DFS - BFS

📌 개요

DFS (Depth-First Search)와 BFS (Breadth-First Search)는 그래프 탐색에서 가장 기본적인 알고리즘으로, 각각 **깊이 우선**과 **너비 우선**의 방식으로 노드를 탐색합니다. 이 두 알고리즘은 트리나 그래프에서 경로 탐색, 연결 요소 탐색, 사이클 탐색, 최단 경로 문제 등에 널리 사용됩니다.

📚 필요 선행 학습 지식 (필수)

선행 지식	설명
그래프 이론	정점(Vertex), 간선(Edge), 인접 리스트/행렬 이해
배열/리스트/큐/스택	스택과 큐의 차이와 활용 방식 이해
재귀 함수	재귀의 원리, 스택 사용, 탈출 조건 (base case)
방문 배열 (visited array)	중복 방문 방지, 사이클 방지
인접 리스트 (adjacency list)	그래프 표현 방식 중 가장 일반적

> 💡 **추천 학습 순서**:
>
> 1. 그래프 기본 개념 (정점, 간선, 방향성)
> 2. 인접 리스트 표현
> 3. DFS/DFS 재귀 구현
> 4. BFS 큐 구현
> 5. 문제 해결 (예: 2D 그리드, N-퀸, 사이클 탐색)

🔍 핵심 개념 정리

항목	DFS	BFS
탐색 방식	깊이 우선 (가장 깊은 브랜치를 우선 탐색)	너비 우선 (가장 가까운 노드부터 탐색)
사용 자료구조	재귀 또는 스택 (stack)	큐 (queue)
탐색 순서	1 → 2 → 3 → 4 → 5 (깊이 우선)	1 → 2 → 3 → 4 → 5 (단계별 확장)
메모리 사용	낮음 (재귀 스택)	높음 (전체 레벨을 저장)
최단 경로	❌ (가장 깊은 경로)	✅ (최단 거리)
사이클 탐색	가능 (재방문 방지 필요)	가능 (방문 여부 체크)

항목	DFS	BFS
응용 예시	경로 탐색, 연결 요소, 순열 생성	최단 경로, 최단 거리, 네트워크 확장

📚 각 알고리즘의 원리와 특징

✅ DFS (Depth-First Search)

- **이론**: 한 노드에서 가능한 모든 경로를 탐색한 후, 다음 경로로 넘어감.
- **특징**:
  - 재귀적으로 구현이 쉽고 직관적.
  - 메모리 효율적 (스택 크기 제한 없이 작동 가능).
  - 사이클 탐색에 유리 (예: 그래프에서 사이클 여부 확인).
- **이용 예**:
  - 그래프의 연결 요소 (connected components) 탐색
  - 경로 탐색 (예: 퀸 문제, N-퀸)
  - 순열/조합 생성 (백트래킹 기반)

✅ BFS (Breadth-First Search)

- **이론**: 시작 노드에서 1단계, 2단계, 3단계 순으로 확장하여 탐색.
- **특징**:
  - 최단 경로를 보장 (단순한 그래프에서 최단 거리 구할 때 유리).
  - 메모리 사용이 많음 (전체 레벨 저장 필요).
- **이용 예**:
  - 최단 경로 (예: 워드 브리핑, 지도 탐색)
  - 최단 거리 문제 (예: 2D 그리드에서 시작점에서 도착점까지의 최단 거리)
  - 탐색 범위 확장 (예: 움직임 제한된 자동차의 이동)

🚀 Node 이동 순서 (예시 그래프)

예시: 그래프 `G`에서 시작 노드 `1`부터 시작

DFS 순서 (재귀 기반)

1 → 2 → 4 → 5 → 3 → 6

> (깊이 우선 탐색: 1에서 2로, 2에서 4, 5로, 5에서 끝나고 3, 6으로 이동)

BFS 순서 (큐 기반)

1 → 2 → 3 → 4 → 5 → 6

> (너비 우선: 1에서 2,3으로 확장 → 그 후 4,5,6으로 확장)

💡 C++ 예제 (재귀 + 반복)

1. DFS (재귀 기반)

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

void dfs(int node, vector<vector<int>>& graph, vector<bool>& visited) {
    visited[node] = true;
    cout << node << " ";

    for (int neighbor : graph[node]) {
        if (!visited[neighbor]) {
            dfs(neighbor, graph, visited);
        }
    }
}

// 전체 DFS 함수
void dfsTraversal(int start, vector<vector<int>>& graph, int n) {
    vector<bool> visited(n, false);
    cout << "DFS 순서: ";
    dfs(start, graph, visited);
    cout << endl;
}
```

> ✅ **사용법**: `graph[i]` 는 노드 i의 인접 노드를 저장 (0-based)

---

2. BFS (큐 기반)

```cpp
#include <iostream>
#include <vector>
```

```cpp
#include <queue>
using namespace std;

void bfs(int start, vector<vector<int>>& graph, int n) {
    vector<bool> visited(n, false);
    queue<int> q;

    q.push(start);
    visited[start] = true;

    cout << "BFS 순서: ";
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        cout << node << " ";

        for (int neighbor : graph[node]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
            }
        }
    }
    cout << endl;
}
```

> ✅ **사용법**: 큐를 통해 각 레벨을 순차적으로 탐색

📌 각 알고리즘의 종류

종류	설명
DFS (재귀)	간단하고 직관적, 메모리 효율적
DFS (반복)	스택을 수동으로 관리 (재귀보다 안정성 높음)
BFS (큐)	너비 우선 탐색, 최단 경로에 최적
DFS with backtracking	조합/순열 문제 (예: 1~8 중 3개 고르기)
BFS with priority queue	최단 경로 (예: 다익스트라 알고리즘 기반)

🚨 주의사항 및 실무 적용

항목	주의사항
재귀 깊이 제한	C++에서 재귀 깊이가 너무 깊으면 스택 오버플로우 발생 (예: 10000 depth)
메모리 최적화	DFS는 메모리 적음, BFS는 전체 레벨 저장 → 메모리 부족 시 문제 발생
사이클 탐색	DFS에서 사이클 탐색 가능, 그러나 방문 체크 필수
무향 그래프 vs 방향 그래프	방향성 여부에 따라 탐색 방향이 달라질 수 있음

📚 예제 문제 (문제 해결 연계)

문제	해결 방법
N-퀸 문제	DFS + 백트래킹 (각 행에 퀸 배치)
단순한 경로 탐색	DFS (깊이 우선 탐색)
최단 거리 (무게 없는 그래프)	BFS (단계별 확장)
연결 요소 (connected components)	DFS (각 컴포넌트에서 탐색)
트리의 높이 계산	DFS (깊이 추적)

📌 정리 요약

항목	**DFS**	**BFS**
탐색 방식	깊이 우선	너비 우선
자료구조	재귀/스택	큐
최단 경로	❌	✅
메모리	낮음	높음
사용 목적	경로 탐색, 사이클 탐색	최단 거리, 네트워크 확장

📝 추가 팁

- **테스트 코드 작성**: 작은 그래프(3~5개 노드)로 테스트 후 확장
- **테스트 케이스**:
  - 사이클 포함 그래프

    - 무방향 그래프
    - 방향 그래프
    - 단절된 그래프 (disconnected)
- **버그 예방**: `visited` 배열 초기화, `queue` 비어있을 때 조건 체크

📌 이 정리가 필요한 경우

- **알고리즘 학습** (입시, 인터뷰, 프로그래밍 경쟁)
- **그래프 문제 해결** (예: 네트워크, 지도, 로봇 경로)
- **백트래킹 문제** (예: 조합, 순열, N-퀸)

📌 관련 노트 (추가 참고)

- 백트래킹 - N-퀸
- 그래프 이론
- 큐와 스택 비교
- 재귀 vs 반복

> ✅ **요약**:
>
> - **DFS**: 깊이 우선, 재귀/스택, 최단 경로 ❌
> - **BFS**: 너비 우선, 큐, 최단 경로 ✅
> - **핵심**: 방문 배열, 자료구조 선택, 문제 유형에 맞는 선택