# Justify Your Choice of Data Structure

I decided to use an adjacency matrix for my data structure to compare connected vertices. My vector of Vertex attributes populates the adjacency. I chose this over using a tree for usability and debugging. An adjacency matrix is very easy to make and print to compare if you are getting your expected results. Writing into the adjacency matrix takes N^2 time with the nested for loops.

After I built the adjacency matrix I used a vector of structs to build my search table and do depth-first search. This was easier that doing a 3-dimensional array to loop back and forth and increment each touch until getting back to the 1st vertex’s 2nd touch. In my depth-first search I set the found vertex to the new vertex so it jumped line to line in the matrix and then stopped when it checked every line. I was worried doing this with recursion and 1000 graph point could fill up my memory on the stack so I stick with the while loop at would take up less space.

# Describe Your Algorithm

My algorithm broken down into the four main components that are all called from my main function.

## fillGraph

This is the first step in the problem, it will take the number of loons and loop through each one to input all vertices into the vector graphPoints.

## calcDistance

This is where all the calculations happen to take the inputted graphPoints and update the attributes of each vertex to the closest point and second closest points.

This is all done with attributes in the Vectex stuct. Each vertex will store its coordinates, two closest vertices, 1st and 2nd western and southern most points for tie breakers.

Nested for loops are used to loop through each vertex at each vertex, using N2 time. For each edge’s distance there are 4 options

* **Edge distance < closest distance**
  + if closest and second closest are equal
    - If western most point and second western most point =
      * If western and second western are equal do southern-most tiebreaker
        + if edge’s southern point < scndSouthPoint, set second closest = first and first equal to the new edge’s attributes
        + else, only replace the first closest with new edge attributes
      * else if western most point < scndWestern most point, set second closest = first and first equal to the new edge’s attributes
      * else, only replace the first closest with new edge attributes
  + else, set second closest = first and first equal to the new edge’s attributes
* **Edge distance = closest distance**
  + If edge’s x coord = current western most point
    - If edge’s y coord < current southern most point
      * If closest distance = second closest distance
        + If 1st and 2nd closest western most point are equal

If southern-most < scnd southern-most, set second closest = first and first equal to the new edge’s attributes

else, only replace the first closest with new edge attributes

* + - * + else if, western most < second western most, set second closest = first and first equal to the new edge’s attributes
        + else, only replace the first closest with new edge attributes
      * else, set second closest = first and first equal to the new edge’s attributes
  + else if, graph’s x coord < western most point
    - if closest = second closest
      * if 1st western most point = 2nd western most point
        + if 1st southern-most point < 2nd southern-most point, set second closest = first and first equal to the new edge’s attributes
        + else, only replace the first closest with new edge attributes
      * else if 1st western most < 2nd western most, set second closest = first and first equal to the new edge’s attributes
      * else, only replace the first closest with new edge attributes
    - else, set second closest = first and first equal to the new edge’s attributes
  + else, only replace the first closest with new edge attributes
* **Edge distance < second closest distance**
  + Replace the second closest with new edge attributes
* **Edge distance = second closest distance**
  + If edge’s x coord = 2nd western most point
    - If edge’s y coord = 2nd southern-most point, replace the second closest with new edge attributes
  + Else if edge’s x coord < 2nd western most point
    - Replace the second closest with new edge attributes

## fillAdjMatrix

Nested for loop to loop through all rows and columns of the adjacency matrix (initialized to all zeros) and set 1’s where any row connects to any column in the matrix

## depthFirstSearch

I do depth-first search to check if every vertex is strongly connected. The way I check this is doing N \*2 to compare if the search table’s first attribute’s touch2 = the maxTouch of N\*2

Maxtouch = graphPoints.size() \* 2

While touch <= maxTouch && searchTable[0].touch2 is not set

Do

If found in the adjacency matrix && inline == false

Check if this connection is already in the search table  
 If not in the search table, add it  
 Set I to j to move to the next row in the adjacency table  
 Set “in line” Boolean = true  
 j++

While j<graphPoints.size() && inline = false  
 if in line boolean = false (no connection found in the line)

Loop though the search table to find the last touch 2

If no touch2 found it will add the next touch to the last element in the search table  
else, add the next touch to the element before the touch found, increment touch

If searchTable[0].touch2 = maxTouch print yes (all strongly connected)

Else, print no