Algorithm

동양미래대학교 강은영

그래프 탐색 알고리즘: DFS/BFS

- 탐색(Search)이란 많은 양의 데이터 중에서 원하는 데이터를 찾는 과정을 말합니다.
- 대표적인 그래프 탐색 알고리즘으로는 DFS와 BFS가 있습니다.
- DFS/BFS는 코딩 테스트나 인터뷰에서 매우 자주 등장하는 유형입니다.

스택 과 큐 자료구조

- 먼저 들어 온 데이터가 나중에 나가는 형식(선입후출)의 자료구조입니다.
- 입구와 출구가 동일한 형태로 스택을 시각화할 수 있습니다.

- 먼저 들어 온 데이터가 먼저 나가는 형식(선입선출)의 자료구조입니다.
- 큐는 입구와 출구가 모두 뚫려 있는 터널과 같은 형태로 시각화 할 수 있습니다.

DFS와 BFS(Breadth-First Search)

- DFS는 깊이 우선 탐색이라고도 부르며, 그래프에서 깊은 부분을 우선적으로 탐색하는 알고리즘입니다. – 스택과 재귀함수

- BFS는 너비 우선 탐색이라고도 부르며, 그래프에서 가까운 노드부터 우선적으로 탐색하는 알고리즘입니다. - 큐

<문제> 미로 탈출: 문제 설명

- 동빈이는 N × M 크기의 직사각형 형태의 미로에 갇혔습니다. 미로에는 여러 마리의 괴물이 있어 이를 피해 탈출해야 합니다.
- 동빈이의 위치는 (1, 1)이며 미로의 출구는 (N, M)의 위치에 존재하며 한 번에 한 칸씩 이동할 수 있습니다. 이때 괴물이 있는 부분은 0으로, 괴물이 없는 부분은 1로 표시되어 있습니다. 미로는 반드시 탈출할 수 있는 형태로 제시됩니다.
- 이때 동빈이가 탈출하기 위해 움직여야 하는 최소 칸의 개수를 구하세요. 칸을 셀 때는 시작 칸과 마지막 칸을 모두 포함해서 계산합니다.

<문제> 미로 탈출: 문제 조건

난이도 ●◐○ | **풀이 시간** 30분 | **시간제한** 1초 | **메모리 제한** 128MB

입력 조건

- 첫째 줄에 두 정수 N, M(4≤N, M≤200)이 주어집니다. 다음 N개의 줄에는 각각 M개의 정수(0 혹은 1)로 미로의 정보가 주어집니다. 각각의 수들은 공백 없이 붙어서 입력으로 제시됩니다. 또한 시작 칸과 마지막 칸은 항상 1입니다.

출력 조건

- 첫째 줄에 최소 이동 칸의 개수를 출력합니다.

입력 예시

```
5 6
101010
111111
000001
111111
111111
```

출력 예시

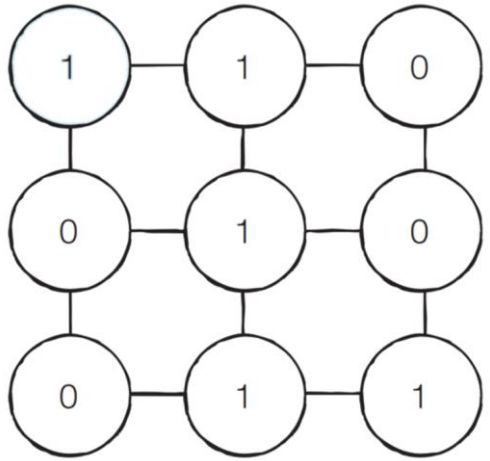
```
10
```

<문제> 미로 탈출: 문제 해결 아이디어

- BFS는 시작 지점에서 가까운 노드부터 차례대로 그래프의 모든 노드를 탐색합니다.
- 상, 하, 좌, 우로 연결된 모든 노드로의 거리가 1로 동일합니다.
  - 따라서 (1, 1) 지점부터 BFS를 수행하여 모든 노드의 최단 거리 값을 기록하면 해결할 수 있습니다.
- 예시로 다음과 같이 3 X 3 크기의 미로가 있다고 가정합시다.

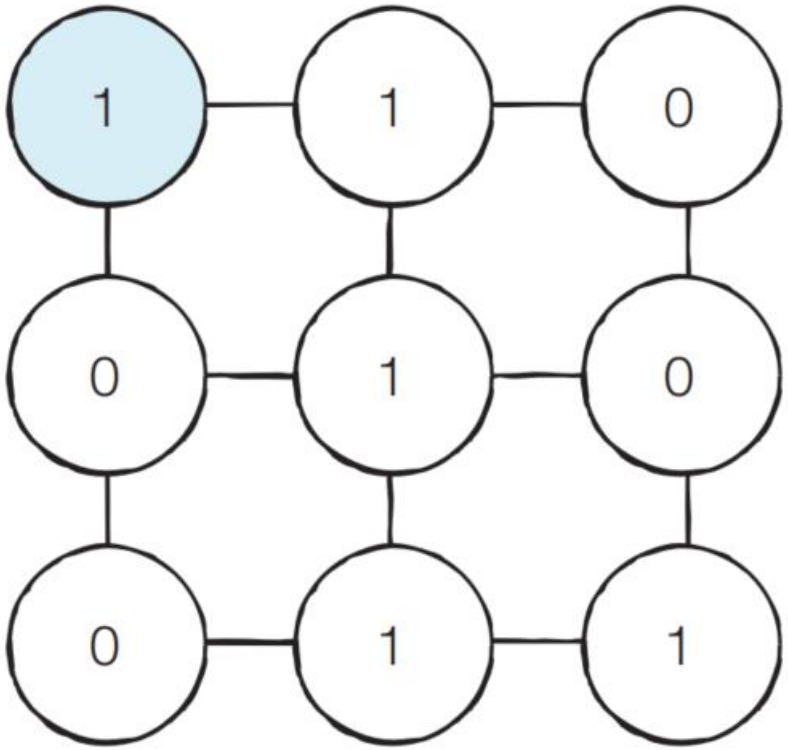
```
110
010
011
```

그래프 표현



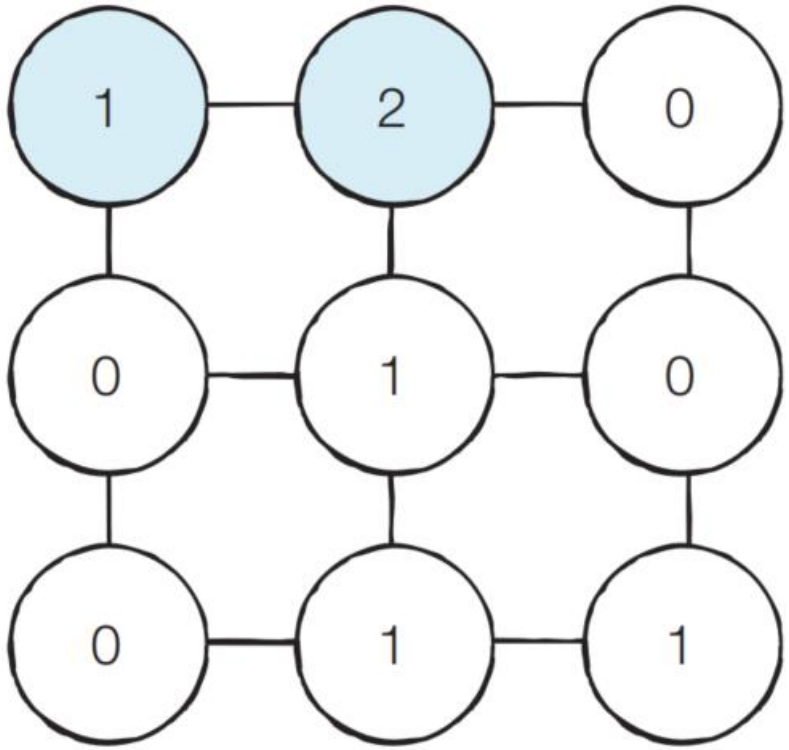
<문제> 미로 탈출: 문제 해결 아이디어

- [Step 1] 처음에 (1, 1)의 위치에서 시작합니다.



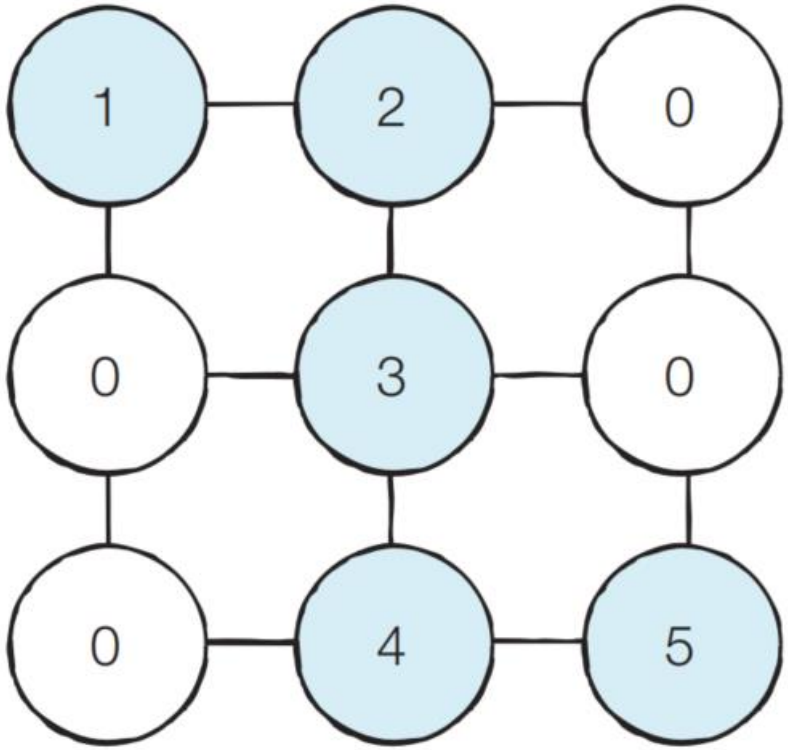
<문제> 미로 탈출: 문제 해결 아이디어

- [Step 2] (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 바꾸게 됩니다.



<문제> 미로 탈출: 문제 해결 아이디어

- [Step 3] 마찬가지로 BFS를 계속 수행하면 결과적으로 다음과 같이 최단 경로의 값들이 1씩 증가하는 형태로 변경됩니다.



Collections라이브러리 deque클래스

- 파이썬 collections 라이브러리의 deque를 사용해 큐를 구현한다.
- 첫번째 원소를 제거할때 popleft(), 마지막원소 제거 pop() 리스트:O(N) deque:O(1)
- 첫번째 원소 삽입 appendleft(x), 마지막 원소 삽입append(x) 리스트:O(N) deque:O(1)

```
from collections import deque

data = deque([2,3,4])
data.appendleft(1)
data.append(5)
print(data)
Print(list(data)) #리스트 자료형으로 변환
```

실행 결과:

```
deque([1,2,3,4,5])
[1,2,3,4,5]
```

Conunter

- 파이썬 collections 라이브러리의 Counter는 등장 횟수를 세는 기능을 제공합니다.
- 리스트와 같은 반복 가능한(iterable) 객체가 주어졌을 때 내부의 원소가 몇 번씩 등장했는지를 알려줍니다.

```
from collections import Counter

cnter = Counter(['red', 'blue', 'red', 'green', 'blue', 'blue'])

print(cnter['blue']) # 'blue'가 등장한 횟수 출력
print(cnter['green']) # 'green'이 등장한 횟수 출력
print(dict(cnter)) # 사전 자료형으로 반환
```

실행 결과:

```
3
1
{'red': 2, 'blue': 3, 'green': 1}
```

<문제> 미로 탈출: 답안 예시(Python) - BFS

```python
# 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
            # 벽인 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]
```

```python
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]

# BFS를 수행한 결과 출력
print(bfs(0, 0))
```

<문제> 미로 탈출: 답안 예시(C++) - BFS

```
int bfs(int x, int y) {
    // 큐(Queue) 구현을 위해 queue 라이브러리 사용
    queue<pair<int, int> > q;
    q.push({x, y});
    // 큐가 빌 때까지 반복하기
    while(!q.empty()) {
        int x = q.front().first;
        int y = q.front().second;
        q.pop();
        // 현재 위치에서 4가지 방향으로의 위치 확인
        for (int i = 0; i < 4; i++) {
            int nx = x + dx[i];
            int ny = y + dy[i];
            // 미로 찾기 공간을 벗어난 경우 무시
            if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
            // 벽인 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;
                q.push({nx, ny});
            }
        }
    }
    // 가장 오른쪽 아래까지의 최단 거리 반환
    return graph[n - 1][m - 1];
}
```

```
#include <bits/stdc++.h>

using namespace std;

int n, m;
int graph[201][201];

// 이동할 네 가지 방향 정의 (상, 하, 좌, 우)
int dx[] = {-1, 1, 0, 0};
int dy[] = {0, 0, -1, 1};

int main(void) {
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            scanf("%1d", &graph[i][j]);
        }
    }
    cout << bfs(0, 0) << '₩n';
    return 0;
}
```

<문제> 미로 탈출: 답안 예시(Java) - BFS

```
import java.io.*;
import java.util.*;

public class Solution {
    public static int N,M;
    public static int [][] graph = new int [200][200];
    //이동할 네 가지 방향 정의 {상, 하, 좌 ,우}
    public static int dx[] = {-1, 1, 0, 0};
    public static int dy[] = {0, 0, -1, 1};

    public static void main(String[] args){
        Scanner scanner = new Scanner(System.in);
        // N, M 입력
        N = scanner.nextInt();
        M = scanner.nextInt();
        scanner.nextLine(); //버퍼 지우기
        System.out.println("n " + N + "m" + M);

        // 2차원 배열의 정보 입력 받기
        for(int i=0; i <N; i++){
            String str = scanner.nextLine();
            for(int j=0; j<M; j++){
                graph[i][j] = str.charAt(j) - '0';
                    }
        }
        scanner.close();
        System.out.println(" result = " + bfs(0, 0));
    }
```

```
    // BFS로 특정 노드를 방문하고 상하좌우로 연결된 모든 노드들도 방문
    public static int bfs(int x, int y){
        Queue<Node> queue = new LinkedList<>();
        queue.add(new Node(x,y));
        //큐가 빌 때까지 반복
        while(!queue.isEmpty()){
            Node node = queue.poll();
            x = node.getX();
            y = node.getY();

            for(int i=0;i<4;i++){
                int nx = x + dx[i];
                int ny = y + dy[i];

                //미로 범위를 벗어나면 무시
                if(nx <0 || ny < 0 || nx >= N || ny >= M) continue;
                //한 번 왔던 위치면 무시 , 괴물이 있는 위치면 무시
                if(graph[nx][ny]==1){
                    graph[nx][ny] = graph[x][y] + 1;
                    queue.add(new Node(nx, ny));
                }
            }
        }
        // 가장 오른쪽 아래까지의 최단 거리 반환
        return graph[N-1][M-1];
    }
     static class Node{
        final private int x;
        final private int y;
        Node(int x, int y){
            this.x = x;
            this.y = y;
        }
        public int getX() {    return x;          }
        public int getY(){      return y;        }
    }
}}
```

<문제> 미로 탈출: DFS 로도 가능

- https://www.acmicpc.net/problem/2178 백준온라인저지

출력

첫째 줄에 지나야 하는 최소의 칸 수를 출력한다. 항상 도착위치로 이동할 수 있는 경우만 입력으로 주어진다.

예제 입력 1 복사

```
4 6
101111
101010
101011
111011
```

예제 출력 1 복사

```
15
```

예제 입력 2 복사

```
4 6
110110
110110
111111
111101
```

예제 출력 2 복사

```
9
```

예제 입력 3 복사

```
2 25
1011101110111011101110111
1110111011101110111011101
```

예제 출력 3 복사

```
38
```

<문제> 미로 탈출: 답안 예시(C++) - DFS

```cpp
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
int n, m;
int graph[201][201];
// 이동할 네 가지 방향 정의 (상, 하, 좌, 우)
int dx[] = {-1, 1, 0, 0};
int dy[] = {0, 0, -1, 1};
int main(void) {
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            scanf("%1d", &graph[i][j]);
        }
    }
    cout << dfs(0, 0) << '₩n';
    return 0;
}
```

```cpp
int dfs(int x, int y) {
    // 스택(Stack) 구현을 위해 stack 라이브러리 사용
    stack<pair<int, int> > st;
    st.push({x, y});
    // 스택이 빌 때까지 반복하기
    while(!st.empty()) {
        int x = st.top().first;
        int y = st.top().second;
        st.pop();
        // 현재 위치에서 4가지 방향으로의 위치 확인
        for (int i = 0; i < 4; i++) {
            int nx = x + dx[i];
            int ny = y + dy[i];
            // 미로 찾기 공간을 벗어난 경우 무시
            if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
            // 벽인 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;
                st.push({nx, ny});
            }
        }
    }
    // 가장 오른쪽 아래까지의 최단 거리 반환
    return graph[n - 1][m - 1];
}
```

<문제> 미로 탈출: 답안 예시(Java) – DFS

```java
import java.io.*;
import java.util.*;

public class Solution {
    public static int N,M;
    public static int [][] graph = new int [200][200];
    //이동할 네 가지 방향 정의 {상, 하, 좌 ,우}
    public static int dx[] = {-1, 1, 0, 0};
    public static int dy[] = {0, 0, -1, 1};

    public static void main(String[] args){
        Scanner scanner = new Scanner(System.in);
        // N, M 입력
        N = scanner.nextInt();
        M = scanner.nextInt();
        scanner.nextLine(); //버퍼 지우기
        System.out.println("n " + N + "m" + M);

        // 2차원 배열의 정보 입력 받기
        for(int i=0; i <N; i++){
            String str = scanner.nextLine();
            for(int j=0; j<M; j++){
                graph[i][j] = str.charAt(j) - '0';
            }
        }
        scanner.close();
        System.out.println("result = " + dfs(0,0));
    }
```

```java
    private static int dfs(int x, int y)
    {
        Stack<Node> stack = new Stack<Node>();
        Node node = new Node(x, y);
        stack.push(node);
        while (!stack.empty()) {
            node = stack.pop();
            x = node.getX();
            y = node.getY();

            for(int i=0;i<4;i++){
                int nx = x + dx[i];
                int ny = y + dy[i];
                //미로 범위를 벗어나면 무시
                if(nx <0 || ny < 0 || nx >= N || ny >= M ) continue;

                //한 번 왔던 위치면 무시 , 괴물이 있는 위치면 무시
                if  (graph[nx][ny]==1 ){
                    System.out.println("[" + nx + "," + ny + "]");
                    graph[nx][ny] = graph[x][y] + 1;
                    stack.push(new Node(nx, ny));
                }
            }
        }
        return graph[N-1][M-1];
    }
    static class Node{
        final private int x;
        final private int y;
        Node(int x, int y){
            this.x = x;
            this.y = y;
        }
        public int getX() {            return x;        }
        public int getY(){            return y;        }
    }
}
```