CS29003 Algorithms Laboratory

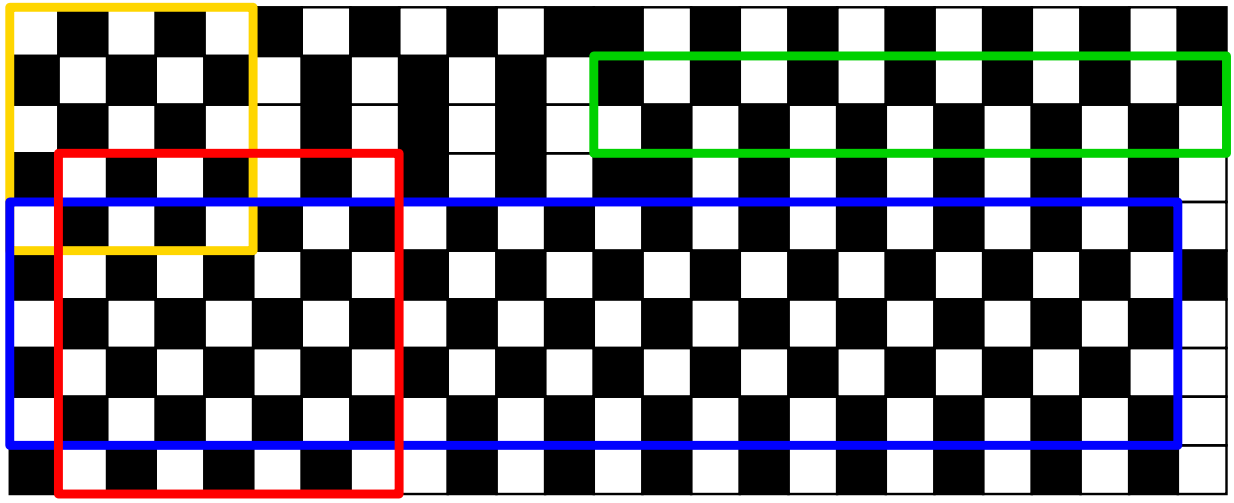
Laboratory Test

Date: 28–April–2020 Maximum Marks: 40

The Foobarland Chess Society (FCS) has built an indoor stadium for international chess tournaments. Its floor is to be covered by an $m \times n$ array of square tiles. Each tile is either white or black. An $s \times t$ array of tiles is said to have the chessboard pattern if every two adjacent tiles are of different colors, where adjacency is with respect to a shared edge (not a shared corner). By this definition, a 1×1 array always has the chessboard pattern (irrespective of its color). Ideally, the entire $m \times n$ floor of the stadium should have the chessboard pattern. However, specific instructions to this effect were not supplied to the tiling engineers. After the finishing of the tiling stage, the president of the FCS notices the problem. He wants to restore the chessboard pattern to the entire floor. In order to do that, he wants to identify the largest rectangle in the floor, that has the chessboard pattern (he will retile the remaining part). He seeks your help in solving this problem. You are supposed to write a program to do the following two tasks.

1. Find the largest square in the $m \times n$ array, that has the chessboard pattern.
2. Find the largest rectangle in the $m \times n$ array, that has the chessboard pattern.

The size of a square/rectangle is the number of tiles in it. The following figure illustrates the problem for a 10×25 array of tiles, and the solutions. The largest square highlighted in red is a 7×7 square of size 49, and the largest rectangle highlighted in blue is a 5×24 rectangle of size 120. Two smaller subarrays of chessboard pattern of dimensions 5×5 (yellow square) and 2×13 (green rectangle) are also shown.



Data types and data structures (4)

In this test, it is your responsibility to maintain your data types and structures, and write every function (like sorting if needed) explicitly. **No STL data types or functions are allowed.** You need to work with several two-dimensional arrays. You may go for dynamic allocation after you read m and n from the user. Assume that each of m and n is $\leqslant 50$. So you can use 50×50 arrays, and work in $m \times n$ parts of them. When you pass such an array to a function, it is mandatory to pass the column dimension 50 (the row dimension may be skipped). Of course, you need to pass m and n additionally to the functions as separate parameters.

We assume that the tiling is stored in an $m \times n$ array. This may be an array of characters (***W* means white, and *B* means black**), or an array of integers (**0 means white, and 1 means black**). Let us call this array F (faulty floor in Foobarland). Denote $\boxed{N = \max(m,n).}$

Finding the largest square by exhaustive search (4)

Write a function *essquare* to implement the following algorithm. For each valid index i, j in the input array F, first find the maximum length L of a square that can be placed in the input array with the top left corner at index i, j. Now, for $l = L, L-1, L-2, \ldots, 1$, find whether the $l \times l$ square with top left corner at i, j stores a chessboard pattern (you may write a function *ischessboard* for this check). Break as soon as the largest l is

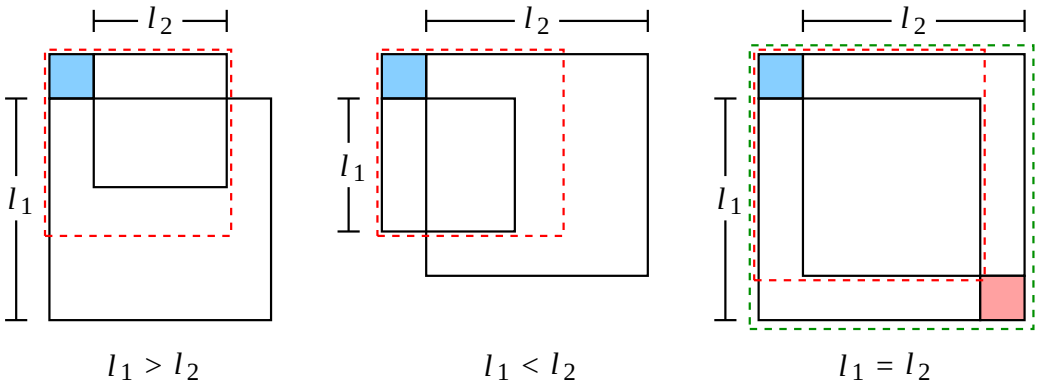
located. In the worst case, the loop runs up to $l = 1$, and a 1×1 subarray always has the chessboard pattern. Maximize the largest l values so detected over all indices i, j. The running time of this algorithm is $O(N^5)$.

Finding the largest rectangle by exhaustive search **(4)**

The function *esrectangle* for this task is similar to *essquare*. For each valid index i, j in the input array F, find the largest numbers K and L such that a $K \times L$ rectangle can fit in F with its top left corner at i, j. For all k, l in the range $1 \leqslant k \leqslant K$ and $1 \leqslant l \leqslant L$, find whether the $k \times l$ subarray with top left corner at i, j has the chessboard pattern (you can use the same function *ischessboard*—a square is also a rectangle). Now, maximize the size kl over all valid possibilities of k and l. Again, note that $k = l = 1$ always works, so this maximum is well-defined. Maximize the maximum size over all indices i, j. The running time is $O(N^6)$.

Finding the largest square by dynamic programming **(8)**

The huge running times of exhaustive search imply that there are scopes for improvements. Here, a dynamic-programming approach is suggested to find the largest square in just $O(N^2)$ time. Prepare an $m \times n$ table T such that $T(i, j)$ stores the length of the largest square of chessboard pattern with the top left corner at index i, j. For $i = m - 1$ or $j = n - 1$, we have $T(i, j) = 1$ (initialization). Take $0 \leqslant i \leqslant m - 2$ and $0 \leqslant j \leqslant n - 2$. If the (i, j)-th tile has the same color as the $(i + 1, j)$-th or the $(i, j + 1)$-th tile, we have $T(i, j) = 1$. Otherwise, look at the values $l_1 = T(i + 1, j)$ and $l_2 = T(i, j + 1)$. Three cases are illustrated in the following figure. The (i, j)-th tile is shaded in blue.



If $l_1 > l_2$ (see the left part of the figure), we have $T(i, j) = l_2 + 1$ (the red dashed rectangle). If a larger square can start at i, j, then the value of l_2 would have been higher. Likewise, if $l_1 < l_2$ (see the middle part of the figure), we have $T(i, j) = l_1 + 1$. The case $l_1 = l_2$ is the trickiest (see the right part of the figure). Look at the tile shaded in pink. If this tile is of the correct color, then $T(i, j) = l_1 + 1 = l_2 + 1$ (the green dashed rectangle). Otherwise, $T(i, j) = l_1 = l_2$ (the red dashed rectangle). Again if any larger square of chessboard pattern can start from (i, j), then neither l_1 nor l_2 was maximal.

Clearly, the table is to be filled from larger row indices to smaller row indices, and in each row, from larger column indices to smaller column indices. This is just the reverse of the row-major order. Write an **iterative** function to populate T. **Do not use memoization.** Maximize $T(i, j)$ over all i, j. Write a function *dpsquare* for implementing this algorithm.

Finding the largest rectangle by dynamic programming **(8)**

Write an **iterative** function *dprectangle* for this problem. The function should run in $O(N^3)$ time. Do not maintain the dimensions k and l if the largest rectangle with top left corner at i, j is a $k \times l$ rectangle. You nevertheless need to maintain two $m \times n$ arrays storing something else, so that at each i, j, you spend $O(N)$ time to calculate the largest rectangle of chessboard pattern starting at i, j. **Do not use memoization.**

The *main*() function **(4)**

- The user first enters m and n. The user then enters an $m \times n$ array of 0's and 1's. Note again that: **0 means white, and 1 means black.** Store these colors (as W/B or as 0/1) in the input array F.
- Call *essquare*, and print the solution. Show the largest square (or rectangle) found as a subarray of W and B. The part of F outside this square (or rectangle) is shown as a sequence of dots (see the sample I/O section below). You would better write a function for printing in this format.

- Call *esrectangle*, and print the solution in the same format.
- Call *dpsquare*, and print the solution in the same format.
- Call *dprectangle*, and print the solution in the same format.

Input/output (4 marks for correctness, 4 marks for formatting) **(8)**

```
10 25
0 1 0 1 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

+++ Exhaustive search: Square
    . . . . . .
    . . . . . .
    . . . . . .
    . W B W B W B W . . . . . .
    . B W B W B W B . . . . . .
    . W B W B W B W . . . . . .
    . B W B W B W B . . . . . .
    . W B W B W B W . . . . . .
    . B W B W B W B . . . . . .
    . W B W B W B W . . . . . .
    Area = 49

+++ Exhaustive search: Rectangle
    . . . . . .
    . . . . . .
    . . . . . .
    . . . . . .
    W B W B W B W B W B W B W B W B W B W B W B W B .
    B W B W B W B W B W B W B W B W B W B W B W B W .
    W B W B W B W B W B W B W B W B W B W B W B W B .
    B W B W B W B W B W B W B W B W B W B W B W B W .
    W B W B W B W B W B W B W B W B W B W B W B W B .
    . . . . . .
    Area = 120

+++ Dynamic Programming: Square
    . . . . . .
    . . . . . .
    . . . . . .
    . W B W B W B W . . . . . .
    . B W B W B W B . . . . . .
    . W B W B W B W . . . . . .
    . B W B W B W B . . . . . .
    . W B W B W B W . . . . . .
    . B W B W B W B . . . . . .
    . W B W B W B W . . . . . .
    Area = 49

+++ Dynamic Programming: Rectangle
    . . . . . .
    . . . . . .
    . . . . . .
    . . . . . .
    W B W B W B W B W B W B W B W B W B W B W B W B .
    B W B W B W B W B W B W B W B W B W B W B W B W .
    W B W B W B W B W B W B W B W B W B W B W B W B .
    B W B W B W B W B W B W B W B W B W B W B W B W .
    W B W B W B W B W B W B W B W B W B W B W B W B .
    . . . . . .
    Area = 120
```

Submit a single C/C++ source file. Do not use global/static variables. Do not use STL.