Maximal Rectangle

Given a 2D binary matrix filled with 0's and 1's, find the largest rectangle containing all ones and return its area.

Solution 1

The DP solution proceeds row by row, starting from the first row. Let the maximal rectangle area at row i and column j be computed by [right(i,j) - left(i,j)]*height(i,j).

All the 3 variables left, right, and height can be determined by the information from previous row, and also information from the current row. So it can be regarded as a DP solution. The transition equations are:

> left(i,j) = max(left(i-1,j), cur*left), cur*left can be determined from the current row
>
> right(i,j) = min(right(i-1,j), cur*right), cur*right can be determined from the current row
>
> height(i,j) = height(i-1,j) + 1, if matrix[i][j]=='1';
>
> height(i,j) = 0, if matrix[i][j]=='0'

The code is as below. The loops can be combined for speed but I separate them for more clarity of the algorithm.

```
class Solution {public:
int maximalRectangle(vector<vector<char> > &matrix) {
    if(matrix.empty()) return 0;
    const int m = matrix.size();
    const int n = matrix[0].size();
    int left[n], right[n], height[n];
    fill_n(left,n,0); fill_n(right,n,n); fill_n(height,n,0);
    int maxA = 0;
    for(int i=0; i<m; i++) {
        int cur_left=0, cur_right=n;
        for(int j=0; j<n; j++) { // compute height (can do this from either side)
            if(matrix[i][j]=='1') height[j]++;
            else height[j]=0;
        }
        for(int j=0; j<n; j++) { // compute left (from left to right)
            if(matrix[i][j]=='1') left[j]=max(left[j],cur_left);
            else {left[j]=0; cur_left=j+1;}
        }
        // compute right (from right to left)
        for(int j=n-1; j>=0; j--) {
            if(matrix[i][j]=='1') right[j]=min(right[j],cur_right);
            else {right[j]=n; cur_right=j;}
        }
        // compute the area of rectangle (can do this from either side)
        for(int j=0; j<n; j++)
            maxA = max(maxA,(right[j]-left[j])*height[j]);
    }
    return maxA;
}
```

};

If you think this algorithm is not easy to understand, you can try this example:

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

The vector "left" and "right" from row 0 to row 2 are as follows

row 0:

```
l: 0 0 0 3 0 0 0
r: 7 7 7 4 7 7 7
```

row 1:

```
l: 0 0 2 3 2 0 0
r: 7 7 5 4 5 7 7
```

row 2:

```
l: 0 1 2 3 2 1 0
r: 7 6 5 4 5 6 7
```

The vector "left" is computing the left boundary. Take (i,j)=(1,3) for example. On current row 1, the left boundary is at j=2. However, because matrix[1][3] is 1, you need to consider the left boundary on previous row as well, which is 3. So the real left boundary at (1,3) is 3.

I hope this additional explanation makes things clearer.

written by morrischen2008 original link here

Solution 2

This question is similar as [Largest Rectangle in Histogram]:

You can maintain a row length of Integer array H recorded its height of '1's, and scan and update row by row to find out the largest rectangle of each row.

For each row, if matrix[row][i] == '1'. H[i] +=1, or reset the H[i] to zero. and accroding the algorithm of [Largest Rectangle in Histogram], to update the maximum area.

```
public class Solution {
    public int maximalRectangle(char[][] matrix) {
        if (matrix==null||matrix.length==0||matrix[0].length==0)
            return 0;
        int cLen = matrix[0].length;    // column length
        int rLen = matrix.length;       // row length
        // height array
        int[] h = new int[cLen+1];
        h[cLen]=0;
        int max = 0;


        for (int row=0;row<rLen;row++) {
            Stack<Integer> s = new Stack<Integer>();
            for (int i=0;i<cLen+1;i++) {
                if (i<cLen)
                    if(matrix[row][i]=='1')
                        h[i]+=1;
                    else h[i]=0;

                if (s.isEmpty()||h[s.peek()]<=h[i])
                    s.push(i);
                else {
                    while(!s.isEmpty()&&h[i]<h[s.peek()]){
                        int top = s.pop();
                        int area = h[top]*(s.isEmpty()?i:(i-s.peek()-1));
                        if (area>max)
                            max = area;
                    }
                    s.push(i);
                }
            }
        }
        return max;
    }
}
```

written by wangyushawn original link here

Solution 3

We can apply the maximum in histogram in each row of the 2D matrix. What we need is to maintain an int array for each row, which represent for the height of the histogram.

Please refer to https://leetcode.com/problems/largest-rectangle-in-histogram/ first.

Suppose there is a 2D matrix like

1 1 0 1 0 1

0 1 0 0 1 1

1 1 1 1 0 1

1 1 1 1 0 1

First initiate the height array as 1 1 0 1 0 1, which is just a copy of the first row. Then we can easily calculate the max area is 2.

Then update the array. We scan the second row, when the matrix[1][i] is 0, set the height[i] to 0; else height[i] += 1, which means the height has increased by 1. So the height array again becomes 0 2 0 0 1 2. The max area now is also 2.

Apply the same method until we scan the whole matrix. the last height arrays is 2 4 2 2 0 2, so the max area has been found as 2 * 4 = 8.

Then reason we scan the whole matrix is that the maximum value may appear in any row of the height.

Code as follows:

```
public class Solution {
public int maximalRectangle(char[][] matrix) {
    if(matrix == null || matrix.length == 0 || matrix[0].length == 0) return 0;

    int[] height = new int[matrix[0].length];
    for(int i = 0; i < matrix[0].length; i ++){
        if(matrix[0][i] == '1') height[i] = 1;
    }
    int result = largestInLine(height);
    for(int i = 1; i < matrix.length; i ++){
        resetHeight(matrix, height, i);
        result = Math.max(result, largestInLine(height));
    }

    return result;
}

private void resetHeight(char[][] matrix, int[] height, int idx){
    for(int i = 0; i < matrix[0].length; i ++){
        if(matrix[idx][i] == '1') height[i] += 1;
        else height[i] = 0;
    }
}

public int largestInLine(int[] height) {
    if(height == null || height.length == 0) return 0;
    int len = height.length;
    Stack<Integer> s = new Stack<Integer>();
    int maxArea = 0;
    for(int i = 0; i <= len; i++){
        int h = (i == len ? 0 : height[i]);
        if(s.isEmpty() || h >= height[s.peek()]){
            s.push(i);
        }else{
            int tp = s.pop();
            maxArea = Math.max(maxArea, height[tp] * (s.isEmpty() ? i : i - 1 - s
.peek()));
            i--;
        }
    }
    return maxArea;
}
```

}

written by bu.will.9 original link here