**ECE374 Assignment 6**

Due 03/27/2023

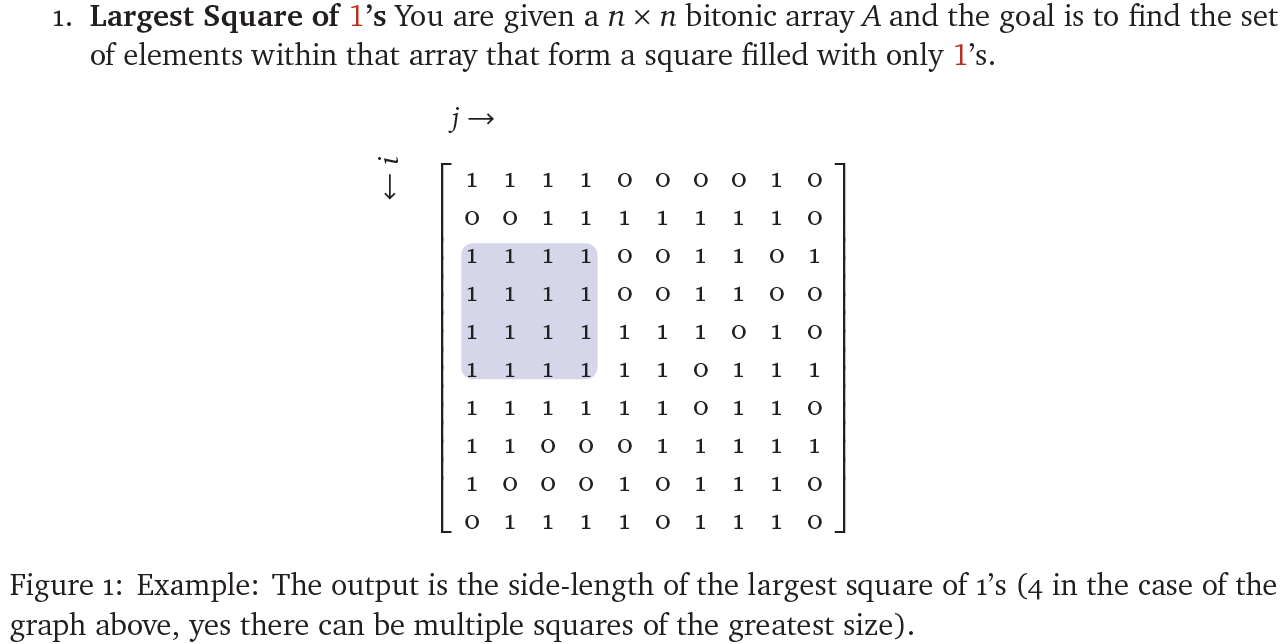
**Group & netid**

*Chen Si chensi3*

*Jie Wang jiew5*

*Shitian Yang sy39*

**Problem 1**



Solution:

Intuition:

1. We could use a n×n table T to store the value of the maximum square side lengths for each element on the source array A, with each element T[i, j] stores the maximum side length of the square whose upper-left corner is A[i, j].

2. Recurrence Cases

(1) Base Case

If the element at A[i, j] is 0, we should mark T[i, j] as 0, as we can’t construct a square of 1s with a upper-left corner of 0.

Since we are expanding the largest square of 1 to the upper-left direction, we should mark the base case as the bottom row T[n, :] and the right-most column T[:, n] to be themselves.

(2) General Case

To determine the max side length of the square with upper-left corner at [i, j], with A[i, j]=1, we should check the row i+1, the column j+1, and the intersection element at [i+1, j+1]. We should add 1 to the max side length if and only if we don’t find any 0 in row i+1, column j+1, and intersection element at [i+1, j+1]. Therefore, we have in this case. If we have a 0 in any one of T[i, j+1], T[i+1,j], T[i+1,j+1], we can’t construct a square that extends to the bottom-right, so we would only have 1 as side length. If we have a minimum value of m in T[i, j+1], T[i+1,j], T[i+1,j+1], we could assert that the largest all-1 matrix guaranteed by the row i+1, the column j+1, and the space between them have a side-length of m, and we could therefore add 1 to obtain a largest side length for [i, j].

(3) Therefore, we would find the final result value to be the maximum value in T.

Recurrence Relation:

Therefore, the algorithm is:

**LargestSquare**(A):

n = A.sidelength

T = table(n, n)

// Base case: last row and last column as themselves

for i 🡨 1 to n:

T[i, n] = A[i, n]

T[n, i] = A[n, i]

// Iterate through table

// From bottom-right to upper-left

for i 🡨 n-1 to 1:

for j 🡨 n-1 to 1:

if (A[i, j]==0):

T[i, j] = 0

else:

T[i, j] = 1 + max(T[i+1, j], [i, j+1], [i+1, j+1])

return max(T)

Since we build a n×n table and fill each element with constant time (take max value of the three “ancestors”), we could have a run time of O(n^2).