1.2 NumberOfIsland

3.4.5 Maximum Depth Of BinaryTree

6. Max Of Islands

7. Word Ladder

8. Word Search

9. Remove Invalid Parentheses

10.11 Maze

1,2) 섬의 수(Number Of Island)

설명

m x n binary grid가 주어집니다. 각 cell의 값 중에 '1' 은 땅이고 '0' 은 물입니다.
섬은 물에 의해 둘러싸입니다.
섬은 수직 또는 수평으로 인접하는 땅(1)을연결함으로써 형성됩니다.
섬은 grid의 네 모서리가 모두 물로 둘러싸여 있다고 가정 할 수 있습니다.
섬의 개수를 리턴하세요

입출력

Input: grid = [
["1","1","1","1","0"],
["1","1","1","1","0"],
["1","1","0","0","0"],
["0","0","0","0","0"]
]
Output: 1

Input: grid = [
["1","1",0,0,"0"],
["1","1",0,0,"0"],
["1","1","0","0","0"],
["0","0","0",1,1]
]
Output: 2

문제 Format

```
class Solution {
    public int solve(int[][] grid) {
    }}
```

제한사항

m == grid.length
n == grid[i].length
1 <= m, n <= 300
grid[i][j] is '0' or '1'

Solution

substring

1. 문제분석

- 문제를 정확히 이해
- 분석 내용 정리(Devide & Conquer)

2. 규칙찾기

- 분석 내용으로 규칙을 찾는다

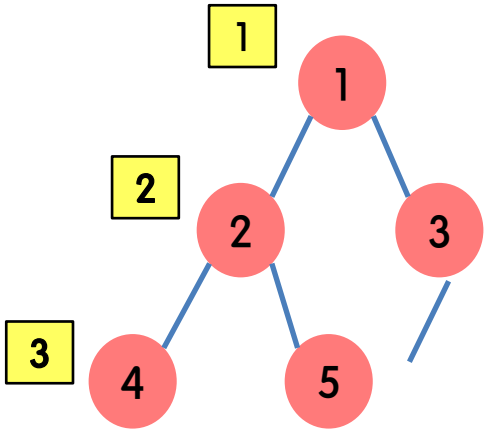
3. 코딩화

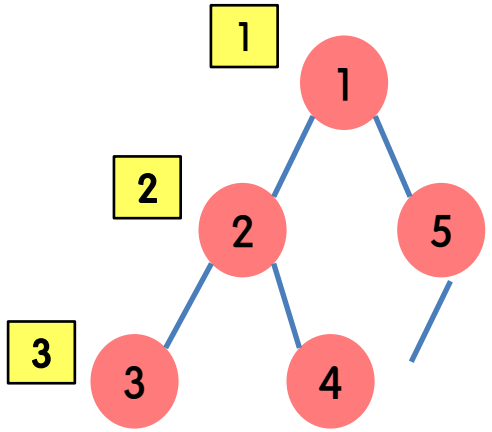
- 분석 내용으로 알맞은 구현방법 찾기

4. 알고리즘 적용

- 알고리즘 정하고 담을 그릇 정한다
(사전지식 필요)

bfs dfs 개념 설명





Queue

Breadth First Search

근처에 있는것부터 찾는 알고리즘

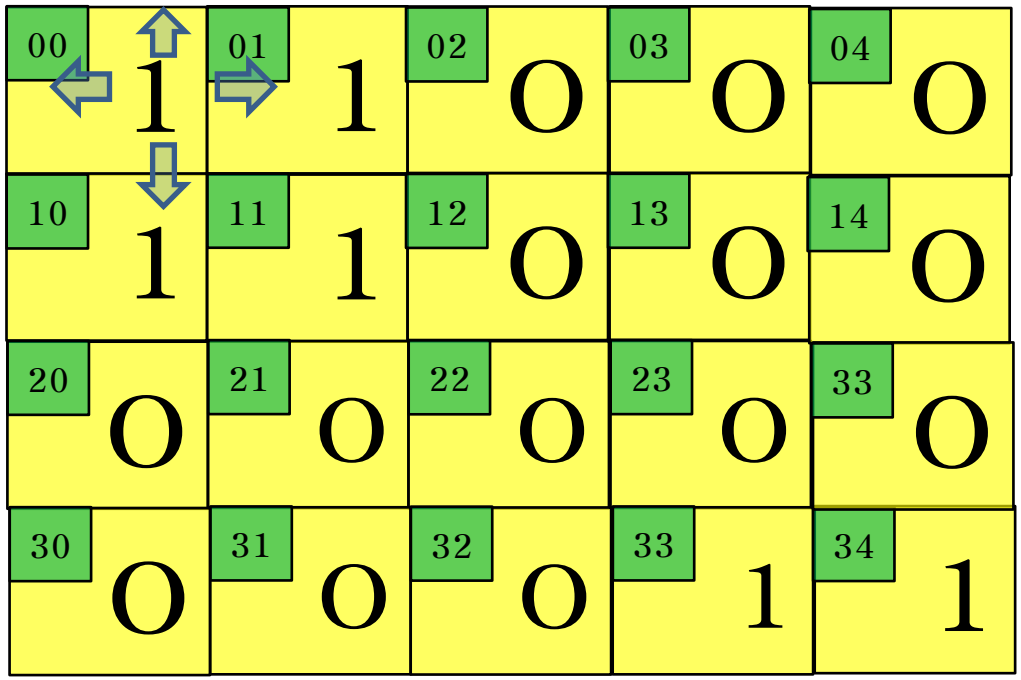
(다익스트라, 페이스북)

Stack

Depth First Search

깊이 우선순위 탐색 ,알파고

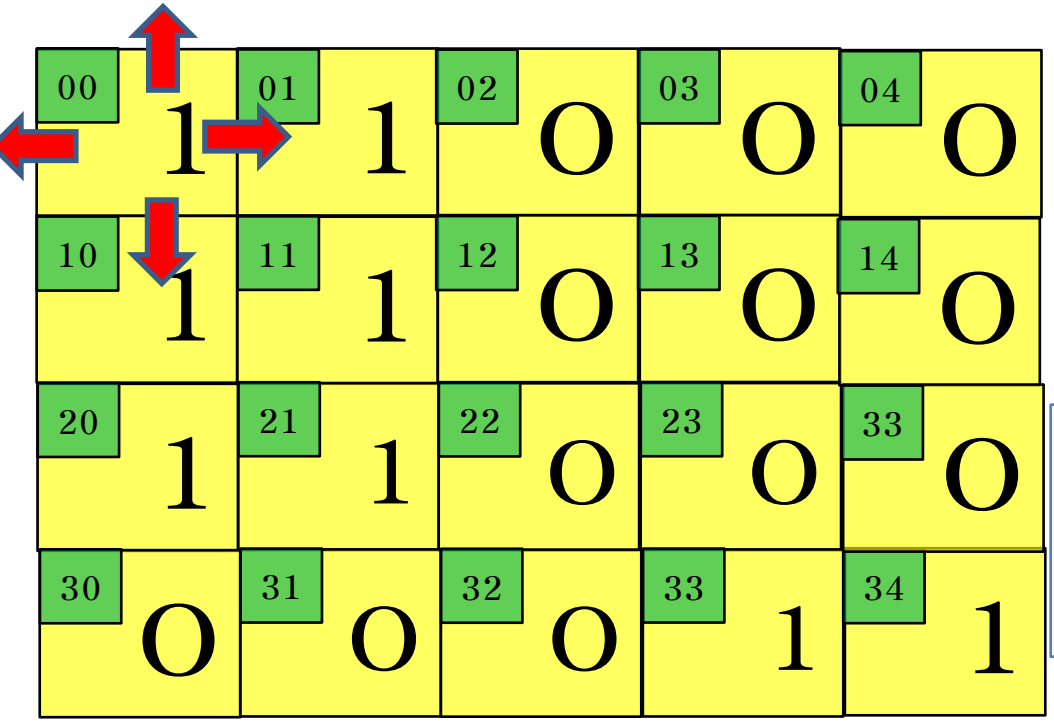
문제분석

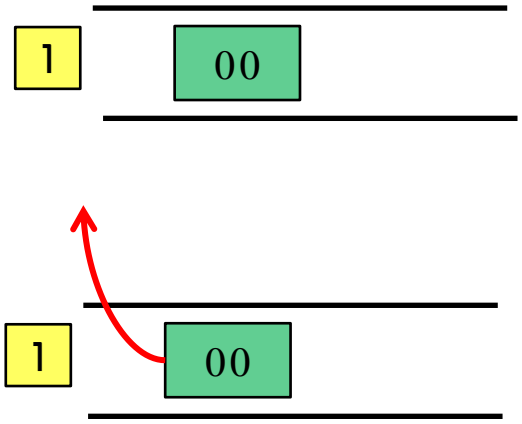


1. 1은 땅이고, 0은 바다입니다.
2. 1을 인식하고 0이 나오면 가지 않으면 된다.
3. For문을 2번돌리면서

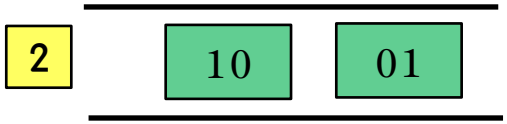
문제분석

BFS(Queue)



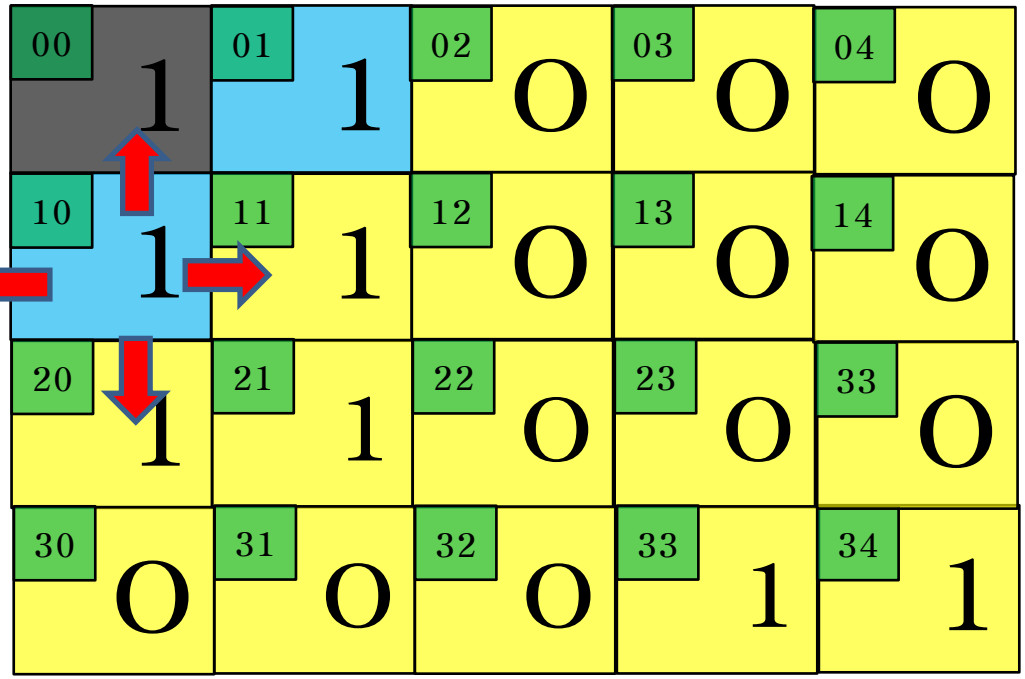


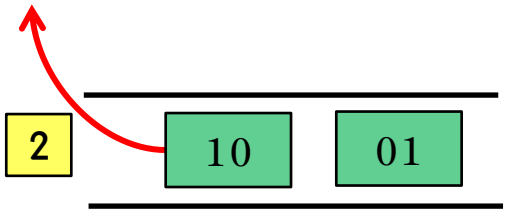
1. Queue에서 poll 한후
2. 4방향으로 체크시작
3. 공식: 조건체크 부분 수행



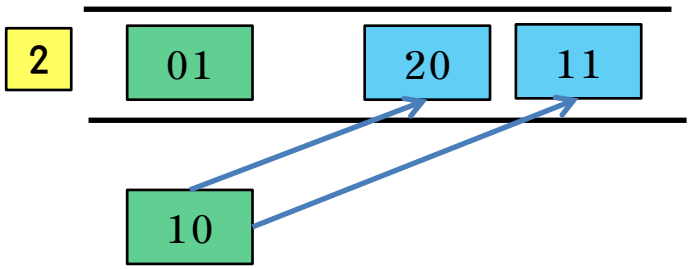
문제분석

BFS(Queue)



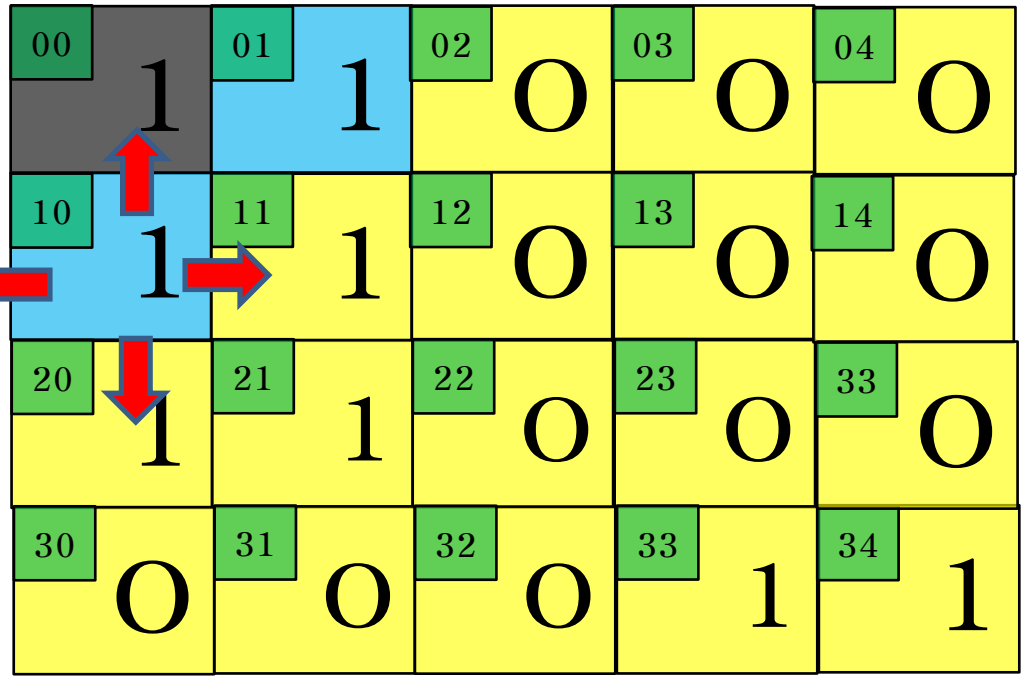


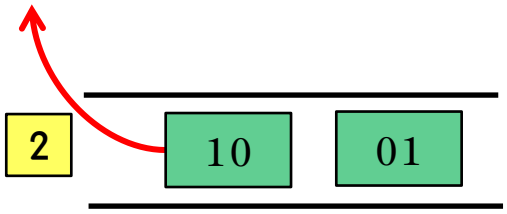
1. Queue에서 poll 한후
2. 4방향으로 체크시작
3. 공식: 조건체크 부분 수행



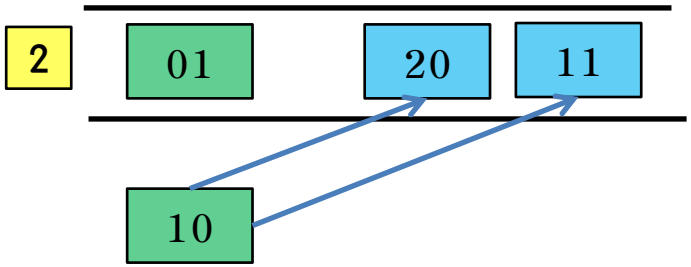
문제분석

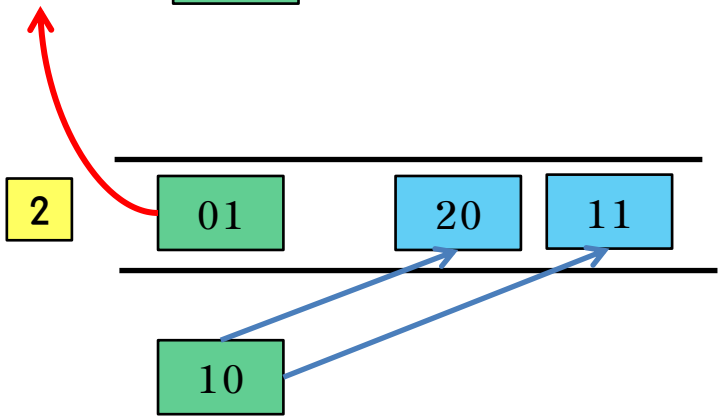
BFS(Queue)

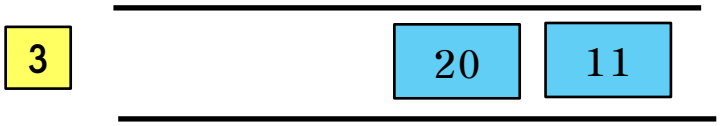




1. Queue에서 poll 한후
2. 4방향으로 체크시작
3. 공식: 조건체크 부분 수행

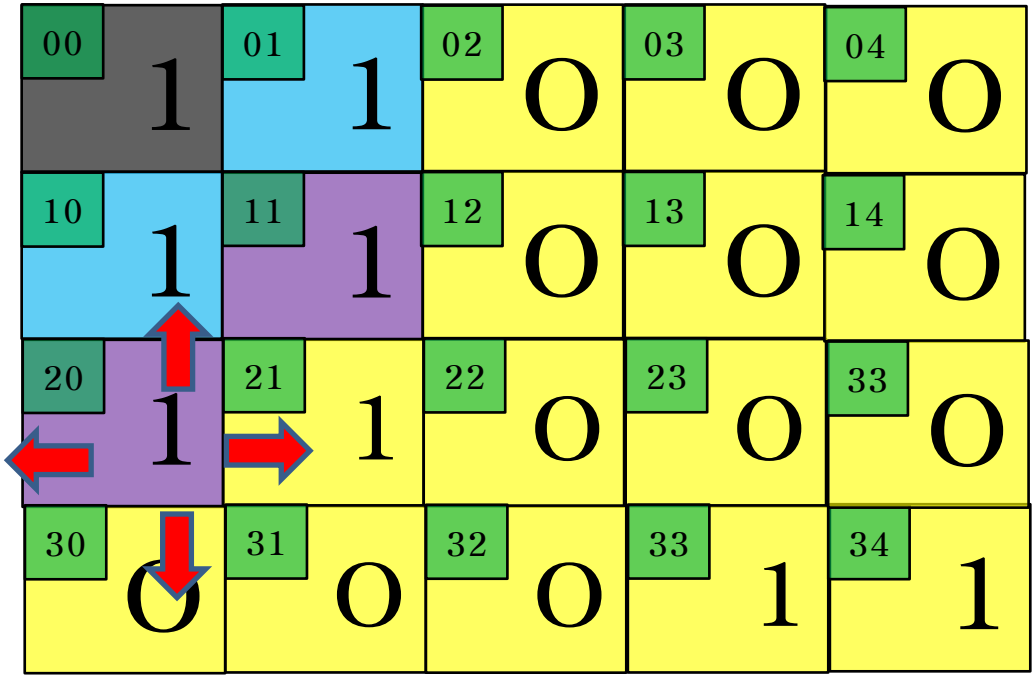


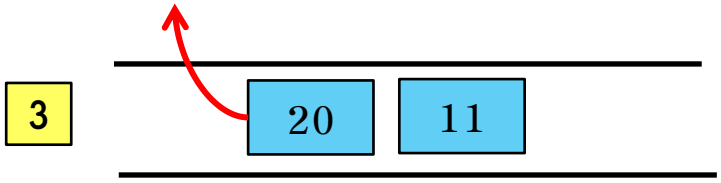




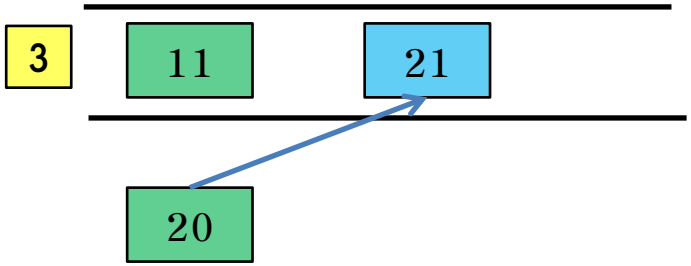
문제분석

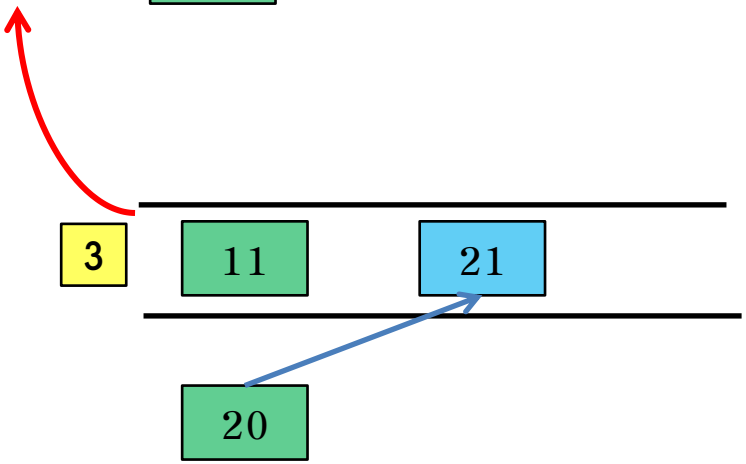
BFS(Queue)





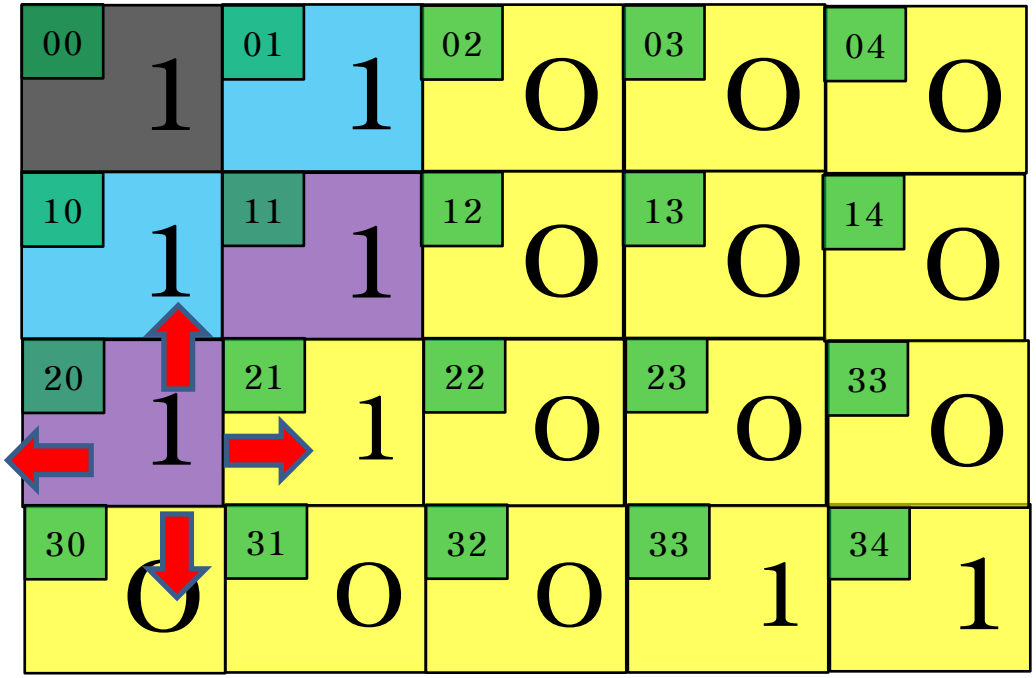
1. Queue에서 poll 한후
2. 4방향으로 체크시작
3. 공식: 조건체크 부분 수행

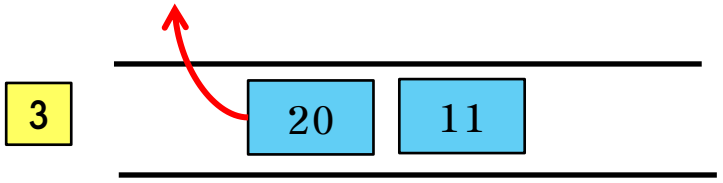




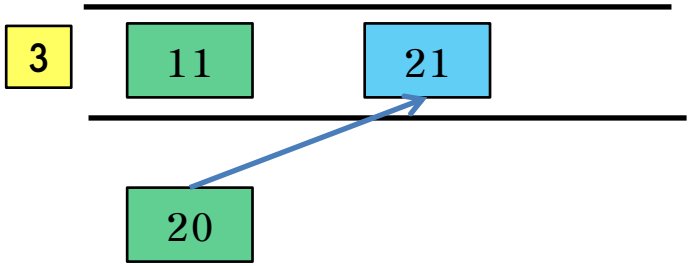
문제분석

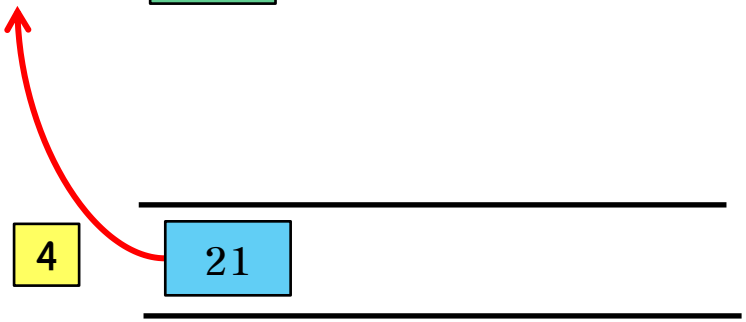
BFS(Queue)





1. Queue에서 poll 한후
2. 4방향으로 체크시작
3. 공식: 조건체크 부분 수행





참고

1 항목 셋팅

이차원배열 사이즈 & 방향 설정(4방향)

```
int m, n;
m = grid.length;
n = grid[0].length;
int[][] dirs = { { -1, 0 }, { 1, 0 }, { 0, 1 }, { 0, -1 } };
boolean[][] visited = new boolean[m][n];
```

참고

2 맞는 조건을 찾아내는 부분

1. Grid에서 위치(i ,j)를 찾아낸다.

```
for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
                if (grid[i][j] == 'l') {
                    count = bfs(grid, i, j, visited);
                    break;
                }
        }
}
```

참고

3 Queue 생성 및 셋팅

```
public int bfs(char[][] grid, int x, int y, boolean[][] visited) {
                visited[x][y] = true;
                Queue<int[]> q = new LinkedList<>();
                q.offer(new int[] { x, y, 0 });
```

참고

4

조건체크해서 큐에 넣는 부분
1. 마이너스 좌표체크
2. m*n 범위체크
3. grid[x][y] 값체크(문제제시값)
4. 위 조건 만족시 queue에 넣고 다시 한칸씩 이동

|| or조건

```
if( x1<0 || x1>= m || y1<0 || y1>=n || visited[x1][y1] == true || grid[x1][y1] == 'X')
  continue;
```

&& and조건

```
if (x1>=0 && y1 >= 0 && x1 < m && y1 < n && visited[x1][y1] != true && grid[x1][y1] != 'X')
 {


 }
```

1. Basic_BFS

외워야할 부분

1 방향 설정 & 이차원배열 사이즈

```
int m, n;
m = grid.length;
n = grid[0].length;
int[][] dirs = { { -1, 0 }, { 1, 0 }, { 0, 1 }, { 0, -1 } };
```

2 맞는 조건을 찾아내는 부분

```
for (int i = 0; i < grid.length; i++) {
    for (int j = 0; j < grid[i].length; j++) {
        if (grid[i][j] == '1') {
            count++;
            bfs(grid, i, j);
        }
    }
}
```

1. Basic_BFS

외워야할 부분

3 큐에서 빼네는 부분

4 조건체크해서 큐에 넣는 부분(&&)
1. 마이너스 좌표체크 2. m*n 범위체크 3. grid[x][y] 값체크(문제제시값)

```
while (!queue.isEmpty()) {
        int[] point = queue.poll();
        for (int[] dir : dirs) {
                int x1 = point[0] + dir[0];
                int y1 = point[1] + dir[1]
                if (x1 >= 0 && y1 >= 0 && x1 < grid.length && y1 < grid[0].length &&
                   grid[x1][y1] == '1') {
                grid[x1][y1] = '0';

                }
      }
}
```

1. Basic_BFS

요약

1 방향 설정 & 이차원배열 사이즈

```
int m, n;
m = grid.length;
n = grid[0].length;
int[][] dirs = { { -1, 0 }, { 1, 0 }, { 0, 1 }, { 0, -1 } };
```

2 맞는 조건을 찾아내는 부분

```
for (int i = 0; i < grid.length; i++) {
        for (int j = 0; j < grid[i].length; j++) {
                if (grid[i][j] == '1') {
                count++;
                bfs(grid, i, j);
                }
        }}
```

3 Queue 생성 및 셋팅

4 조건체크해서 큐에 넣는 부분(&&)
1. 마이너스 좌표체크 2. m*n 범위체크 3. grid[x][y] 값체크(문제제시값)

```
while (!queue.isEmpty()) {
        int[] point = queue.poll();
        for (int[] dir : dirs) {
                int x1 = point[0] + dir[0];
                int y1 = point[1] + dir[1]
         if (x1 >= 0 && y1 >= 0 && x1 < grid.length && y1 < grid[0].length &&
            grid[x1][y1] == '1') {
                grid[x1][y1] = '0';
                }
    }
}
```

|| or조건

```
if( x1<0 || x1>= m || y1<0 || y1>=n ||
 visited[x1][y1] == true || grid[x1][y1] == 'X')
  continue;
```

&& and조건

```
if (x1>=0 && y1 >= 0 && x1 < m && y1 < n
&& visited[x1][y1] != true && grid[x1][y1] !=
'X') { }
```

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(M*N)
   대상 : char[][] grid
   이유 : m은 rows, n은 columns

공간복잡도

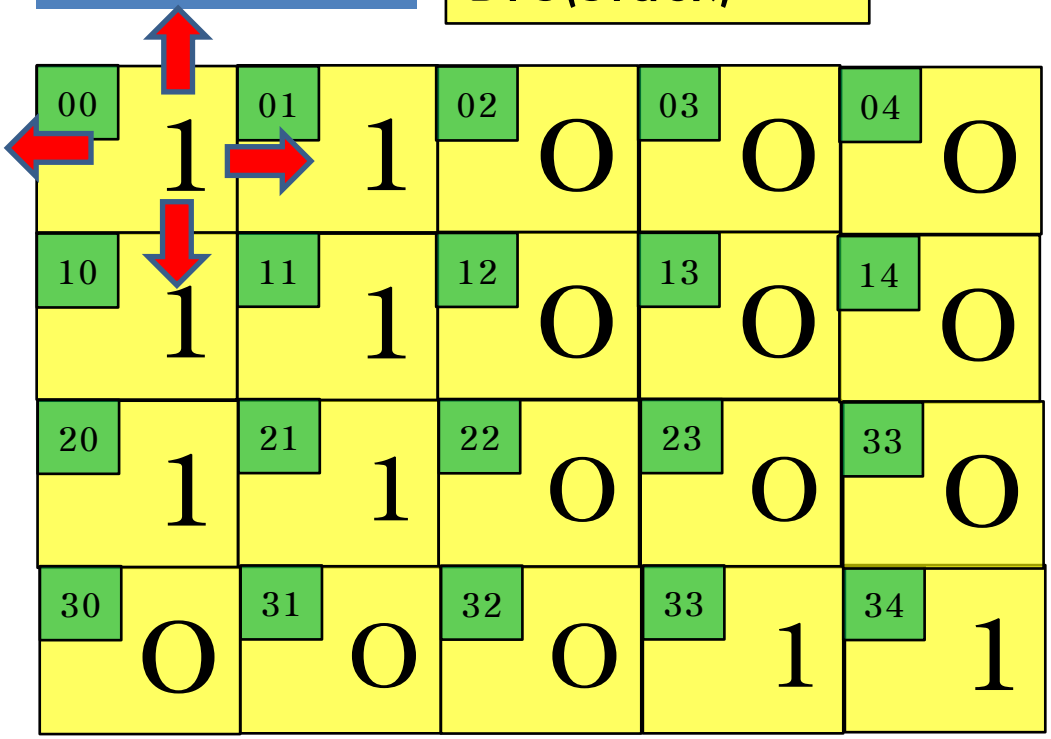
2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(min(m, n))
   대상 : Queue<int[]> q = new LinkedList<>();
   이유 : 큐에 좌표값 저장 계산 O(max(m, n)) O(min(m, n)) 중에 하나다

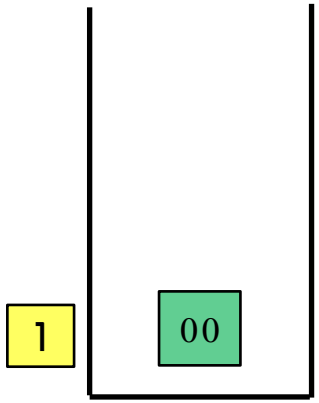
참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)

문제분석

DFS(Stack)



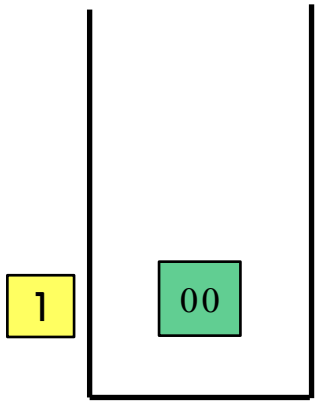


1. Stack에 (0,0) push
2. 4방향으로 체크시작
3. 공식 조건체크 부분 수행

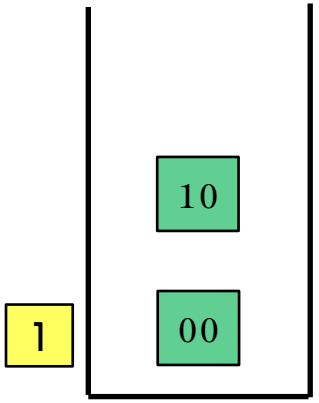
문제분석

DFS(Stack)

00	01	02	03	04
1	1	0	0	0
10	11	12	13	14
1	1	0	0	0
20	21	22	23	33
1	1	0	0	0
30	31	32	33	34
0	0	0	1	1

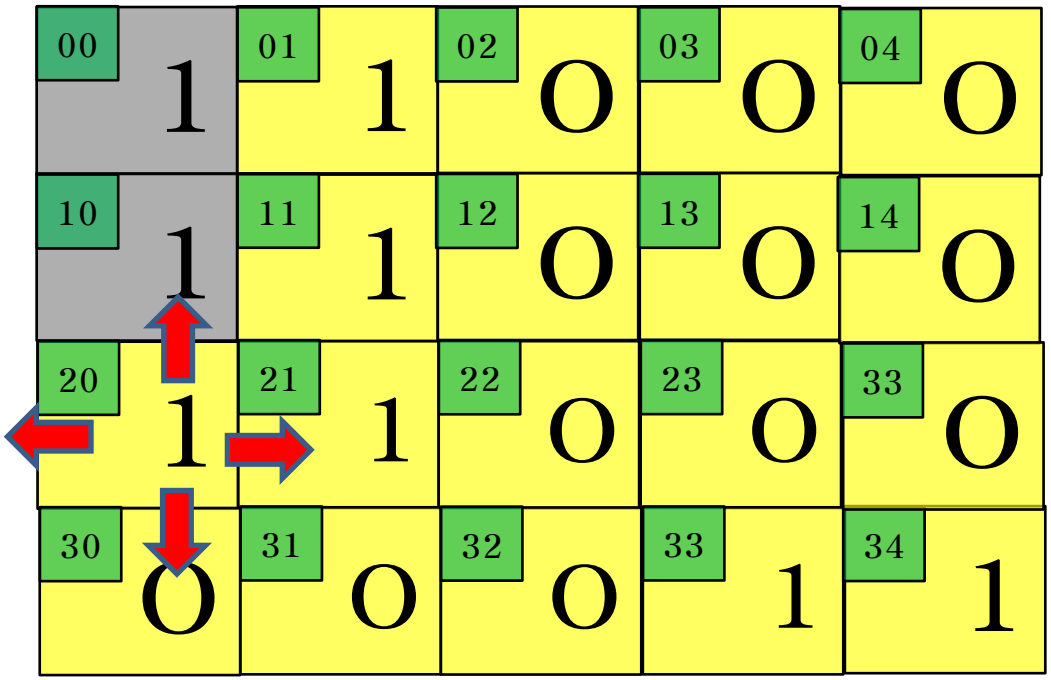


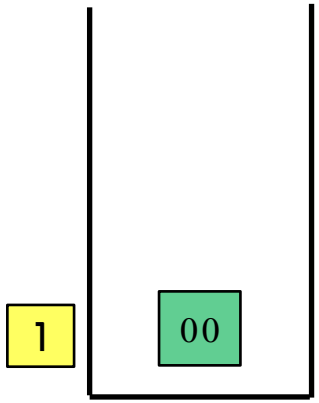
1. Stack에 (0,0) push
2. 4방향으로 체크시작
3. 공식 조건체크 부분 수행



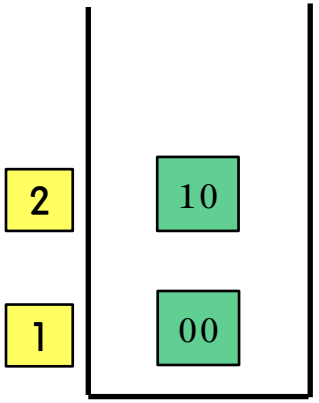
문제분석

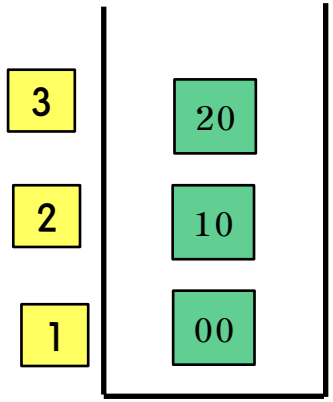
DFS(Stack)





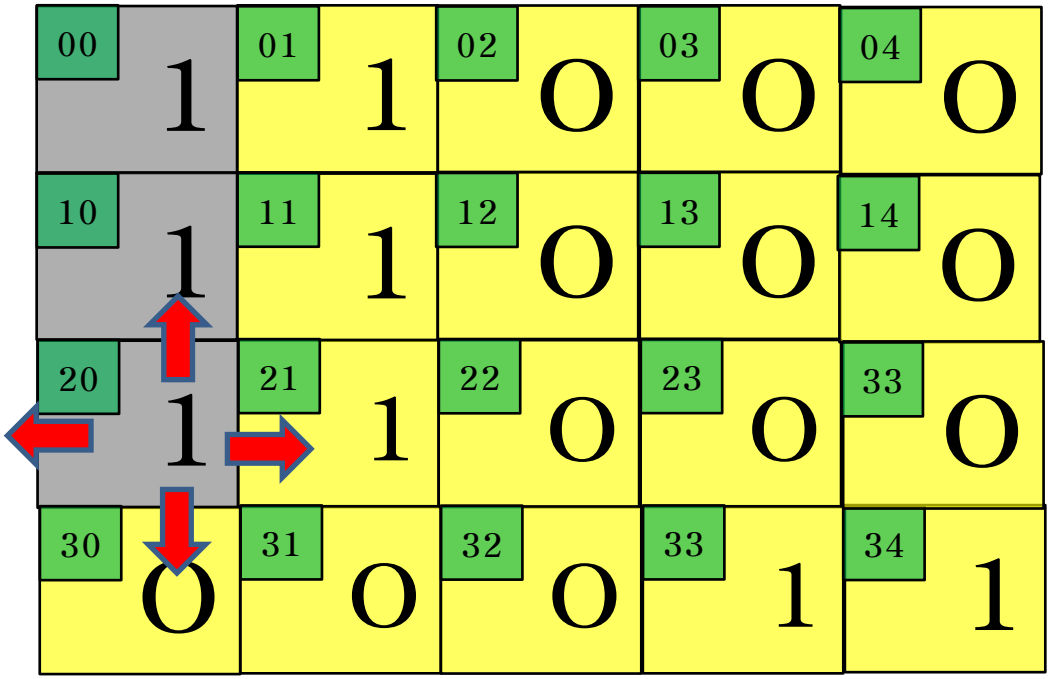
1. Stack에 (0,0) push
2. 4방향으로 체크시작
3. 공식 조건체크 부분 수행

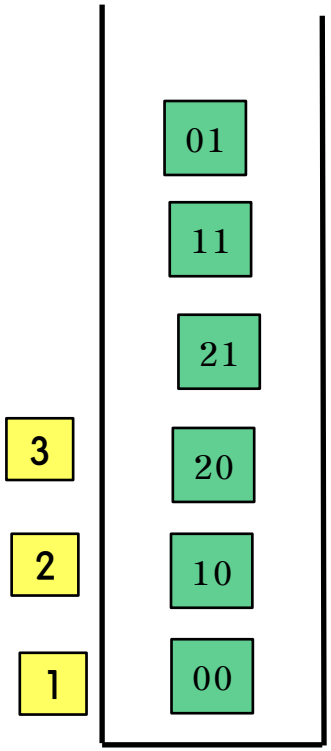




문제분석

DFS(Stack)





2. Basic_DFS

외워야할 부분

1 방향 설정 & 이차원배열 사이즈

```
int m, n;
m = grid.length;
n = grid[0].length;
int[][] dirs = { { -1, 0 }, { 1, 0 }, { 0, 1 }, { 0, -1 } };
```

2 맞는 조건을 찾아내는 부분

```
for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (grid[i][j] == '1') {
            count++;
            merge(grid, i, j);
            }
        }
}
```

2. Basic_DFS

외워야할 부분

3 재귀를 이용한다(stack개념)

4 조건체크해서 함수 재호출(재귀)
1. 마이너스 좌표체크 2. m*n 범위체크 3. grid[x][y] 값체크(문제제시값)

```
public void dfs(char[][] grid, int i, int j) {
    if (i < 0 || i >= m || j < 0 || j >= n || grid[i][j] != '1')
        return;
    grid[i][j] = 'X';
    for(int[] dir: dirs) {
        dfs(grid, i+dir[0], j+dir[1]);
    }
}
```

2. Basic_DFS

요약

1 방향 설정 & 이차원배열 사이즈

```
int m, n;
m = grid.length;
n = grid[0].length;
int[][] dirs = { { -1, 0 }, { 1, 0 }, { 0, 1 }, { 0, -1 } };
```

2 맞는 조건을 찾아내는 부분

```
for (int i = 0; i < grid.length; i++) {
        for (int j = 0; j < grid[i].length; j++) {
                if (grid[i][j] == '1') {
                count++;
                dfs(grid, i, j);
                }
        }}
```

3 재귀를 이용한다(stack개념)

4 조건체크해서 함수 재호출(재귀)
1. 마이너스 좌표체크 2. m*n 범위체크 3. grid[x][y] 값체크(문제제시값)

```
public void dfs(char[][] grid, int i, int j) {
    if (i < 0 || i >= m || j < 0 || j >= n || grid[i][j] != '1')
            return;
    grid[i][j] = 'X';
    for(int[] dir: dirs) {
            dfs(grid, i+dir[0], j+dir[1]);
    }
}
```

|| or조건

```
if( x1<0 || x1>= m || y1<0 || y1>=n ||
 visited[x1][y1] == true || grid[x1][y1] == 'X')
  continue;
```

&& and조건

```
if (x1>=0 && y1 >= 0 && x1 < m && y1 < n
&& visited[x1][y1] != true && grid[x1][y1] !=
'X') { }
```

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(M*N)
   대상 : char[][] grid
   이유 : m은 rows, n은 columns

공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(M*N) in worst case
   대상 : 내부 stack 생성
   이유 :

참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)

문제 3,4,5) Maximum Depth Of BinaryTree

Problem

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Note: A leaf is a node with no children.

Example:

Given binary tree [3,9,20,null,null,15,7],

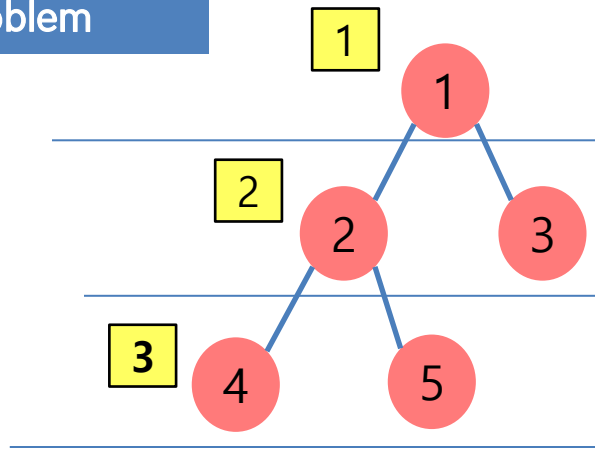
```
    3
   / \
  9  20
    /  \
   15   7
```

return its depth = 3.

Maximum Depth Of BinaryTree

BFS-queue

Problem



Queue[1] , Size =1, left, right 검색해서 Queue에 넣는다

Queue[2,3] Size =2 , 2일때 left, right 검색해서 Queue에 넣는다, 순간적으로 [3, 4, 5] 가됩니다. 3을 빼서검색할 는 left, right가없습니다. For문에서 size=2를 만족해서 나와버립니다

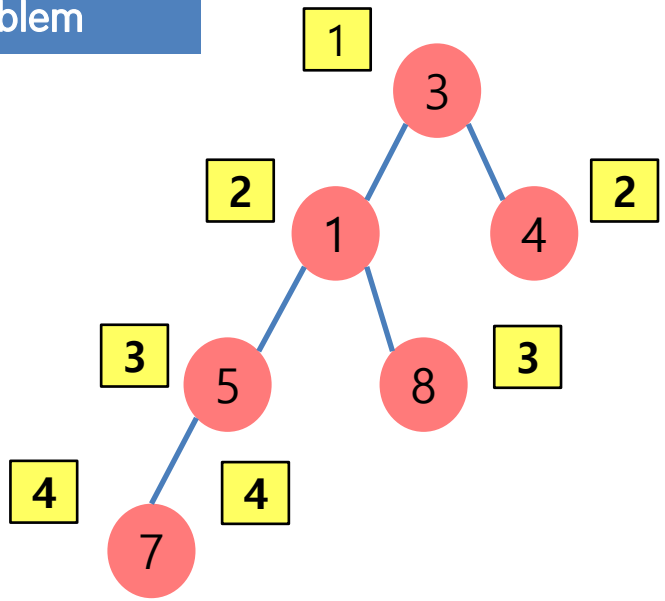
Queue[4,5] Size =2 , 4일때 left, right 가없습니다.
5일때도left, right가없습니다. For문에서 size=2를 만족해서 나와버립니다

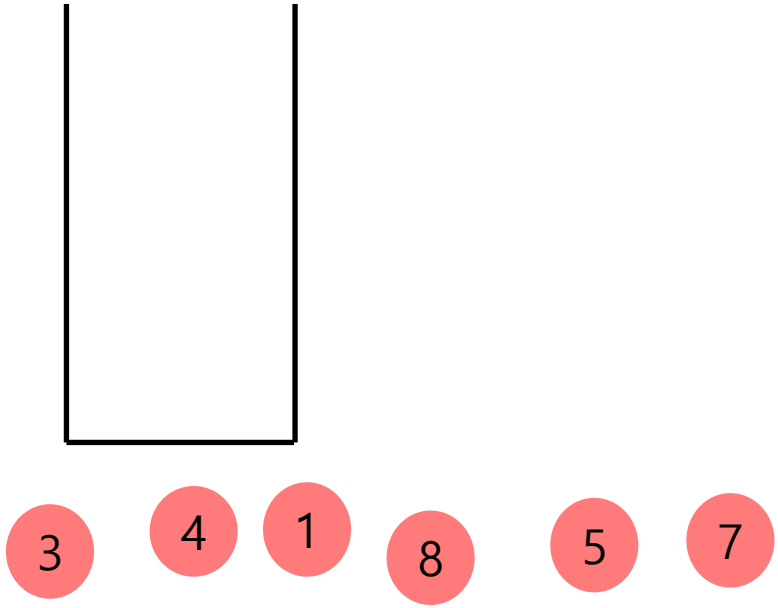
Queue[2,3]

Maximum Depth Of BinaryTree

DFS stack

Problem





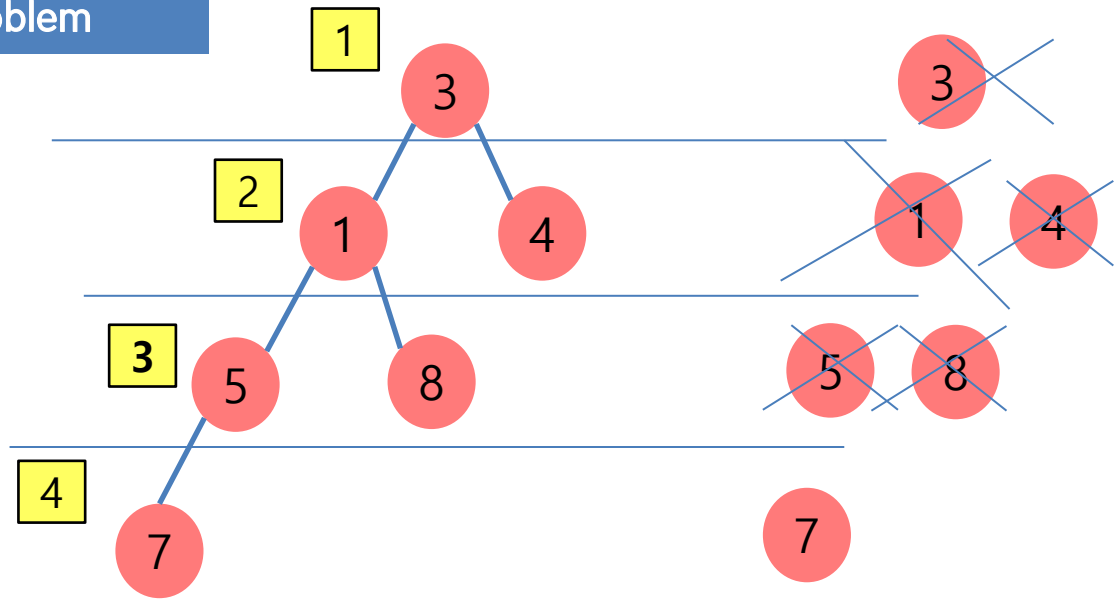
Example

1. DFS방식 스택을 이용한다.

Maximum Depth Of BinaryTree

BFS-queue

Problem



Example

1. Queue 방식, 근접해서 한칸씩
2.

문제 6) Max Area Of Islands

Problem

문제설명)
2차원배열 그리드가 0과 1 인 경우, 1이 육지(섬) 0이 바다입니다.
육지는 4 방향 (수평 또는 수직)으로 연결된 1 (육지를 나타냄)의 그룹입니다.
그리드의 네 모서리 모두가 물로 둘러싸여 있다고 가정 할 수 있습니다. 주어진 2D 배열에서 섬의 최대 면적을 찾으십시오.
(섬이 없으면 최대 면적은 0입니다.)

Note)
The length of each dimension in the given grid does not exceed 50

예제)

```
{{1,1,0,1,1},
 {0,1,1,0,0},
 {0,0,0,0,0},
 {1,1,0,1,1},
 {1,0,1,1,1},
 {1,0,1,1,1}};
```

island must be connected 4-directionally.
그래서 답은 8, 12는 안됨

Max Area Of Islands

Problem

{1, 1, 0, 1, 1},
{0, 1, 1, 0, 0}
{0, 0, 0, 0, 0}
{1, 1, 0, 1, 1}
{1, 0, 1, 1, 1}
{1, 0, 1, 1, 1}

1은 육지, 0 은 바다
Output :8

Example

{00, 01, 02, 03, 04},
{10, 11, 12, 13, 14},
{20, 21, 22, 23, 24},
{30, 31, 32, 33, 34},
{40, 41, 42, 43, 44},
{50, 51, 52, 53, 54},

1. Number of island
섬의 개수를 구한다.

2. Count 변수를 두고
4,2,4,8
max= Math.max(max, area);

문제7) Word Ladder

Problem

Given two words (*beginWord* and *endWord*), and a dictionary's word list, find the length of shortest transformation sequence from *beginWord* to *endWord*, such that:
Only one letter can be changed at a time.
Each transformed word must exist in the word list. Note that *beginWord* is *not* a transformed word.

Note:

Return 0 if there is no such transformation sequence.
All words have the same length.
All words contain only lowercase alphabetic characters.
You may assume no duplicates in the word list.
You may assume *beginWord* and *endWord* are non-empty and are not the same.

Example 1:

Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]

Output: 5

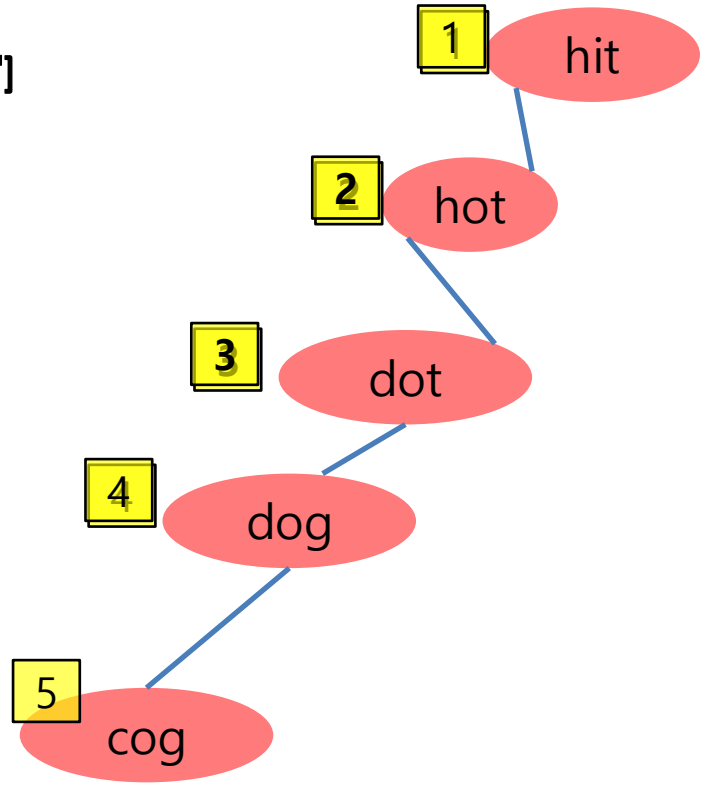
Explanation: As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog", return its length 5.

Word Ladder

Problem

Input: beginWord = "hit", endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]

Output: 5
"hit" -> "hot" -> "dot" -> "dog" -> "cog",



Example

1. hit ait, hat , hia zit, hzt, hiz	Queue size 1 hot
2. hot aot hat hoa zot hzt hoz	Queue size 3 Dot,lot,hot
3. dot ,lot, hot log	Queue size 10 Dot,lot.hot,log..
4. 10개	33개

문제 8) Word Search

Problem

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

Example:

```
board =
[
  ['A','B','C','E'],
  ['S','F','C','S'],
  ['A','D','E','E']
]
```

Given word = "ABCCED", return true.
Given word = "SEE", return true.
Given word = "ABCB", return false.

WordSearch

Problem

{'A','B', C','E'},
{'S','F', 'C','S'},
{'A','D','E','E'}
String word = "ABCCED";

Output : true;

Example

1. Dfs 문제
2. 제한 조건 찾기
   'A' 를 찾았다면

   {00, 01, 02, 03},
   {10, 11, 12, 13},
   {20, 21, 22, 23},

   좌표, visited, word

문제 9) Remove Invalid Parentheses

Problem

Remove the minimum number of invalid parentheses in order to make the input string valid. Return all possible results.

Note: The input string may contain letters other than the parentheses (and).

Example 1:

Input: "()())()"
Output: ["()()()", "(())()"]
Example 2:

Input: "(a)())()"
Output: ["(a)()()", "(a())()"]
Example 3:

Input: ")("
Output: [""]

Remove Invalid Parentheses

Problem

Input: "(a)())()"
Output: ["(a)()()", "(a())()"]

Input: "()())()"
Output: ["()()()", "(())()"]

Ex) 012345
substring(0,3): 012
substring(3) : 345

Solution

0	1	2	3	4	5	6	7
(	a	)	(		)		)	(		)

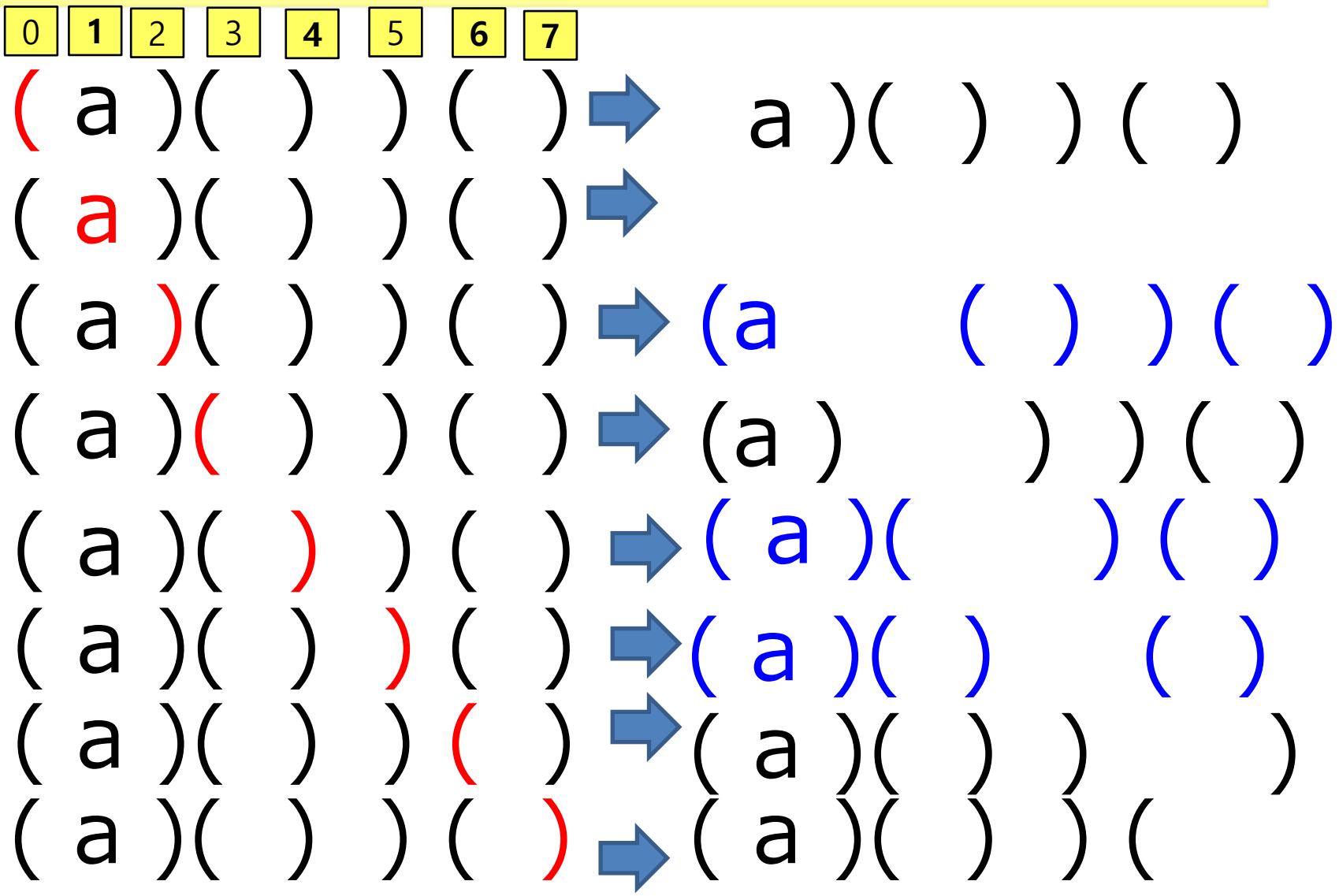
1. Queue에 저장
2. 케이스별로 valid 체크
3. Brace 체크, 쌍으로 존재
4. newStr 큐에 저장

i=0 + a) ( )) ( )

i=2 (a + ( )) ( )

i=5 (a) ( ) + ( )

Remove Invalid Parentheses



Remove Invalid Parentheses

a) ( )) ( )

a) ( )) ( )

a) ( )) ( )

a ( )) ( )

a))) ( )

(a ( )) ( )

1,2) 미로(Maze)

설명

m x n grid maze(미로)가 주어집니다.
Maze에서 각셀은 빈 공간(0으로 표시)과 벽(1로 표시)입니다. 빈 공간에 공이 있습니다.

1. 공은 **위, 아래, 왼쪽 또는 오른쪽** 으로 굴러 빈 공간을 통과 할 수 있지만 벽에 부딪 힐 때까지 굴러가는 것을 멈추지 않습니다. 공이 멈 추면 다음 방향을 선택할 수 있습니다.

2. 공의 start, destination 위치는 start = [$start_{row}$, $start_{col}$]
destination = [$destination_{row}$, $destination_{col}$]

3. 공이 목적지에서 멈출 수 있으면 true 리턴 , 그렇지 않으면 false를 리턴 합니다.

미로의 경계는 모두 벽 이라고 가정 할 수 있습니다 (예제 참조).

입출력

Input: maze = [[0,0,1,0,0],
[0,0,0,0,0],
[0,0,0,1,0],
[1,1,0,1,1],
[0,0,0,0,0]],
start = [0,4], destination = [4,4]
Output: true
Explanation: : left -> down -> left -> down -> right -> down -> right.

Input: maze = [[0,0,1,0,0],
[0,0,0,0,0],
[0,0,0,1,0],
[1,1,0,1,1],
[0,0,0,0,0]],
start = [0,4], destination = [3,2]
Output: false

문제 Format

```
public boolean hasPath(int[][] maze, int[] start,
                       int[] destination) {
    }
```

제한사항

m == maze.length
n == maze[i].length
1 <= m, n <= 100
maze[i][j] is 0 or 1.
start.length == 2
destination.length == 2
0 <= $start_{row}$, $destination_{row}$ <= m
0 <= $start_{col}$, $destination_{col}$ <= n
공과 목적지는 모두 빈 공간에 존재하며
처음에는 같은 위치에 있지 않습니다.
미로에는 **최소 2 개의 빈 공간이** 있습니다.

Solution

1. 문제분석

- 문제를 정확히 이해
- 분석 내용 정리(Devide & Conquer)

2. 규칙찾기

- 분석 내용으로 규칙을 찾는다

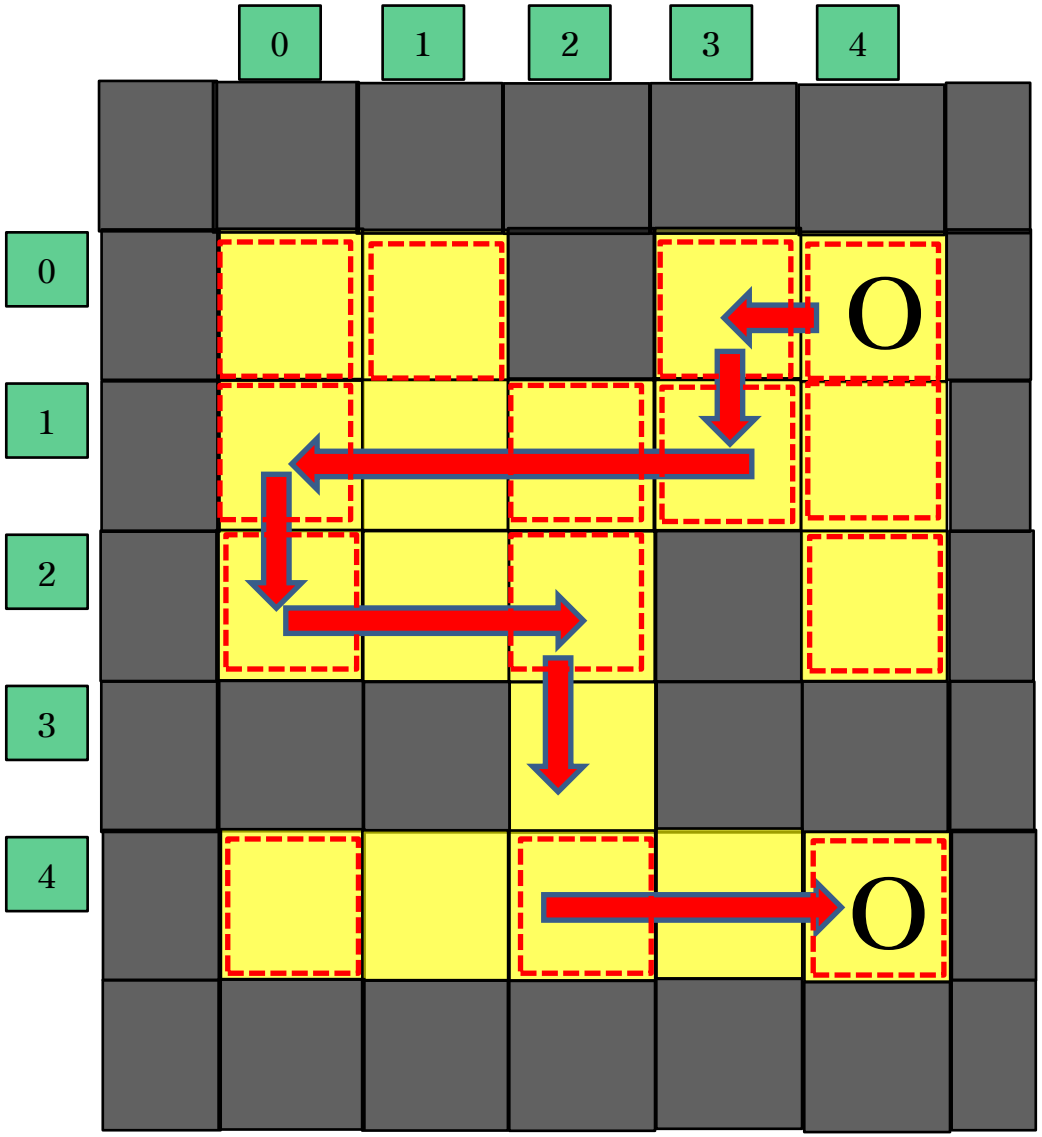
3. 코딩화

- 분석 내용으로 알맞은 구현방법 찾기

4. 알고리즘 적용

- 알고리즘 정하고 담을 그릇 정한다
(사전지식 필요)

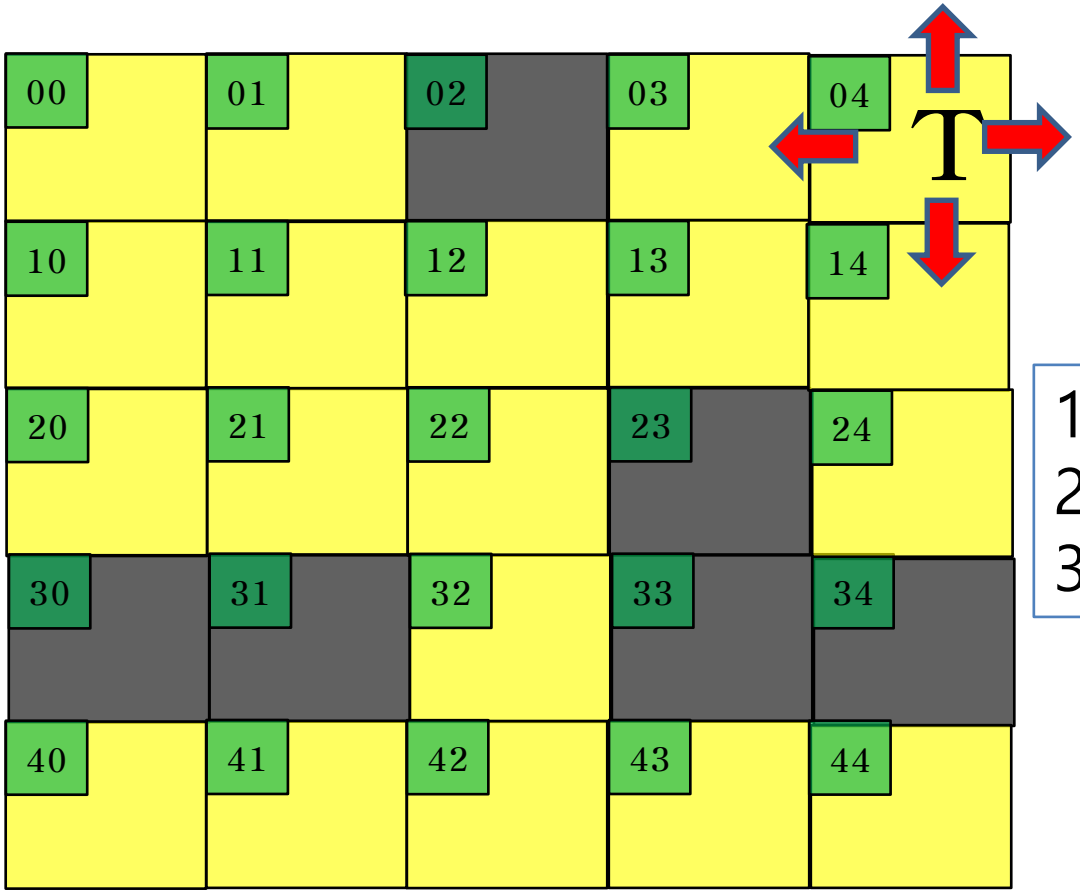
문제분석

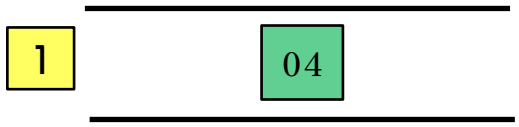


1. 0은 빈공간이고, 1은 벽
2. start = [0,4],
   destination = [4,4]
3. left -> down -> left -> down -> right -> down -> right.

문제분석

BFS(Queue)

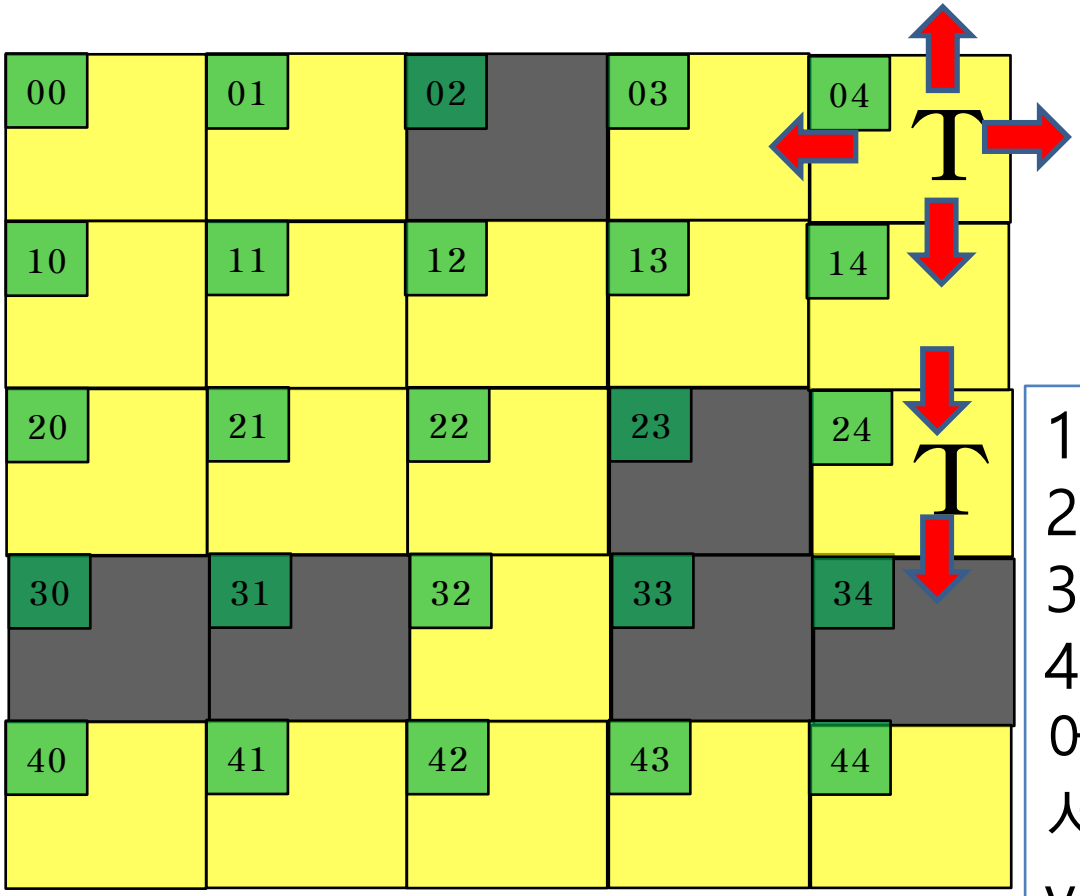


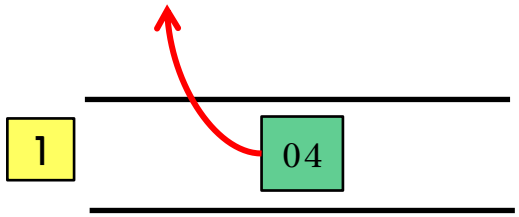


1. Queue에서 poll 한후
2. 4방향으로 체크시작
3. 공식 조건체크 부분 수행

문제분석

BFS(Queue)





1. Queue에서 poll 한후
2. 4방향으로 체크시작
3. 공식: 조건체크 부분 수행
4. (3,4)까지 도착한후

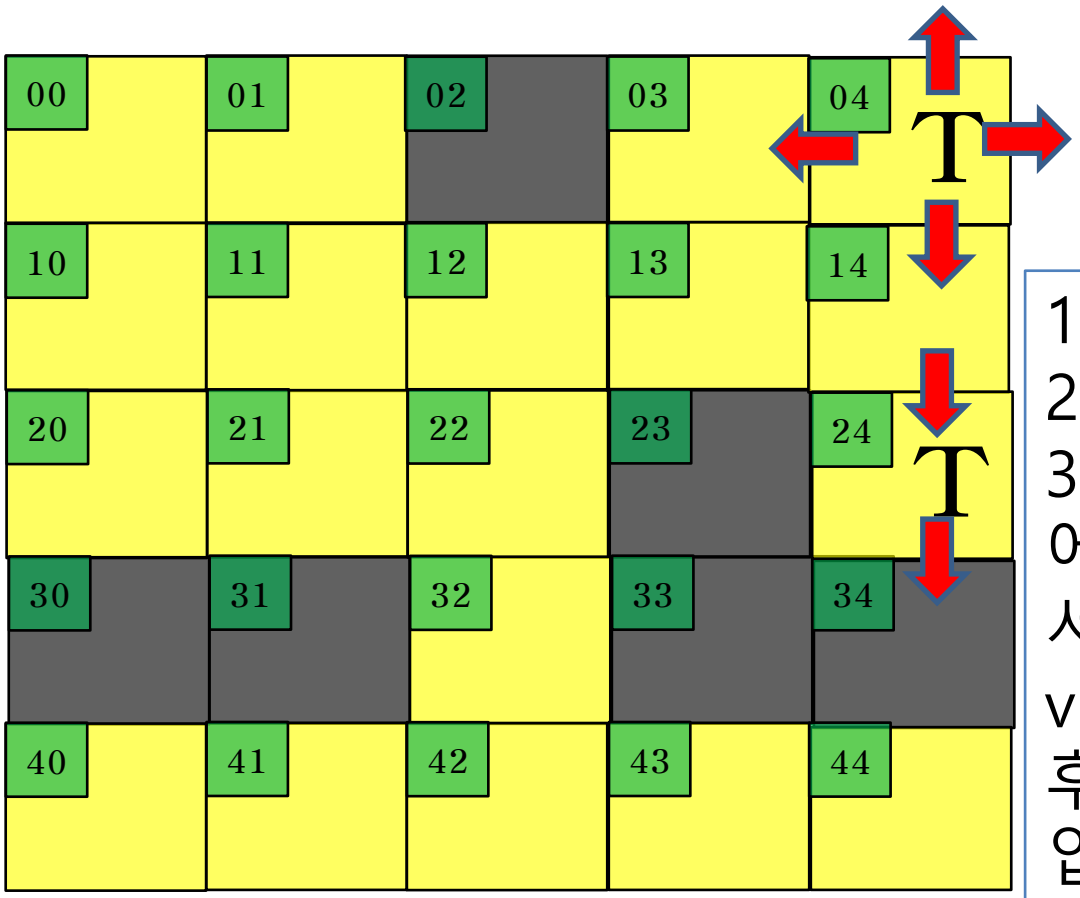
에러 체크 한다. While문에서 올린 좌표를 –로 원상복귀

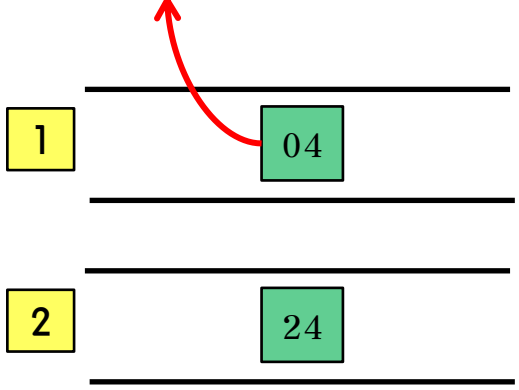
visited[x][y]==true인지 체크 후 방문한적이 없으면 true 입력

5. queue.offer(2,4)

문제분석

BFS(Queue)



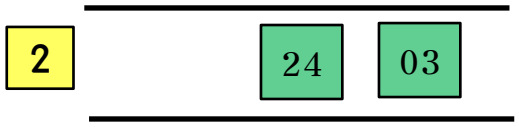


1. 4방향 (0,3)
2. 공식 조건체크 부분 수행
3. (0,2)까지 도착한후

에러 체크 한다. While문에서 올린 좌표를 –로 원상복귀

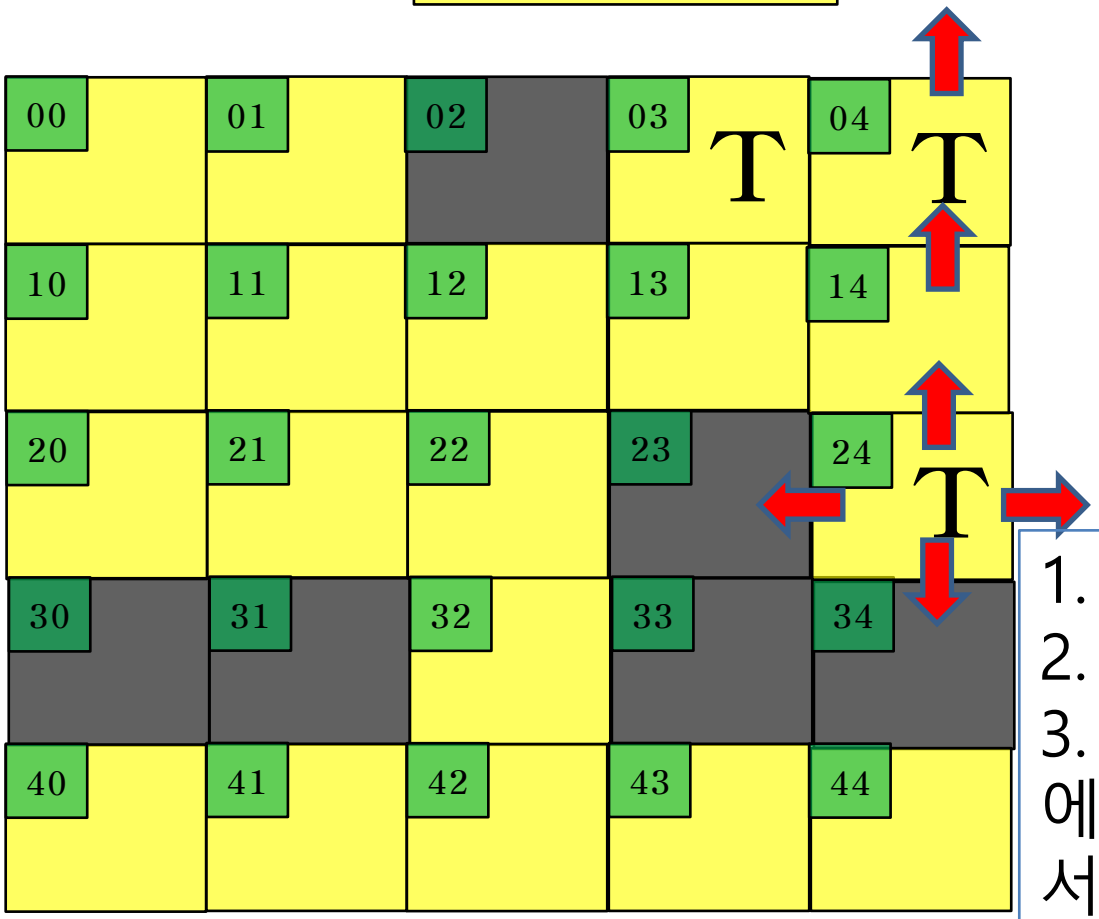
visited[x][y]==true인지 체크 후 방문한적이 없으면 true 입력

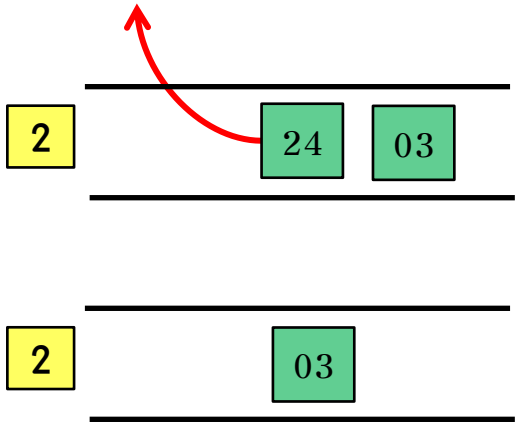
5. queue.offer(0,3)



문제분석

BFS(Queue)





1. (2,4)를 4방향으로 체크
2. 공식: 조건체크 부분 수행
3. (-1,4)까지 도착한후

에러 체크 한다. While문에서 올린 좌표를 –로 원상복귀 visited[x][y]==true인지 체크하는데 (0,4)가 true이므로 원래 좌표 (2,4)로 복귀

1. Basic_BFS

요약

1 방향 설정 & 이차원배열 사이즈

```
int m, n;
m = grid.length;
n = grid[0].length;
int[][] dirs = { { -1, 0 }, { 1, 0 }, { 0, 1 }, { 0, -1 } };
```

2 맞는 조건을 찾아내는 부분

```
for (int i = 0; i < grid.length; i++) {
        for (int j = 0; j < grid[i].length; j++) {
                if (grid[i][j] == '1') {
                count++;
                bfs(grid, i, j);
                }
        }}
```

3 Queue 생성 및 셋팅

4 조건체크해서 큐에 넣는 부분(&&)
1. 마이너스 좌표체크 2. m*n 범위체크 3. grid[x][y] 값체크(문제제시값)

```
while (!queue.isEmpty()) {
        int[] point = queue.poll();
        for (int[] dir : dirs) {
                int x1 = point[0] + dir[0];
                int y1 = point[1] + dir[1]
         if (x1 >= 0 && y1 >= 0 && x1 < grid.length && y1 < grid[0].length &&
            grid[x1][y1] == '1') {
                grid[x1][y1] = '0';
                }
    }
}
```

|| or조건

```
if( x1<0 || x1>= m || y1<0 || y1>=n ||
 visited[x1][y1] == true || grid[x1][y1] == 'X')
  continue;
```

&& and조건

```
if (x1>=0 && y1 >= 0 && x1 < m && y1 < n
&& visited[x1][y1] != true && grid[x1][y1] !=
'X') { }
```

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(M*N)
   대상 : int[][] grid
   이유 : m은 rows, n은 columns

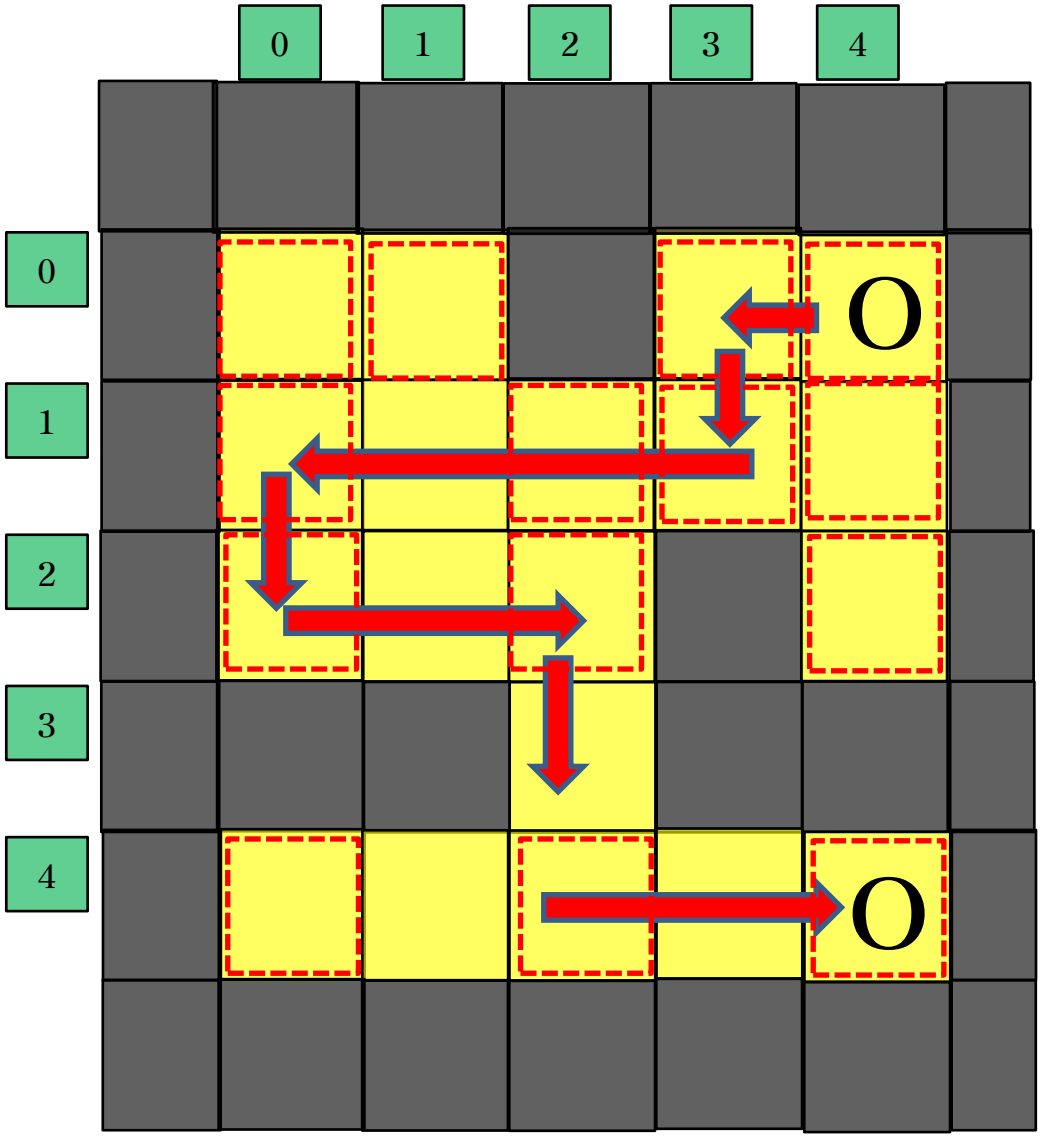
공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(M*N)
   대상 : boolean[][] visited = new boolean[m][n];
   이유 : m은 rows, n은 columns

참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)

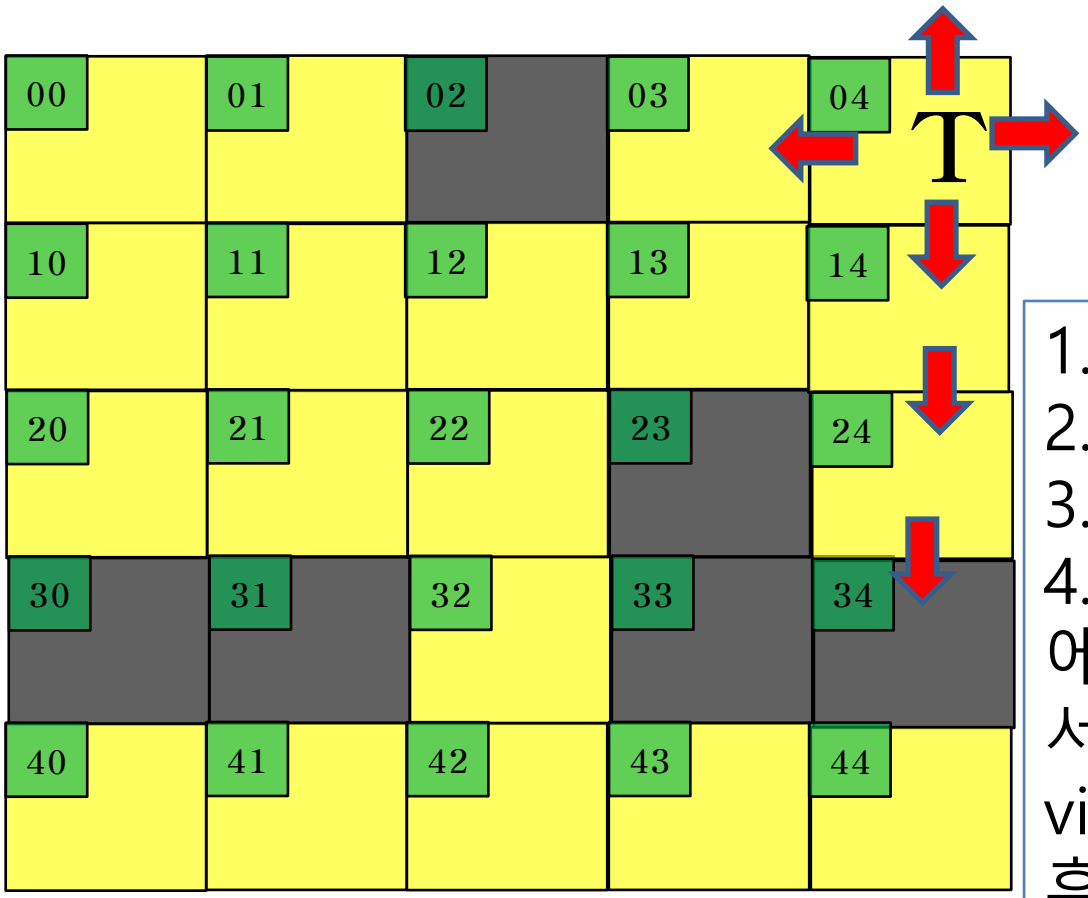
문제분석

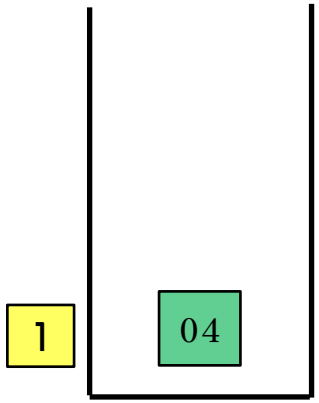


1. 0은 빈공간이고, 1은 벽
2. start = [0,4],
   destination = [4,4]
3. left -> down -> left -> down -> right -> down -> right.

문제분석

DFS(Stack)



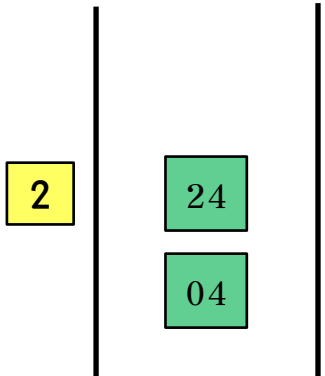


1. Stack에 (0,4) push
2. 4방향으로 체크시작
3. 공식: 조건체크 부분 수행
4. (3,4)까지 도착한후

에러 체크 한다. While문에서 올린 좌표를 –로 원상복귀

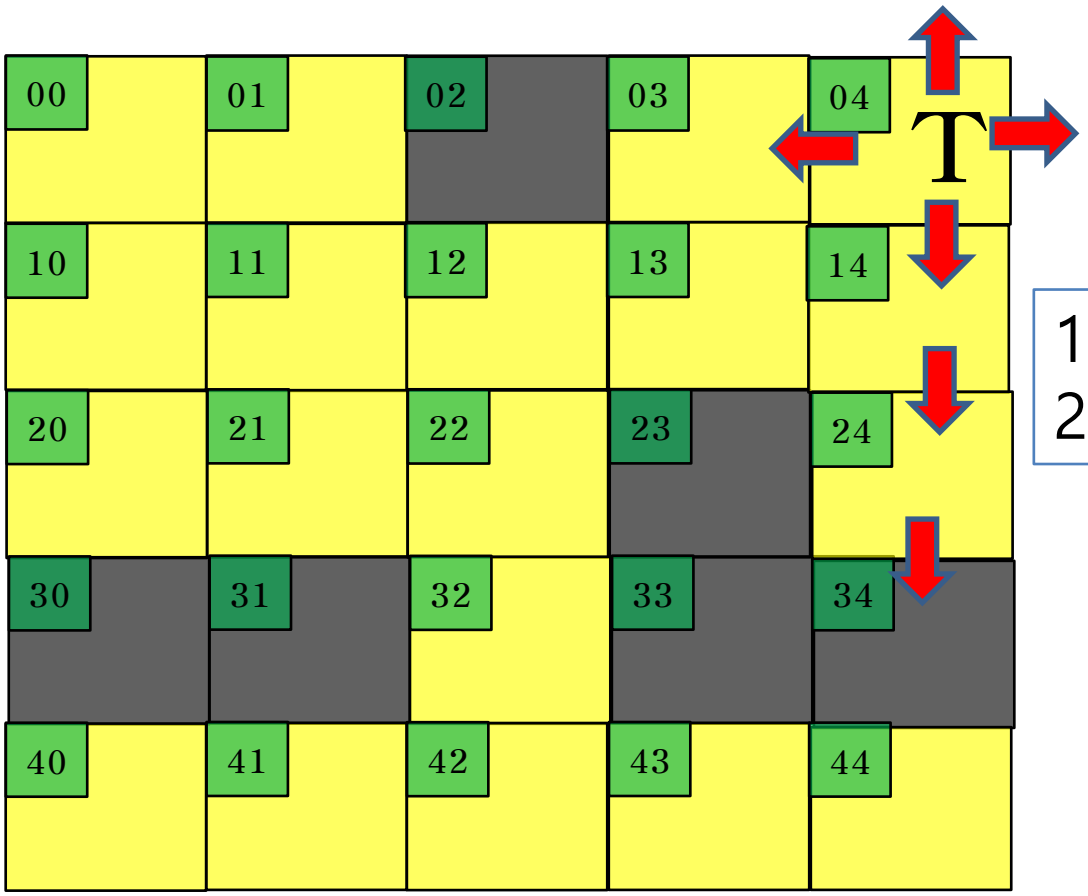
visited[x][y]==true인지 체크 후 방문한적이 없으면 true 입력

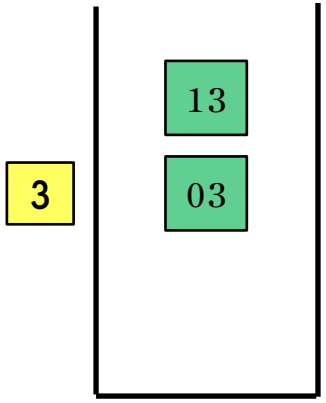
5. queue.offer(2,4)



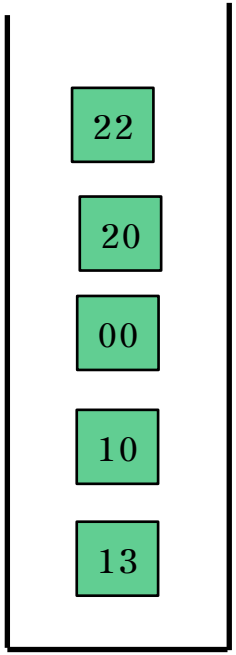
문제분석

DFS(Stack)





1. (0,3)를 꺼내서
2. 4방향으로 체크시작



2. Basic_DFS

요약

1 방향 설정 & 이차원배열 사이즈

```
int m, n;
m = grid.length;
n = grid[0].length;
int[][] dirs = { { -1, 0 }, { 1, 0 }, { 0, 1 }, { 0, -1 } };
```

2 맞는 조건을 찾아내는 부분

```
for (int i = 0; i < grid.length; i++) {
        for (int j = 0; j < grid[i].length; j++) {
                if (grid[i][j] == '1') {
                count++;
                dfs(grid, i, j);
                }
        }}
```

3 재귀를 이용한다(stack개념)

4 조건체크해서 함수 재호출(재귀)
1. 마이너스 좌표체크 2. m*n 범위체크 3. grid[x][y] 값체크(문제제시값)

```
public void dfs(char[][] grid, int i, int j) {
    if (i < 0 || i >= m || j < 0 || j >= n || grid[i][j] != '1')
            return;
    grid[i][j] = 'X';
    for(int[] dir: dirs) {
            dfs(grid, i+dir[0], j+dir[1]);
    }
}
```

|| or조건

```
if( x1<0 || x1>= m || y1<0 || y1>=n ||
 visited[x1][y1]  || grid[x1][y1] == 'X')
  continue;
```

&& and조건

```
if (x1>=0 && y1 >= 0 && x1 < m && y1 < n
&& visited[x1][y1] != true && grid[x1][y1] !=
'X') { }
```

시간복잡도/공간복잡도 계산

시간복잡도

1. 대상(Source) : 문제에서 입력받은 파라미터(array 등) (속도)
   Time Complexity : O(M*N)
   대상 : int[][] grid
   이유 : m은 rows, n은 columns

공간복잡도

2. 대상(Source) : 실제 사용되는 저장 공간을 계산(메모리 사용량)
   예) 프로그램을 실행 및 완료하는데 필요한 저장공간
   Space Complexity : O(M*N)
   대상 : boolean[][] visited = new boolean[m][n];
   이유 : m은 rows, n은 columns

참고

O(1) : 스택, 큐,Map
O(n) : for문 => 데이터를 한번씩 다 호출하니까 (제일 많음)
O(log N) : sort, prirotiyQueue, binary Search Tree, Tree
O(Klog N) : k번만큼 소팅하는 경우
O(n^2) : 이중for문
O(m*n) : 이중for문인데, n이 다른경우bfs,dfs 류 (예 n=100 인데 m=5인경우)