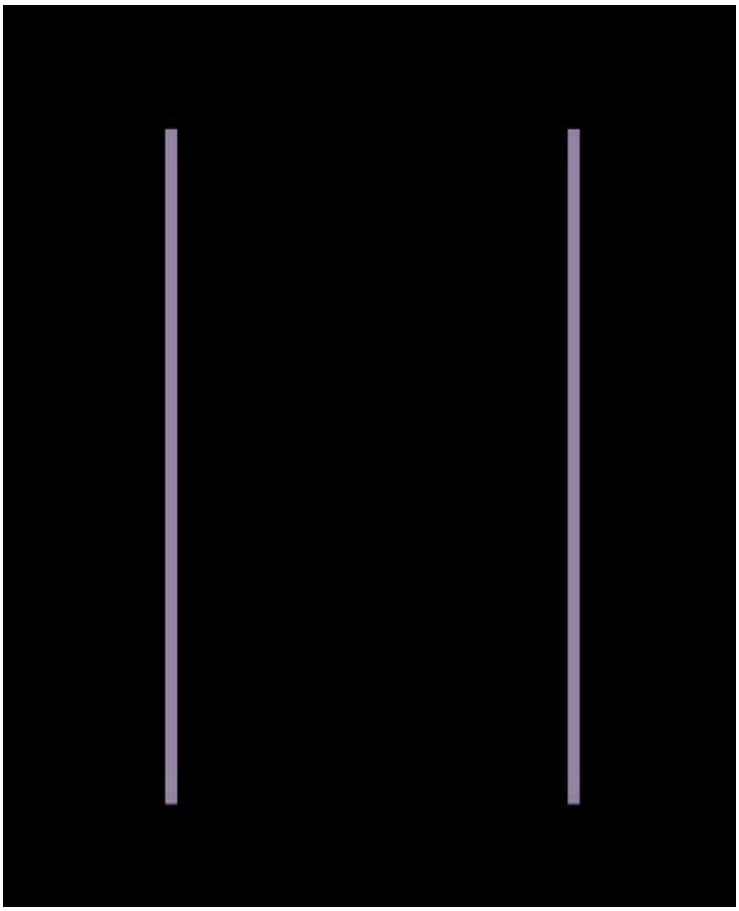
🔷

BFS

1. 큐 자료구조 (복습)

- 먼저 들어온 데이터가 먼저 나가는 형식(선입선출)의 자료구조



출처: https://ai-rtistic.com/2022/01/22/data-structure-queue/

- 파이썬에서 큐 사용하는 법

```
from collections import deque

queue = deque()

queue.append(1)
queue.append(2)
queue.append(3)
queue.append(4)
queue.append(5)
queue.popleft()
queue.popleft()
queue.popleft()

print(queue)
# deque([4, 5])
```

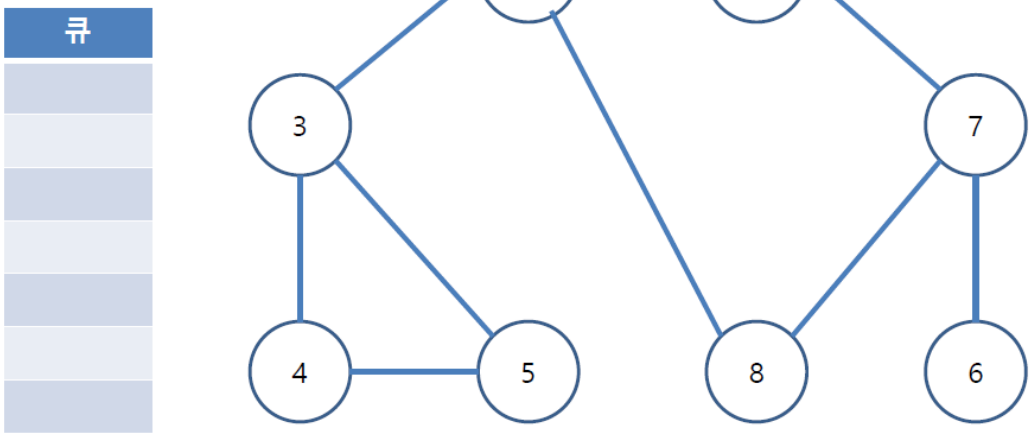
- 리스트를 사용하지 않고 데크를 사용하는 이유 :
  - 리스트 자료형으로 큐를 구현할 수 있지만 시간복잡도에서 데크가 더 빠르다!
    - append( ) → O(1)
    - pop( ) → O(1)
    - pop(0) → O(N)
    - popleft( ) → O(1)

2. BFS (Breadth-First Search)

- **BFS**는 **너비 우선 탐색**이라고도 부르며, 그래프에서 가까운 노드부터 우선적으로 탐색하는 알고리즘이다.
- BFS는 큐 자료구조를 이용하며, 구체적인 동작 과정은 다음과 같다.
  1. 탐색 시작 노드를 큐에 삽입하고 방문 처리를 한다.
  2. 큐에서 노드를 꺼낸 뒤에 해당 노드의 인접 노드 중에서 방문하지 않은 노드를 모두 큐에 삽입하고 방문 처리한다.
  3. 더 이상 2번의 과정을 수행할 수 없을 때까지 반복한다.

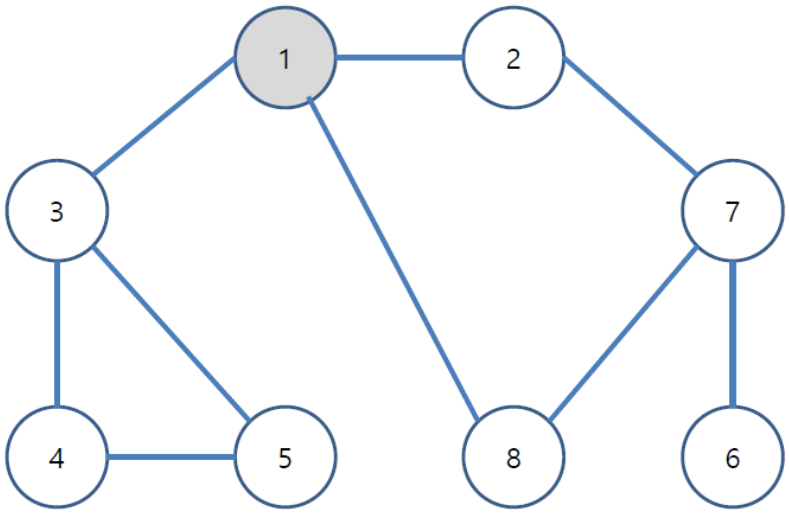
▼ 그림으로 이해하기

- [step 0] 그래프를 준비한다. (방문기준 : 번호가 낮은 인접 노드부터)
  - 시작노드: 1



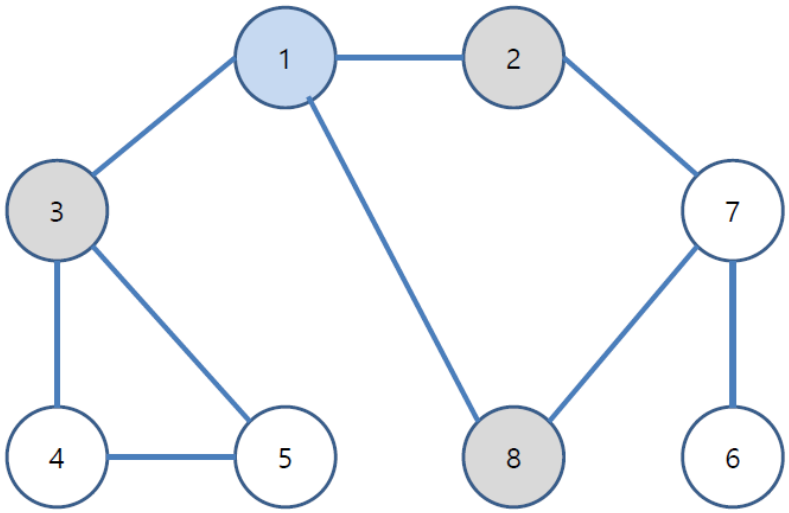
- [step 1] 시작 노드인 ‘1’을 큐에 삽입하고 방문 처리한다.

큐
1



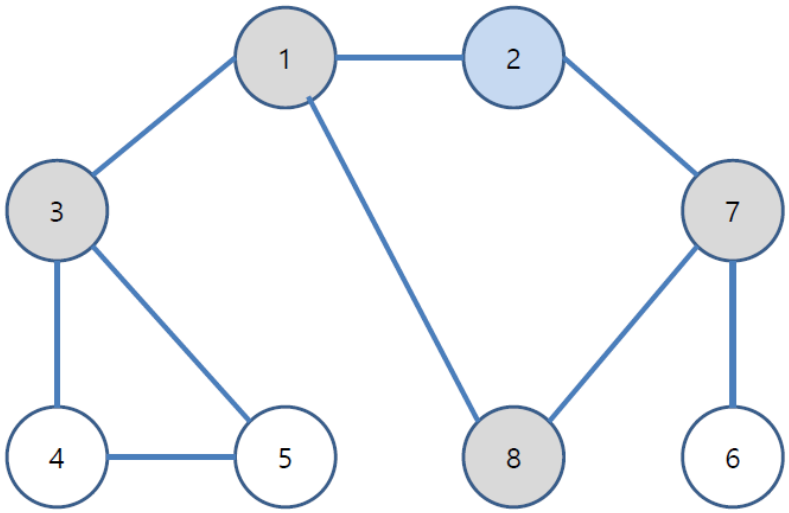
- [step 2] 큐에서 노드 ‘1’을 꺼내 방문하지 않은 인접 노드 ‘2’, ‘3’, ‘8’을 큐에 삽입하고 방문 처리한다.

큐
8
3
2



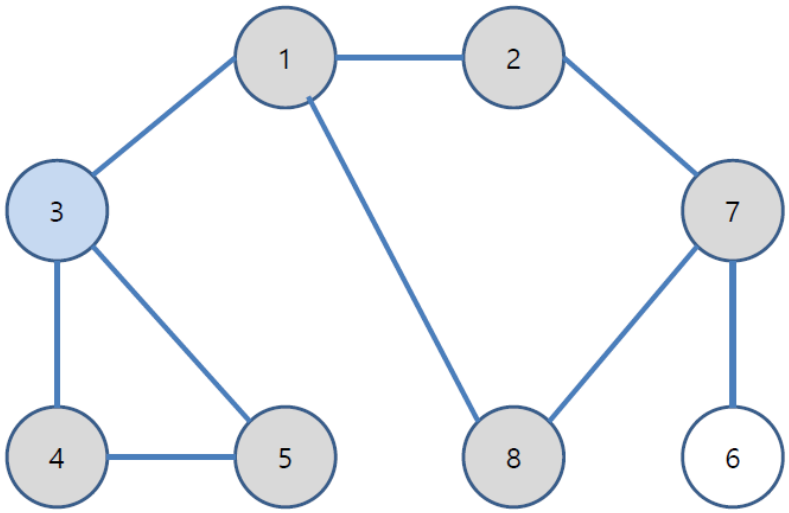
- [step 3] 큐에서 노드 ‘2’를 꺼내 방문하지 않은 인접 노드 ‘7’을 큐에 삽입하고 방문 처리한다.

큐
7
8
3



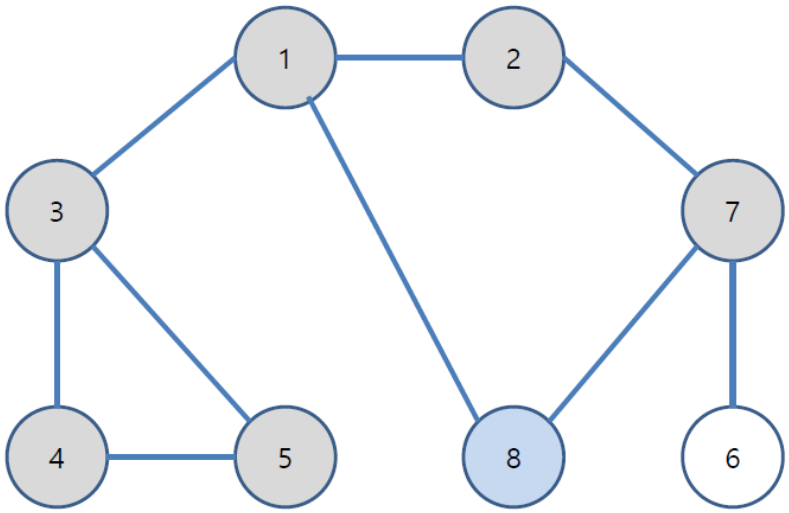
- [step 4] 큐에서 노드 ‘3’을 꺼내 방문하지 않은 인접 노드 ‘4’, ‘5’를 큐에 삽입하고 방문 처리한다.

큐
5
4
7
8

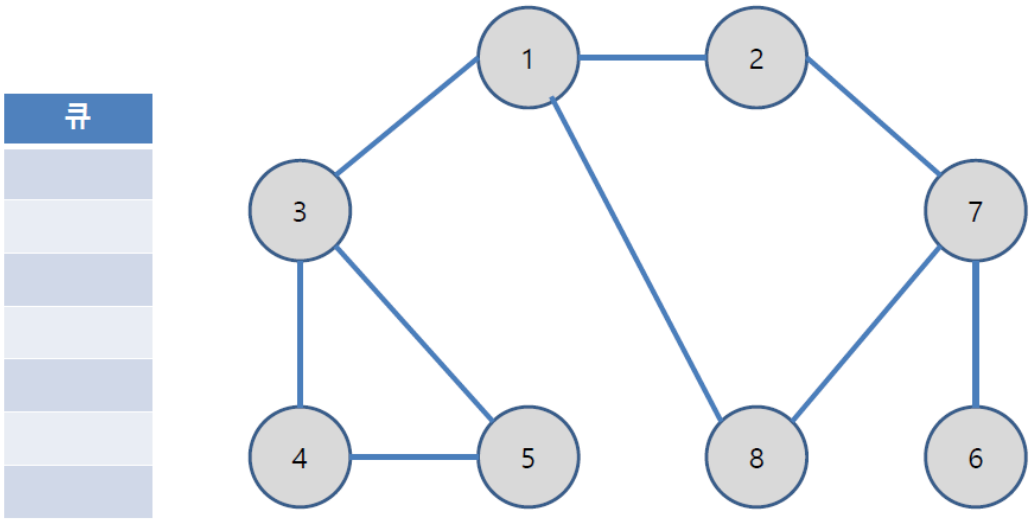


- [step 5] 큐에서 노드 ‘8’을 꺼내고 방문하지 않은 인접 노드가 없으므로 무시한다.

큐
5
4
7



- 이러한 과정을 반복하여 전체 노드의 탐색 순서(큐에 들어간 순서)는 다음과 같다.



탐색 순서 : 1 → 2 → 3 → 8 → 7 → 4 → 5 → 6

2, 3, 8은 시작노드에서부터 거리가 1이고, 7, 4, 5는 시작노드에서부터 거리가 2이다. 가장 거리가 먼 6은 마지막에 탐색된다.

- BFS는 너비 우선 탐색이라는 이름처럼 시작노드에서부터 **가까운 노드를 우선적으로** 탐색한다. BFS 알고리즘은 주로 그래프에서 모든 간선의 비용이 동일한 조건에서 최단 거리를 구하는 문제를 효과적으로 해결할 수 있는 알고리즘이다.

```
from collections import deque

# BFS 함수 정의
def bfs(graph, start, visited):
    # 큐(Queue) 구현을 위해 deque 라이브러리 사용
    queue = deque([start])
    # 현재 노드를 방문 처리
    visited[start] = True
    # 큐가 빌 때까지 반복
    while queue:
        # 큐에서 하나의 원소를 뽑아 출력
```

```
        v = queue.popleft()
        print(v, end=' ')
        # 해당 원소와 연결된, 아직 방문하지 않은 원소들을 큐에 삽입
        for i in graph[v]:
            if not visited[i]:
                queue.append(i)
                visited[i] = True

# 각 노드가 연결된 정보를 리스트 자료형으로 표현(2차원 리스트)
graph = [
  [],
  [2, 3, 8],
  [1, 7],
  [1, 4, 5],
  [3, 5],
  [3, 4],
  [7],
  [2, 6, 8],
  [1, 7]
]

# 각 노드가 방문된 정보를 리스트 자료형으로 표현(1차원 리스트)
visited = [False] * 9

# 정의된 BFS 함수 호출
bfs(graph, 1, visited)
```

3. BFS 관련 문제 (미로탈출)

- 문제 상황 요약:
    - N x M 크기의 직사각형 미로를 탈출해야 된다. 시작 위치는 (1,1)이며 미로의 출구는 (N,M)의 위치에 존재하며 한 번에 한 칸씩 이동할 수 있다. 이때, 1은 이동할 수 있는 칸을 나타내고, 0은 이동할 수 없는 칸을 나타낸다. 미로를 탈출하기 위해 움직여야 하는 최소 칸의 개수를 구하시오. 칸을 셀 때는 시작 칸과 마지막 칸을 모두 포함해서 계산한다.
- 입력예시:

5 6

101010

111111

000001

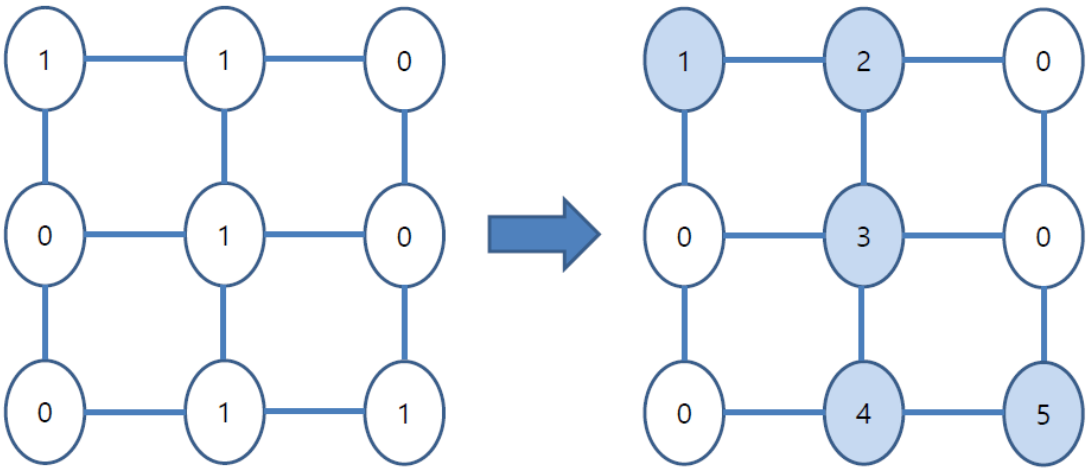
111111

111111

- 출력예시:

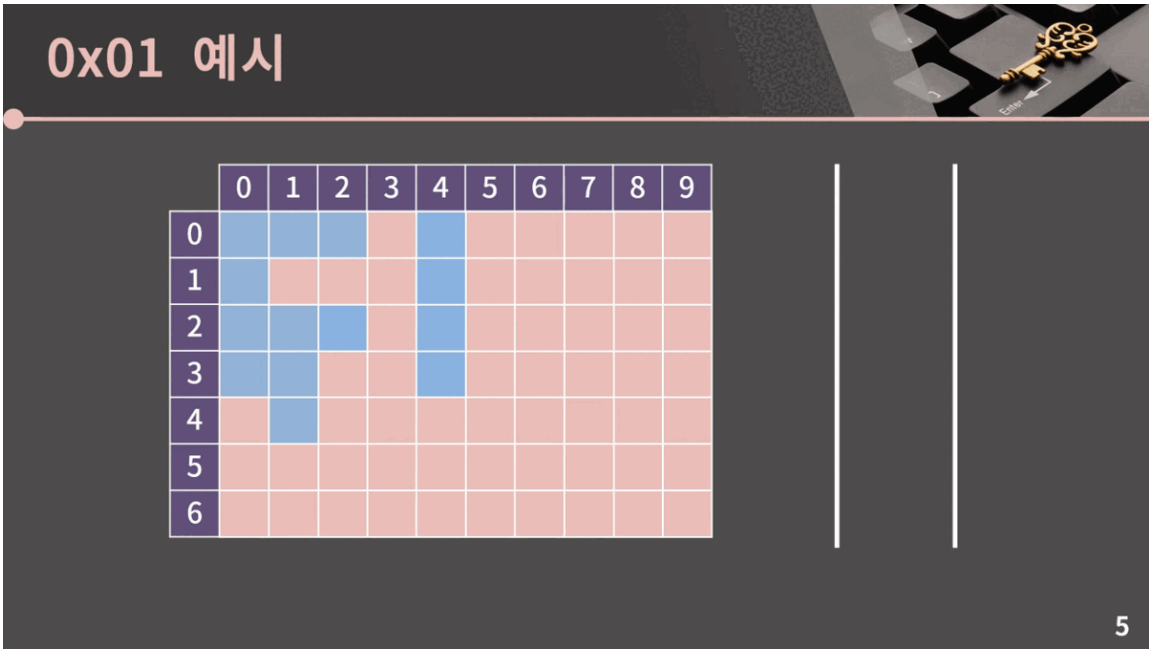
10

- 문제 해결 아이디어
  - BFS는 시작 지점에서 가까운 노드부터 차례대로 그래프의 모든 노드를 탐색한다.
  - 상, 하, 좌, 우로 연결된 모든 노드로의 거리가 1로 동일하다.
    - 따라서 (1,1)지점부터 BFS를 수행하여 모든 노드의 최단거리 값을 기록하면 해결할 수 있다.



  - (1,1) 좌표에서 상, 하, 좌, 우로 탐색을 진행하면 바로 옆 노드인 (1,2) 위치의 노드를 방문하게 되고 새롭게 방문하는 (1,2) 노드의 값을 2로 바꾼다.
  - BFS를 계속 수행하면 위의 그림처럼 최단 경로의 값들이 1씩 증가하는 형태로 변경된다.

▼ 미로탈출 아이디어 (그림으로 이해하기)



▼ 문제 해결 코드

```
from collections import deque

# N, M을 공백을 기준으로 구분하여 입력 받기
n, m = map(int, input().split())
# 2차원 리스트의 맵 정보 입력 받기
graph = []
for i in range(n):
    graph.append(list(map(int, input())))

# 이동할 네 가지 방향 정의 (상, 하, 좌, 우)
dx = [-1, 1, 0, 0]
dy = [0, 0, -1, 1]

# BFS 소스코드 구현
def bfs(x, y):
    # 큐(Queue) 구현을 위해 deque 라이브러리 사용
    queue = deque()
    queue.append((x, y))
    # 큐가 빌 때까지 반복하기
    while queue:
        x, y = queue.popleft()
        # 현재 위치에서 4가지 방향으로의 위치 확인
        for i in range(4):
            nx = x + dx[i]
            ny = y + dy[i]
            # 미로 찾기 공간을 벗어난 경우 무시
            if nx < 0 or nx >= n or ny < 0 or ny >= m:
                continue
            # 이동할 수 없는 경우 무시
            if graph[nx][ny] == 0:
```

```
                continue
            # 해당 노드를 처음 방문하는 경우에만 최단 거리 기록
            if graph[nx][ny] == 1:
                graph[nx][ny] = graph[x][y] + 1
                queue.append((nx, ny))
    # 가장 오른쪽 아래까지의 최단 거리 반환
    return graph[n - 1][m - 1]

# BFS를 수행한 결과 출력
print(bfs(0, 0))
```

기본 문제(추천) : 백준 2178 미로탐색

공통 문제 : 백준 2206 벽 부수고 이동하기

개인 문제 : BFS 문제 모음