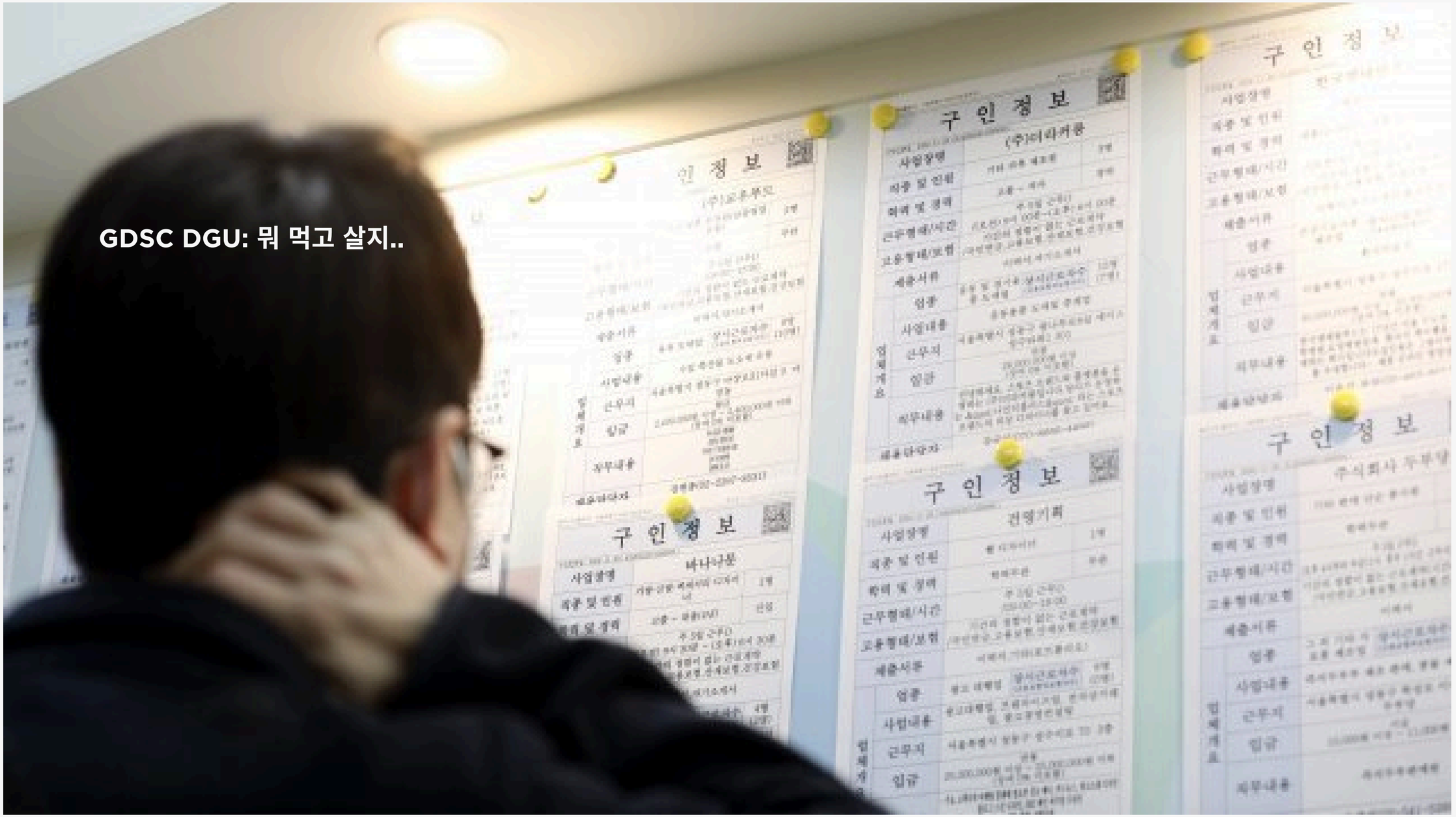
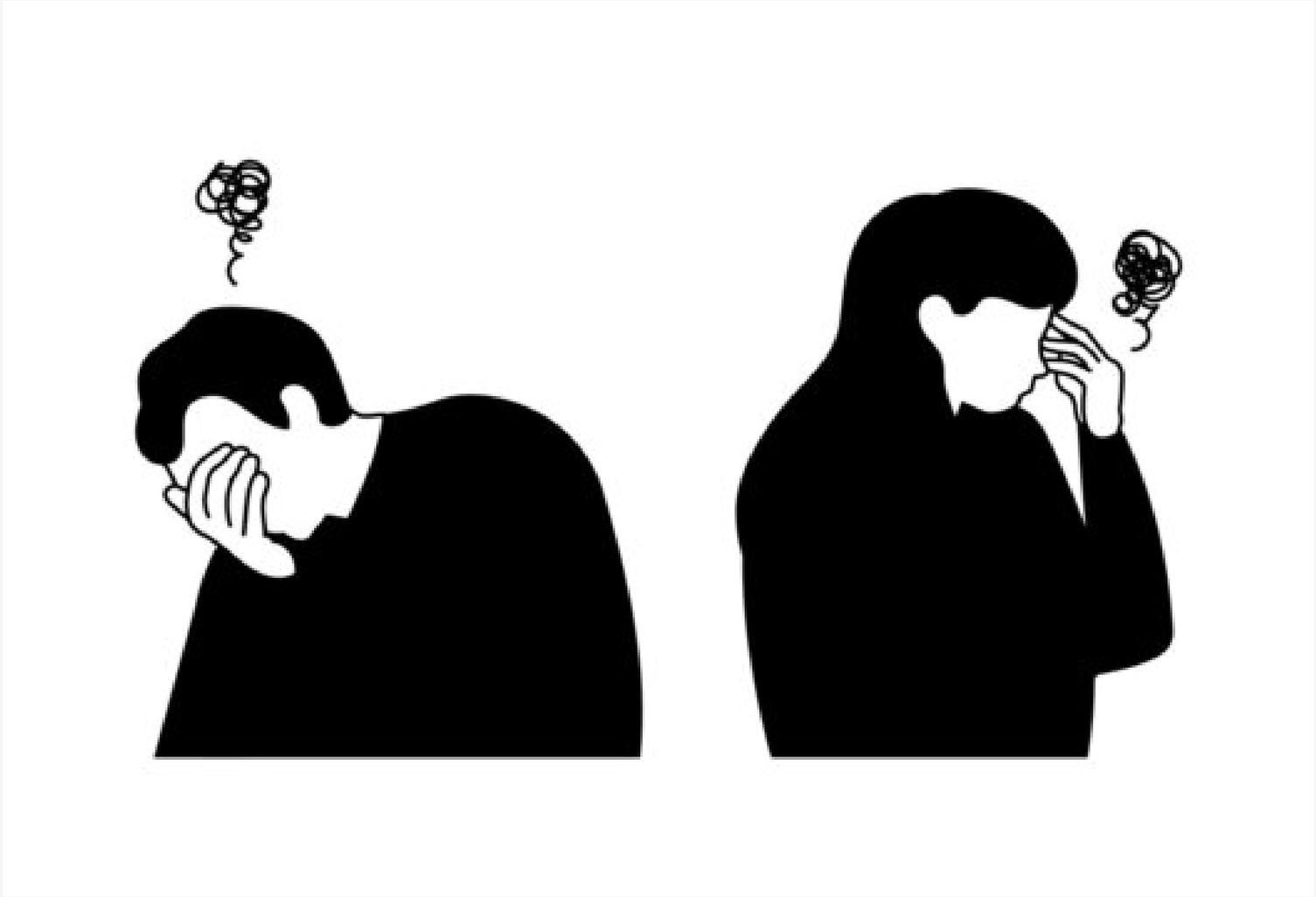
ALGORITHM

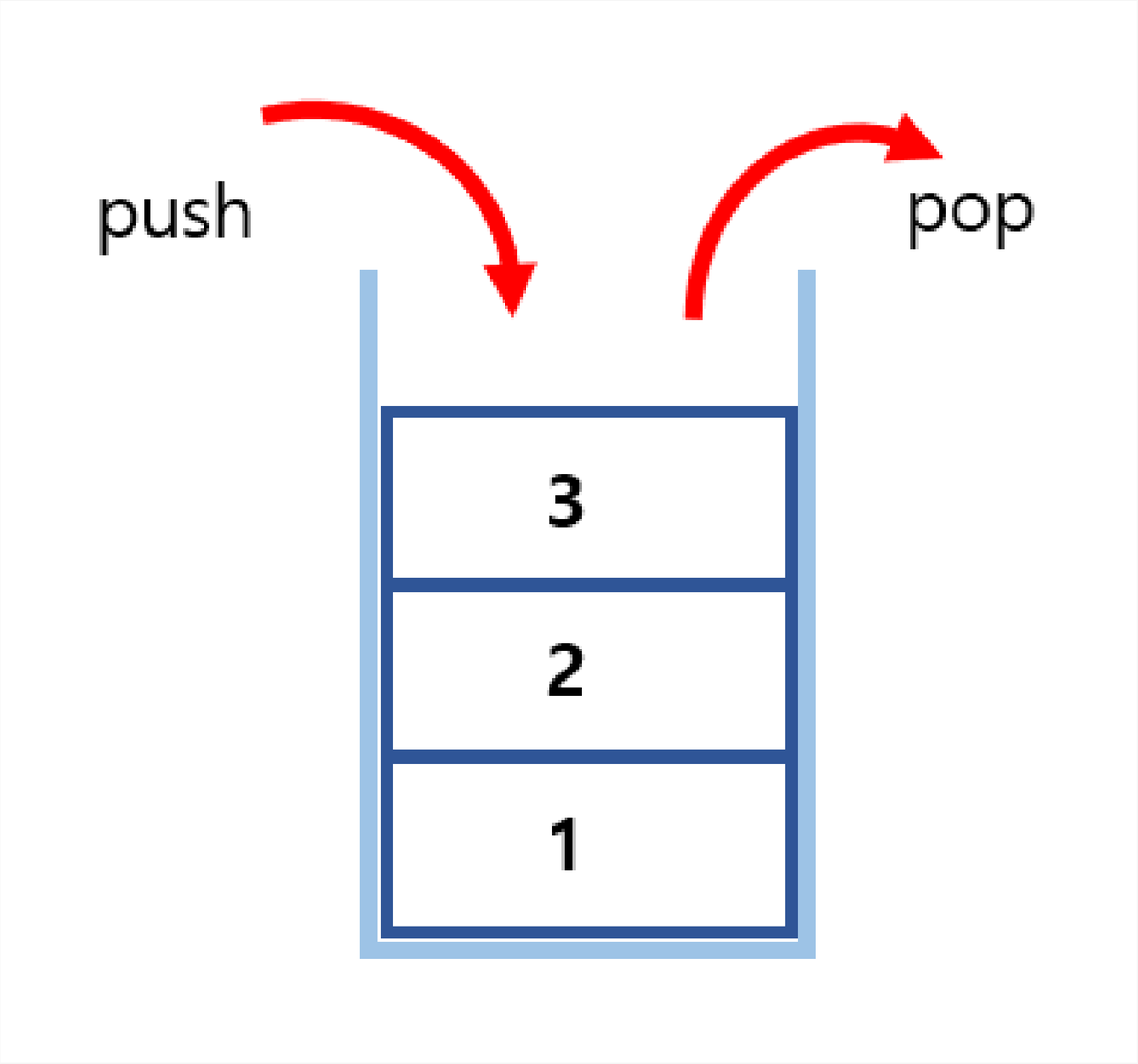

GDSC DGU: 뭐 먹고 살지..
구 인 정 보
구 인 정 보
구 인 정 보
구 인 정 보
구 인 정 보

I DONT KNOW..



PROBLEM EXAMPLE

STACK



괄호 성공 다국어

☆ 한국어

실버 IV

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	128 MB	209084	98638	70789	45.966%

문제

괄호 문자열(Parenthesis String, PS)은 두 개의 괄호 기호인 '(' 와 ')' 만으로 구성되어 있는 문자열이다. 그 중에서 괄호의 모양이 바르게 구성된 문자열을 올바른 괄호 문자열(Valid PS, VPS)이라고 부른다. 한 쌍의 괄호 기호로 된 "( )" 문자열은 기본 VPS 이라고 부른다. 만일 x 가 VPS 라면 이것을 하나의 괄호에 넣은 새로운 문자열 "(x)"도 VPS 가 된다. 그리고 두 VPS x 와 y를 접합(concatenation)시킨 새로운 문자열 xy도 VPS 가 된다. 예를 들어 "(())()"와 "((()))" 는 VPS 이지만 "(()(", "(())())" , 그리고 "(()" 는 모두 VPS 가 아닌 문자열이다.

여러분은 입력으로 주어진 괄호 문자열이 VPS 인지 아닌지를 판단해서 그 결과를 YES 와 NO 로 나타내어야 한다.

```
3
( (
) )
( ) ) ( ( )
```

```
NO
NO
NO
```

문제의 중요한 포인트

1. 올바른 괄호의 짝을 찾는다.
2. 괄호의 짝은 가장 가까운 ‘(’와 ‘)’이다.

(( ) ((( )) ( ) (

(= PUSH　　) = POP

주식가격 제출 내역

문제 설명

초 단위로 기록된 주식가격이 담긴 배열 prices가 매개변수로 주어질 때, 가격이 떨어지지 않은 기간은 몇 초인지를 return 하도록 solution 함수를 완성하세요.

제한사항

- prices의 각 가격은 1 이상 10,000 이하인 자연수입니다.
- prices의 길이는 2 이상 100,000 이하입니다.

입출력 예

prices	return
[1, 2, 3, 2, 3]	[4, 3, 1, 1, 0]

100000 길이의 배열? → **N^2 이면 100초**

문제의 중요한 포인트

N의 시간복잡도로 해결

스택에서 N의 시간복잡도?

인접한 값만 비교

+ 기본 결과 값 생성

solution.py

```python
def solution(prices):
    answer = [i for i in range(len(prices)-1, -1, -1)]
    stack = list()
    for i in range(len(prices)):
        while stack and prices[stack[-1]] > prices[i]:
            j = stack.pop()
            answer[j] = i - j
        stack.append(i)
    return answer
```

입출력 예

prices	return
[1, 2, 3, 2, 3]	[4, 3, 1, 1, 0]

QUEUE

DFS VS BFS

DFS

1	2	3
10	11	4
9	12	5
8	7	6

BFS

1	2	3
2	3	4
3	4	5
4	5	6

최단거리 = BFS

탈출 성공 다국어

☆ 한국어

골드 IV

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	128 MB	54614	19278	13171	33.355%

문제

사악한 암흑의 군주 이민혁은 드디어 마법 구슬을 손에 넣었고, 그 능력을 실험해보기 위해 근처의 티떱숲에 홍수를 일으키려고 한다. 이 숲에는 고슴도치가 한 마리 살고 있다. 고슴도치는 제일 친한 친구인 비버의 굴로 가능한 빨리 도망가 홍수를 피하려고 한다.

티떱숲의 지도는 R행 C열로 이루어져 있다. 비어있는 곳은 '.'로 표시되어 있고, 물이 차있는 지역은 '*', 돌은 'X'로 표시되어 있다. 비버의 굴은 'D'로, 고슴도치의 위치는 'S'로 나타내어져 있다.

매 분마다 고슴도치는 현재 있는 칸과 인접한 네 칸 중 하나로 이동할 수 있다. (위, 아래, 오른쪽, 왼쪽) 물도 매 분마다 비어있는 칸으로 확장한다. 물이 있는 칸과 인접해있는 비어있는 칸(적어도 한 변을 공유)은 물이 차게 된다. 물과 고슴도치는 돌을 통과할 수 없다. 또, 고슴도치는 물로 차있는 구역으로 이동할 수 없고, 물도 비버의 소굴로 이동할 수 없다.

티떱숲의 지도가 주어졌을 때, 고슴도치가 안전하게 비버의 굴로 이동하기 위해 필요한 최소 시간을 구하는 프로그램을 작성하시오.

고슴도치는 물이 찰 예정인 칸으로 이동할 수 없다. 즉, 다음 시간에 물이 찰 예정인 칸으로 고슴도치는 이동할 수 없다. 이동할 수 있으면 고슴도치가 물에 빠지기 때문이다.

예제 입력 1 복사

```
3 3
D.*
...
.S.
```

예제 출력 1 복사

```
3
```

두 물체가 이동하는 경우, 방해 역할을 먼저

벽 부수고 이동하기 성공

☆

골드 III

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	192 MB	143589	37736	23596	23.345%

문제

N×M의 행렬로 표현되는 맵이 있다. 맵에서 0은 이동할 수 있는 곳을 나타내고, 1은 이동할 수 없는 벽이 있는 곳을 나타낸다. 당신은 (1, 1)에서 (N, M)의 위치까지 이동하려 하는데, 이때 최단 경로로 이동하려 한다. 최단경로는 맵에서 가장 적은 개수의 칸을 지나는 경로를 말하는데, 이때 시작하는 칸과 끝나는 칸도 포함해서 센다.

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

한 칸에서 이동할 수 있는 칸은 상하좌우로 인접한 칸이다.

맵이 주어졌을 때, 최단 경로를 구해 내는 프로그램을 작성하시오.

예제 입력 1 복사

```
6 4
0100
1110
1000
0000
0111
0000
```

예제 출력 1 복사

```
15
```

문제의 중요한 포인트

벽을 한번 무시할 수 있다

하나의 벽 무시로 답이 매우 달라짐

하나만 부셔야 하기 때문에

어디서 부셔야 하는지 최적의 경우 판단

해결방법

1. Y, X, WALL 3가지 상태를 큐에 전달
   - C++: 구조체 OR PAIR 사용
   - PYTHON: LIST 혹은 튜플 사용
2. 3차원 방문 배열로 이동 제어
   - VISITED[Y][X][2]
     - VISITED[Y][X][0]: (Y, X)에서 벽을 0번 부숨
     - VISITED[Y][X][1]: (Y, X)에서 벽을 1번 부숨

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;
int dir[4][2] = { {0,1},{1,0},{0,-1},{-1,0} };
int r,c;
int visited[1000][1000][2];
int bfs(int row, int col, vector<string> &graph ) {
    queue<pair<pair<int, int>, int>> q;
    q.push({ {0,0},1 });
    visited[0][0][1] = 1;

    while (!q.empty()) {
        // 큐에 존재하는 가장 오래된 녀석 꺼냄
        int current_r = q.front().first.first;
        int current_c = q.front().first.second;
        int block = q.front().second;
        q.pop();

        if (current_r == r - 1 && current_c == c - 1) { //도착지에 도달하면 return
            return visited[current_r][current_c][block];
        }

        // 상, 하, 좌, 우 4방향으로 탐색
        for (int i = 0; i < 4; i++) {
            int next_r = current_r + dir[i][0];
            int next_c = current_c + dir[i][1];
            if (next_r >= 0 && next_r < r&& next_c >= 0 && next_c < c) {

                //다음 칸이 벽이고 뚫을 수 있을 때 -> 기존에 벽을 부순 적이 없음
                if (graph[next_r][next_c] == '1' && block) {
                    q.push({ {next_r,next_c} ,0 });
                    visited[next_r][next_c][block - 1] = visited[current_r][current_c][block] + 1;
                }

                //다음 칸이 0이고 방문하지 않았을 때 -> 벽과 관계없이 갈 수 있음
                else if (graph[next_r][next_c] == '0' && visited[next_r][next_c][block] == 0) {
                    q.push({ {next_r,next_c},block });
                    visited[next_r][next_c][block] = visited[current_r][current_c][block] + 1;
                }
            }
        }

    }
    return -1;

}
```

THANK YOU.