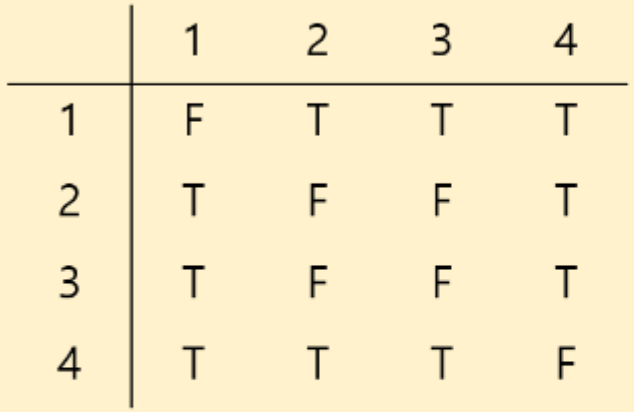
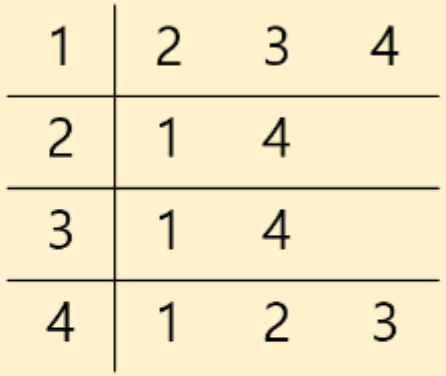
DFS(Dreadth First Search)

깊이우선탐색

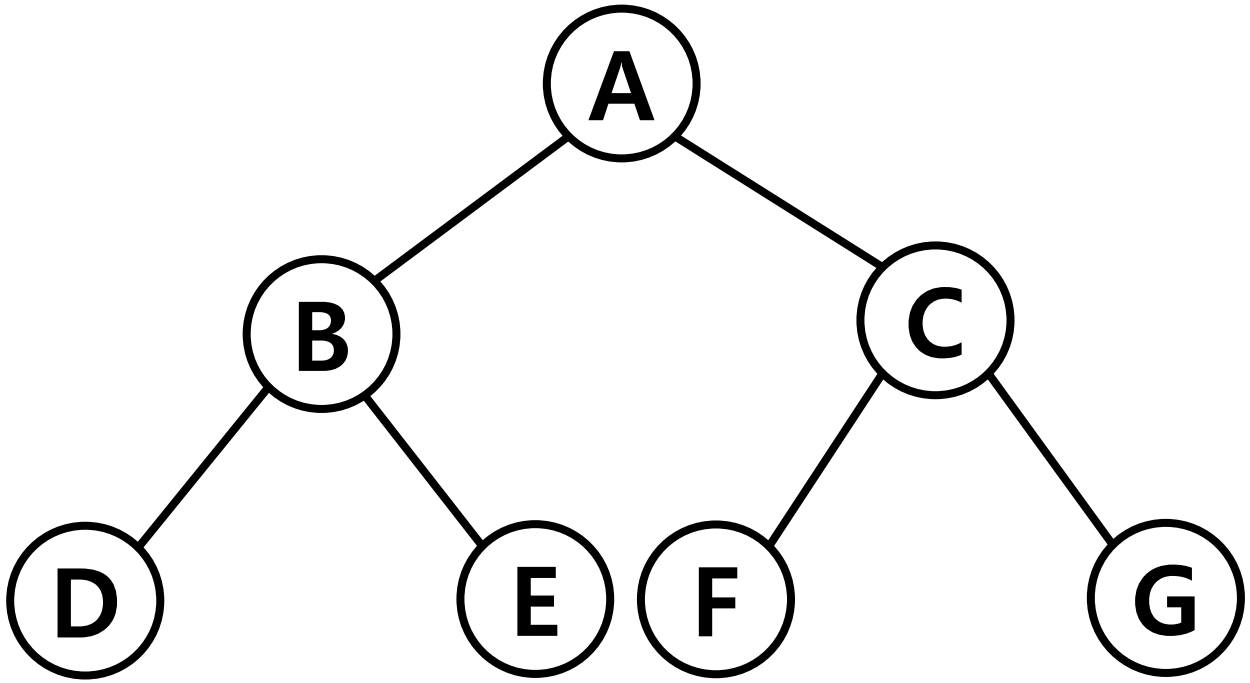
- 트리(Tree), 그래프(Graph)를 깊이를 우선으로 탐색하는 알고리즘
- 깊이를 우선 탐색하기에 답이 되는 경로가 여러개인 경우에도 최단경로임을 보장할 수 없다.
- 특정 위치에서 출발하여 모든 이웃하는 노드들을 탐색하여 해를 찾아 모든 노드들을 검사하는 무정보 탐색이다.
- 스택(Stack) 또는 재귀를 통해 구현 할 수 있다.
- 어떤 노드를 방문했었는지 여부를 반드시 검사 해야 한다.

깊이우선탐색(DFS) 탐색 시간 복잡도

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

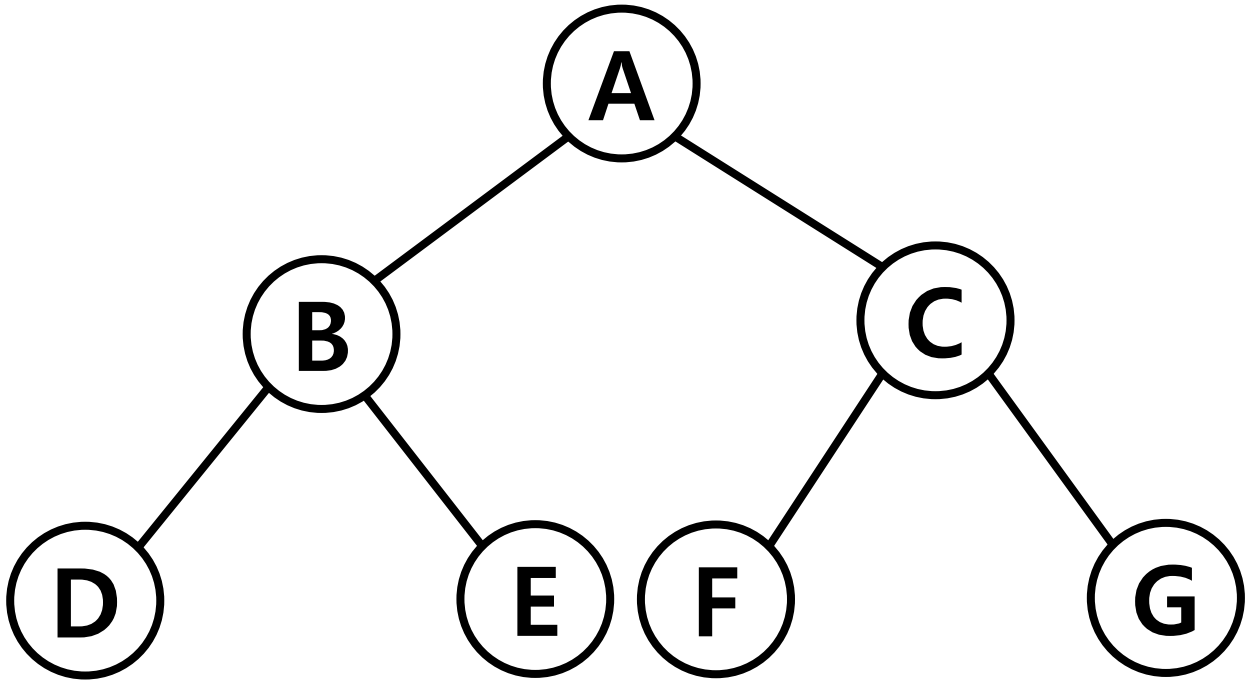
인접행렬: **O(V^2)**
인접리스트: **O(V+E)**

깊이우선탐색(DFS)



A 루트노드부터 탐색을 시작하는 경우
왼쪽부터 탐색을 시작하면
A→B→D→E→C→F→G
순으로 탐색하게 된다.

깊이우선탐색(DFS) vs 너비우선탐색(BFS)

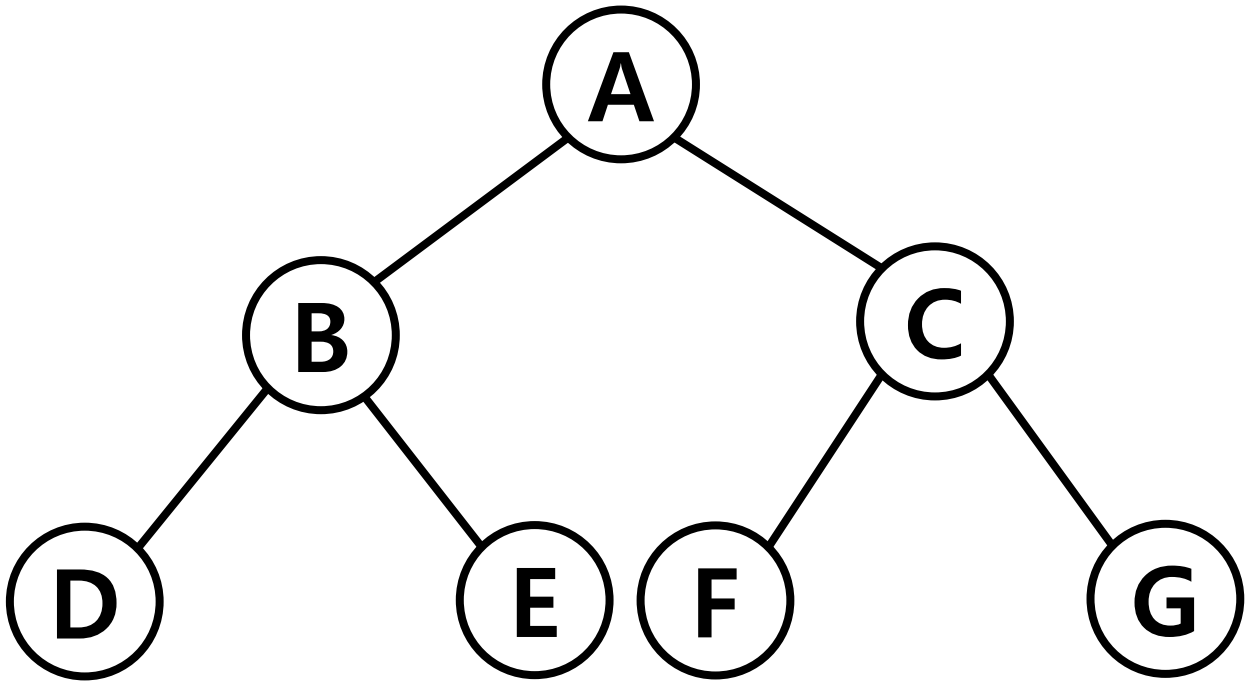


DFS
- 스택(STACK)를 사용해 구현
- 탐색 순서: A→B→D→E→C→F→G

BFS
- 큐(QUEUE)를 사용해 구현
- 탐색 순서: A→B→C→D→E→F→G

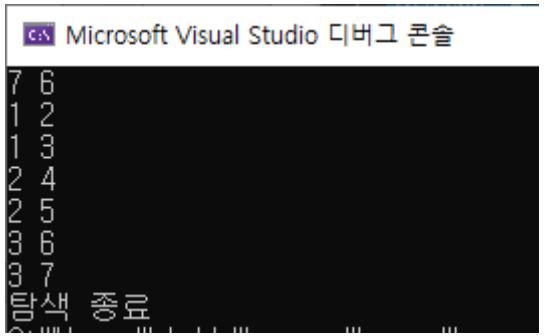
깊이우선탐색(DFS) 탐색 구현

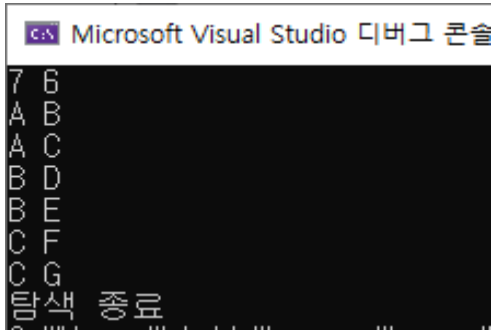
재귀, 인접리스트로 구현



인접리스트를 사용하는 이유
- 실제 연결된 노드만 저장
- 모든 vector의 원소의 개수 합 = 간선의 개수
- 전체 노드 탐색 시 시간복잡도 O(E)

입력
정점 개수(V) 간선 개수(E)
연결할 두 정점을 간선 개수 만큼 입력 받는다.





깊이우선탐색(DFS) 탐색 코드

```cpp
#include <iostream>
#include <queue>
#include<stack>
#include<vector>
#include<algorithm>

using namespace std;

// 정점에 대한 정보와 비용이 담긴 그래프
vector<int> tree[100001];

// 정점 방문체크용 배열
vector<int> visted(10001);

void DFS(int current_node)
{
    // 초기 위치 세팅
    visted[current_node] = true;

    // 현재 정점의 인접한 정점 push
    for (int idx = 0; idx < tree[current_node].size(); idx++)
    {
        int next = tree[current_node][idx];

        if (visted[next] == true) continue;

        DFS(next);
    }
}
```

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

    // 정점 개수와 간선 개수
    int V, E;
    cin >> V >> E;

    // 초기 세팅
    for (int idx = 1; idx <= E; idx++)
    {
        char start, end;
        cin >> start >> end;

        start -= 64;
        end -= 64;
        tree[start].push_back(end);
        tree[end].push_back(start);
    }

    // DFS(Depth First Search)
    DFS(1);

    cout << "탐색 종료";

    return 0;
}
```