백준 2933번 - 미네랄

시간제한	메모리 제한	제출	정답	맞은 사람	정답 비율
1초	128MB	8504	2296	1463	24.750%

문제

창영과 상근은 한 동굴을 놓고 소유권을 주장하고 있다. 두 사람은 막대기를 서로에게 던지는 방법을 이용해 누구의 소유인지를 결정하기로 했다. 싸움은 동굴에서 벌어진다. 동굴에는 미네랄이 저장되어 있으며, 던진 막대기가 미네랄을 파괴할 수도 있다.

동굴은 R행 C열로 나타낼 수 있으며, R×C칸으로 이루어져 있다. 각 칸은 비어있거나 미네랄을 포함하고 있으며, 네 방향 중 하나로 인접한 미네랄이 포함된 두 칸은 같은 클러스터이다.

창영은 동굴의 왼쪽에 서있고, 상근은 오른쪽에 서있다. 두 사람은 턴을 번갈아가며 막대기를 던진다. 막대를 던지기 전에 던질 높이를 정해야 한다. 막대는 땅과 수평을 이루며 날아간다.

막대가 날아가다가 미네랄을 만나면, 그 칸에 있는 미네랄은 모두 파괴되고 막대는 그 자리에서 이동을 멈춘다.

미네랄이 파괴된 이후에 남은 클러스터가 분리될 수도 있다. 새롭게 생성된 클러스터가 떠 있는 경우에는 중력에 의해서 바닥으로 떨어지게 된다. 떨어지는 동안 클러스터의 모양은 변하지 않는다. 클러스터는 다른 클러스터나 땅을 만나기 전까지 계속해서 떨어진다. 클러스터는 다른 클러스터 위에 떨어질 수 있고, 그 이후에는 합쳐지게 된다.

동굴에 있는 미네랄의 모양과 두 사람이 던진 막대의 높이가 주어진다. 모든 막대를 던지고 난 이후에 미네랄 모양을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 동굴의 크기 R과 C가 주어진다. (1 ≤ R,C ≤ 100)

다음 R개 줄에는 C개의 문자가 주어지며, '.'는 빈 칸, 'x'는 미네랄을 나타낸다.

다음 줄에는 막대를 던진 횟수 N이 주어진다. (1 ≤ N ≤ 100)

마지막 줄에는 막대를 던진 높이가 주어지며, 공백으로 구분되어져 있다. 모든 높이는 1과 R사이이며, 높이 1은 행렬의 가장 바닥, R은 가장 위를 의미한다. 첫 번째 막대는 왼쪽에서 오른쪽으로 던졌으며, 두 번째는 오른쪽에서 왼쪽으로, 이와 같은 식으로 번갈아 가며 던진다.

공중에 떠 있는 미네랄 클러스터는 없으며, 두 개 또는 그 이상의 클러스터가 동시에 떨어지는 경우도 없다. 클러스터가 떨어질 때, 그 클러스터 각 열의 맨 아래 부분 중 하나가 바닥 또는 미네랄 위로 떨어지는 입력만 주어진다.

출력

입력 형식과 같은 형식으로 미네랄 모양을 출력한다.

예제 입력1

```
5 6
......
..xx..
..x...
..xx..
.xxxx.
1
3
```

예제 출력1

```
......
......
..xx..
..xx..
.xxxx.
```

(이하 생략)

출처

출처

알고리즘 분류

- 구현
- 그래프 이론
- 그래프 탐색
- 너비 우선 탐색
- 시뮬레이션
- 깊이 우선 탐색

접근 방법

조금(?) 복잡한 구현 문제이다. 문제의 요구사항에 맞게 $N-k$ 행의 가장 왼쪽 또는 오른쪽 원소를 하나 삭제하면 되는데, 까다로웠던 부분은 제거된 미네랄로 인하여 클러스터가 분해될 수 있다는 부분이었다.

먼저, 미네랄이 제거되어 클러스터가 분해되는 상황을 생각해보자.

연결되어 있는 미네랄들은 하나의 클러스터를 이룬다. 이것을 우리는 `area`로 칭하기로 한다.

미네랄이 제거될 때마다, 특정 미네랄 위치로부터 시작하여 연결된 모든 미네랄들의 `area id`을 동일하게 만드는 것으로 클러스터들을 구분할 수 있다. (그래프 탐색으로 인접한 미네랄들의 id를 동일하게 만들 수 있다)

우리는 해당 클러스터에 포함된 모든 미네랄의 위치들을 `mp`라는 탐색 트리에 저장한 후, `mp`에 저장된 id 값을 통하여 모든 클러스터들에 대하여 해당 클러스터에 포함된 모든 미네랄의 위치를 불필요한 탐색 없이 가져올 수 있다.

다음으로 고려해야할 부분은 클러스터가 공중에 떠 있는지에 대한 판단이다. 관찰을 통하여 우리는 다음과 같은 사실을 알 수 있다.

> 클러스터가 공중에 떠 있는 경우는 클러스터에 포함된 모든 미네랄의 위치를 1번 이상 평행이동을 시킬 수 있어야한다. <이때 평행이동은 row기준 + 방향이다.>

평행이동이 가능한 조건은 이동하고자 하는 곳이 유효하며(즉, board 밖을 벗어나지 않으며) 다른 클러스터와 겹치지 않는 위치여야한다.

해당 로직은 50 ~ 69번 줄에 구현이 되어 있다.

소스코드

```
#define FASTIO cin.tie(0)->sync_with_stdio(false), cout.tie(0)
////////////////////////////////////////////////////////////////
#include <bits/stdc++.h>
using namespace std;
int R, C, area[105][105];
map<int, vector<pair<int,int>>> mp;
const int dir[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
char board[105][105];
bool destroy(int row, int opt){
    int i = 0;
    if(opt == 0) {
        i = 0;
        while (i < C && board[row][i] == '.') i++;
        if (i == C)
            return false;
    }else{
        i = C-1;
        while(i >= 0 && board[row][i] == '.') i--;
        if(i == -1)
            return false;
    }
    board[row][i] = '.';
    return true;
}
void DFS(int r, int c, const int id){
    for(int d=0; d<4; d++){
        int nr = r + dir[d][0], nc = c + dir[d][1];
```

```
        if(nr < 0 || nc < 0 || nr >= R || nc >= C) continue;
        if(area[nr][nc] != -1 || board[nr][nc] != 'x') continue;
        area[nr][nc] = id;
        mp[id].push_back({nr, nc});
        DFS(nr, nc, id);
    }
}
void adjust(){
    memset(area, -1, sizeof(area));
    mp.clear();
    int id = 0;
    for(int i=0; i<R; i++){
        for(int j=0; j<C; j++){
            if(board[i][j] == 'x' && area[i][j] == -1){
                area[i][j] = id;
                mp[id].push_back({i, j});
                DFS(i, j, id);
                id += 1;
            }
        }
    }
    // 덩어리 상태의 block들을 y축 평행이동
    for(auto &p : mp){
        int mpID = p.first;
        vector<pair<int,int>> &v = p.second;
        sort(v.begin(), v.end(), greater<pair<int,int>>());
        int k = 0;
        bool flag = true; // 평행이동을 더 진행해도 되는가?
        while(flag){
            k++;
            for(const auto& [y, x] : v) {
                if(y + k >= R) flag = false;
                if(area[y+k][x] != -1 && area[y+k][x] != area[y][x]) flag =
false;
            }
        }
        k--;
        for(auto& [y,x] : v) {
            board[y][x] = '.', board[y+k][x] = 'x';
            area[y][x] = -1, area[y+k][x] = mpID;
            y += k;
        }
    }

}
int main(void){
    FASTIO;
    cin >> R >> C;
    for(int i=0; i<R; i++){
```

```
        for(int j=0; j<C; j++)
            cin >> board[i][j];
    }

    int n;
    cin >> n;
    vector<int> target(n);
    for(auto &item : target)
        cin >> item;

    for(int i=0; i<target.size(); i++){
        int &tar = target[i];
        bool isDestroyed = false;
        isDestroyed = destroy(R-tar, i % 2);
        if(isDestroyed)
            adjust();
    }

    for(int i=0; i<R; i++){
        for(int j=0; j<C; j++)
            cout << board[i][j];
        cout << '\n';
    }

    return 0;
}

```