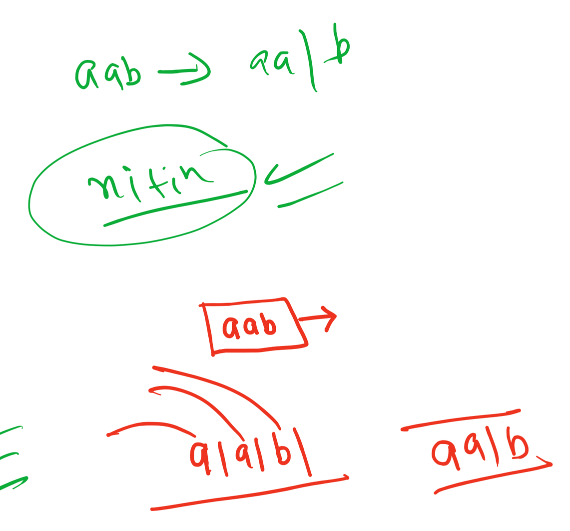
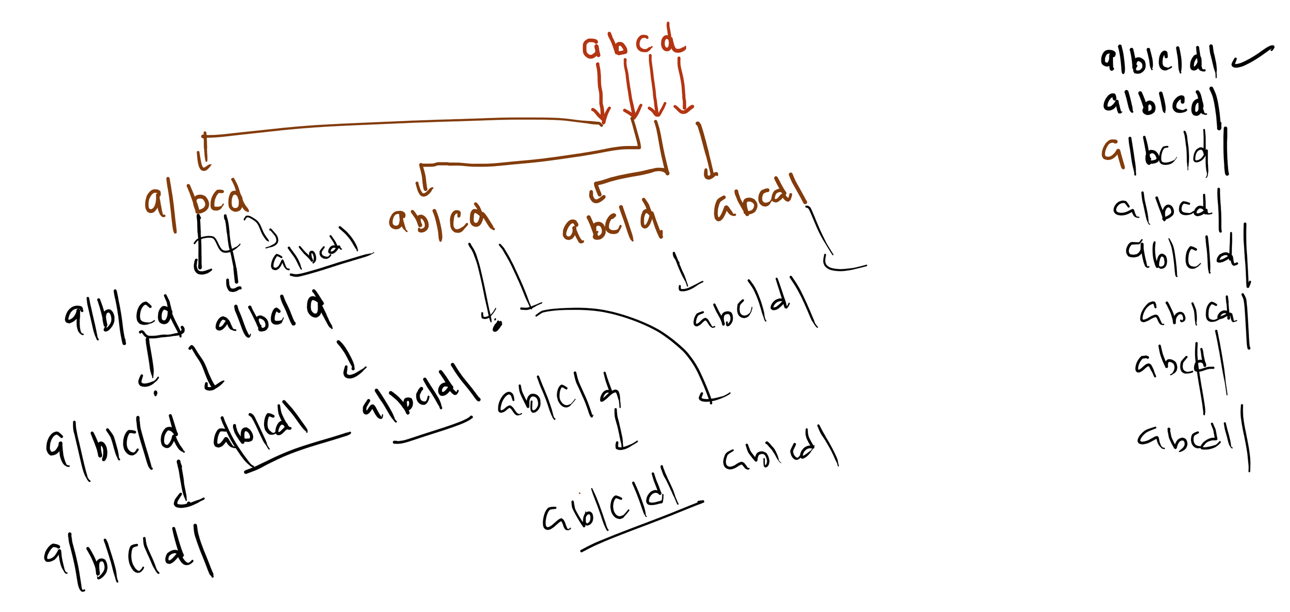
Given a string `s`, partition `s` such that every substring of the partition is a **palindrome**.
Return *all possible palindrome partitioning of* `s`.

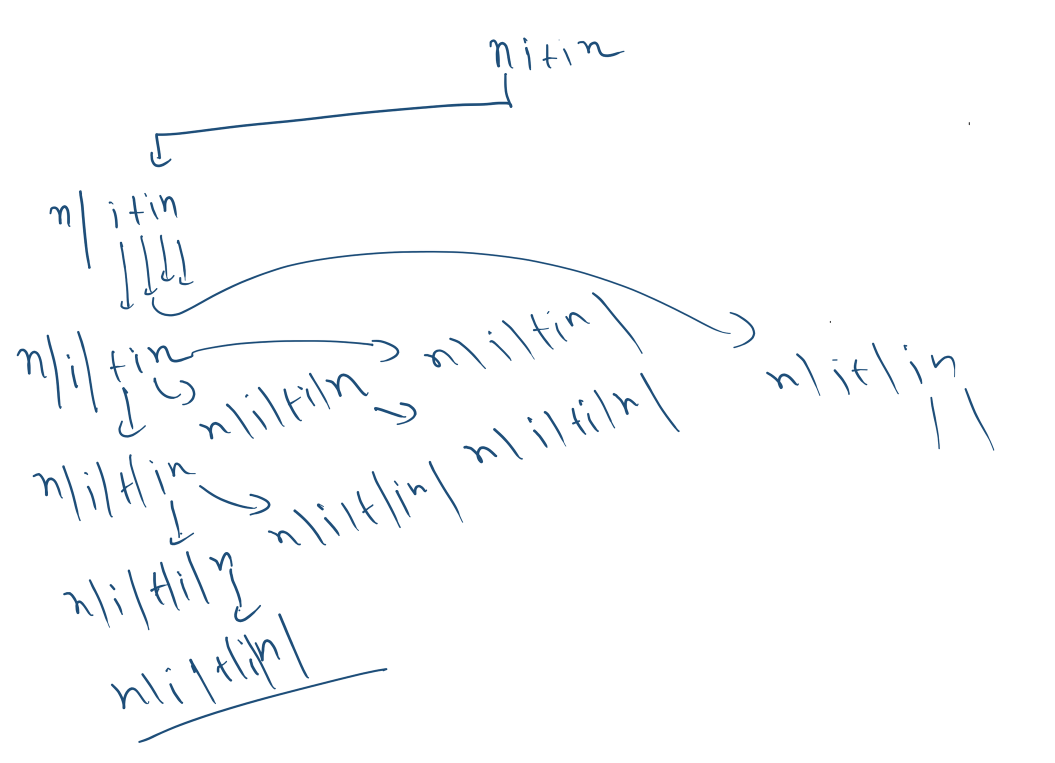
Example 1:

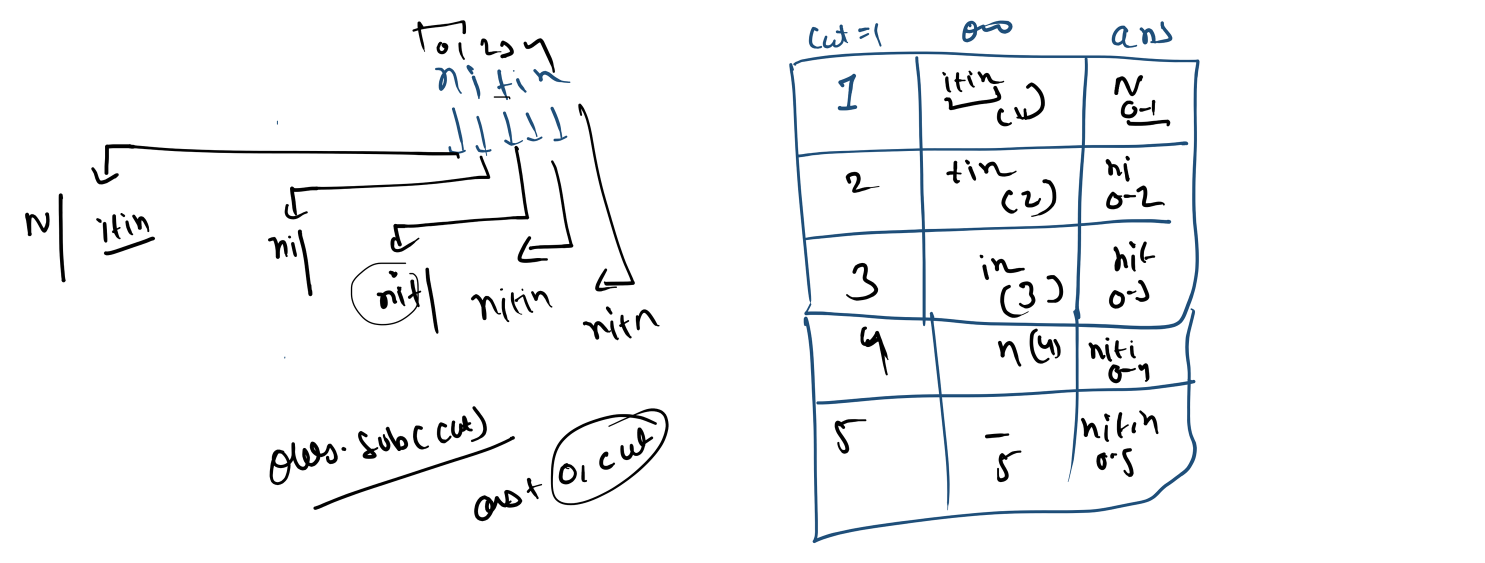
```
Input: s = "aab"
Output: [["a","a","b"],["aa","b"]]
```



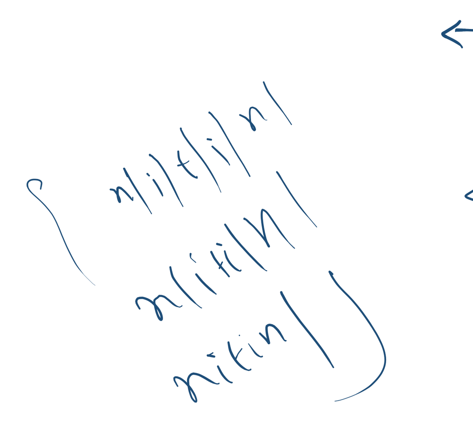

```
n|i|t|i|n|
n|i|t|in|
n|i|ti|n|
n|i|tin|
n|it|i|n|
n|it|in|
n|iti|n|
n|itin|
ni|t|i|n|
ni|t|in|
ni|ti|n|
ni|tin|
nit|i|n|
nit|in|
niti|n|
nitin|
```



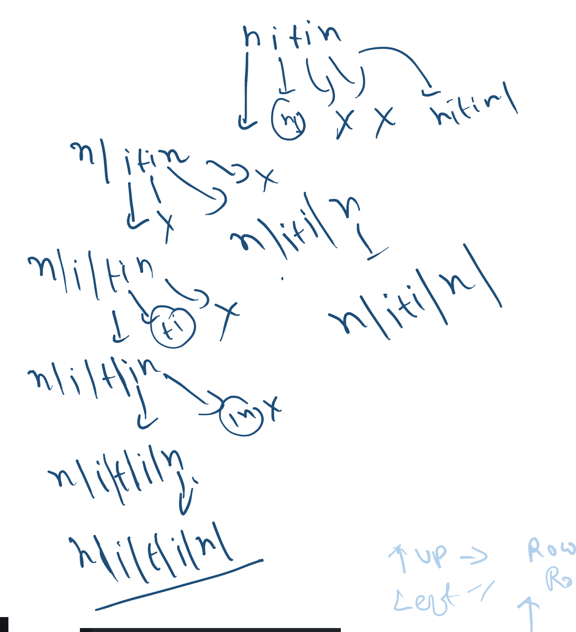





```
n|i|t|i|n|
n|i|t|in|
n|i|ti|n|
n|i|tin|
n|it|i|n|
n|it|in|
n|iti|n|
n|itin|
ni|t|i|n|
ni|t|in|
ni|ti|n|
ni|tin|
nit|i|n|
nit|in|
niti|n|
nitin|
```


```
n|i|t|i|n|
n|i|t|in|
n|i|ti|n|
n|i|tin|
n|it|i|n|
n|it|in|
n|iti|n|
n|itin|
ni|t|i|n|
ni|t|in|
ni|ti|n|
ni|tin|
nit|i|n|
nit|in|
niti|n|
nitin|
```

Rat Chases its cheese

You are given an N*M grid. Each cell (i,j) in the grid is either blocked, or empty. The rat can move from a position towards left, right, up or down on the grid.

Initially rat is on the position (1,1). It wants to reach position (N,M) where it's cheese is waiting for. If a path exists-it is always unique. Find that path and help the rat reach its cheese.

Input format

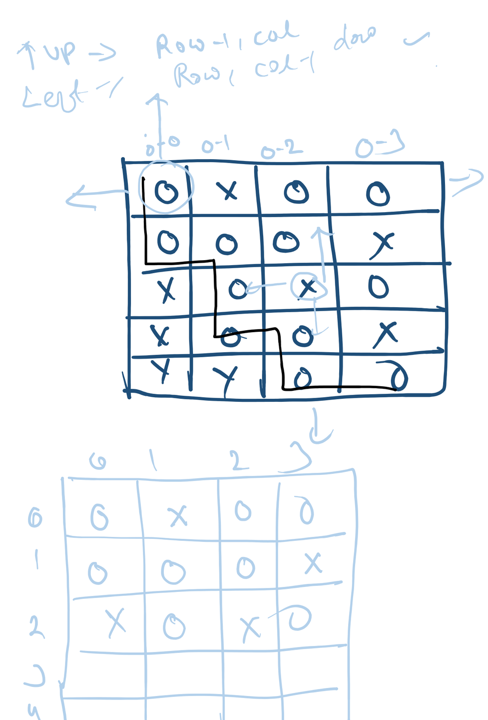
```
First line contains 2 integers  N and M  denoting the rows and columns in the grid.
 Next N line contains M characters each. An 'X' in position (i,j) denotes that the cell is blocked and ans 'O' denotes that the cell is empty.
```

Input

```
5 4
OXOO
OOOX
XOXO
XOOX
XXOO
```

Output

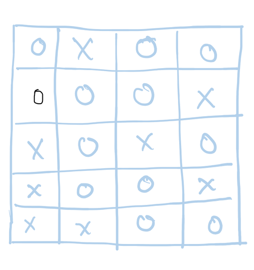
```
1 0 0 0
1 1 0 0
0 1 0 0
0 1 1 0
0 0 1 1
```



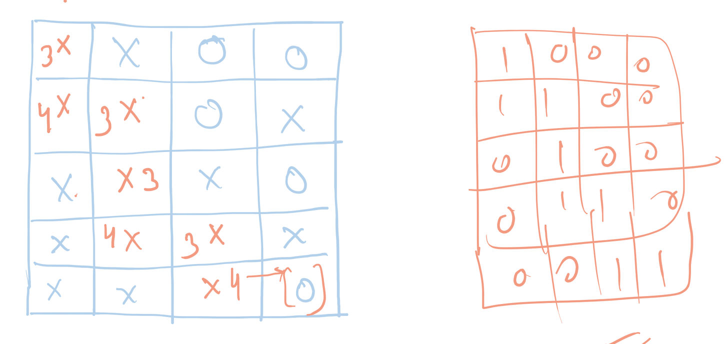
```java
public static void main(String[] args) {
    // TODO Auto-generated method stub
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    int m = sc.nextInt();
    char[][] maze = new char[n][m];
    for (int i = 0; i < maze.length; i++) {
        String s = sc.next();
        for (int j = 0; j < s.length(); j++) {
            maze[i][j]=s.charAt(j);
        }
    }

}
```

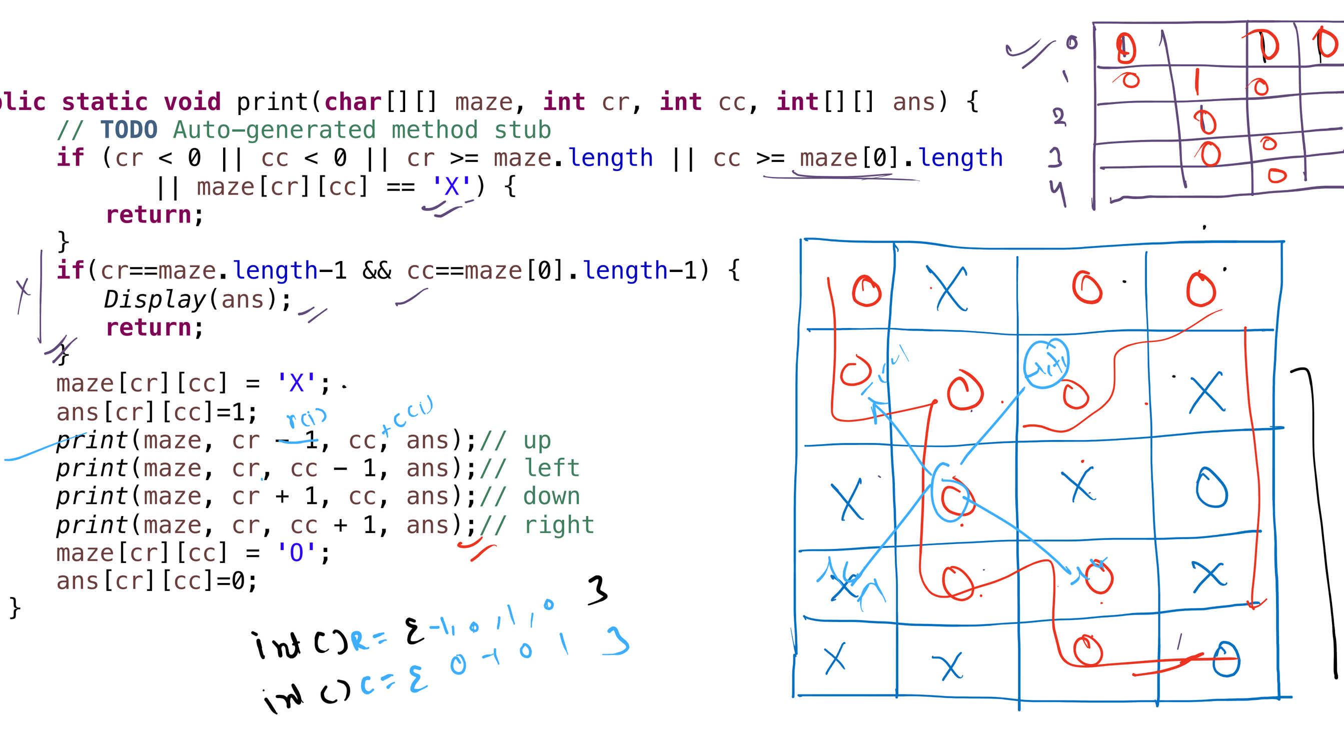
```java
public static void print(char[][] maze, int cr, int cc) {
    // TODO Auto-generated method stub
    if (cr < 0 || cc < 0 || cr >= maze.length || cc >= maze[0].length) {
        return;
    }
    print(maze, cr - 1, cc);// up
    print(maze, cr, cc - 1);// left
    print(maze, cr + 1, cc);// down
    print(maze, cr, cc + 1);// right
}
```

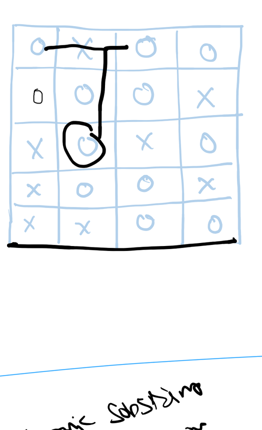


```java
public static void print(char[][] maze, int cr, int cc) {
    // TODO Auto-generated method stub
    if (cr < 0 || cc < 0 || cr >= maze.length || cc >= maze[0].length || maze[cr][cc] == 'X') {
        return;
    }
    maze[cr][cc] = 'X';
    print(maze, cr - 1, cc);// up
    print(maze, cr, cc - 1);// left
    print(maze, cr + 1, cc);// down
    print(maze, cr, cc + 1);// right
    maze[cr][cc] = '0';
}
```



```java
public static void print(char[][] maze, int cr, int cc, int[][] ans) {
    // TODO Auto-generated method stub
    if (cr < 0 || cc < 0 || cr >= maze.length || cc >= maze[0].length
            || maze[cr][cc] == 'X') {
        return;
    }
    if(cr==maze.length-1 && cc==maze[0].length-1) {
        Display(ans);
        return;
    }
    maze[cr][cc] = 'X';
    ans[cr][cc]=1;
    print(maze, cr - 1, cc, ans);// up
    print(maze, cr, cc - 1, ans);// left
    print(maze, cr + 1, cc, ans);// down
    print(maze, cr, cc + 1, ans);// right
    maze[cr][cc] = '0';
    ans[cr][cc]=0;
}
```





I
(i) Count total Palindromic Substring
(ii) Print all Palindromic Substring
(iii) longest length palindromic substring
(iv) Count all odd len Palindromic substring
(v) Count all even len Palindromic substring
(vi) longest odd len ""
(vii) long even ""