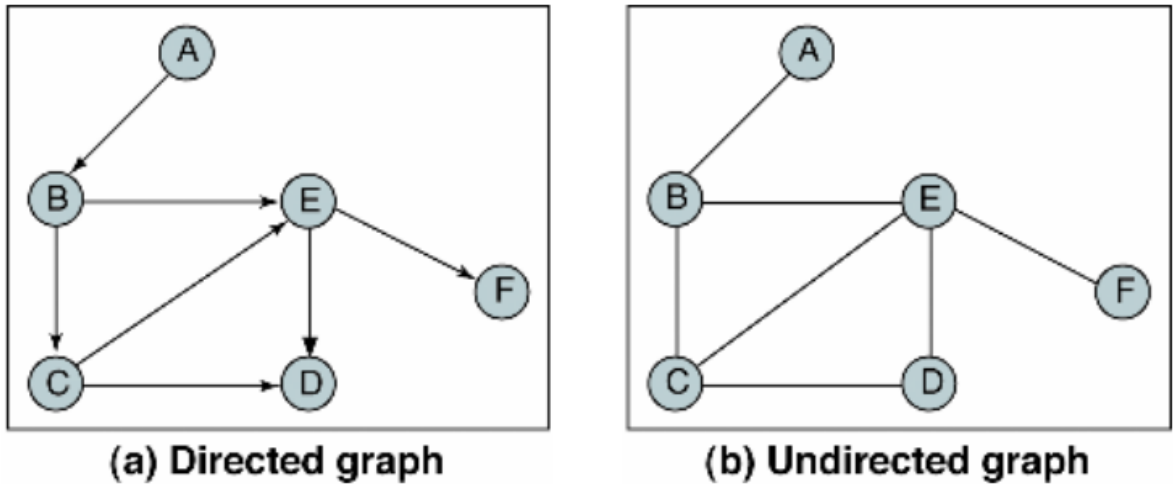
그래프

그래프(graph)

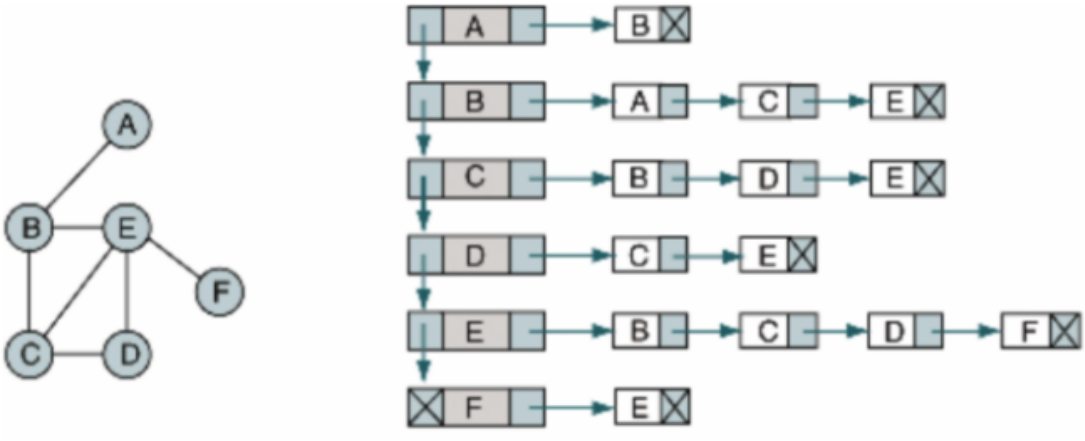
- 노드와 그 노드를 연결하는 간선을 하나로 모아놓은 비선형 자료구조
- 방향성에 따라 무방향(undirected) 그래프와 단방향(directed) 그래프로 나뉘며 간선에 가중치(weighted)를 할당하는 가중치 그래프가 있다.



- 단방향 그래프 : 간선에 방향성이 존재하는 그래프 <A,B>로 표시한다.
- 무방향 그래프 : 간선을 통해 양방향으로 이동하며, (A,B)와 같이 쌍으로 표현하며 (A,B)와 (B,A)가 동일함

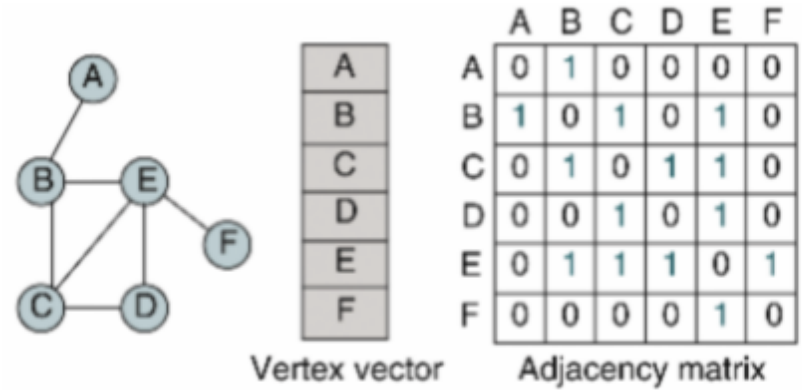
그래프의 구현

1. 인접 리스트(Adjacency list) 방식



- 그래프를 표현하는 가장 일반적인 방법
- 그래프 내에 적은 숫자의 간선만을 가지는 경우에 용이
- 각각의 정점에 인접한 정점들을 리스트로 표현한 것
- 배열이나 연결리스트를 이용하여 구현

2. 인접 행렬(Adjacency matrix) 방식



- 정점들을 2차원 배열로 구현한 것
- 그래프에 간선이 많이 존재하는 밀집 그래프의 경우에 용이
- 항상 n^2개의 메모리 공간이 필요

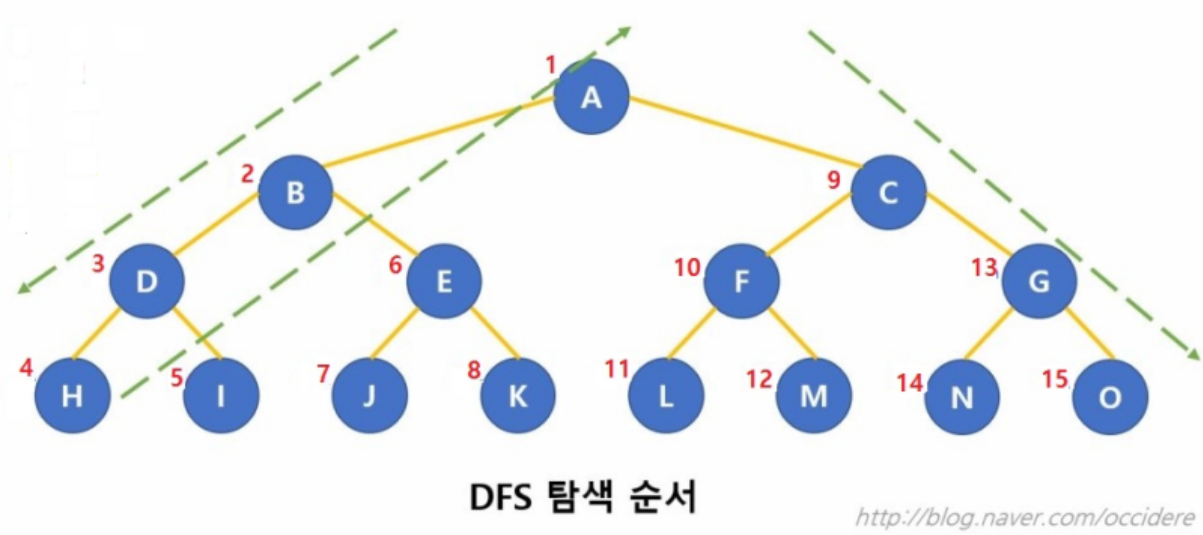
그래프 구현 코드(in javascript)

```
class Graph {
  constructor() {
    this.nodes = {};
  }

  addNode(node) {
    // 그래프에 노드 추가
    this.nodes[node] = this.nodes[node] || [];
  }

  contains(node) {
    // 그래프에 해당 노드가 존재하는지 여부를 반환
    let result = null;
    this.nodes[node] ? (result = true) : (result = false);

    return result;
  }

  removeNodes(node) {
    // 그래프에 노드 삭제
    this.nodes[node] ? delete this.nodes[node] : this.node[node];
  }

  addEdge(fromNode, toNode) {
    // 노드 사이에 간선을 추가
    this.nodes[fromNode].push(toNode);
    this.nodes[toNode].push(fromNode);
  }

  removeEdge(fromNode, toNode) {
    // 노드 사이의 간선을 삭제
    let node = this.nodes(fromNode);
    if (this.nodes[fromNode].includes(toNode) && this.nodes[toNode].includes(fromNode)) {
      this.nodes[fromNode][node.indexOf(toNode)] = '';
      this.nodes[toNode][node.indexOf(fromNode)] = '';
    }
  }
}
```

그래프의 깊이 우선 탐색

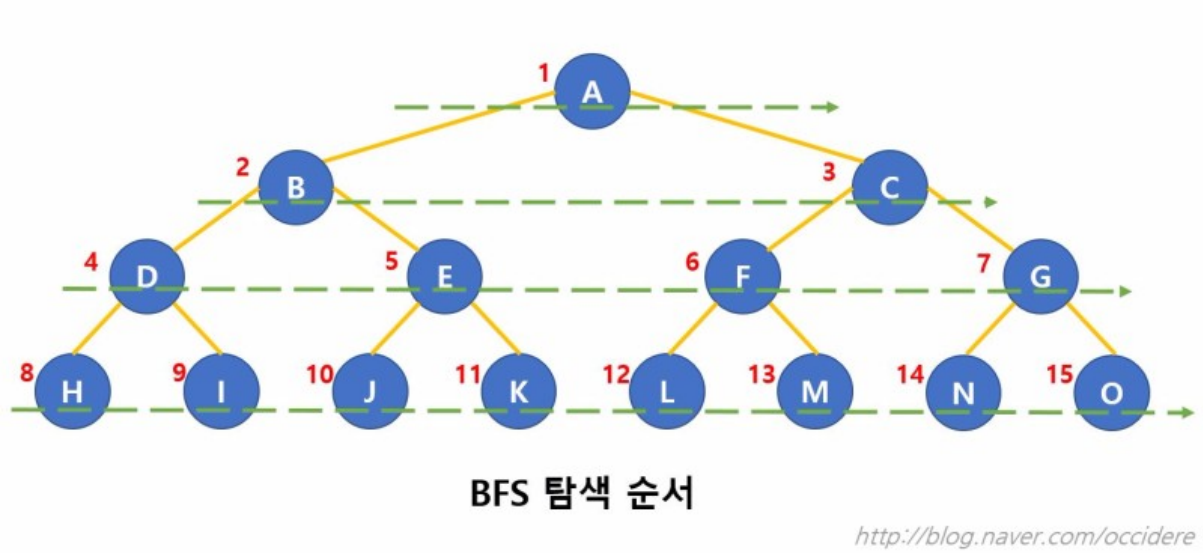
- 탐색 방식
  : 자신과 연결된 정점을 선택해, 그 정점에서 연결된

- 특징
  : 넓이가 넓은 그래프에 비해 높은 성능
- 자료구조
  : 스택(Stack)



그래프의 너비 우선 탐색

- 탐색 방식
  : 자신과 연결된 주변 정점부터 탐색해 나감
- 특징
  : 깊이가 깊은 그래프에 비해 높은 성능
- 자료구조
  : 큐(Queue)



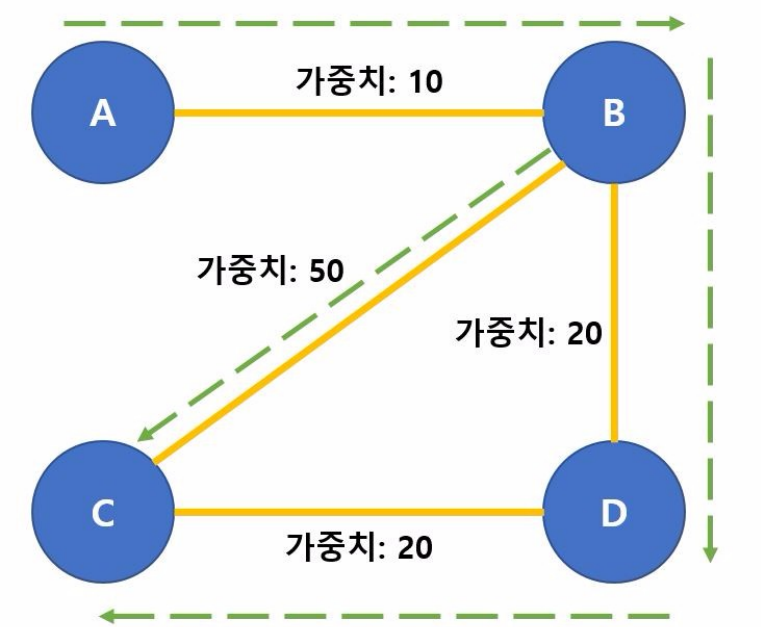
최단 경로 알고리즘

1) BFS

- 모든 weight가 1이라고 가정하면 깊이 = 최단 거리
- 가중치가 없을 때 사용

2) 다익스트라

- BFS의 응용으로 가중치가 다를 때 사용
- 어느 한 정점에서 모든 정점까지의 최단경로를 찾는데 사용
- 매 탐색마다 **해당 정점까지의 가중치의 합을 최소값으로 갱신**
- 우선순위 큐(최소 힙)를 사용하여 시간을 단축
- 음이 아닌 가중치에 대해서만 사용이 가



A에서 C로 가는데 (A->B->C), (A->B->D->C) 중
어느 길을 이용하는 것이 최단 경로일까?

http://blog.naver.com/occidere

최소 스패닝 트리(Minimum Spanning Tree / MST)

- 스패닝 트리 : 그래프 내의 모든 정점을 포함하는 트리, 그래프의 최소 연결 부분 그래프이다. ⇒ 간선의 수가 가장 작음
- MST : 스패닝 트리 중에서 사용된 간선들의 가중치 합이 최소인 트리
  - 간선의 가중치의 합이 최소여야 함
  - n개의 정점을 가지는 그래프에 대해 반드시 n-1개의 간선을 사용해야함
  - 사이클이 포함되면 안됨
- 통신망, 도로망, 유통망에서 길이, 구축 비용, 전송 시간등을 최소로 구축하려는 경우 사용

구현 방법

1. Kruskal MST 알고리즘
2. Prim MST 알고리즘

> 출처

https://velog.io/@nomadhash/Data-Structure-자바스크립트로-그래프-Graph-구현하기

https://m.blog.naver.com/occidere/220923695595