tạo struct Point

Tạo một cấu trúc Point với các thành phần:

- int x: Tọa độ x

- int y: Tọa độ y

Hàm importMN(filePath, M, N, matrix):

Mở tệp tại đường dẫn filePath để đọc

Đọc M từ tệp

Đọc N từ tệp

Khởi tạo ma trận matrix kích thước M x N

Duyệt qua từng i từ 0 đến M-1:

Duyệt qua từng j từ 0 đến N-1:

Đọc matrix[i][j] từ tệp

Đóng tệp

Hàm find(x, y, M, N, matrix):

Tạo một ngăn xếp stack rỗng

Đẩy (x, y) vào stack

Diện tích = 0

Trong khi stack không rỗng:

Lấy phần tử từ đỉnh stack và gán vào current

Pop phần tử từ stack

Gán x và y từ current.x, current.y

Nếu x nằm trong [0, M-1] và y nằm trong [0, N-1] và matrix[x][y] là 'O':

Đặt matrix[x][y] thành 'X'

Tăng diện tích lên 1

Đẩy (x+1, y) vào stack

Đẩy (x-1, y) vào stack

Đẩy (x, y+1) vào stack

Đẩy (x, y-1) vào stack

Trả về diện tích

Hàm findLargestCleanableArea(M, N, matrix):

maxArea = 0

startX = []

startY = []

Duyệt qua từng i từ 0 đến M-1:

Duyệt qua từng j từ 0 đến N-1:

Nếu matrix[i][j] là 'O':

diện tích = dfs(i, j, M, N, matrix)

Nếu diện tích lớn hơn maxArea:

Làm sạch startX

Làm sạch startY

maxArea = diện tích

Thêm (i, j) vào startX

Thêm (i, j) vào startY

Ngược lại nếu diện tích bằng maxArea:

Thêm (i, j) vào startX

Thêm (i, j) vào startY

Trả về startX, startY, maxArea

Hàm writeResultToFile(filePath, startX, startY, maxArea):

Mở tệp tại đường dẫn "filePath" để ghi

Ghi startX, startY, maxArea vào tệp

Đóng tệp

Hàm main():

inputFilePath = "/Users/thanhhuong/Desktop/CTDL/inpHB.txt"

outputFilePath = "outHB.txt"

M, N, maxArea = 0

matrix = []

importMN(inputFilePath, M, N, matrix)

startX, startY, maxArea = findLargestCleanableArea(M, N, matrix)

randomIndex = số ngẫu nhiên từ 0 đến độ dài của startX - 1

selectedX = startX[randomIndex]

selectedY = startY[randomIndex]

writeResultToFile(outputFilePath, giá trị tại selectedX, gía trị lại selectedY, maxArea)

Giải thích các thuật toán sẽ dùng và phân tích các bước chính phải làm để giải quyết vấn đề.

thuật toán bắt đầu từ ý tưởng

- Ban đầu ta sẽ bắt đầu từ phần tử O đầu tiên trong mảng.

- Tiếp theo, ta duyệt qua các vị trí ở 4 hướng (trên, dưới, trái, phải). Nếu gặp O thì ta

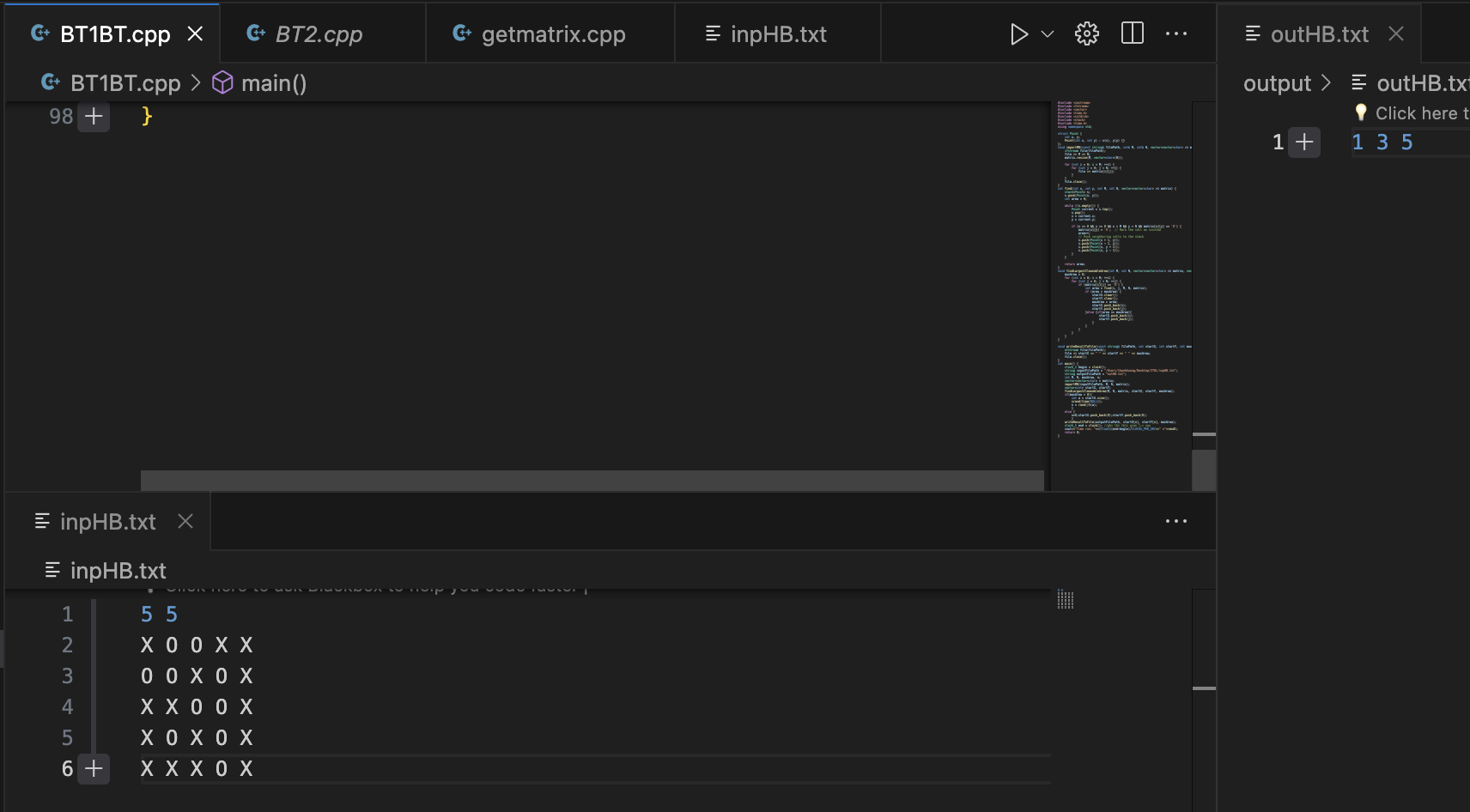
gắn X cho vị trí hiện tại và di chuyển đến vị trí O đó, lặp lại hành động đó đến khi không còn O nữa.

- Di chuyển đến vị trí có O gần nhất và lặp lại hành động trên.

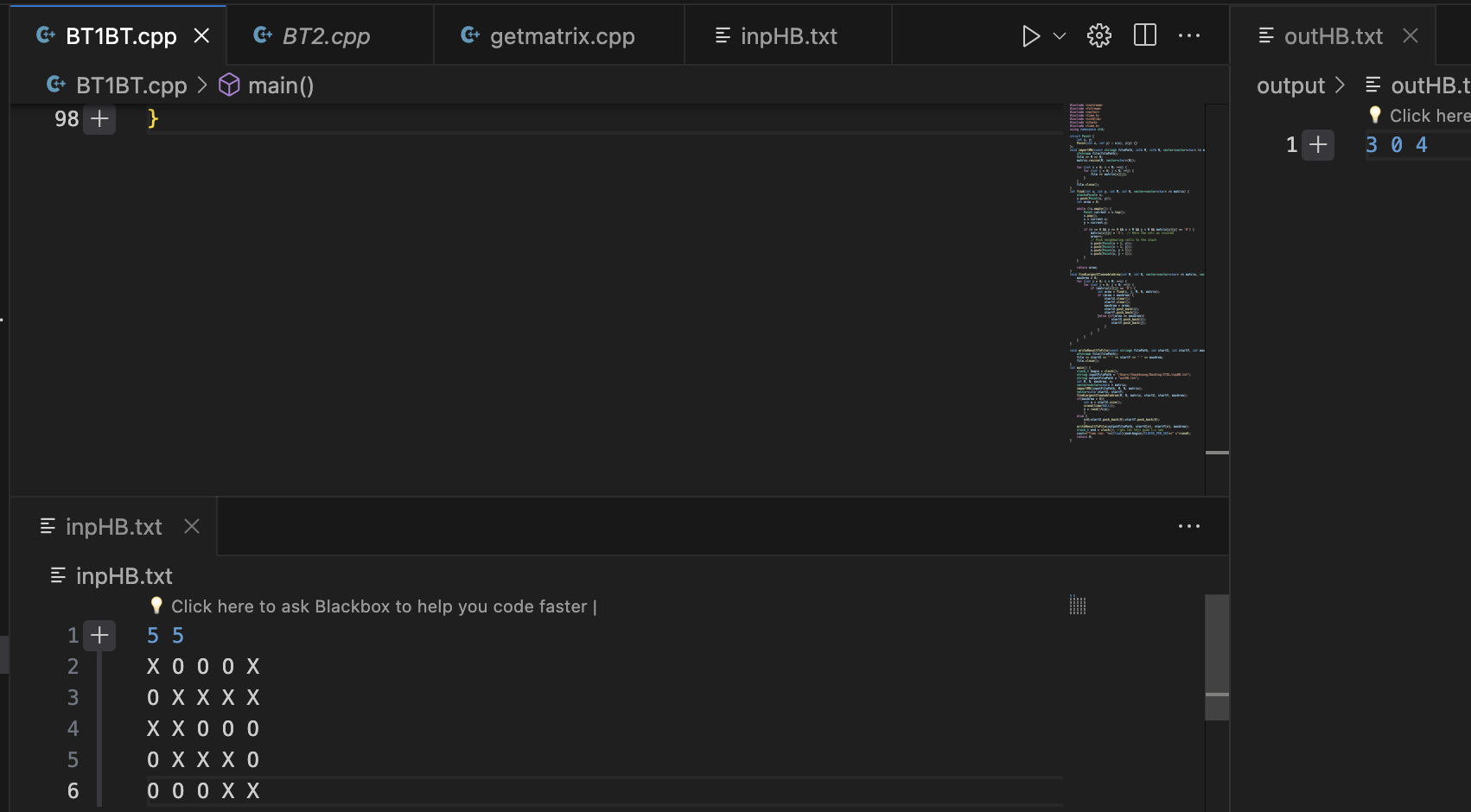
Mô tả quá trình tạo trường hợp để kiểm tra thuật toán (test case) và kết quả thu được khi chạy chương trình.

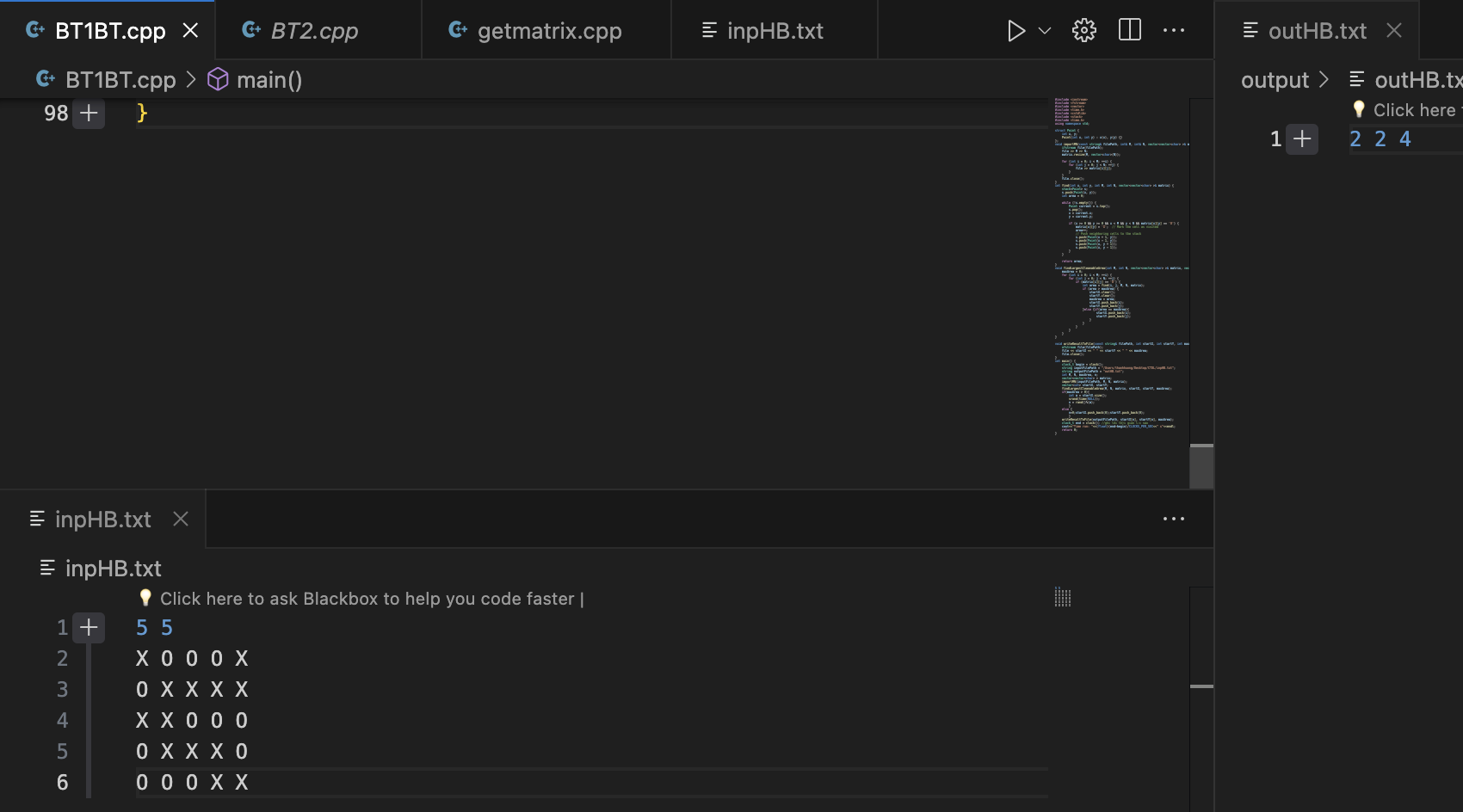
Đầu tiên là chạy code với yêu cầu đề bài

***1.Trường hợp chỉ có duy nhất 1 khu vực “O” lớn nhất***



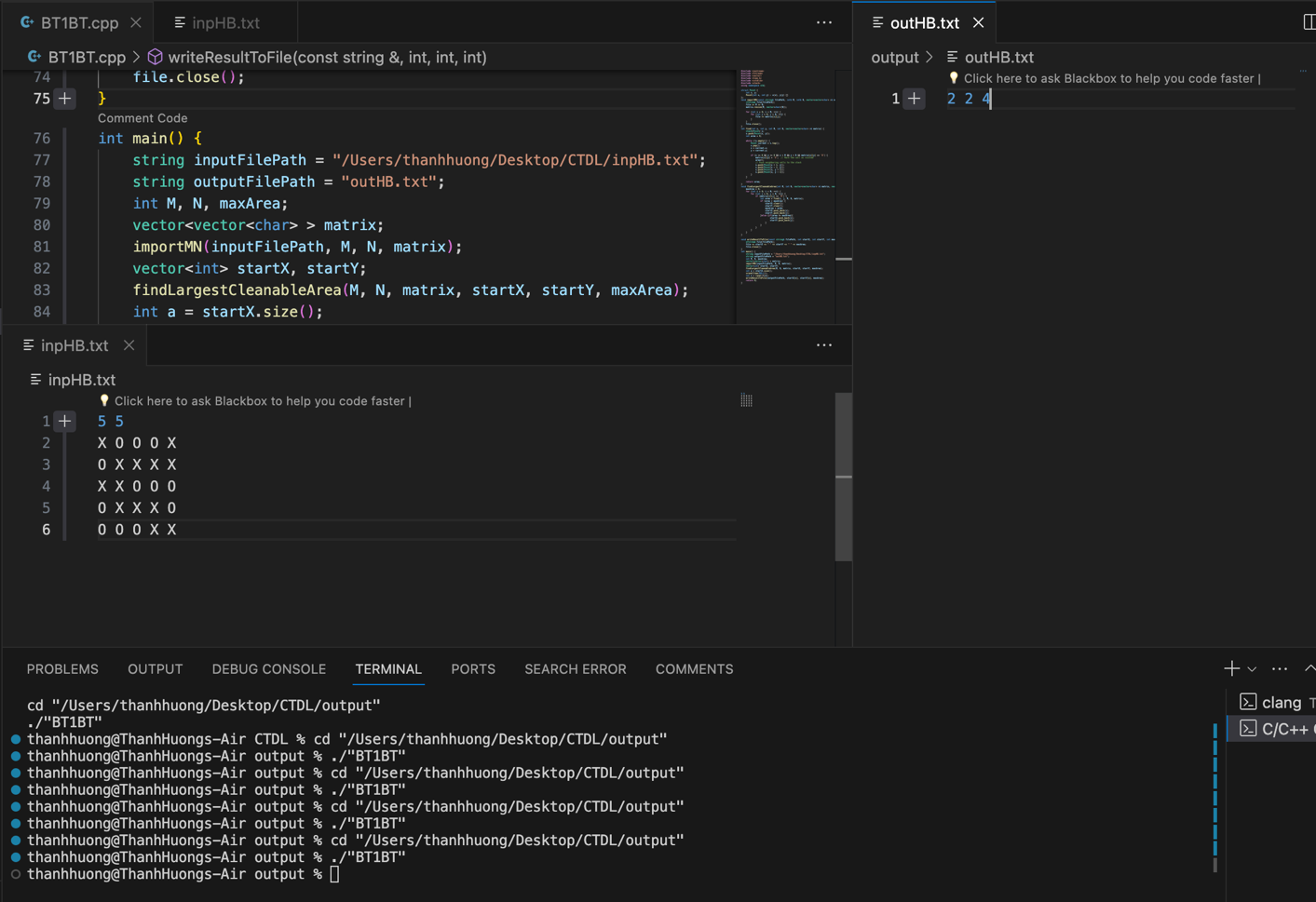
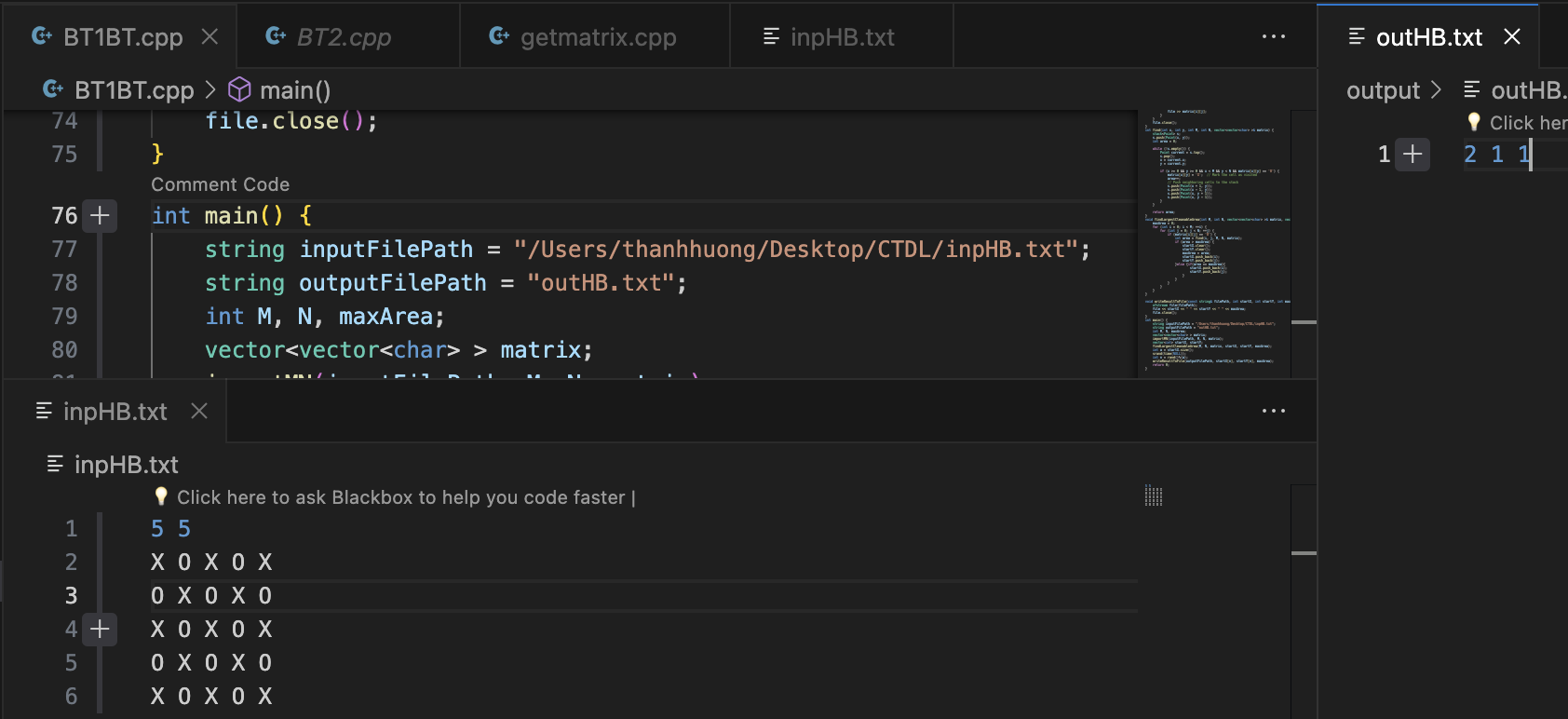
***2. Trường hợp có 2 khu vực “O” bằng nhau***





Thì ta sẽ nhận được cả 2 kết quả nếu thử chạy code nhiều lần

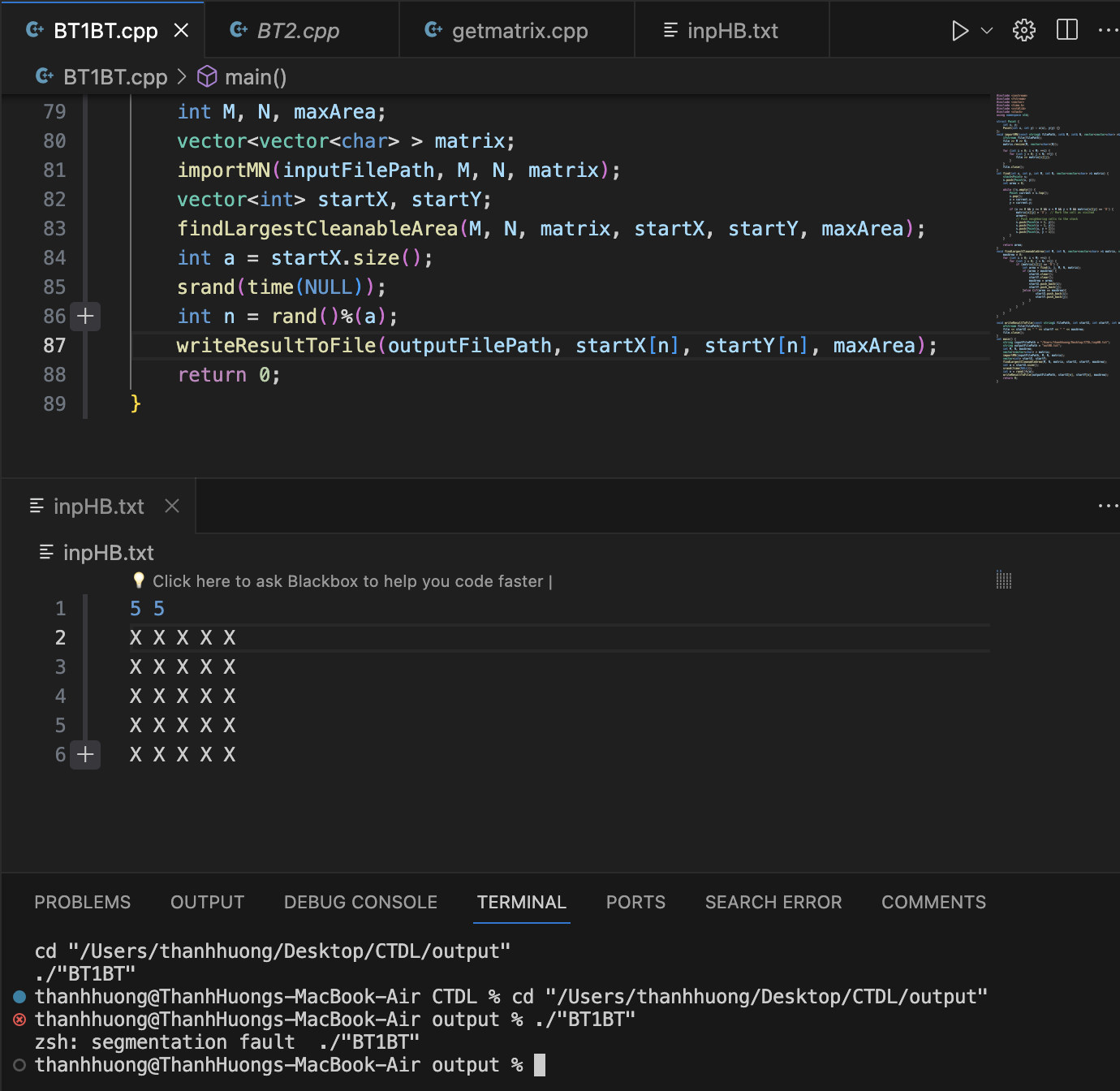
***3. Tiếp theo là trường hợp “X” và “O” xen kẻ và giữu nguyên kích thước 5x5***



Tương tự ở trường hợp 2, ta có thể nhận được 12 kết quả ngẫu nhiên vì ma trận trên có

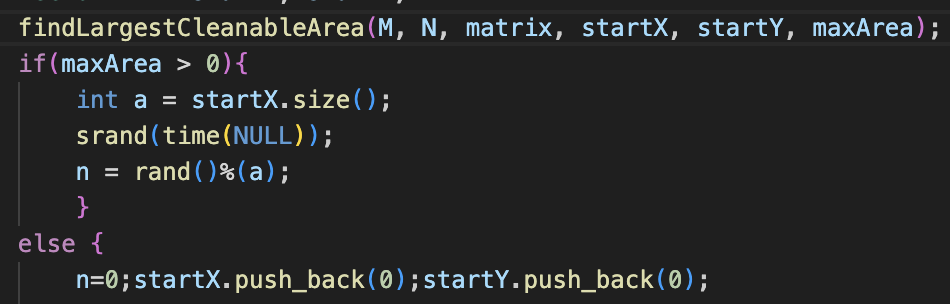
12 khu vực giống nhau

***4.Trường hợp chỉ toàn là “X”, giữ nguyên kích thước 5x5***

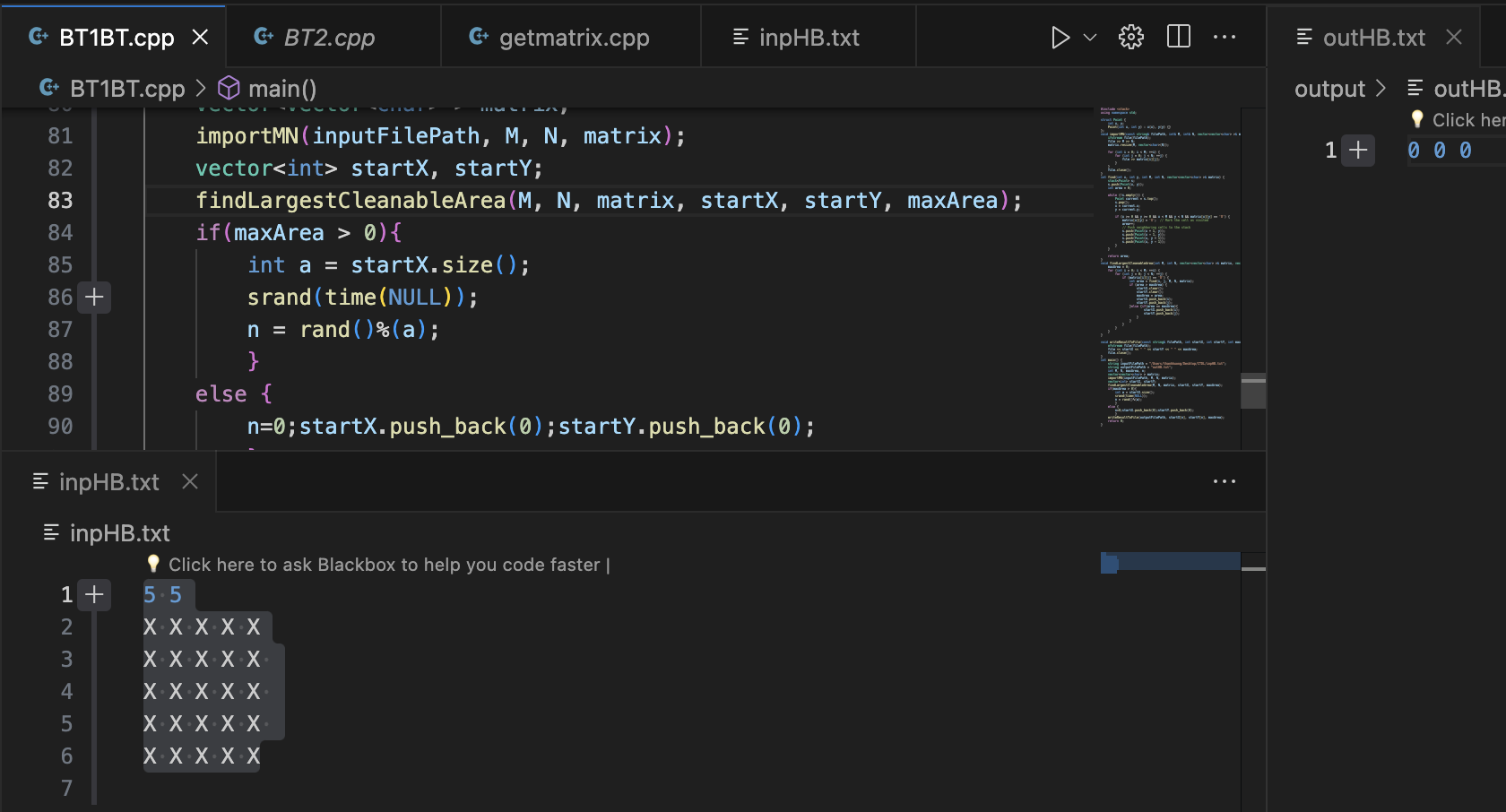


Ta nhận được lỗi “zsh: segmentation fault ./"BT1BT"”

Nguyên nhân là do lúc này đoạn mã không tìm thấy giá trị area và toạ độ bắt đầu, nên vùng nhớ không được cấp phát để lưu giá trị, vậy nên khi ta bắt đầu lấy size của vector và tính toán sẽ gặp lỗi.



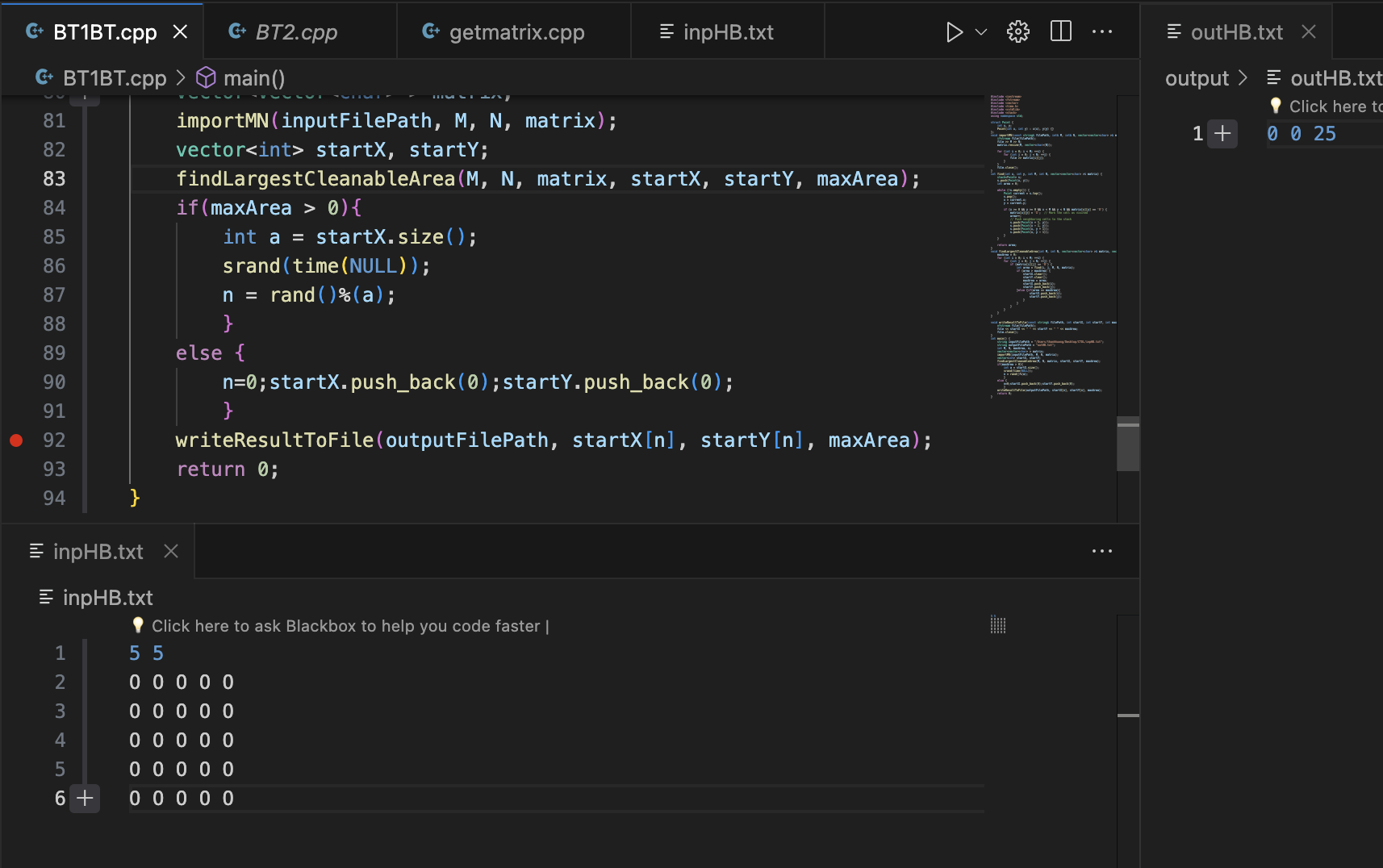
Vậy ta sẽ chỉnh sửa một chút đề khi gặp trường họp tương tự, code vẫn sẽ in kết quả



Kết quả sau khi chỉnh sửa, ta được

*Và đương nhiên, sau khi chỉnh sửa một chút, các trường hợp trên vẫn ra đúng kết quả*

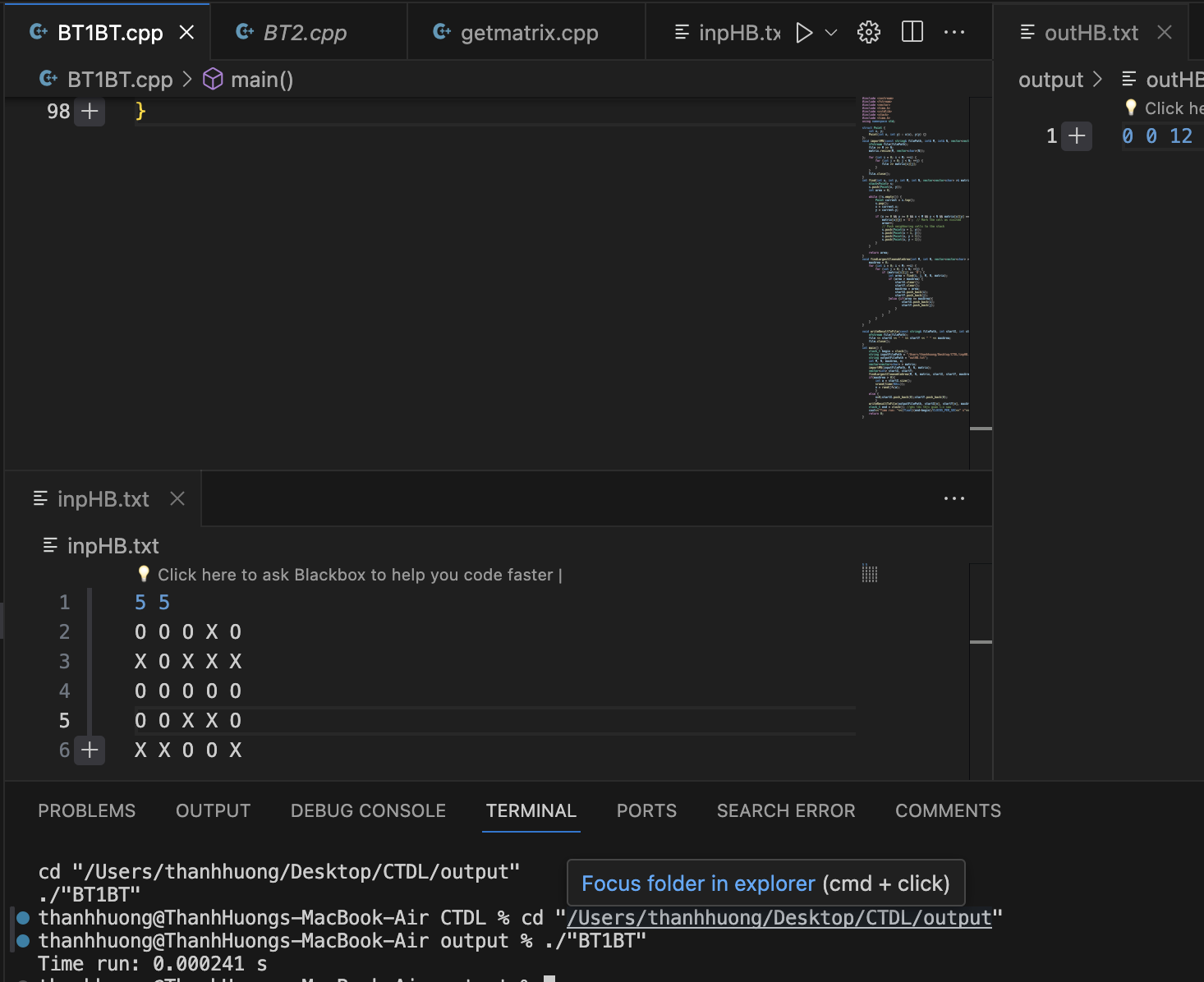
**5.Trường hợp tất cả chỉ toàn là “O”, giữ nguyên kích. 5x5**

****

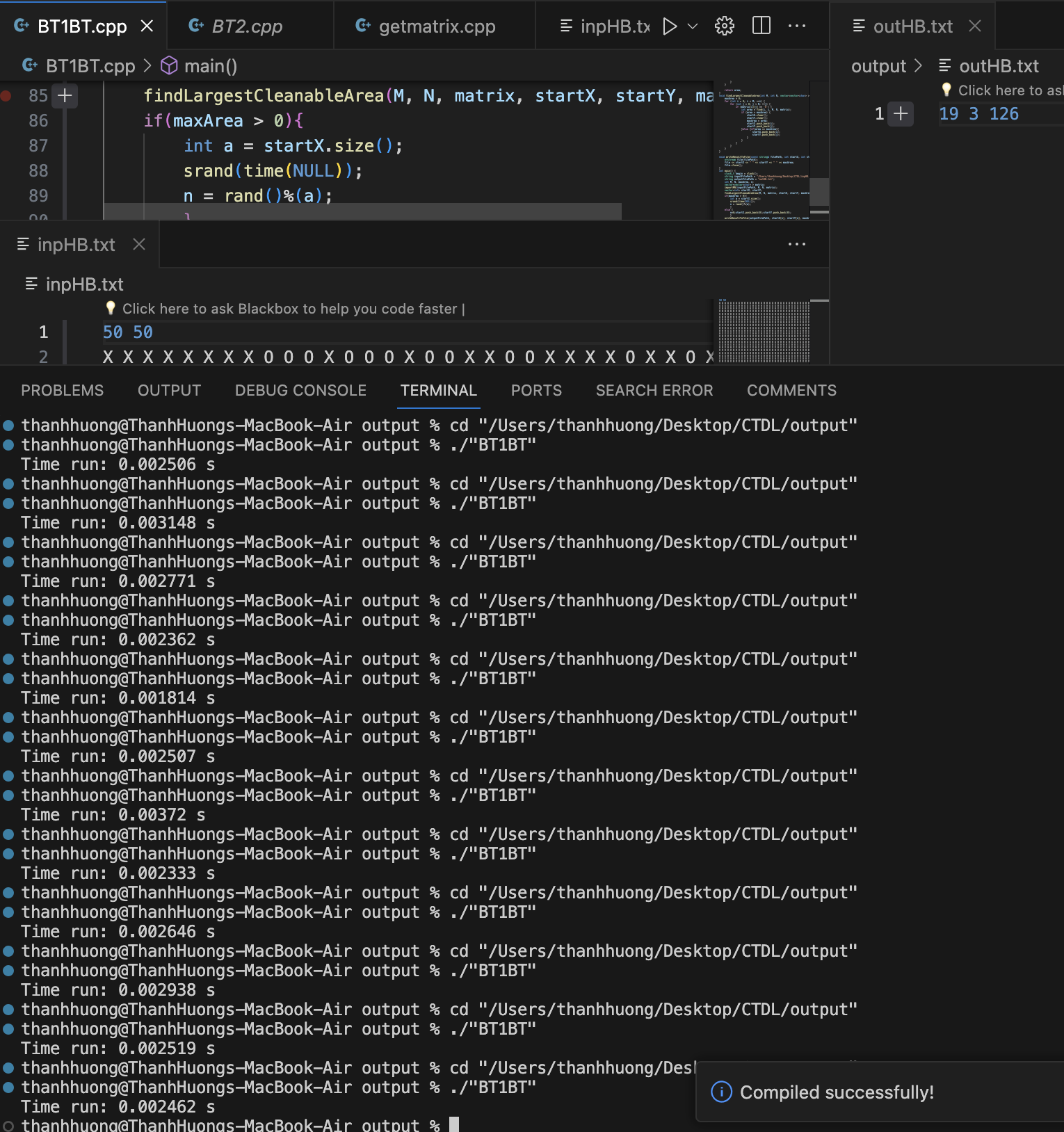
Kết quả

**6. Khảo sát thời gian xử lý:**

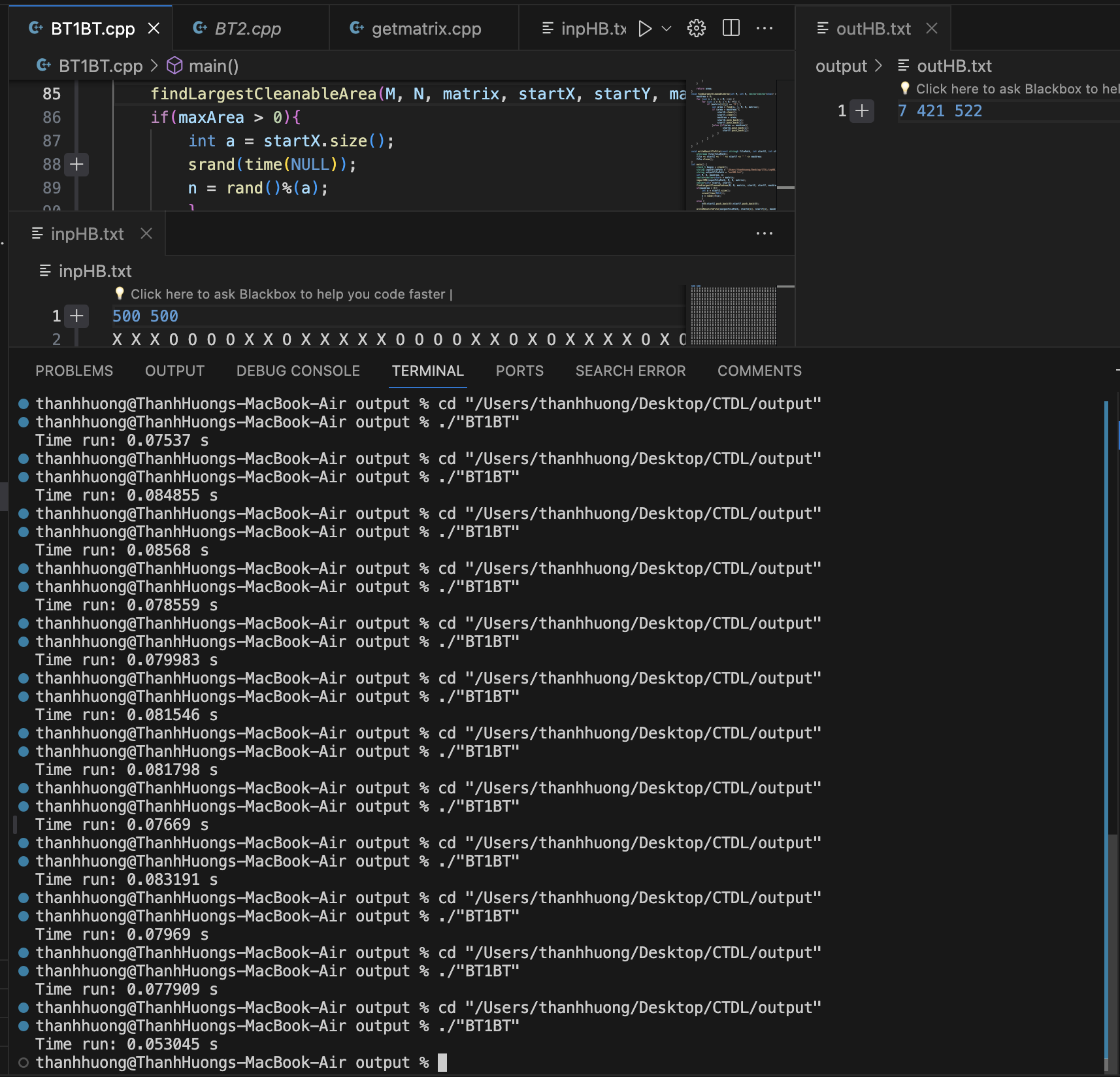
Bây giờ ta thử tăng kích thước của ma trận lên



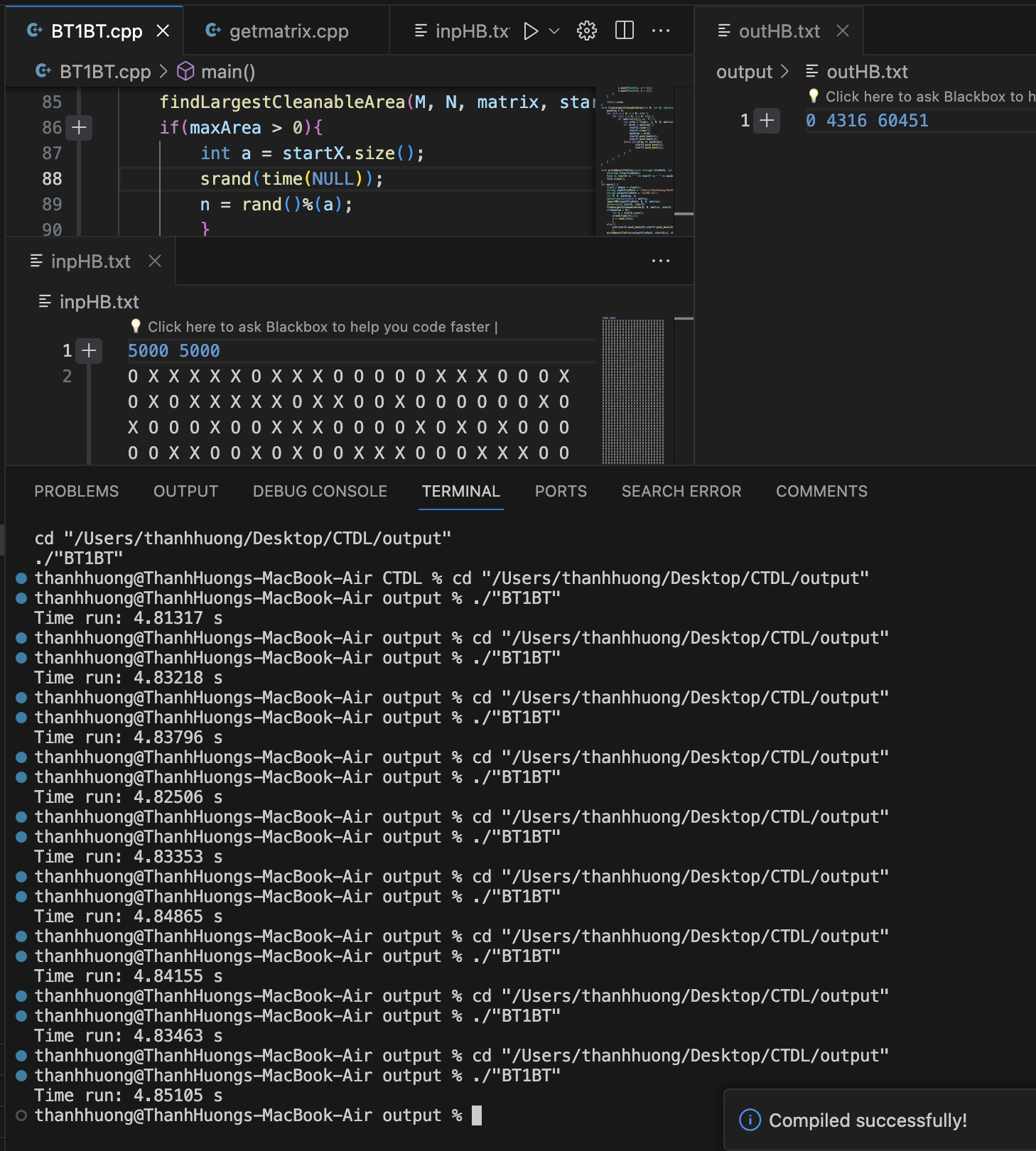
Với ma trận 5x5, ta tốn rất ít thời gian xử lý



Và ma trận cỡ 50x50. Với nhiều lần chạy, thời gian hoàn thành chương trình đâu đó vào khoảng 3-5ms.



Với ma trận 500x500 thì thời gian hoàn thành chương trình rơi vào khoảng 8ms.



Ma trận với kích thứo 5000x5000, ta có thể thấy trung bình thời gian hoàn thành chương trình từ 4 – 5s, và mỗi lần chạy không chênh nhau quá nhiều. Vì ta có thể thấy ở ảnh bên dưới, trong hàm Tìm O, ta sử dụng Stack để duyệt các ô liên tục.

