一、题目说明

题目，85. Maximal Rectangle，计算只包含1的最大矩阵的面积。难度是Hard！

二、我的解答

看到这个题目，我首先想到的是dp，用`dp[i][j]`表示第i行第j列元素向右下角计算的最大面积。后来发现从`dp[i+1][j]`、`dp[i][j+1]`和`dp[i+1][j+1]`计算`dp[i][j]`几乎没有任何规律可循。

然后，我就想用`down_dp[i][j]`和`right_dp[i][j]`两个dp，但遗憾的是还是没成功。

后面看了大神的写法，其实`down_dp[i][j]`然后"向右找同一行"计算即可。代码如下：

```
class Solution{
    public:
        int maximalRectangle(vector<vector<char>>& matrix){
            if(matrix.empty()) return 0;
            int m = matrix.size();
            int n = matrix[0].size();

            vector<vector<int>> down_dp(m,vector<int>(n,0));
            int result = 0;
            //最后一行
            for(int j=n-1;j>=0;j--){
                if(matrix[m-1][j]=='0'){
                    down_dp[m-1][j] = 0;
                }else if(j==n-1){
                    down_dp[m-1][j] =  1;
                    result = max(1,result);
                }else{
                    down_dp[m-1][j] =  1;
                    result = max(1,result);
                    int tmp = 1;
                    for(int t=j+1;t<n;t++){
                        if(down_dp[m-1][t]>0){
                            tmp++;
                            result = max(tmp,result);
                        }else{
                            break;
                        }
                    }
                }
            }

            //最后一列
            for(int i=m-1;i>=0;i--){
                if(matrix[i][n-1]=='0'){
                    down_dp[i][n-1] = 0;
                }else if(i==m-1){
                    down_dp[i][n-1] =  1;
                    result = max(1,result);
                }else{
                    down_dp[i][n-1] =  down_dp[i+1][n-1] + 1;
                    result = max(down_dp[i][n-1],result);
                }
            }
```

```
            for(int j=n-1;j>=0;j--){//列
                for(int i=m-2;i>=0;i--){
                    if(matrix[i][j]=='0'){
                        down_dp[i][j] = 0;
                    }else if(matrix[i][j]=='1'){
                        down_dp[i][j] = down_dp[i+1][j] + 1;
                        result = max(down_dp[i][j],result);
                        int temp = 1,curMin=down_dp[i][j],curMax = down_dp[i]
[j];

                        //向右找同一行
                        for(int t=j+1;t<n;t++){
                            if(down_dp[i][t]>0){
                                temp++;
                                curMin = min(curMin,down_dp[i][t]);
                                curMax = temp * curMin;
                                result = max(curMax,result);
                            }else{
                                break;
                            }
                        }
                    }
                }
            }

            return result;
        }
};
```

性能如下：

```
Runtime: 28 ms, faster than 45.92% of C++ online submissions for Maximal
Rectangle.
Memory Usage: 11.1 MB, less than 61.11% of C++ online submissions for Maximal
Rectangle.
```

三、优化措施

上面代码，先计算最后一行，最后一列，然后向上计算。其实完全可以合并起来的。

```
class Solution{
    public:
        int maximalRectangle(vector<vector<char>>& matrix){
            if(matrix.empty()) return 0;
            int m = matrix.size();
            int n = matrix[0].size();

            vector<vector<int>> down_dp(m,vector<int>(n,0));
            int result = 0;

            for(int j=n-1;j>=0;j--){//列
                for(int i=m-1;i>=0;i--){
```

```
                    if(matrix[i][j]=='0'){
                        down_dp[i][j] = 0;
                    }else if(matrix[i][j]=='1'){
                        if(i<m-1){
                            down_dp[i][j] = down_dp[i+1][j] + 1;
                        } else{
                            down_dp[i][j] = 1;
                        }

                        result = max(down_dp[i][j],result);
                        int temp = 1,curMin=down_dp[i][j],curMax = down_dp[i]
[j];

                        //向右找同一行
                        for(int t=j+1;t<n;t++){
                            if(down_dp[i][t]>0){
                                temp++;
                                curMin = min(curMin,down_dp[i][t]);
                                curMax = temp * curMin;
                                result = max(curMax,result);
                            }else{
                                break;
                            }
                        }
                    }
                }
            }

            return result;
        }
};
```