

Algorithm Study

12.16.2024 김홍주

#2206 벽 부수고 이동하기/Gold3

Q. 벽 부수고 이동하기

$N \times M$ 행렬로 표현되는 맵이 있다. 이 맵에선 0은 이동할 수 있는 곳, 1은 벽이 있는 곳을 나타낸다.

당신은 (1,1) 에서 (N,M) 위치 까지 최단 경로로 이동하려고 한다. 이때 최단 경로는 시작하는 칸과 끝나는 칸도 포함해서 센다

또한 이동하는 도중에 한 개의 벽을 부수고 이동하는 것이 좀 더 경로가 짧아진다면, 벽을 한 개 까지 부수고 이동하여도 된다

<입력 조건 >

- 첫째 줄에 $N(1 \leq N \leq 1,000)$, $M(1 \leq M \leq 1,000)$ 이 주어진다.
- 다음 N 개의 줄에 M 개의 숫자로 맵이 주어진다.
- (1, 1)과 (N, M)은 항상 0이라고 가정하자.

<출력 조건>

- 첫째 줄에 최단 거리를 출력한다. 불가능할 때는 -1을 출력한다.

[예제]

6	4
0	100
1	110
1	000
0	000
0	111
0	000

Hint

1. BFS

- 최단 경로 탐색문제로 그래프 탐색 알고리즘 중 BFS 사용
 - BFS : 시작점 부터 가까운 노드 부터 점차 탐색하기 때문에 최단 거리를 보장함
-> (N,M)에 처음 도착한 경우, 최단 거리임.
 - DFS : 모든 경로 탐색해야 함으로 비효율적임

2. 3차원으로 벽 부순 횟수 기록

추가 조건으로 칸이 1인 벽을 딱 한번 부술 수 있음

다음 칸(graph[xx][yy])으로 이동 가능한 경우

- (1) 다음 칸 graph[xx][yy] = 0 이고 다음 칸의 방문 여부 = 0 일때
- (2) 다음 칸 이 벽 graph[xx][yy] = 1 이고 현재 벽을 부순 적 없을 때 (새로운 칸을 개척한것임으로 방문 여부 = 0 은 확정)

벽을 부술 수 있는 기회가 주어졌기 때문에 queue와 방문 표기 배열에 벽 부순 횟수도 같이 기록해야 한다.

=> visited [x][y][wall] : 방문 표기 배열을 3차원으로 만든다.

벽을 뚫지 않고 온 경우, 벽을 뚫어서 온 경우를 따로 구분함

visited[x][y][0] : 벽을 뚫지 않고 온 최단 경로 거리

visited[x][y][1] : 벽을 1회 뚫고 온 최단 경로 거리

Hint3 - Flow

1. `visited[x][y][wall]` = 최단 거리

: 벽 뚫은 여부, 방문 여부, 시작점(일단 0,0) ~ 목적지(N,M) 까지 최단 거리

2. BFS를 위한 queue 초기화

- 시작점 (0,0) 에서 출발하며 , 벽 부수지 않은 상태=0 로 큐에 추가
- `visited[0][0][0] = 1` 초기화(출발칸도 경로에 포함됨)

3. BFS 탐색 과정

- queue에서 현재 위치와 벽 부순 상태를 꺼내 상하좌우로 이동 시도
- 다음 이동할 칸이 맵 안에 존재한다는 가정하에

[이동 가능한 경우]

(1) 이동할 칸= 벽 & 현재 벽 안 부수고 온 경우 -> 이동

(2) 이동할 칸 = 빈칸 & 방문 여부 x -> 이동

4. 결과 출력

- 목적지(N,M)에 도달하면 현재까지 이동한 거리를 반환
- BFS 종료될때까지 도착하지 못하면 -> -1 반환

Solution

```
import sys
from collections import deque

# 1. 입력 그래프
N , M = map(int,sys.stdin.readline().split())
graph = []
for n in range(N) :
    graph.append(list(map(int, input())))

#방문 여부 & 시작노드로 부터 최단 경로 거리 ,벽 부순 횟수(3차원)
visited = [[[0]*2 for _ in range(M)] for k in range(N)]

#상하좌우 -> 인접 노드
dx = [0,0,1,-1]
dy = [1,-1,0,0]

# 2. bfs 최단 거리
def bfs():
    #초기 노드 queue 에 추가 , 방문 등록
    dq = deque()
    dq.append((0,0,0))
    visited[0][0][0] = 1
```

1. 입력 처리 및 초기화

- 행렬 크기와 맵 데이터 입력 받기
- 방문 여부 및 최단 거리 나타낼 **visited** 배열 초기화
 - visited[x][y][0]** : 벽 부수지 않고 도착한 최단 거리
 - visited[x][y][1]** : 벽 부수고 도착한 최단 거리

Solution

```
while dq :
    x,y,wall = dq.popleft()
    # 목적지(N,M) 도착한 경우, 이동 횟수 출력
    if x == N-1 and y == M-1 :
        return visited[x][y][wall]

    for i in range(4):#상하좌우로 다음 이동할 칸의 위치
        xx = x+ dx[i] ; yy = y+ dy[i]
        # (1)다음 이동할 곳이 graph 밖에 있는 경우
        if xx<0 or xx>= N or yy <0 or yy>= M:
            continue
        # (2) 다음 이동할 곳이 벽이고, 한번도 벽 안 뚫었을때
        if graph[xx][yy] == 1 and wall == 0 : # and not visited[xx][yy][1]
            visited[xx][yy][1] = visited[x][y][0]+1
            dq.append((xx,yy,1))

        # (3) 다음 이동할 곳이 벽이 아니고, 한번도 방문 하지 않은 곳
        elif graph[xx][yy] == 0 and visited[xx][yy][wall] == 0 : # 그냥 감
            visited[xx][yy][wall] = visited[x][y][wall] +1
            dq.append((xx,yy,wall))

    return -1 #BFS 종료될때 까지 도착점에 도달하지 못하면 -1 출력하기

print(bfs())
```

2. BFS를 위한 queue 초기화

- 시작점 (0,0) 에서 출발하며 , 벽 부수지 않은 상태=0 로 큐에 추가
- visited[0][0][0] =1 초기화(출발칸도 경로에 포함됨)

3. BFS 탐색 과정

- queue에서 현재 위치와 벽 부순 상태를 꺼내 상하좌우로 이동 시도
- 다음 이동할 칸이 맵 안에 존재한다는 점이 전제조건

[이동 가능한 경우]

(1) 이동할 칸= 벽 & 현재 벽 안 부수고 온 경우 -> 이동

(2) 이동할 칸 = 빈칸 & 방문 여부 x -> 이동

4. 결과 출력

- 목적지(N,M)에 도달하면 현재까지 이동한 거리를 반환
- BFS 종료될때까지 도착하지 못하면 -> -1 반환

Assignment

[백준#13460 구슬탈출 2] Gold1

- 문제 : <https://www.acmicpc.net/problem/13460>

#BFS , #구현