그래프

노드와 그 노드를 연결하는 간선들을 하나로 모아 놓은 비선형 자료구조

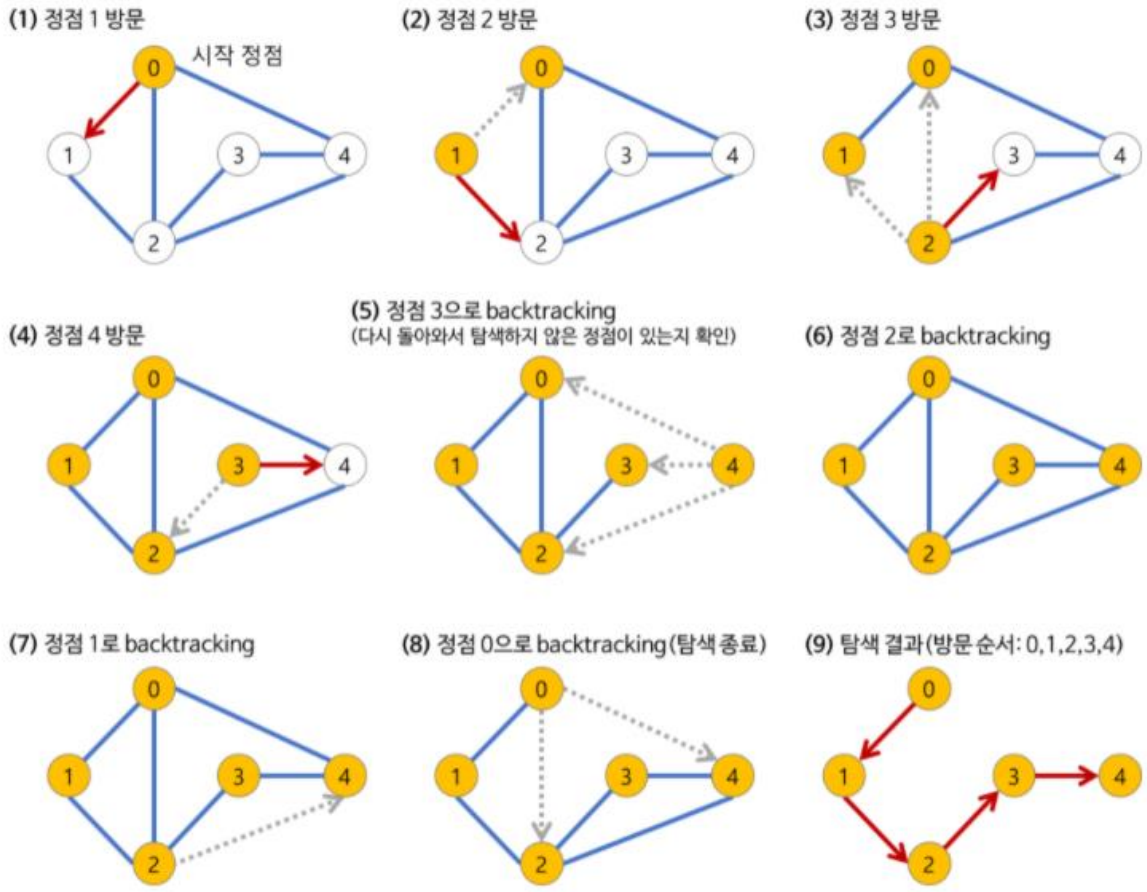
+ 방향 그래프/무방향그래프

그래프 탐색

하나의 정점으로부터 시작해서 차례대로 모든 정점들을 방문

DFS(Depth-First Search) 깊이 우선 탐색

깊이 우선 탐색이란 시작점에서 다음 브랜치로 넘어가기 전에 해당 브랜치를 완벽하게 탐색하는 방법이다.



- 노드 방문시에 반드시 방문(visited)여부를 검사해야 한다 – 무한 루프 방지

구현 알고리즘

1. 루트 노드에서 시작한다.

2. 루트노드와 인접하고 방문 된 적 없는 노드를 방문한다.(가장 깊은 노드까지)

3. 인접하고 방문 된 적 없는 노드가 없을 경우(가장 깊은 노드를 방문 한 뒤) 갈림 길로 돌아와 다른 방향의 노드를 방문한다.

모든 노드를 방문하고자 하는 경우에 이 방법을 선택

BFS보다 좀 더 간단함

검색 속도 자체는 BFS에 비해 느림

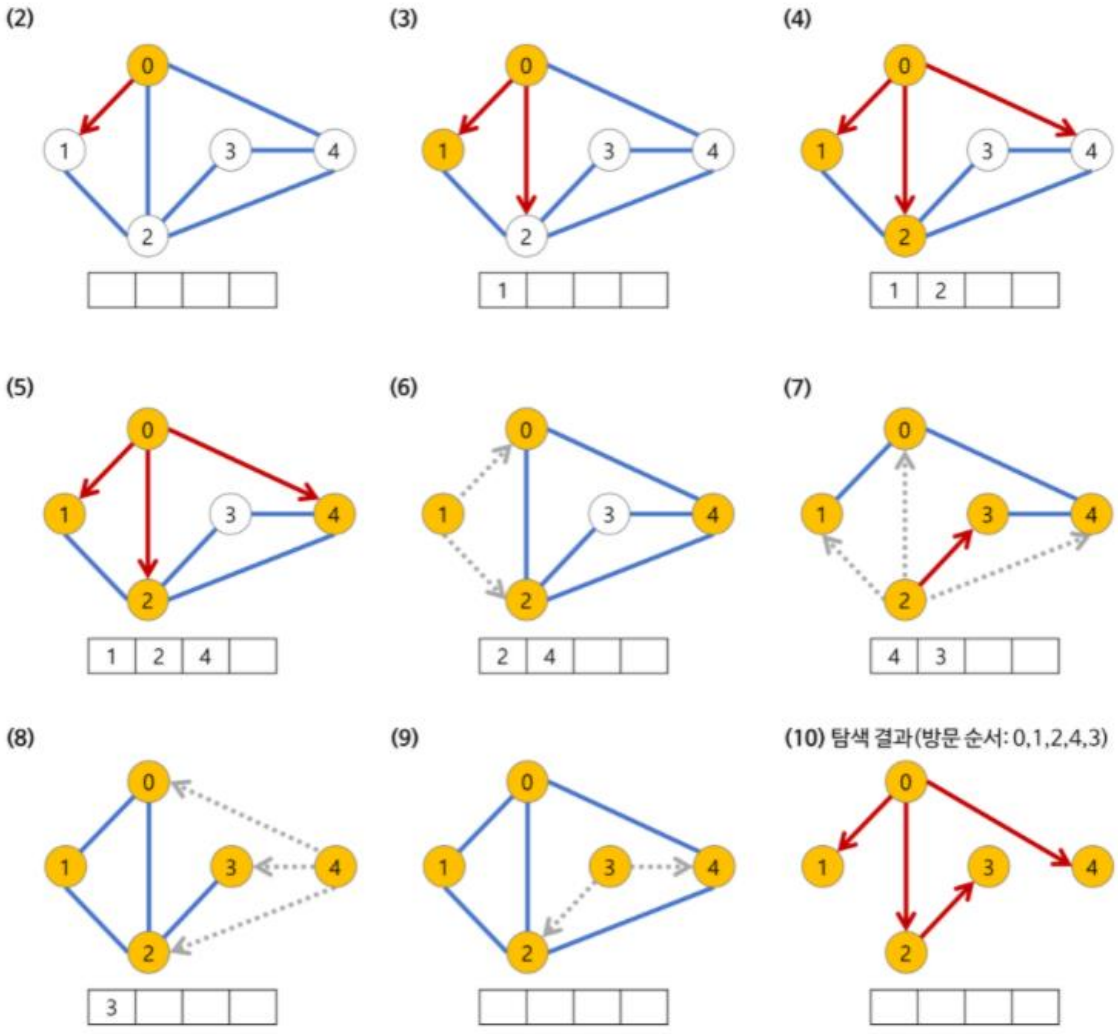
DFS 코드(C++)

```
void DFS(int x) {
    cout << x << " ";
    visit[x] = true; //방문체크

    for (int i = 1; i <= n; i++) {
        //현재 노드와 연결되어 있고, 연결되어있는 노드를 방문하지 않았을때
        if (graph[x][i] == 1 && visit[i] != true) {
            DFS(i);
        }
    }
}
```

BFS(Breadth-First Search) 너비 우선 탐색

너비 우선 탐색이란 시작점에서 인접한 노드를 먼저 탐색하는 방법이다.



- 노드 방문시에 반드시 방문(visited)여부를 검사해야 한다 – 무한 루프 방지

구현 알고리즘

1. 루트노드에서 시작한다.

2. 루트노드와 인접하고 방문 된 적 없고, 큐에 저장되지 않은 노드를 Queue에 넣는다.

3. Queue에서 dequeue하여 가장 먼저 큐에 저장한 노드를 방문한다.

주로 두 노드 사이의 최단경로를 찾고 싶을 때 사용한다.

BFS코드(C++)

```cpp
void BFS(int x) {
    q.push(x);
    visit[x] = true;
    cout << x << " ";

    while (!q.empty()) {
        x = q.front();
        q.pop();

        for (int i = 1; i <= n; i++) {
            //현재 노드와 연결되어 있고, 연결되어있는 노드를 방문하지 않았을때
            if (graph[x][i] == 1 && visit[i] != true) {
                q.push(i);
                visit[i] = true;
                cout << i << " ";
            }
        }
    }
}
```

DFS/BFS를 활용한 문제 유형

1. 그래프의 모든 정점을 방문하는 것이 주요한 문제

단순히 모든 정점을 방문하는 것이 중요한 문제의 경우 두가지 방법 중 편한 것을 사용하면 된다.

2. 경로의 특징을 저장해 두어야 하는 문제

예를 들어 각 정점에 숫자가 적혀 있고 a부터 b까지의 경로를 구하는데 경로에 같은 숫자가 있으면 안된다는 문제 같은 경우

각각의 경로마다 특징을 저장해 두어야 할 때는 DFS를 사용(BFS는 경로의 특징을 가지지 못함)

3. 최단거리를 구해야 하는 문제

미로 찾기나 최단거리를 구해야 할 경우 BFS가 유리

DFS로 경로를 검색할 경우 처음으로 발견되는 해답이 최단거리가 아닐 수 있지만 BFS은 현재 노드에서 가까운 곳부터 찾기 때문에 먼저 찾아지는 해답이 최단거리이기 때문

+ 검색 대상 그래프가 크다면 DFS고려

+ 검색 대상의 규모가 크지 않고 검색 시작 지점으로부터 원하는 대상이 별로 멀지 않다면 BFS