Leetcode problem no. 221

Link- (https://leetcode.com/problems/maximal-square/description/)

Problem Description:

Given an m x n binary matrix filled with 0's and 1's, *find the largest square containing only* 1's *and return its area*.

Example 1:

Input: matrix =
[["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]
Output: 4

Intuition:

We want the **largest square of all 1s** in a binary matrix.
Instead of the classic DP approach, this code uses a **binary search on side length x**:

1. **Preprocessing with prefix row sums (prefRow)**
    a. For each row, store prefix sums so that we can quickly check how many 1s are in any segment of that row.
    b. This allows us to test if a row segment of length x is all 1s in O(1) time.
2. **Binary search over possible square sizes**
    a. Low = 1, High = min(rows, cols).

b. For each candidate size x, scan all possible x × x submatrices.
c. For each row in that square, check if the entire segment of length x is all 1s using prefix sums.
d. If every row inside this square has all 1s, then this size works (here = true).
e. If some square of size x exists, we move low up (try bigger squares); otherwise, shrink high.

3. **Final Answer**
    a. ans keeps track of the largest side length found, and we return ans * ans as the maximum area.

So the approach is: **binary search the side length → verify squares using row prefix sums.**

Code:

```
class Solution {
public int maximalSquare(char[][] matrix) {
int rl = matrix.length;
int cl = matrix[0].length;
int n = Math.min(matrix.length,matrix[0].length);
int[][] prefRow = new int[rl][cl];
for(int i=0;i<rl;i++){
int sum = 0;
for(int j=0;j<cl;j++){
sum += (matrix[i][j]-'0');
prefRow[i][j] = sum;
}
}
int low = 1;
int ans = 0;
int high = n;
while(low<=high){
int x = (low+high)/2;
boolean here = false;
for(int i=0;i<=rl-x;i++){
for(int j=0;j<=cl-x;j++){
boolean found = false;
int sr = i;
```

```
int er = i+x-1;
int sc = j;
int ec = j+x-1;
for(int k=sr;k<=er;k++){
if(prefRow[k][ec]-prefRow[k][sc]+(matrix[k][sc]-'0')!=x){
found = true;
break;
}
}
if(!found) {
here = true;
break;
}
}
if(here) break;
}
if(here){
low = x+1;
ans = x;
}else{
high = x-1;
}
}
return ans*ans;

}
}
```

Time Complexity:

O(n*m*log(min(m,n))

Space Complexity:

O(nm)

Best Time Complexity:

O(nm)

Best space complexity:

O(nm)