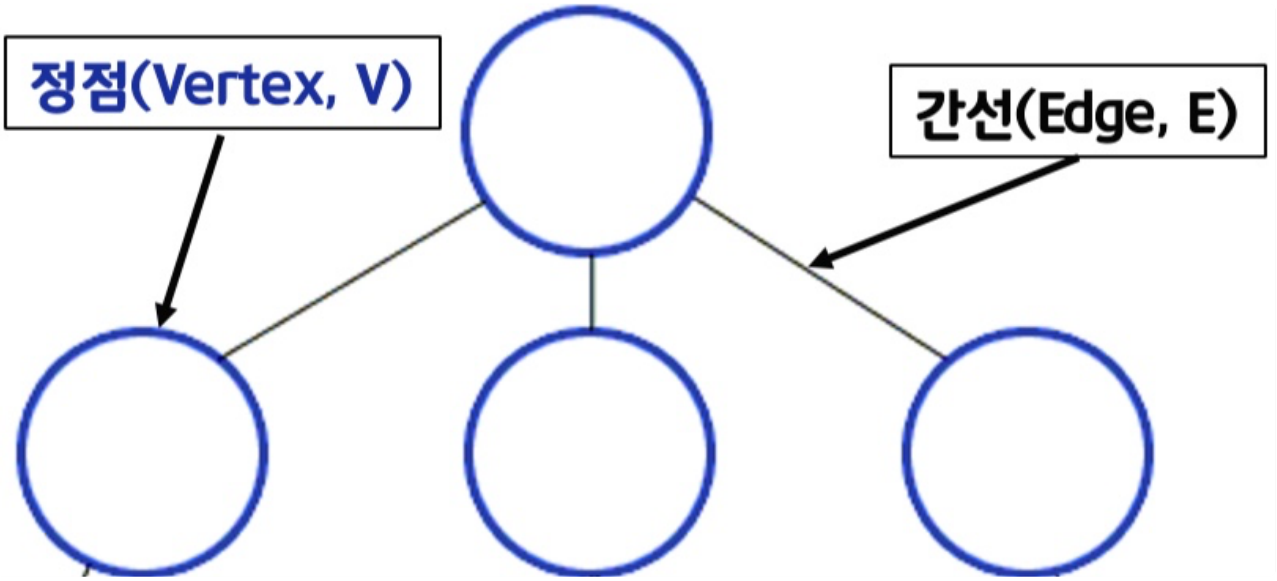
그래프 탐색 알고리즘 (Graph Search Algorithm)

그래프(Graph)란?

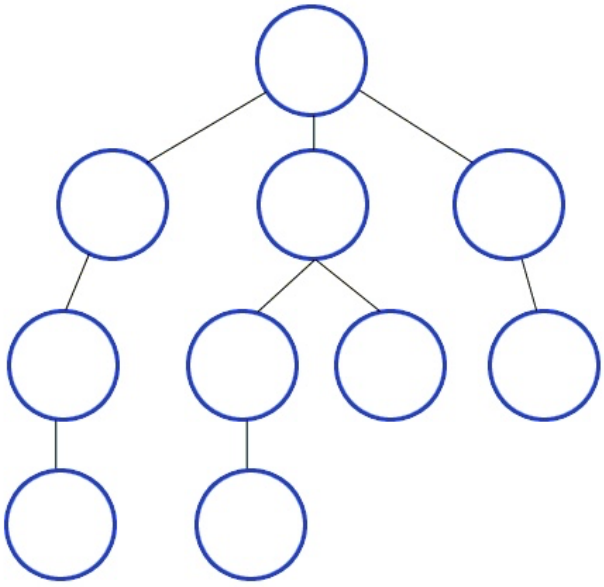


그래프(Graph)는 정점(Vertex)과 간선(Edge)으로 이루어진 추상적인 자료구조 입니다.

그래프 탐색 알고리즘 (Graph Search Algorithm)

그래프 탐색 알고리즘은 그래프에서 특정한 정점을 시작으로 그래프를 탐색하는 알고리즘을 말합니다. 그래프는 노드와 간선으로 구성되며, 그래프 탐색 알고리즘은 이러한 노드와 간선들을 이용해 탐색을 수행합니다.

깊이 우선 탐색 (DFS, Depth-First Search)



깊이 우선 탐색(Depth First Search, DFS)은 가장 기본적인 완전 탐색 알고리즘으로, 재귀를 이용하면 쉽게 구현할 수 있습니다. DFS는 **탐색 공간이 제한**되어 있고, **탐색 공간 내 탐색 목표가 있는지 검사**할 때 유용하게 사용됩니다.

DFS (with Javascript)

```
class Graph {
  constructor(v) {
    this.V = v;
    this.adj = new Array(v);

    for (let i = 0; i < v; i++) {
      this.adj[i] = new Array();
    }
  }

  addEdge(v, w) {
    this.adj[v].push(w);
  }

  DFS(s) {
    const visited = new Array(this.V).fill(false);
    const stack = [];

    stack.push(s);

    while (stack.length) {
      s = stack.pop();

      if (!visited[s]) {
        visited[s] = true;
        console.log(s + " ");
      }

      this.adj[s].forEach((v) => {
        if (!visited[v]) stack.push(v);
      });
    }
  }
}

const graph = new Graph(4);

graph.addEdge(0, 1);
graph.addEdge(0, 2);
graph.addEdge(1, 2);
graph.addEdge(2, 0);
graph.addEdge(2, 3);
graph.addEdge(3, 3);

graph.DFS(2);
```

위 코드에서는 DFS를 구현하기 위해 스택을 사용합니다. 방문한 노드를 스택에 저장하고, 스택이 빌 때까지 반복적으로 노드를 꺼내어 방문하며, 해당 노드와 연결된 노드를 스택에 추가합니다. 이를 반복하여 그래프의 모든 노드를 방문합니다.

백트래킹 (backtracking)

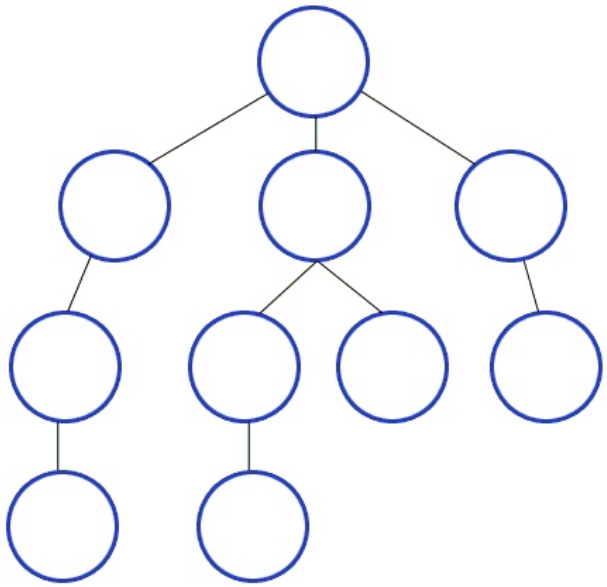
백트래킹 알고리즘은 현재의 상태에서 다음 상태로 넘어가기 전에, 이전 단계로 돌아가서 다른 가능성을 검사하는 것을 반복하여, 모든 경우의 수를 검사합니다. 이전 단계로 돌아가서 다른 가능성을 검사하기 때문에, backtracking이라는 이름이 붙여졌습니다.

백트래킹은 DFS를 기반으로 하는 알고리즘입니다. 일반적인 DFS는 모든 경로를 탐색하지만, 백트래킹은 문제의 조건에 맞지 않으면 해당 경로를 더이상 진행하지 않고 되돌아갑니다.

관련 문제

1987번: 알파벳
https://www.acmicpc.net/problem/1987
https://www.acmicpc.net/problem/1987

너비 우선 탐색 (BFS, Breadth-First Search)



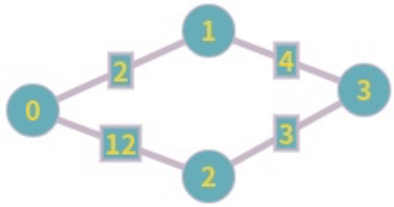
BFS (Breadth-First Search)는 그래프를 탐색하는 방법 중 하나로, 시작 정점에서 거리가 가까운 정점들을 먼저 탐색하는 방법입니다. 즉, 현재 정점과 인접한 정점을 모두 큐에 넣고, 그 다음 큐에서 꺼낸 정점과 인접한 정점들을 모두 큐에 넣는 방식으로 탐색을 진행합니다.

BFS (with Javascript)

```
class Graph {
  constructor(vertices) {
    this.vertices = vertices;
    this.edges = new Map();
    for (let i = 0; i < vertices.length; i++) {
      this.edges.set(vertices[i], []);
    }
  }

  addEdge(v, w) {
    this.edges.get(v).push(w);
    this.edges.get(w).push(v);
  }

  bfs(startingNode) {
    let visited = new Set();
    let queue = [];

    visited.add(startingNode);
    queue.push(startingNode);

    while (queue.length !== 0) {
      let currentNode = queue.shift();
      console.log(currentNode);

      let neighbours = this.edges.get(currentNode);

      for (let i = 0; i < neighbours.length; i++) {
        let neighbour = neighbours[i];
        if (!visited.has(neighbour)) {
          visited.add(neighbour);
          queue.push(neighbour);
        }
      }
    }
  }
}

let g = new Graph([1, 2, 3, 4, 5, 6, 7]);
g.addEdge(1, 2);
g.addEdge(1, 3);
g.addEdge(2, 4);
g.addEdge(2, 5);
g.addEdge(3, 6);
g.addEdge(3, 7);

g.bfs(1);
```

위 코드에서는 BFS 함수에서 Queue를 사용하여 BFS를 구현하고 있습니다. 시작 정점을 먼저 Queue에 넣고, Queue가 빌 때까지 반복문을 실행합니다. 반복문에서는 Queue에서 정점을 하나 꺼내어 출력하고, 해당 정점과 인접한 정점을 모두 Queue에 넣습니다. 이렇게 Queue에 넣은 정점은 이미 방문한 정점이므로 visited 배열을 통해 방문 여부를 체크해줍니다.

BFS vs Dijkstra



BFS로는 최단경로를 찾을 수 없는 그래프

위 그림에서 나타내는 그래프가 위에서 말한 상황을 표현합니다. 그리고 이 그래프는 BFS 로는 0번 정점에서 2번 정점으로 가는 최단경로를 찾을 수 없습니다. 0번 정점에서 2번 정점으로 가는 최단 경로는 우리 눈으로 쉽게 볼 수 있듯, 0→1→3→2 이지만, BFS가 방문하는 순서는, 0→(1,2 번 정점 방문)→3 이기 때문에, BFS로는 해결할 수 없는것이 확실합니다.

현실에서도 모든 도로의 이용시간, 이용요금이 다 같지는 않으니, 위의 BFS만 가지고는 많은 현실속 문제를 해결하긴 힘듭니다.

하지만, 이 다익스트라 알고리즘을 사용하는데 유의해야할 사항이 있습니다. BFS가 가중치가 다른 그래프에서 올바른 결과를 보증 못하듯, 그래프의 간선의 가중치에 **음수**가 있을 경우, 특히 **음수 사이클이 있을 경우** 다익스트라 알고리즘이 올바르게 동작하지 않습니다. 이러한 경우에는 밑에서 소개할 벨만-포드 알고리즘을 사용해야 합니다.

언제 무얼 사용할까?

1. 비가중치 그래프 → BFS/DFS
2. 가중치 그래프, 음의 간선 없음 → 다익스트라 (Dijkstra)
3. 가중치 그래프, 음의 간선 존재 → 벨만 포드 (Bellman Ford)

시간 복잡도

1. BFS/DFS
    - 인접 행렬 구현: O(N^2)
    - 인접 리스트 구현: O(V+E)
2. 다익스트라
    - 인접 행렬 구현: O(N^2)
    - 인접 리스트 구현: O(NlogN)
3. 벨만 포드
    - O(VE) = O(N^3)

관련 문제

문제 - 1 페이지
https://www.acmicpc.net/problemset?sort=ac_desc&algo=127
https://www.acmicpc.net/problemset?sort=ac_desc&algo=127

프로그래머스 스쿨 - 온라인 IT 특화 교육 전문 플랫폼
프로그래머스 스쿨은 IT 분야 교육 전반의 과정을 제공하는 온라인 교육 플랫폼입니다. 프로그래밍, 데이터, 인공지능 등의 학습부터 코딩 테스트, 과제 테…
https://school.programmers.co.kr/learn/courses/30/parts/12421