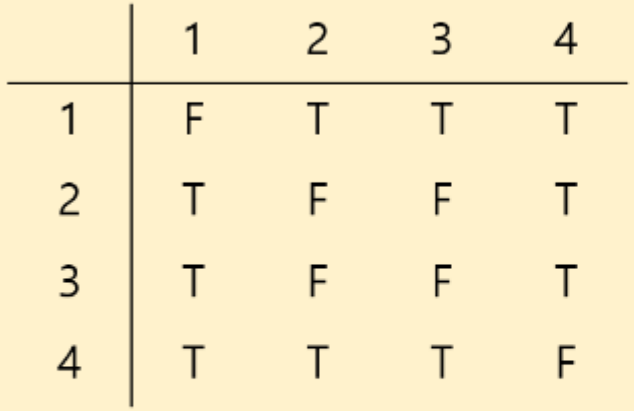
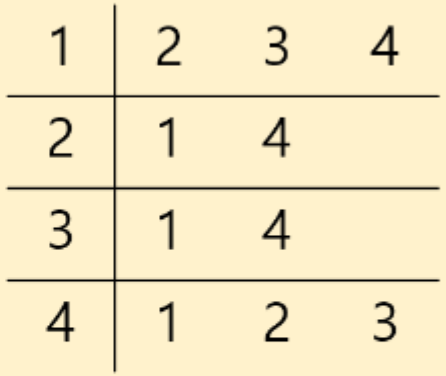
BFS(Breadth First Search)

너비우선탐색

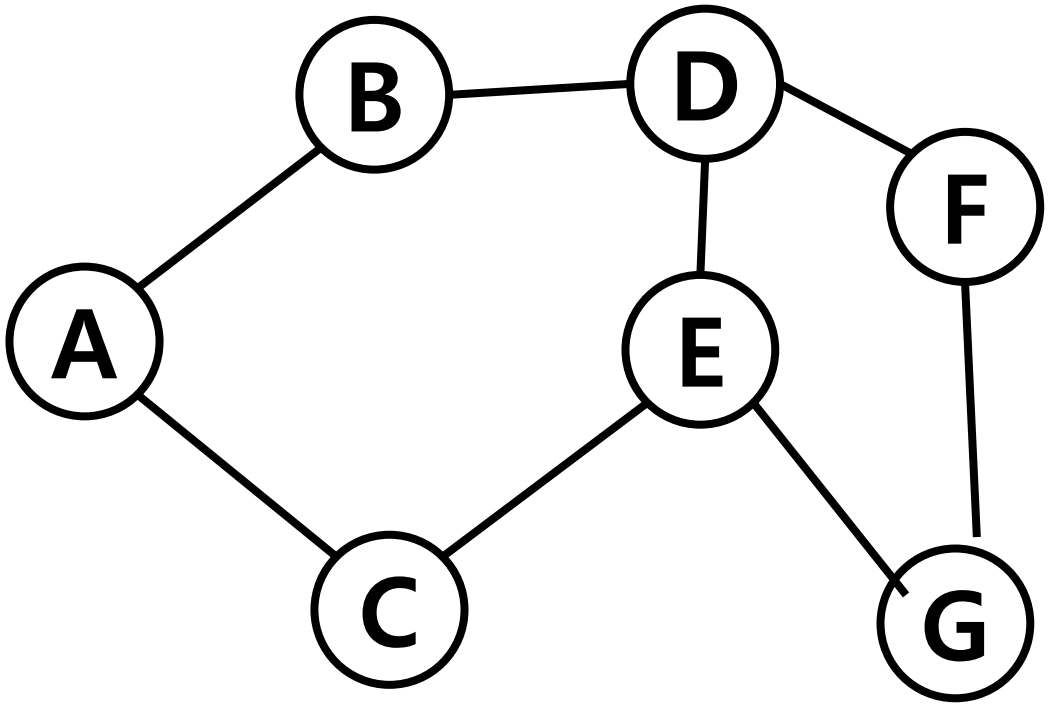
- 트리(Tree), 그래프(Graph)를 너비를 우선으로 탐색하는 알고리즘
- 너비를 우선 탐색하기에 답이 되는 경로가 여러개인 경우에도 최단경로임을 보장한다.
- 특정 위치에서 출발하여 모든 이웃하는 노드들을 탐색하여 해를 찾아 모든 노드들을 검사하는 무정보 탐색이다.
- 방문한 노드들을 차례로 저장한 후 꺼낼 수 있는 자료 구조인 큐(Queue)를 사용하기 때문에 노드가 많을 경우 저장공간이 많이 필요하다.
- 어떤 노드를 방문했었는지 여부를 반드시 검사 해야 한다.

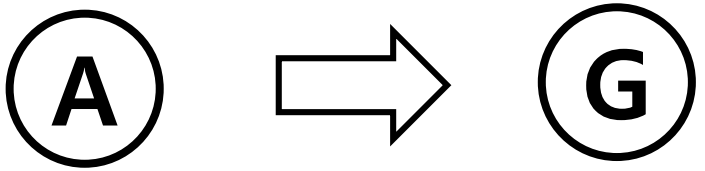
너비우선탐색(BFS) 탐색 시간 복잡도

인접행렬	인접리스트
	
1. 이차원배열 int[][]	1. 링크드리스트 배열 LinkedList[] 2. 어레이리스트 배열 ArrayList[] 3. 어레이리스트를 저장하는 어레이리스트 ArrayList<ArrayList>>

인접행렬: **O(V^2)**
인접리스트: **O(V+E)**

너비우선탐색(BFS) 탐색 조건

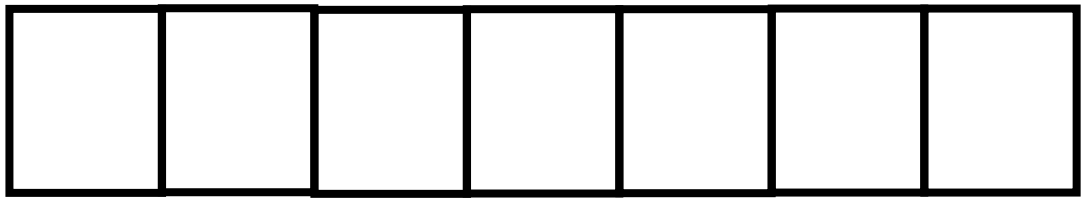




탐색 조건

- 노드 한칸을 이동하는데 걸리는 시간을 1초라고 할때, A → G 까지 탐색하는데 걸리는 최소시간

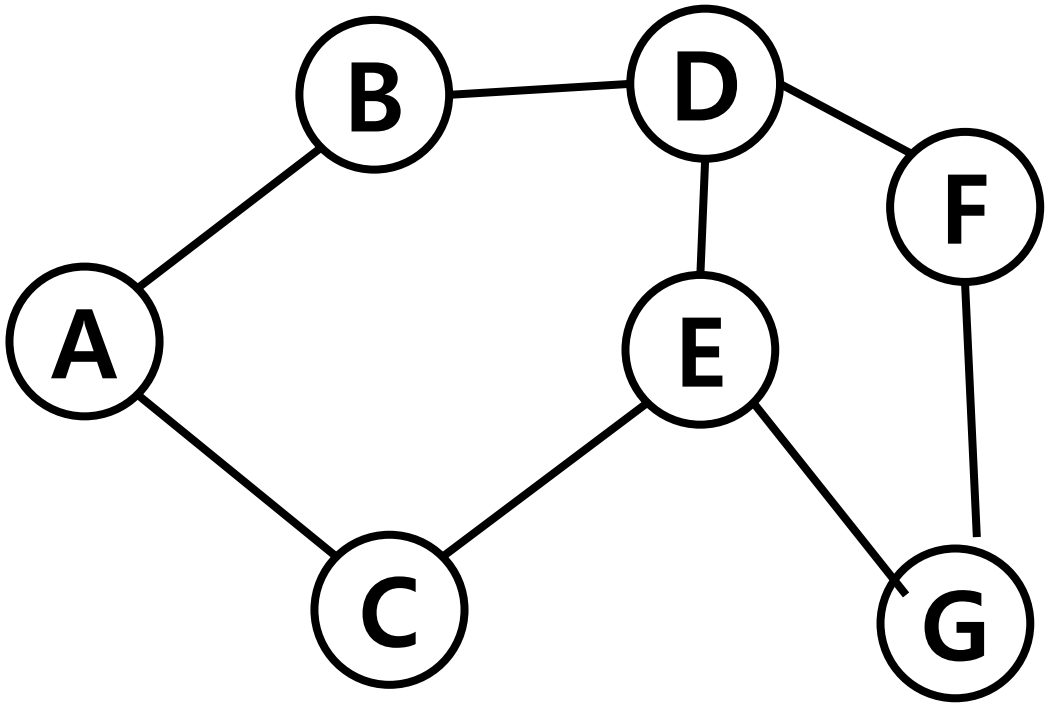
큐(QUEUE)

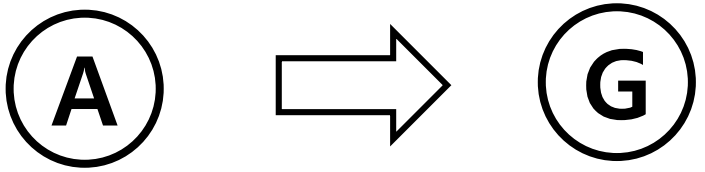


방문 체크용 배열(VISTED)

A	B	C	D	E	F	G

너비우선탐색(BFS) 탐색 과정





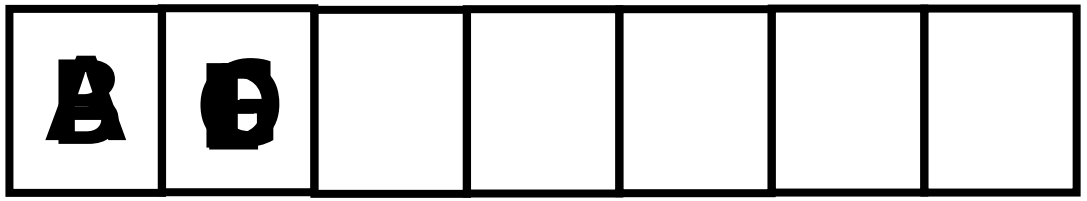
STEP 1

1. visted -1 로 초기화
2. 시작 노드를 push
3. visted 갱신

STEP 2

1. queue front pop
2. 인접한 노드면서 방문하지 않은 노드 push
3. visted 갱신 (인접한 노드가 있을 경우 2, 3 반복)

큐(QUEUE)

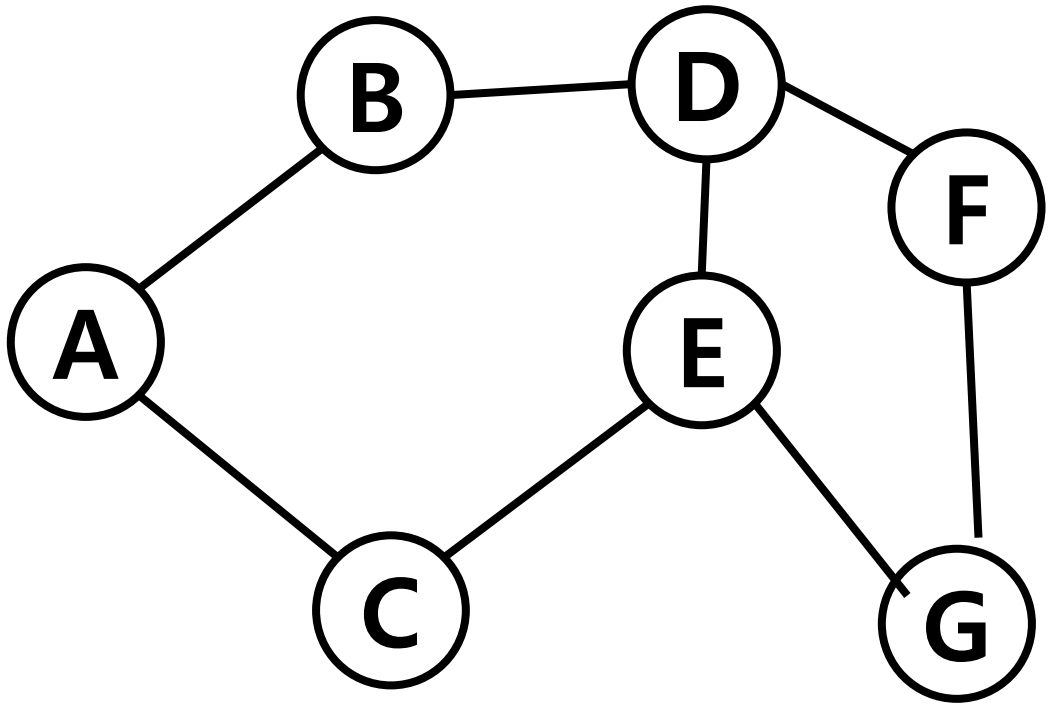


방문 체크용 배열(VISTED)

A	B	C	D	E	F	G
0	1	1	2	2	2	2

너비우선탐색(BFS) 탐색 구현

인접리스트로 구현




```
nodes[A] | B C
nodes[B] | A D
nodes[C] | A E
nodes[D] | B E F
nodes[E] | C D G
nodes[F] | D G
nodes[G] | E F
```

```
nodes[0] | 1 2
nodes[1] | 0 3
nodes[2] | 0 4
nodes[3] | 1 4 5
nodes[4] | 2 3 6
nodes[5] | 3 6
nodes[6] | 4 5
```

너비우선탐색(BFS) 탐색 구현

```cpp
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    // 초기 세팅
    init();

    // bfs 탐색
    bfs(0, 6);

    return 0;
}
```

```cpp
void init()
{
    // visted -1로 초기화
    for (int i = 0; i < 7; i++)
    {
        visted[i] = -1;
    }

    // 인접한 그래드 초기화
    // A:0 B:1 C:2 D:3 E:4 F:5 G:6
    nodes[0].push_back(1);
    nodes[0].push_back(2);

    nodes[1].push_back(0);
    nodes[1].push_back(3);

    nodes[2].push_back(0);
    nodes[2].push_back(4);

    nodes[3].push_back(1);
    nodes[3].push_back(4);
    nodes[3].push_back(5);

    nodes[4].push_back(2);
    nodes[4].push_back(3);
    nodes[4].push_back(6);

    nodes[5].push_back(3);
    nodes[5].push_back(6);

    nodes[6].push_back(4);
    nodes[6].push_back(5);
}
```

```cpp
void bfs(int start, int end)
{
    queue<int> QUEUE;

    // 시작 위치
    QUEUE.push(start);
    visted[start] = 0;

    while (QUEUE.empty() == false)
    {
        // 현재 노드
        int currentNode = QUEUE.front();
        QUEUE.pop();

        // 현재 노드가 목적지면..
        if (currentNode == end)
        {
            cout << visted[end];
        }

        for (int index = 0; index < nodes[currentNode].size(); index++)
        {
            // 인접한 노드
            int nextNode = nodes[currentNode][index];

            // 방문한적이 있다면 넘어감
            if (visted[nextNode] != -1) continue;

            // 방문한적이 없다면 QUEUE에 인접한 노드 PUSH
            QUEUE.push(nextNode);

            // 방문체크 인접한노드 = 현재노드 + 1
            visted[nextNode] = visted[currentNode] + 1;
        }
    }
}
```

ETC

유사 알고리즘
- 다익스트라(Dijkstra) 알고리즘
- 최소스패닝트리 프림(Prim) 알고리즘

참고 문서
- https://coding-factory.tistory.com/612
- https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

- 코드