SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

알고리즘

<알림>

본 강의는 삼성 청년 SW아카데미의 컨텐츠로
보안서약서에 의거하여
강의 내용을 어떠한 사유로도 임의로 복사,
촬영, 녹음, 복제, 보관, 전송하거나
허가 받지 않은 저장매체를
이용한 보관, 제3자에게 누설, 공개,
또는 사용하는 등의 행위를 금합니다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Day3. BFS

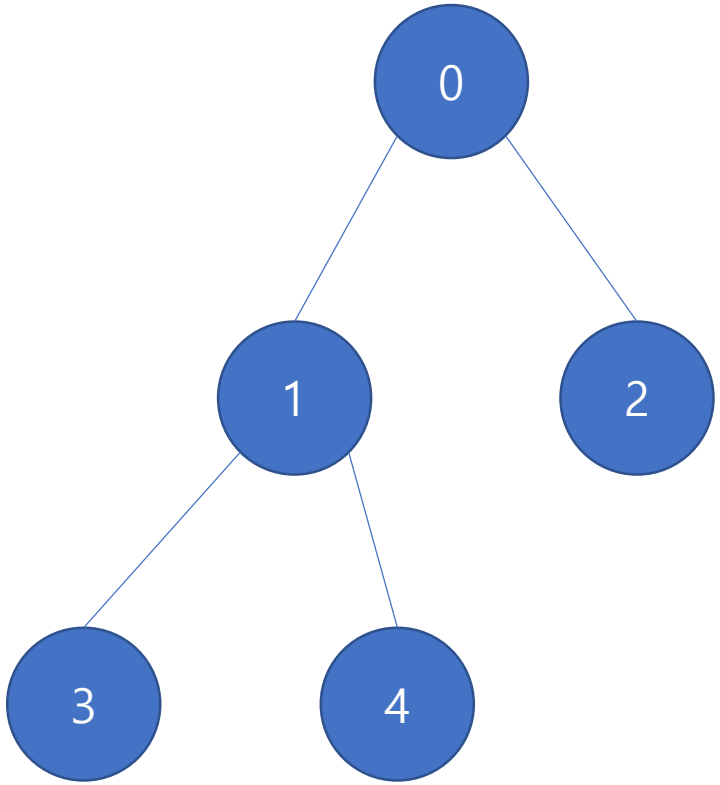
BFS 소개

Breadth First Search 너비 우선 탐색

- 0 → 1 → 2 → 3 → 4

인접한 노드를 우선으로 탐색하게 된다.

- 구현이 DFS보다 직관적

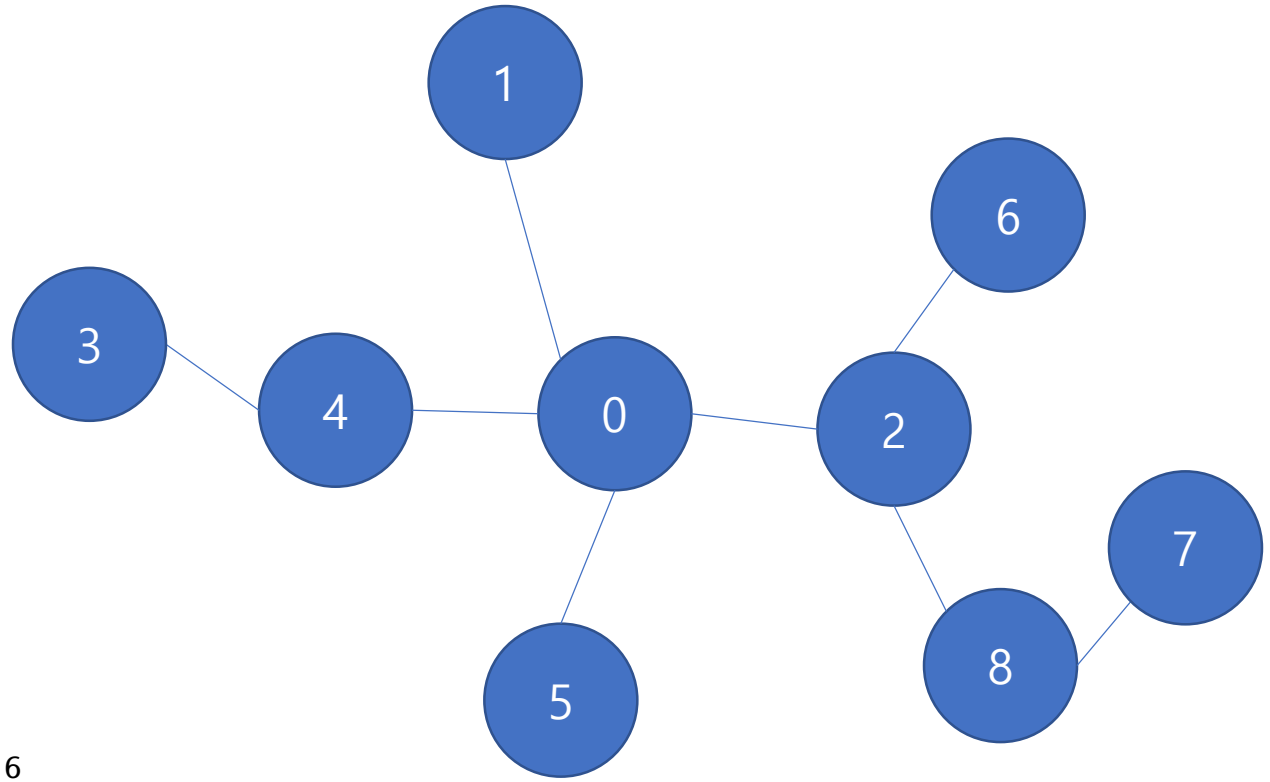


탐색 순서

- DFS : 깊이 우선 탐색 (0→1 →2 →6→8→7→4→3→5)
- BFS : 너비 우선 탐색 (0→1→2→4→5→6→8→3→7)

구현 방식

- DFS : 재귀
- BFS : Queue



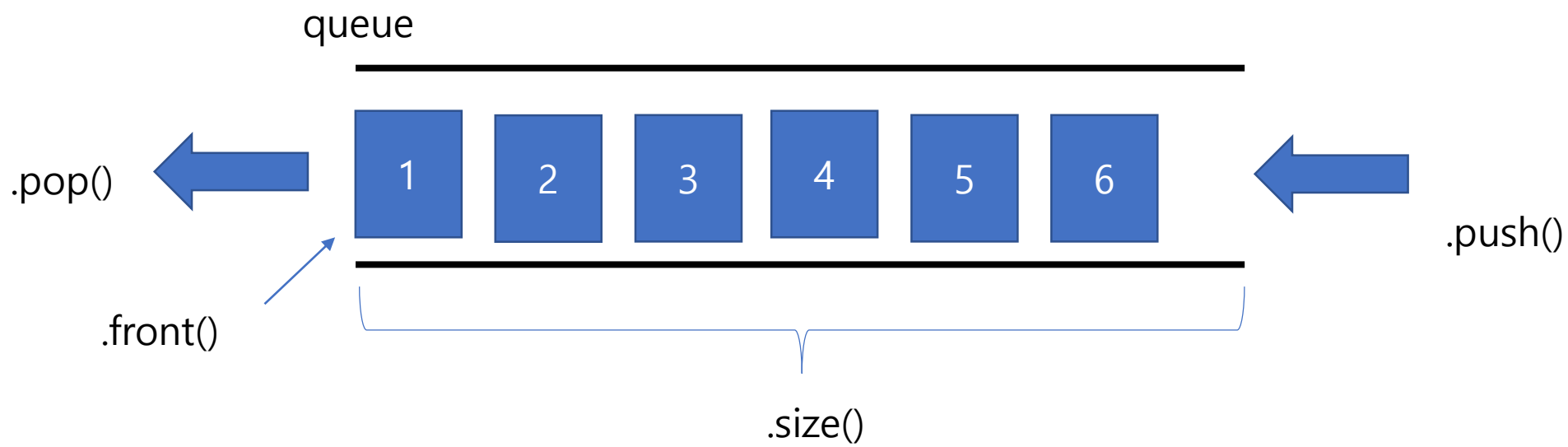
SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

queue

FIFO (First In First Out) : 먼저 들어온 순서대로 나간다.

#include<queue> 필요



queue를 사용하기 위한 API를 잘 학습하자.

- .push()
- .front()
- .empty()
- .size()
- .pop()

```
//.empty() : queue 에 데이터가 비었는 지 true/false return
while (!q.empty()) {
        //.front() : queue에 있는 가장 첫번째 데이터 가져오기
        int b = q.front();
        cout << b << " ";
        //.pop() : queue에 있는 가장 첫번째 데이터 삭제
        q.pop();
}
```

https://gist.github.com/hoconoco/72e22544596fd8aec93aa6241c53a7d3

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

BFS 기본

BFS의 구현을 위한 가장 기본적인 틀이다.

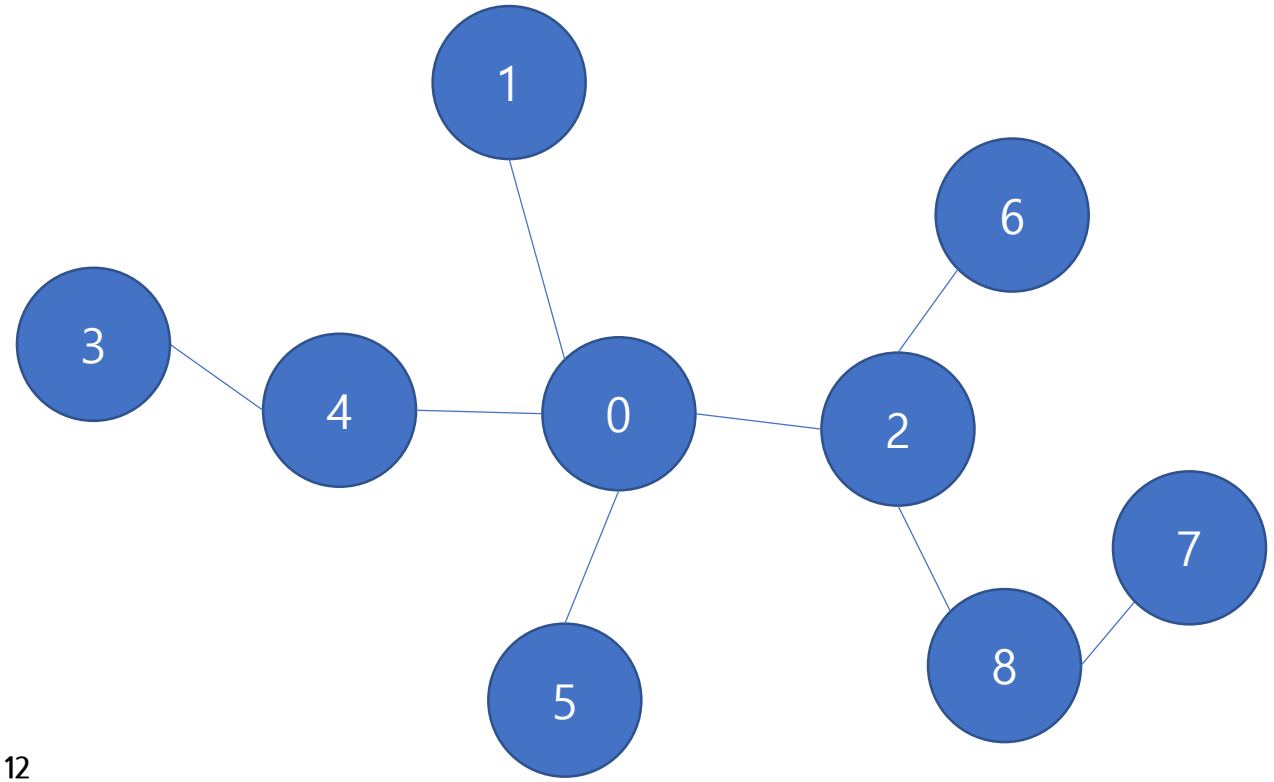
1. **대기열 준비 (Queue 준비)**
2. **시작 노드 Queue 에 삽입**
3. **큐 (대기열) 에 맨 앞에 노드부터 추출 및 확인(now)**
4. **now를 기준으로 연결되어 있는 노드들을 큐에 등록(next 후보 검색)**
5. **next 후보들을 큐에 등록**
6. **모든 노드들을 확인할 때까지 3~5 번까지 계속 반복**
   - 대기열이 비게 된다.

BFS 구현 방식

그래프를 표현하는 방식으로 구분할 수 있다.

1. 인접리스트 방식
2. 인접행렬 방식

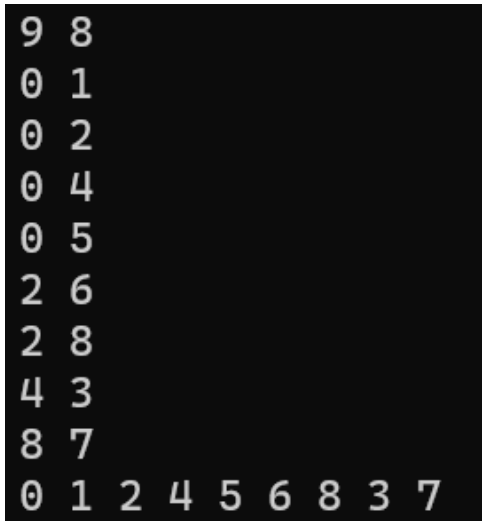
순서대로 학습한다.



방법1. 인접리스트 방식이다.

- vector 사용

구현 원리만 잘 이해하자!




```
void bfs(int start) {
        //1.대기열 생성
        queue<int> q;
        //2.시작 노드 대기열에 삽입
        q.push(start);
        //3~5 반복
        while (!q.empty()) {
                //3. 맨 앞 노드 꺼내서 확인 및 추출
                int now = q.front(); q.pop();
                cout << now << " ";
                //4. 맨 앞 노드와 인접/연결되어있는 노드(next) 탐색
                for (int i = 0; i < arr[now].size(); i++) {
                        //5. next 노드를 대기열에 삽입
                        int next = arr[now][i];
                        q.push(next);
                }
        }
}
```

https://gist.github.com/hoconoco/bc4afd475e9ed6cea83afe792c6a651e

방법2. 인접행렬 방식이다.

- **배열 사용**

구현 원리만 잘 이해하자!

```
void bfs(int start) {
        //1.대기열 생성
        queue<int> q;
        //2.시작 노드 대기열에 삽입
        q.push(start);

        //3~5 반복
        while (!q.empty()) {
                //3. 맨 앞 노드 꺼내서 확인 및 추출
                int now = q.front(); q.pop();

                cout << now << " ";

                //4. 맨 앞 노드와 인접/연결되어있는 노드(next) 탐색
                for (int next = 0; next < N; next++) {
                        //0이면 건너뛰기
                        if (arr[now][next] == 0) continue;
                        //5. next 노드를 대기열에 삽입
                        q.push(next);
                }
        }
}
```

https://gist.github.com/hoconoco/9886910e24020b1f753a59a703c9bc5c

현재 코드는 단방향 그래프일 때 구현이다.

양방향으로 구현하자.

- **입력할 때, to→from 도 연결하면 된다.**

- **실행한다.**
  - 어마어마하다.
  - 제대로 된 탐색이 되지 않는다.
  - why? → 중복되기 때문
  - DFS에서 확인한 바와 같다.
  - 똑같이 visited 로 방문 처리한다.

```
for (int i = 0; i < M; i++) {
    int from, to;
    cin >> from >> to;
    //양방향 연결
    arr[from][to] = 1;
    arr[to][from] = 1;
}

//시작노드 0부터 시작
bfs(0);
```

방법1 인접리스트 + visited

- **중복 방지**

```
void bfs(int start) {
    //1.대기열 생성
    queue<int> q;
    //2.시작 노드 대기열에 삽입
    q.push(start);
    //방문 체크!
    visited[start] = 1;

    //3~5 반복
    while (!q.empty()) {
        //3. 맨 앞 노드 꺼내서 확인 및 추출
        int now = q.front(); q.pop();
        cout << now << " ";
        //4. 맨 앞 노드와 인접/연결되어있는 노드(next) 탐색
        for (int i = 0; i < map[now].size(); i++) {
            //5. next 노드를 대기열에 삽입
            int next = map[now][i];
            //방문했다면 건너뛰기
            if (visited[next] == 1) continue;
            //노드 방문 체크
            visited[next] = 1;
            q.push(next);
        }
    }
}
```

https://gist.github.com/hoconoco/5568177b5f10d3379d857497e2ae1900

방법2 인접행렬 + visited

- **중복 방지**

```
void bfs(int start) {
    //1.대기열 생성
    queue<int> q;
    //2.시작 노드 대기열에 삽입
    q.push(start);
    //방문 체크!
    visited[start] = 1;
    //3~5 반복
    while (!q.empty()) {
        //3. 맨 앞 노드 꺼내서 확인 및 추출
        int now = q.front(); q.pop();
        cout << now << " ";

        //4. 맨 앞 노드와 인접/연결되어있는 노드(next) 탐색
        for (int next = 0; next <= N; next++) {
            //연결 안되면 건너뛰기
            if (map[now][next] == 0 ) continue;
            //방문했다면 건너뛰기
            if (visited[next] == 1) continue;
            //next 노드 방문 체크!
            visited[next] = 1;
            //5. next 노드를 대기열에 삽입
            q.push(next);
        }
    }
}
```

https://gist.github.com/hoconoco/d2e2cf5171b277dba8746e56ead85d55

BFS 응용

BFS 문제 유형

BFS는 그래프를 완전 탐색하는 알고리즘이다.

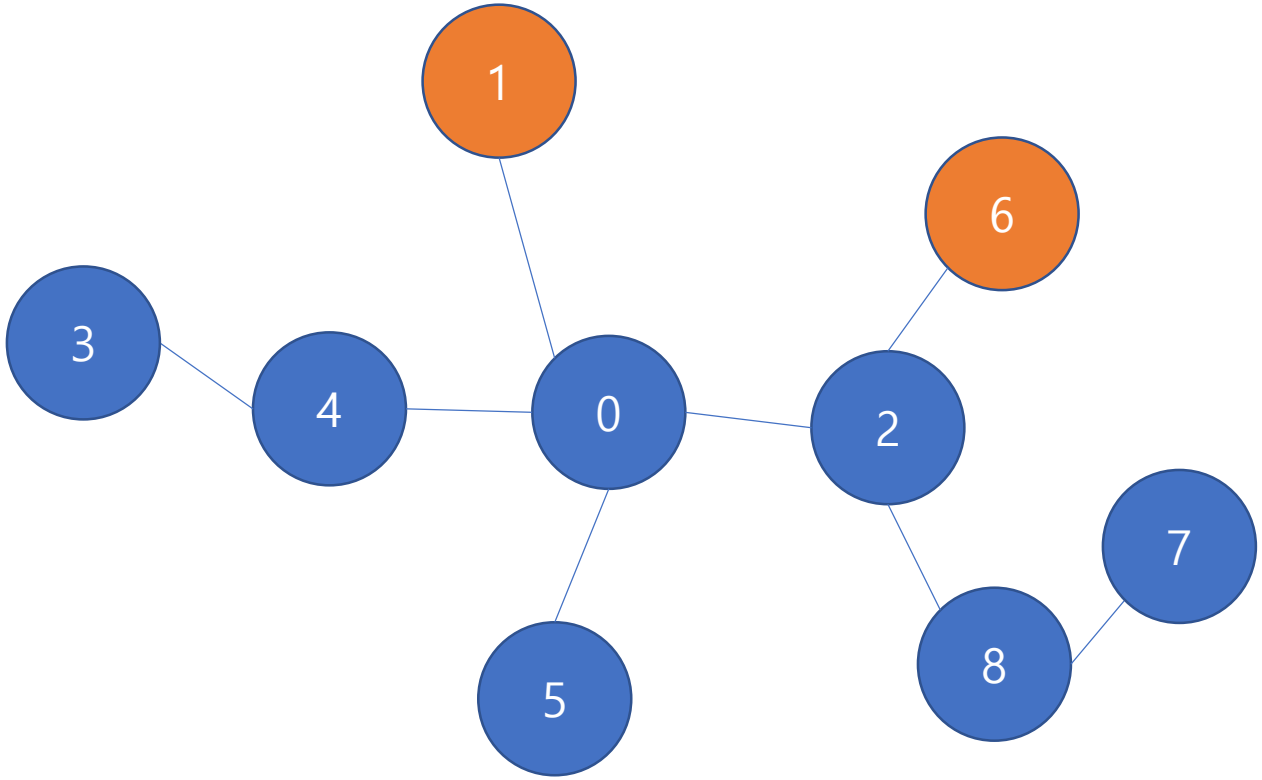
다양한 문제 유형 풀이에 사용된다.

- 최단 거리 문제
- 미로 탈출
- 최소 이동 횟수
- 연결 요소 개수
- 군집의 크기
- 최소 연산 횟수로 숫자 변경
- 위상 정렬
- …

대표적인 최단 경로 문제 유형이 있다.

최단 경로를 찾는 게 목적

- 1번 노드부터 6번 노드까지의 거리 : 3
- 0번 노드부터 번 노드까지의 거리 : 2

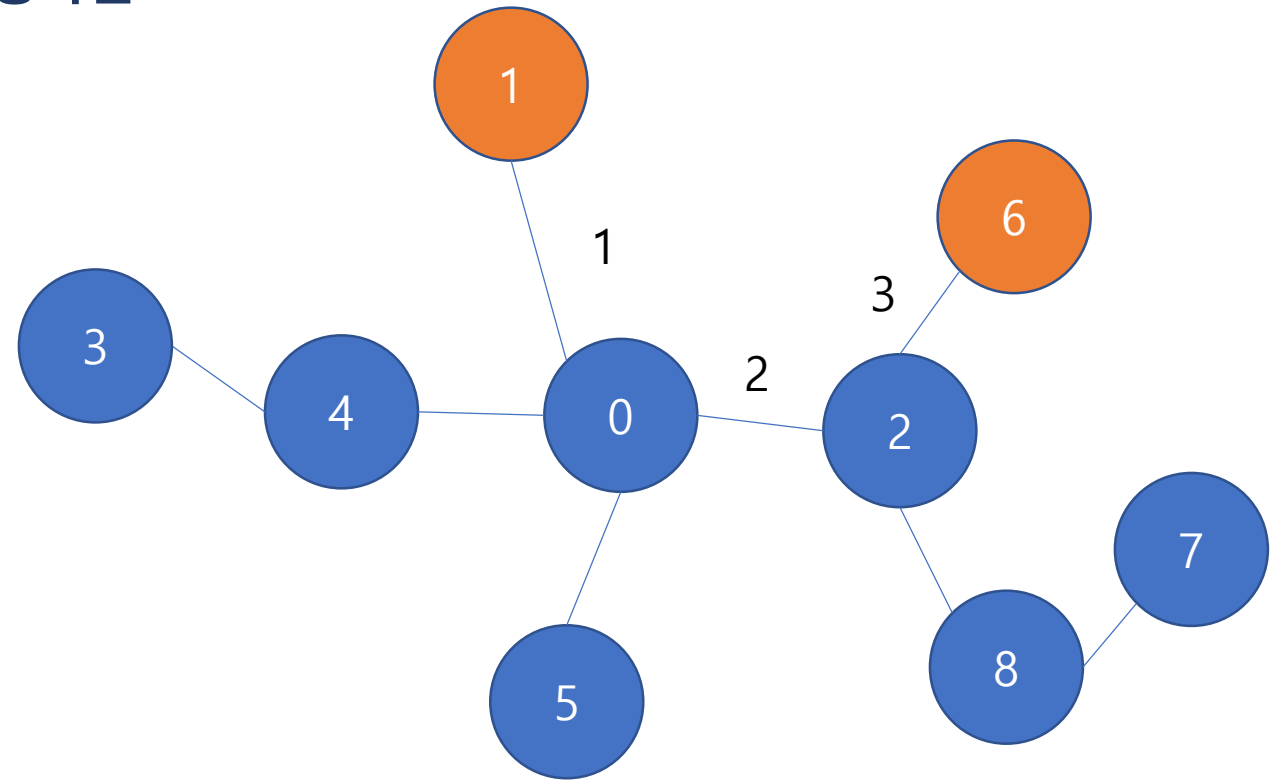


경로 찾는 방식

현재 그래프는 모든 노드가 연결된 상황이다.

시작 노드 → 도착 노드까지 경로를 찾는 방법은?!

- **visited[] 방문 기록용 노드에 1씩 더하면서 이동하면?**
- **몇 번만에 도착하는 지 파악할 수 있다!**



방법1. 인접리스트 + visited(경로 기록)

```
0 1 2 4 5 6 8 3 7
노드 번호 : 0, visited : 1
노드 번호 : 1, visited : 2
노드 번호 : 2, visited : 2
노드 번호 : 3, visited : 3
노드 번호 : 4, visited : 2
노드 번호 : 5, visited : 2
노드 번호 : 6, visited : 3
노드 번호 : 7, visited : 4
노드 번호 : 8, visited : 3
```

```
//4. 맨 앞 노드와 인접/연결되어있는 노드(next) 탐색
for (int i = 0; i < arr[now].size(); i++) {
        int next = arr[now][i];
        //visited[next]는 이제 0보다 큰 값이 담기게 된다.
        if (visited[next] > 0) continue;
        //next 노드를 방문하면서, now+1 의 값을 기록
        visited[next] = visited[now] + 1;
        //5. next 노드를 대기열에 삽입
        q.push(next);
}
```

https://gist.github.com/hoconoco/afde7c21a05f31e909842f60c998c1e7

방법2. 인접행렬 + visited(경로 기록)

```
0 1 2 4 5 6 8 3 7
노드 번호 : 0, visited : 1
노드 번호 : 1, visited : 2
노드 번호 : 2, visited : 2
노드 번호 : 3, visited : 3
노드 번호 : 4, visited : 2
노드 번호 : 5, visited : 2
노드 번호 : 6, visited : 3
노드 번호 : 7, visited : 4
노드 번호 : 8, visited : 3
```

```
//4. 맨 앞 노드와 인접/연결되어있는 노드(next) 탐색
for (int next = 0; next < N; next++) {
        //연결 안되면 건너뛰기
        if (arr[now][next] == 0) continue;
        //visited[next]는 이제 0보다 큰 값이 담기게 된다.
        if (visited[next] > 0) continue;
        //next 노드를 방문하면서, now+1 의 값을 기록
        visited[next] = visited[now] + 1;
        //5. next 노드를 대기열에 삽입
        q.push(next);
}
```

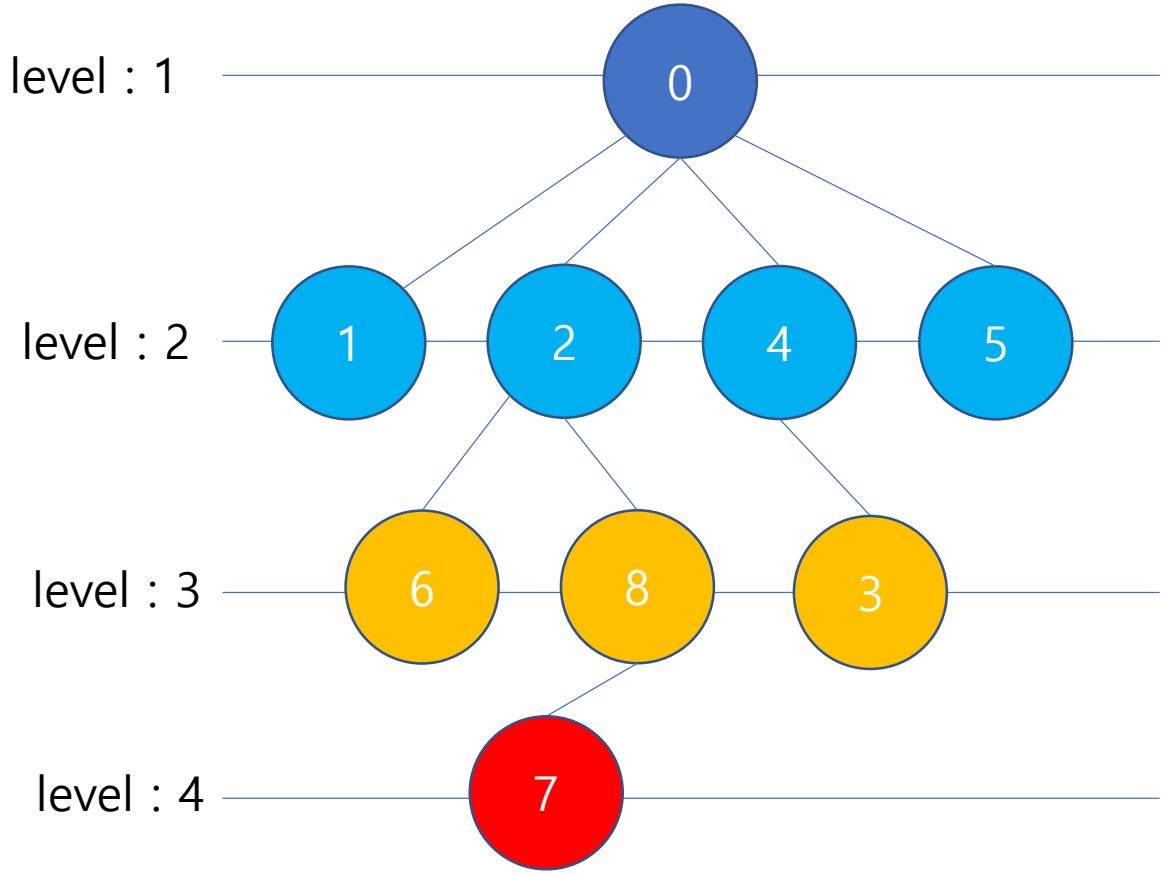
https://gist.github.com/hoconoco/95567e69341cd660de021ceee5f00409

[참고] visited와 level

visited의 값은 그래프를 트리 구조로 보면, level과 같다.

- **level : 같은 깊이를 가진 노드들의 집합**
- **0번 노드를 기준으로 트리 구조로 보면, level이라 볼 수 있다.**

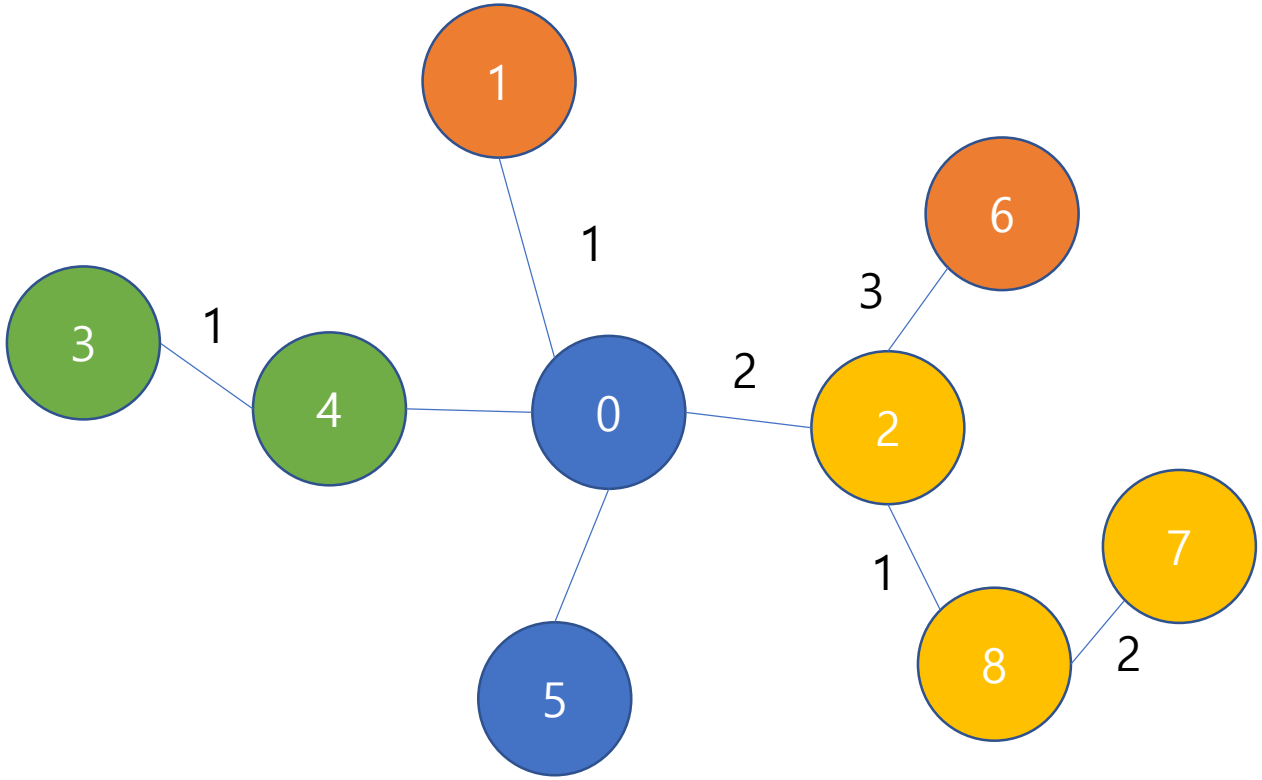
```
0 1 2 4 5 6 8 3 7
노드 번호 : 0, visited : 1
노드 번호 : 1, visited : 2
노드 번호 : 2, visited : 2
노드 번호 : 3, visited : 3
노드 번호 : 4, visited : 2
노드 번호 : 5, visited : 2
노드 번호 : 6, visited : 3
노드 번호 : 7, visited : 4
노드 번호 : 8, visited : 3
```



최단 경로 구하기

이제 visited[]를 이용해서 노드와 노드 사이의 최단경로를 구하자

- 1 → 6 : 3 칸
- 2 → 7 : 2 칸
- 3 → 4 : 1 칸



코드를 작성하고 빌드한다.

방법1. 인접리스트로 구현한 코드

```
3
1 6
거리 : 3
2 7
거리 : 2
3 4
거리 : 1
```

```
//q번 입력받기
int q;
cin >> q;
for (int i = 0; i < q; i++) {
    //출발, 도착 노드 입력
    int from, to;
    cin >> from >> to;
    //visited 초기화
    memset(visited, 0, sizeof(visited));
    //from -> to 까지 bfs 돌리기
    bfs(from, to);
}
```

```
//now 가 현재 도착한 지점 == 목표로 했던 end 인지 체크
if (now == end) {
    //from -> to 까지의 거리를 출력한다.
    cout << "거리 : " << visited[now] << '\n';
    return;
}
```

https://gist.github.com/hoconoco/618ba9a74af607928498970425b7d1e2

방법2. 인접행렬로 구현한 코드

```
//q번 입력받기
int q;
cin >> q;
for (int i = 0; i < q; i++) {
    //출발, 도착 노드 입력
    int from, to;
    cin >> from >> to;
    //visited 초기화
    memset(visited, 0, sizeof(visited));
    //from -> to 까지 bfs 돌리기
    bfs(from, to);
}
```

```
3
1 6
거리 : 3
2 7
거리 : 2
3 4
거리 : 1
```

```
//now 가 현재 도착한 지점 == 목표로 했던 end 인지 체크
if (now == end) {
    //from -> to 까지의 거리를 출력한다.
    cout << "거리 : " << visited[now] << '\n';
    return;
}
```

https://gist.github.com/hoconoco/0187b38f171f6f85fb620b543905645e

[참고] BFS / DFS 공통점 차이점

공통점

- **그래프 탐색, 특정 노드 기준 → 모든 노드를 탐색하는 완전탐색 방법**

차이점

- **DFS**
  - 재귀, 깊이 우선, 한 방향으로 갈 수 있는 데까지 가보고 다른 경로 또 탐색, 다양한 경로 ex) 가장 긴 경로
  - 효율을 높이기 위한 백트래킹/기저 조건 설정 필수
  - N = 20~30, 시간 초과 나지 않게 백트래킹 필수
- **BFS**
  - 큐를 사용, now 기준 발견되는 순서대로 간다.
  - 최대한 적은 node의 개수를 들려서 가는 방법(경로), 최단 경로
  - 해당 경로 외에 다른 경로를 알지 못한다.

A형에 가장 많은 출제 (80%이상)
- DFS+시뮬레이션 (특정 노드 기준 다양한 경로)
- BFS+시뮬레이션 (특정 노드 기준 최단경로)
+ 다익스트라 (가중치에 따른 경로)

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

내일 방송에서 만나요!

삼성 청년 SW 아카데미