**Algorithm 1: The Island Problem**

Assume that you are given a two-dimensional array (a matrix) of potentially unequal height and

width containing only 0 and 1 s. Each 1 represents water, and each 0 represents part of a land mass.

A land mass consists of any number of 0s that are either horizontally or vertically adjacent (but

not diagonally adjacent). The number of adjacent 0s forming a land mass determines its size.

Note that a land mass can twist. In other words, it does not have to be a straight vertical line or a

straight horizontal line. For example, it can be L-shaped.

Design an algorithm that returns the size of the largest possible land mass after changing exactly

one 𝟏 to a 𝟎. Note that the given matrix will always contain at least one 1. The graph may be

mutated or transposed

1. Pseudocode

**function circularTour(city\_distances[], fuel[], mpg):**

**Initialize start to 0**

**Initialize requiredFuel to 0**

*// extra fuel is the fuel left*

**Initialize extraFuel to 0**

**n = length of fuel**

**for i from 0 to n-1 do:**

*// calculate extra fuel by subtracting the distance between the two cities and the fuel that is filled in that city*

**Add (fuel[i] \* mpg - city\_distances[i]) to extraFuel**

//

**if extraFuel < 0 then:**

// Update the starting point to the next city (i + 1)

**Set start to (i + 1)**

// Update the required fuel by adding the negative extra fuel

**Add extraFuel to requiredFuel**

// Reset extraFuel to 0 because now we are starting the index(i+1) as starting city

**Set extraFuel to 0**

// Check if there is enough total fuel (requiredFuel + extraFuel) to complete the circular tour:: using this as we will not traverse the already visited cities because we have calculated the required fuel from those cities which was negative

**if requiredFuel + extraFuel >= 0 then:**

**Return start**

**else:**

**Return -1**

1. Efficiency Class:

The algorithm has linear complexity of O(n). The Algorithm contains a single for loop that iterates through the n cities exactly once, where n is the length of the fuel array.

Explanation:

Operations Inside the Loop: Inside the loop, there are constant-time operations O(1):

Arithmetic operations like additions and subtractions take constant time.

Comparisons (if extraFuel < 0) and assignments also take constant time.

Conclusion: Since the loop iterates through the cities once and the operations inside the loop are constant time, the overall time complexity of the code is O(n), making it linear in terms of the number of cities.

Github Link contains code and Readme.txt file

<https://github.com/shriyab2099/Advanced_Algorithm-project_1-.git>