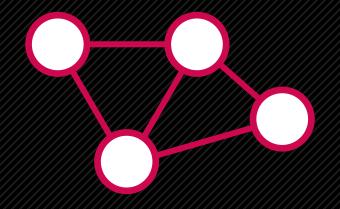


알고리즘 특강 그래프 탐색

코테 단골손님 중 하나인 그래프입니다. DFS/BFS는 정말 능숙하게 사용할 줄 알아야 합니다!







Graph

노드와 그 노드를 잇는 간선을 하나로 모아 놓은 자료구조.

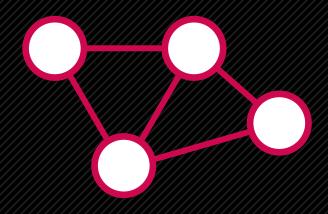
용어 설명



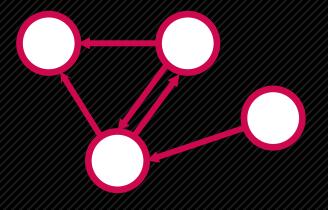


무방향/방향



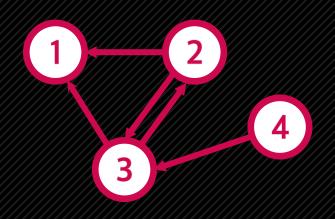


무방향 그래프 간선의 방향이 없음.



방향 그래프 간선에 특정한 방향이 있음.



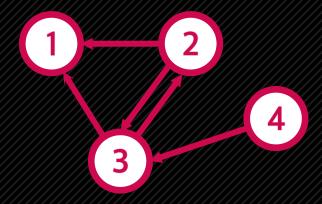


Adjust	Array
---------------	-------

- N*N 크기의 2차원 배열을 생성해서 연결 상태를 나타내는 방법.
- 특정 노드 N와 N_J가 연결되어 있는지 확인: O(1)
- 특정 노드와 연결되어 있는 모든 노드를 확인: O(N)
- 공간 복잡도: O(N*N)

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



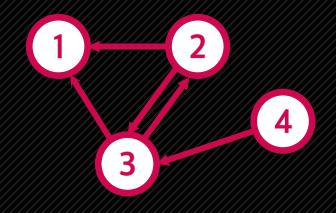


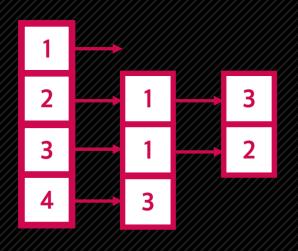
```
#include <iostream>
using namespace std;

int main() {
   int graph[5][5];

   graph[2][1] = 1;
   graph[2][3] = 1;
   graph[3][1] = 1;
   graph[3][2] = 1;
   graph[4][3] = 1;
}
```

구현

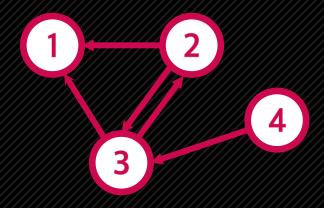




Adjust List

- N개의 리스트를 생성해서 연결 상태를 나타내는 방법.
- 특정 노드 N와 N」가 연결되어 있는지 확인: O(min(Degree(N₁), Degree(N」))
- 특정 노드와 연결되어 있는 모든 노드를 확인: O(Degree(N_i))
- 공간 복잡도: O(M)

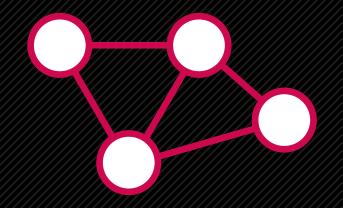


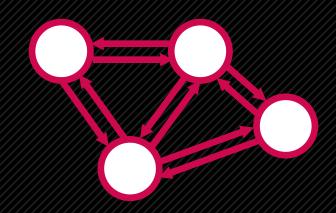


```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   vector<int> graph[5];
   graph[2].push_back(1);
   graph[3].push_back(3);
   graph[3].push_back(1);
   graph[3].push_back(2);
   graph[4].push_back(3);
}
```

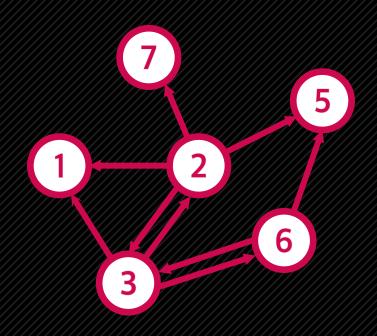
구현에서의 무방향과 방향





● 결국, 무방향 그래프는 모든 간선이 <mark>왕복간선 → 양방향을 다 만들어주면 된</mark>다!

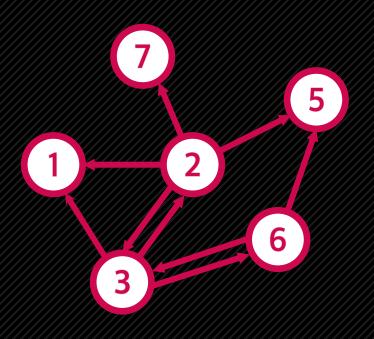
그래프 탐색



Graph Traversal

- 그래프의 모든 노드를 탐색하기 위해 <mark>간선을 따라 순회하는 것</mark>.
- 탐색 방법에 따라 BFS (Breadth First Search), DFS (Depth First Search)로 나뉜다.
- 그래프의 다양한 응용문제는 탐색을 기반으로 이뤄짐!

그래프 탐색



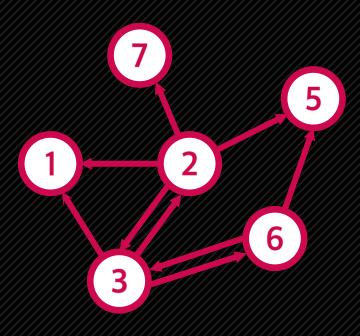
Breadth First Search

- 너비 우선 탐색이라고도 하며, 자신의 자식들부터 순차적으로 탐색.
- 순차 탐색 이후 다른 자식 노드의 자식을 확인하기 위해 큐를 사용.

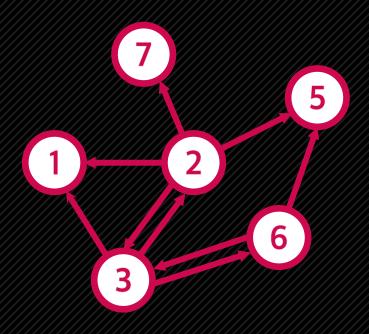
Depth First Search

- <mark>깊이 우선 탐색</mark>이라고도 하며, 최대한 깊게 탐색 후 빠져 나옴.
- 백트래킹의 일종이며, 재귀를 활용함.

그래프 탐색

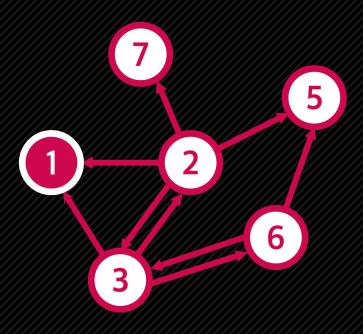


BFS

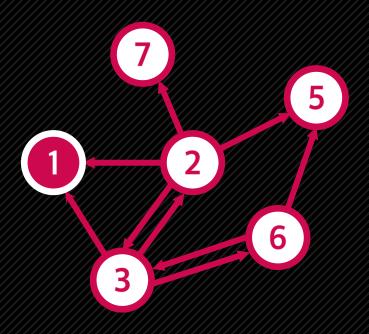


1



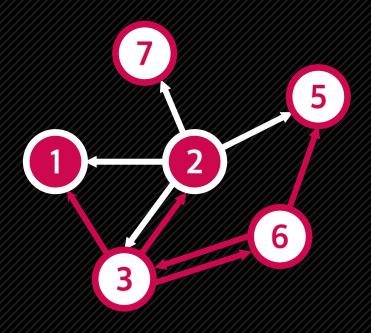


BFS



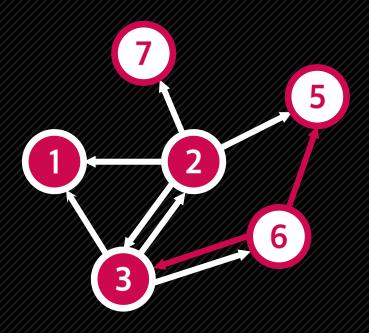
2

BFS



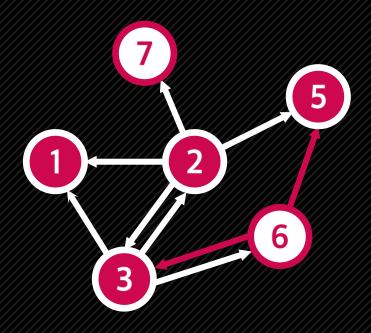








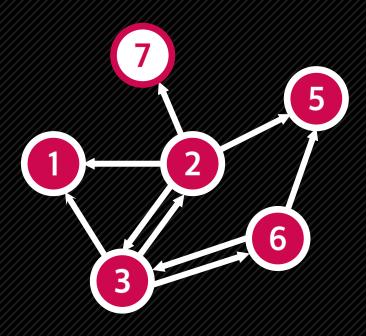




```
function bfs(pos) {
    set Q = Queue
    pos -> Q
    while Q is not empty
        set node = popped element of Q
        for children of pos
            if each child has not been visited
                 visit child
                       child -> Q
}
```

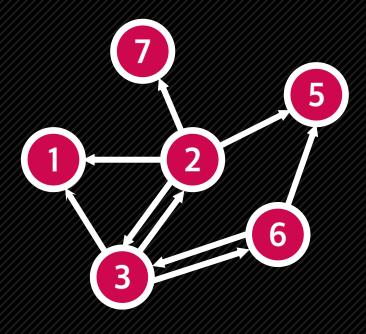


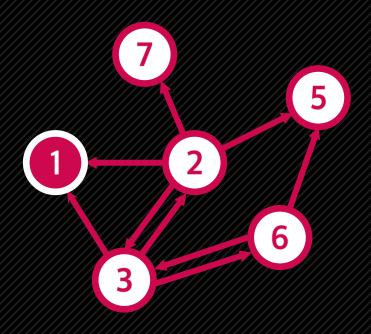








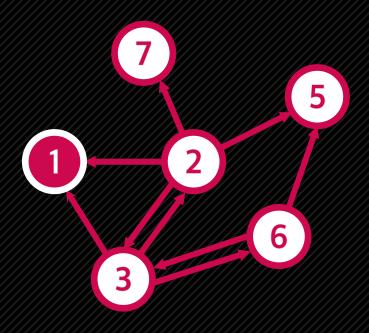




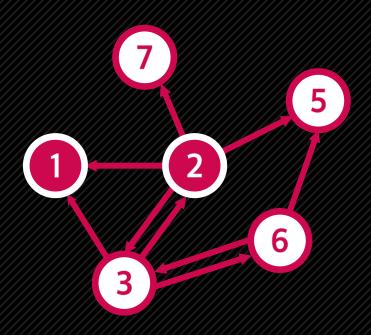
```
function dfs(pos) {
  visit pos
  for children of pos
    if each child has not been visited
        dfs(pos)
}
```

dfs(1)



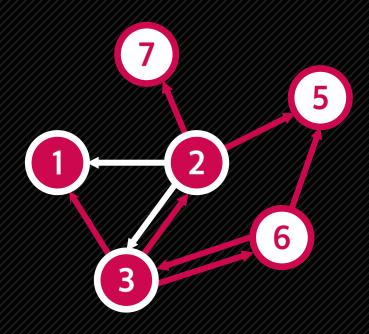


```
function dfs(pos) {
  visit pos
  for children of pos
    if each child has not been visited
        dfs(pos)
}
```



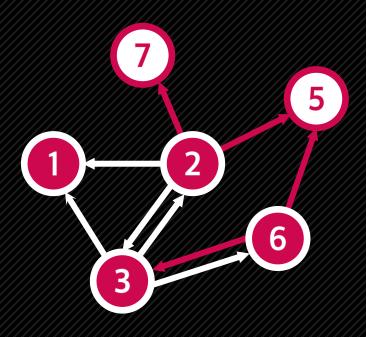
```
function dfs(pos) {
  visit pos
  for children of pos
    if each child has not been visited
        dfs(pos)
}
```

dfs(2)



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

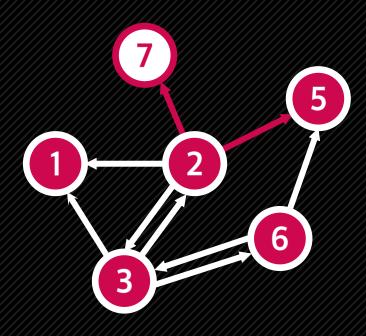
dfs(3) dfs(2)



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

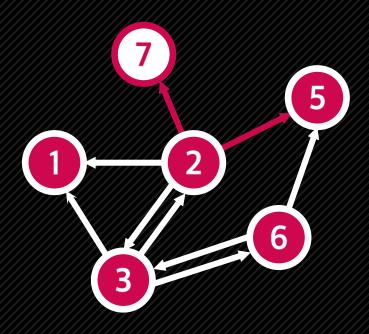
dfs(6) dfs(3) dfs(2)





dfs(5)
dfs(6)
dfs(3)
dfs(2)

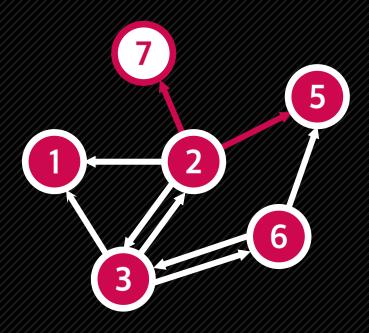




```
function dfs(pos) {
  visit pos
  for children of pos
    if each child has not been visited
        dfs(pos)
}
```

dfs(6) dfs(3) dfs(2)

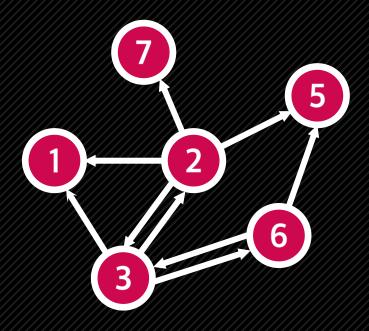




```
function dfs(pos) {
  visit pos
  for children of pos
    if each child has not been visited
        dfs(pos)
}
```

dfs(2)

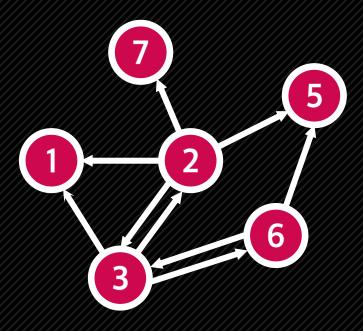




```
function dfs(pos) {
  visit pos
  for children of pos
    if each child has not been visited
        dfs(pos)
}
```







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

연습해보자!

Silver 2 - BFS2+ DFS (#1260)

요약

- 그래프를 BFS와 DFS로 탐색한 결과를 출력하는 프로그램을 작성하시오.
- 단, 방문할 수 있는 정점이 여러 개 인 경우 번호가 작은 정점을 먼저 방문한다.

제약조건

- 정점의 수의 범위는 1 <= N <= 1,000 이다.
- 간선의 수의 범위는 1 <= M <= 10,000 이다.

To be continue,,,