1861. Rotating the Box

You are given an `m x n` matrix of characters `box` representing a side-view of a box. Each cell of the box is one of the following:

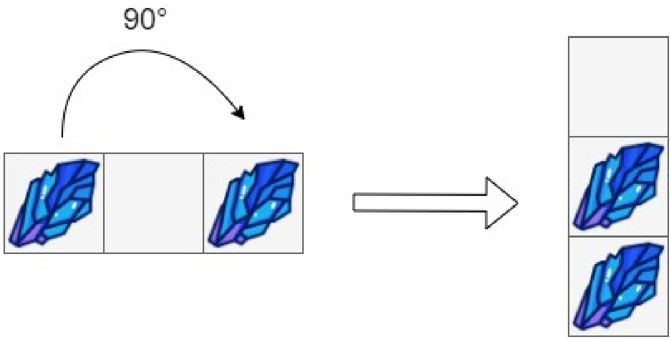
- A stone `'#'`
- A stationary obstacle `'*'`
- Empty `'.'`

The box is rotated **90 degrees clockwise**, causing some of the stones to fall due to gravity. Each stone falls down until it lands on an obstacle, another stone, or the bottom of the box. Gravity **does not** affect the obstacles' positions, and the inertia from the box's rotation **does not** affect the stones' horizontal positions.

It is **guaranteed** that each stone in `box` rests on an obstacle, another stone, or the bottom of the box.

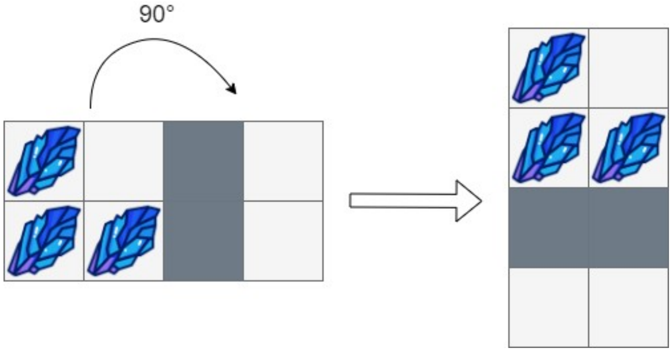
Return *an* `n x m` *matrix representing the box after the rotation described above*.

Example 1:



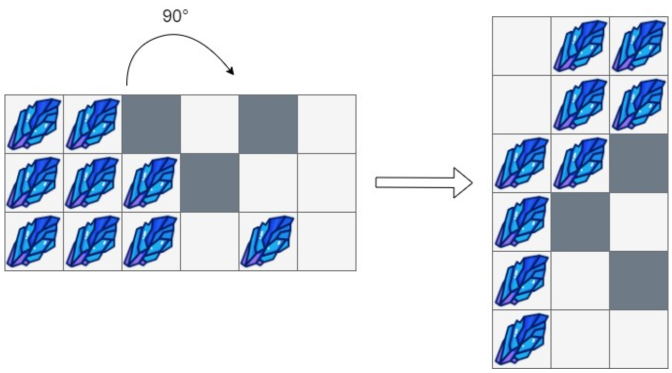
```
Input: box = [["#",".","#"]]
Output: [["."],
         ["#"],
         ["#"]]
```

Example 2:



```
Input: box = [["#",".","*","."],
              ["#","#","*","."]]
Output: [["#","."],
         ["#","#"],
         ["*","*"],
         [".","."]]
```

Example 3:



```
Input: box =
[["#","#","*",".","*","."],
["#","#","#","*",".","."],
["#","#","#",".","#","."]]
Output: [[".","#","#"],
         [".","#","#"],
         ["#","#","*"],
         ["#","*","."],
         ["#",".","*"],
         ["#",".","."]]
```

Constraints:

- `m == box.length`
- `n == box[i].length`
- `1 <= m, n <= 500`
- `box[i][j]` is either `'#'`, `'*'`, or `'.'`.

1861. Rotating the Box

```
/*
    Brute Force: Two pointers to identify the sequence of stones that will fall
    Double passes:gravity effect+Rotation
    Time complexity:  $O\left(m.n^2\right)$
    Space complexity: O(n.m)
*/
typedef std::vector<char> vc;
typedef std::vector<vc> vvc;
class Solution {
    public:
        vvc rotateTheBox(vvc& box){
            int m=box.size();
            int n=box[0].size();

            // Gravity
            for(auto& cur_row: box){
                // Two pointers to identify the sequence of rocks that will fall
                int start=-1,end=-1;
                for(int i=0;i<n;++i){
                    // Stone: Mark the start of the sequence
                    if(cur_row[i]=='#' && start==-1) start=i;

                    // Empty cell: Mark the end of the sequence
                    if(cur_row[i]=='.' && end==-1) end=i;

                    // Obstacle:
                    if(cur_row[i]=='*') start=-1,end=-1;

                    // Not valid sequence: ..###
                    if(start!=-1 && end!=-1 && start>end) end=-1;
                    // Valid sequence
                    if(start!=-1 && end!=-1 && end>start){
                        // Empty cell will be on the top
                        cur_row[start]='.';

                        // All stone will go right (fall)
                        for(int j=start+1;j<=end;++j) cur_row[j]='#';

                        // Mark the start of the new sequence
                        start++;

                        // Don't know its end
                        end=-1;
                    }
                }
            }

            // Rotation
            vvc rot(n,vc(m));

            for(int i=0;i<n;++i){
                for(int j=m-1;j>=0;--j){
                    rot[i][m-j-1]=box[j][i];
                }
            }

            return rot;
        }
};
```

1861. Rotating the Box

```
/*
    One pointer: Mark the lowest empty cell
    Double passes:gravity effect+Rotation
    Time complexity: O(m.n)
    Space complexity: O(n.m)
*/
typedef std::vector<char> vc;
typedef std::vector<vc> vvc;
class Solution {
    public:
        vvc rotateTheBox(vvc& box){
            int m=box.size();
            int n=box[0].size();
            // Gravity
            for(int i=0;i<m;++i){
                // Lowest empty cell: by default last cell
                int lowest_empty_cell_idx=n-1;
                // Process each cell in row `j` from left to right (bottom to top)
                for(int j=n-1;j>=0;--j){
                    // Stone: let it fall to the lowest empty cell
                    if(box[i][j]=='#'){
                        box[i][j]='.';
                        box[i][lowest_empty_cell_idx]='#';
                        lowest_empty_cell_idx--;
                    }
                    // Obstacle: reset `lowest_empty_cell_idx` to the cell
                    // directly on its left (above it)
                    if(box[i][j]=='*') lowest_empty_cell_idx=j-1;
                }
            }
            // Rotation
            vvc rot(n,vc(m));
            for(int i=0;i<n;++i){
                for(int j=m-1;j>=0;--j){
                    rot[i][m-j-1]=box[j][i];
                }
            }
            return rot;}};
```

1861. Rotating the Box

```
  /*
    One pointer: Mark the lowest empty cell
    Single pass:gravity effect+Rotation
    Time complexity: O(m.n)
    Space complexity: O(n.m)
*/
typedef std::vector<char> vc;
typedef std::vector<vc> vvc;
class Solution {
    public:
        vvc rotateTheBox(vvc& box){
            int m=box.size();
            int n=box[0].size();

            // Gravity and rotation
            vvc rot(n,vc(m,'.'));
            for(int i=0;i<m;++i){
                int lowest_empty_cell_idx=n-1;
                for(int j=n-1;j>=0;--j){
                    if(box[i][j]=='#'){
                        rot[lowest_empty_cell_idx][m-i-1]='#';
                        lowest_empty_cell_idx--;
                    }
                    if(box[i][j]=='*'){
                        rot[j][m-i-1]='*';
                        lowest_empty_cell_idx=j-1;
                    }
                }
            }

            return rot;
        }
};
```