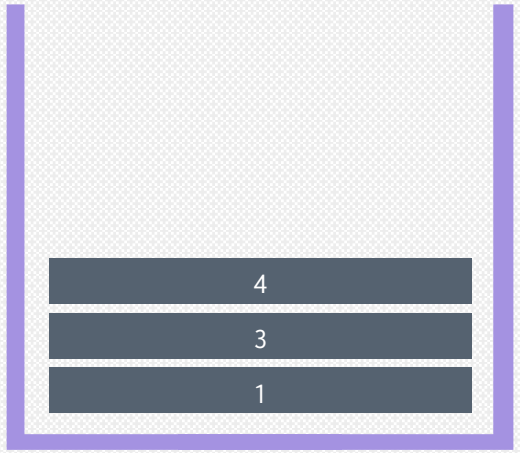




탐색 순서 1 - 2 - 7 - 6 - 8 - 3 - 4 - 5

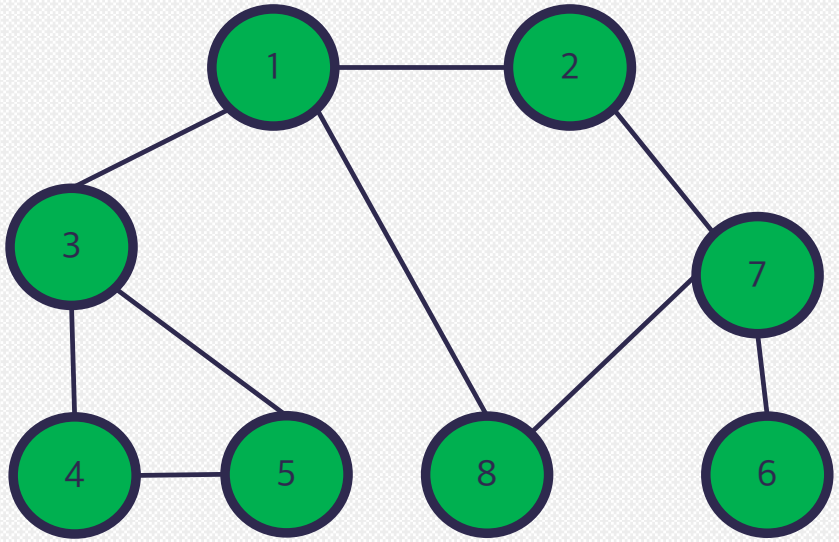
DFS

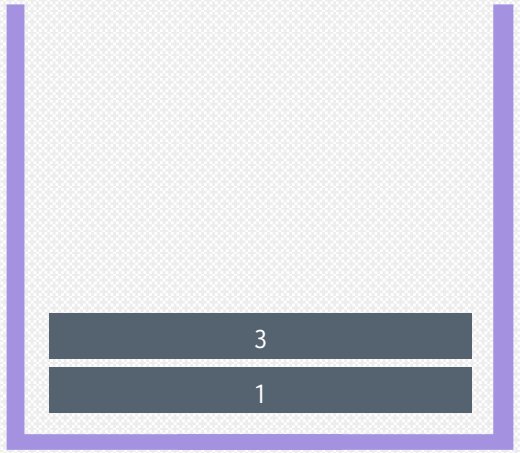




탐색 순서 1 - 2 - 7 - 6 - 8 - 3 - 4 - 5

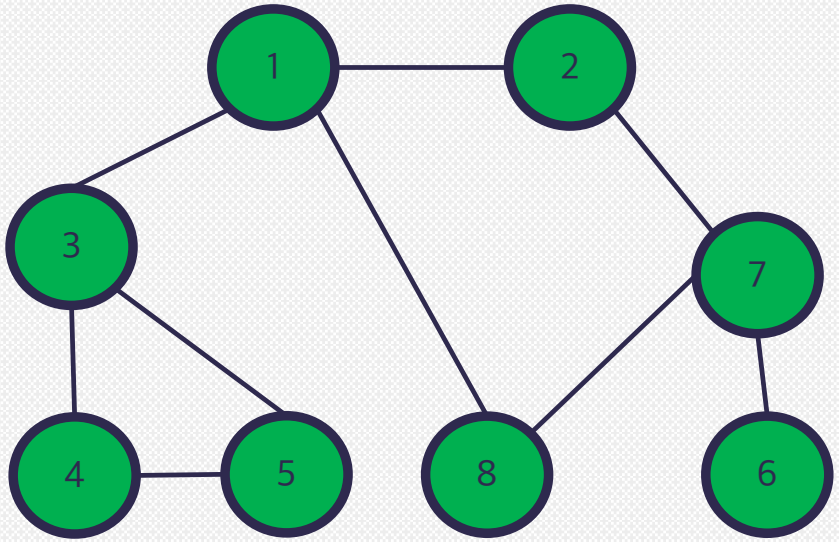
DFS

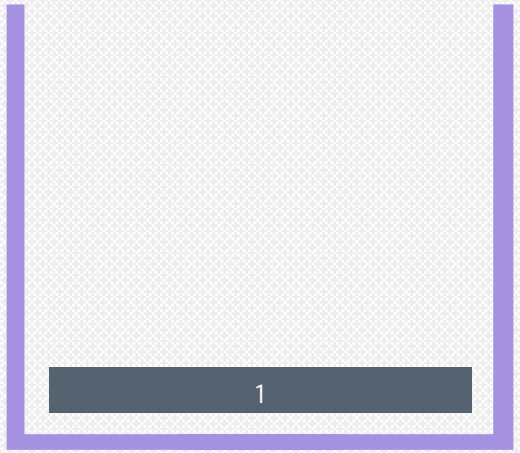




탐색 순서 1 - 2 - 7 - 6 - 8 - 3 - 4 - 5

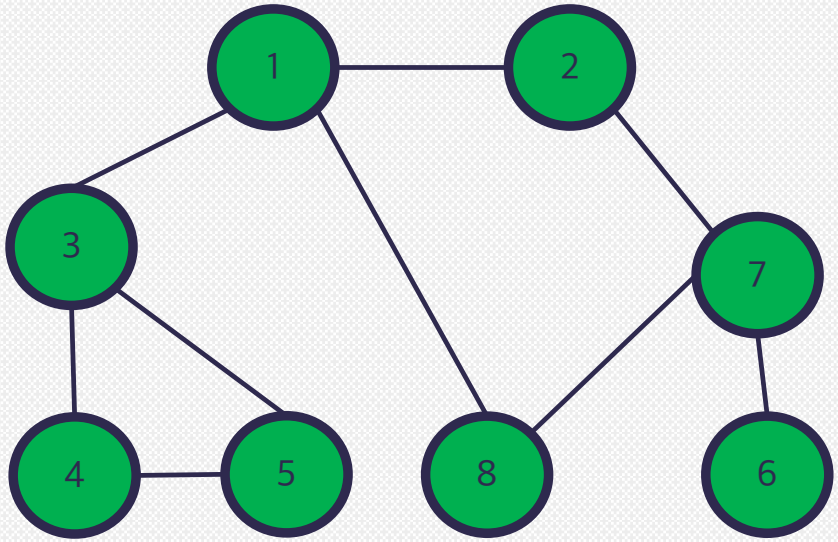
DFS

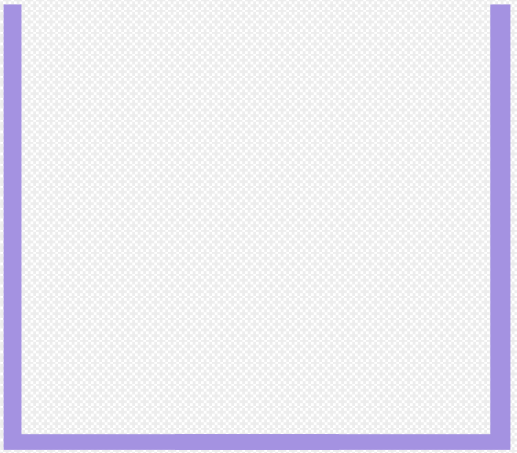




탐색 순서 1 - 2 - 7 - 6 - 8 - 3 - 4 - 5

DFS





탐색 순서 1 - 2 - 7 - 6 - 8 - 3 - 4 - 5

DFS

```python
def dfs(x):
    visited[x] = True

    for next in graph[x]:
        if not(visited[next]):
            dfs(next)
}
```

```java
public static void dfs(int x) {
    visited[x] = true

    for (int i = 0; i < graph.get(x).size(); i++) {
        int y = graph.get(x).get(i);
        if(!visited[y]) dfs(y);
    }
}
```

```java
void dfs(int x) {
    visited[x] = true

    for (int i = 0; i < graph[x].size(); i++) {
        int y = graph[x][i];
        if(!visited[y]) dfs(y);
    }
}
```

이렇게 해도 괜찮을까 ?

```java
for (int y : graph[x]) {
    if(!visited[y]) dfs(y);
}
}
```

BFS

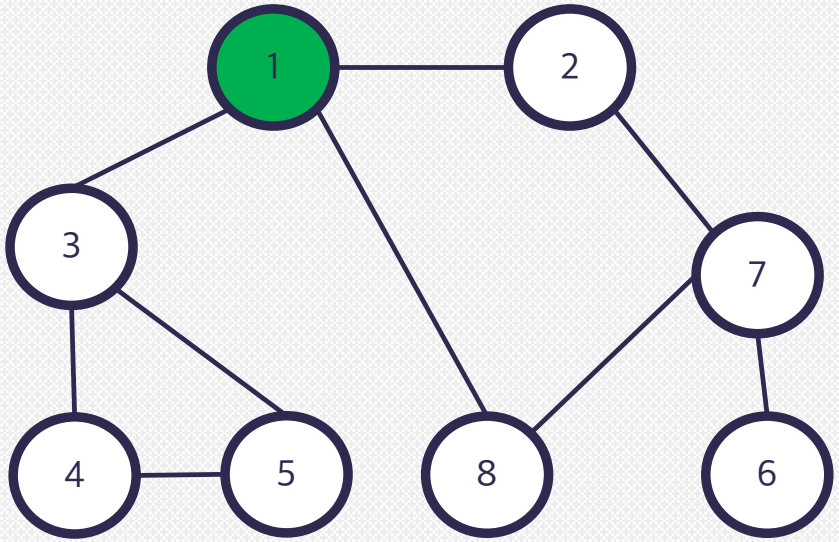
BFS

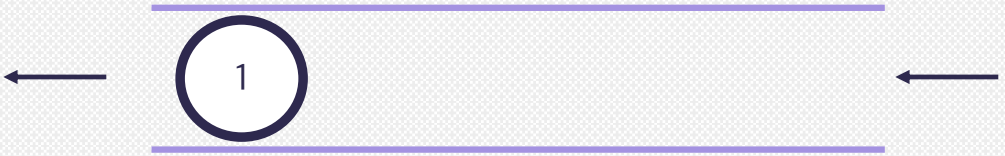
- 너비 우선 탐색이라고 부르며 그래프에서 가까운 노드부터 우선적으로 탐색하는 알고리즘
- 큐 자료구조를 이용

1. 탐색 시작 노드를 큐에 삽입하고 방문 처리
2. 큐에서 노드를 꺼낸 뒤에 해당 노드의 인접 노드 중에서 방문하지 않은 노드를 모두 큐에 삽입하고 방문처리
3. 더 이상 2번의 과정을 수행 할 수 없을 때까지 반복

```
function bfs(pos) {
     set Q = Queue
     pos -> Q
     while Q is not empty
        set node = popped element of Q
        for children of node
        if each child has not been visited
            visit child
            child -> Q
}
```

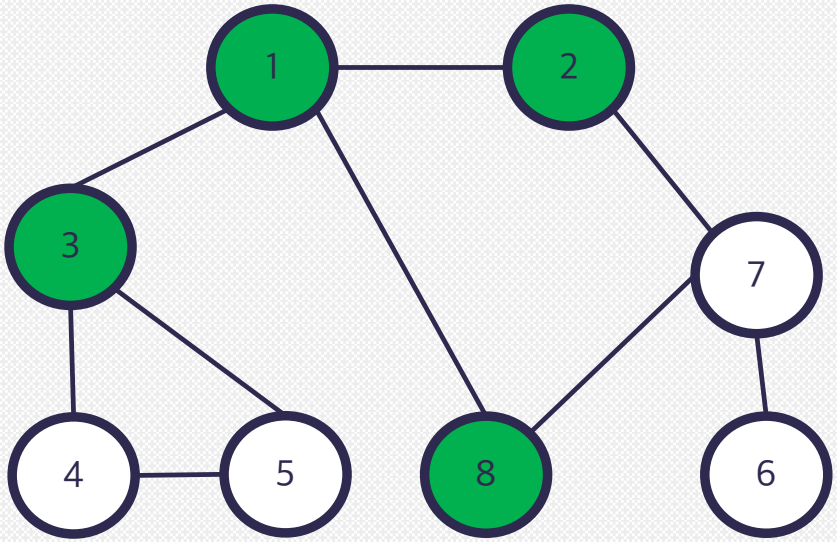
BFS





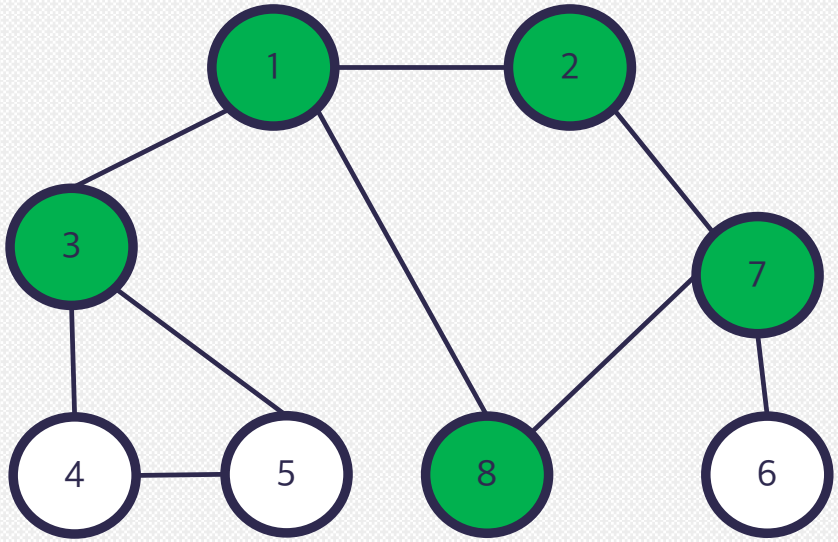
탐색 순서 1

BFS



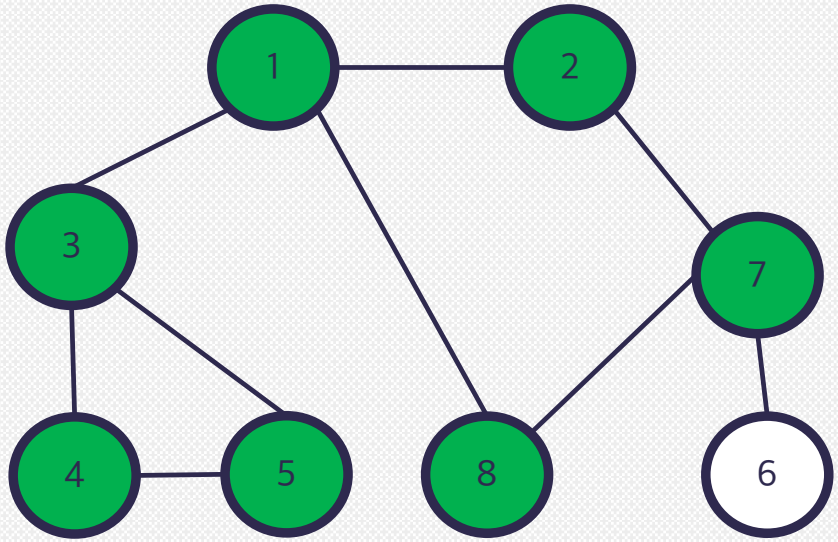
탐색 순서 1 - 2 - 3 - 8

BFS



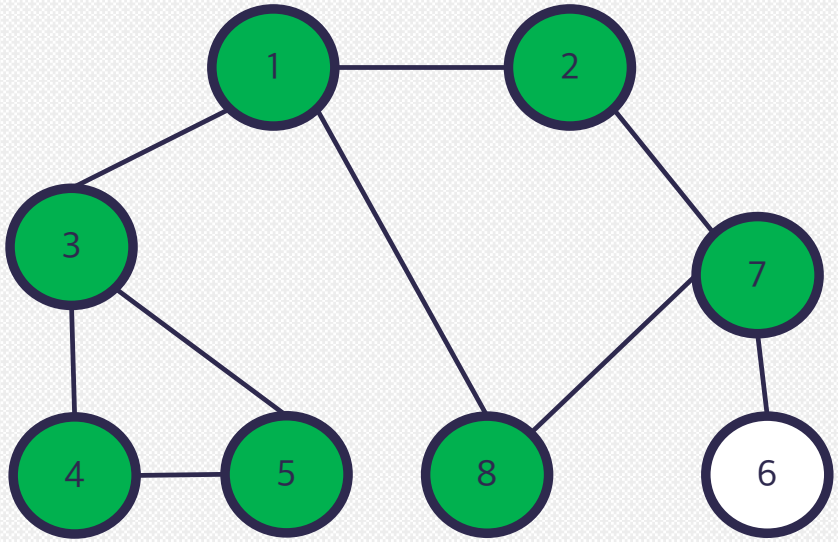
탐색 순서 1 - 2 - 3 - 8 - 7

BFS



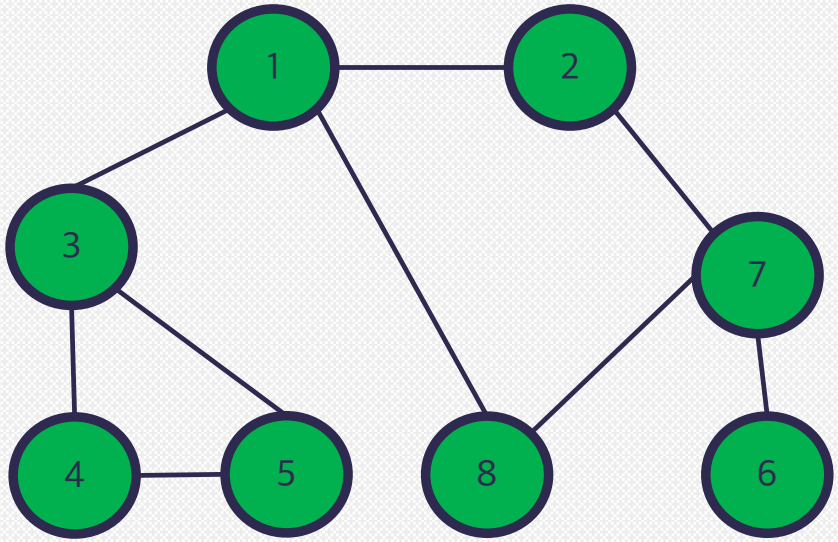
탐색 순서 1 - 2 - 3 - 8 - 7 - 4 - 5

BFS



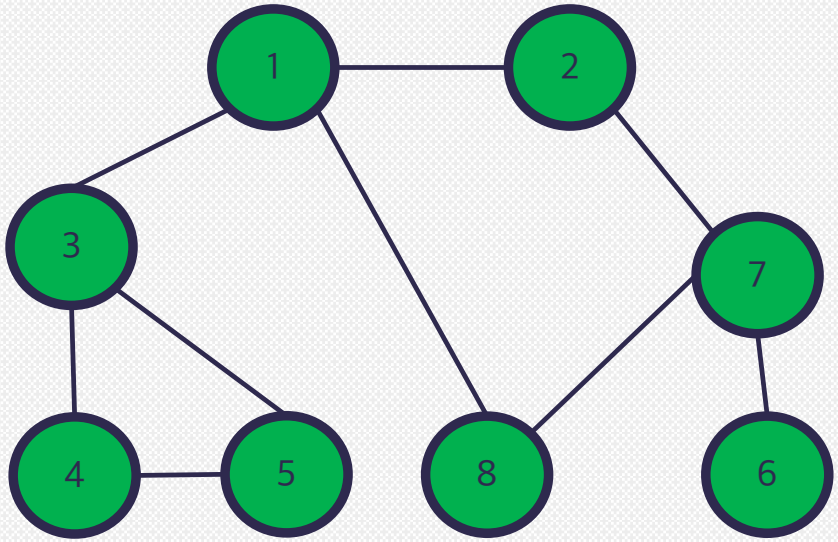
탐색 순서 1 - 2 - 3 - 8 - 7 - 4 - 5

BFS



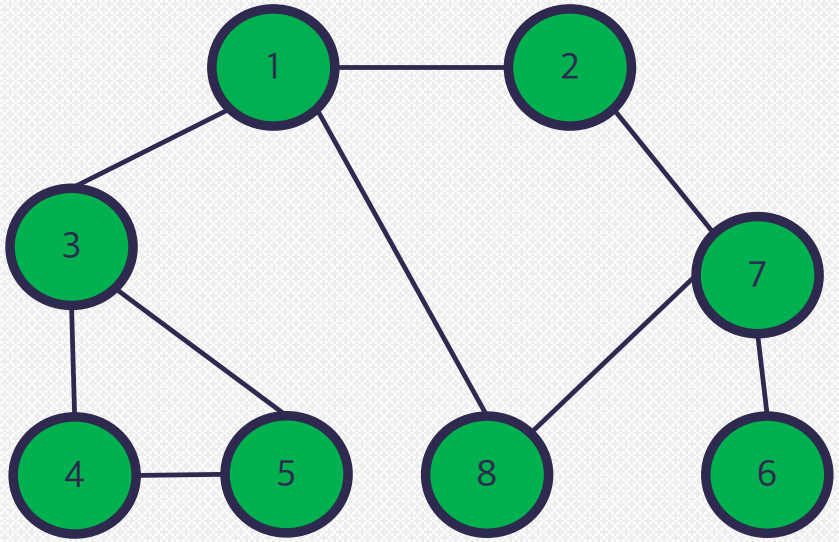
탐색 순서 1 - 2 - 3 - 8 - 7 - 4 - 5 - 6

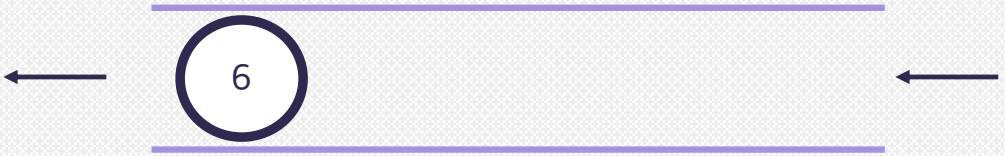
BFS



탐색 순서 1 - 2 - 3 - 8 - 7 - 4 - 5 - 6

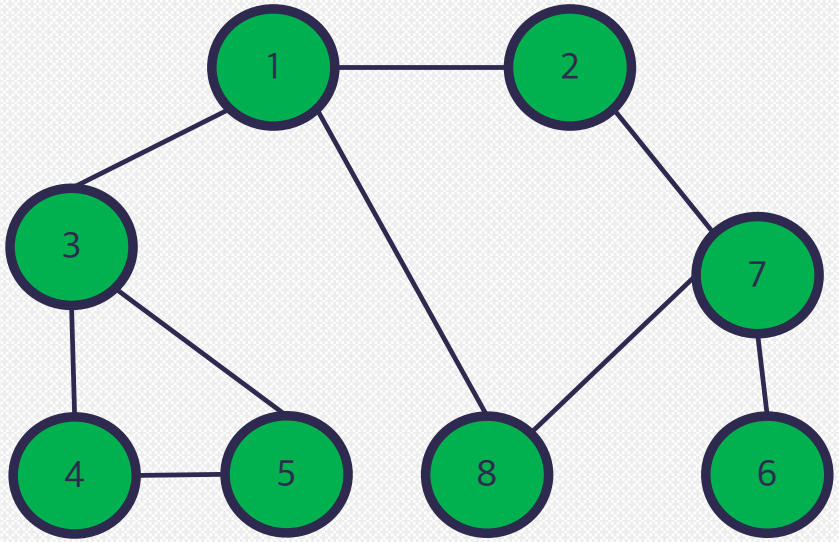
BFS





탐색 순서 1 - 2 - 3 - 8 - 7 - 4 - 5 - 6

BFS



탐색 순서 1 - 2 - 3 - 8 - 7 - 4 - 5 - 6

BFS

```python
def bfs(start):
    q = deque()
    visited[start] = True
    q.append(start)
    while(len(q)):
        node = q.popleft()
        for next in graph[node]:
            if visited[next]:
                continue
            visited[next] = True
            q.append(next)
}
```

```cpp
void bfs(int start) {
    Queue<int> q;
    q.push(start);
    visited[start] = true;

    while(!q.empty()) {
        int x = q.front();
        q.pop();
        for(int i = 0; i<graph[x].size(); i++) {
            int y = graph[x][i];
            if(!visited[y]) {
                visited[y] = true;
                q.push(y);
            }
        }
    }
}
```

```java
public static void bfs(int start) {
    Queue<Integer> q = new LinkedList<>();
    q.offer(start);
    visited[start] = true;

    while(!q.isEmpty()) {
        int x = q.poll();
        for(int i = 0; i<graph.get(x).size(); i++) {
            int y = graph.get(x).get(i);
            if(!visited[y]) {
                visited[y] = true;
                q.offer(y);
            }
        }
    }
}
```

이렇게 해도 괜찮을까 ?

```java
for (int y : graph[x]) {
    if(!visited[y]) dfs(y);
}
}
```

실전 문제

- **음료수 얼려 먹기**

  개념 이론이 부족하면 스스로 학습 후 답지 없이 문제 풀어보기

- **미로 탈출**

  개념 이론이 부족하면 스스로 학습 후 답지 없이 문제 풀어보기