Audrey Clark
March 8, 2017
Assignment 4

CS 405: Algorithm Analysis II

Given the vertices of a convex polygon, I found the chords that create triangles between points with a minimal triangular perimeter.

My solution to this problem begins with creating a node and polygon class. The nodes of the graph are held in an ArrayList called "vertices" with an integer key denoting the vertex's identifier and corresponding node held in the ArrayList. Each node in the graph has an adjacency list represented by an ArrayList stored in the object itself.

To find the triangles in the polygon, I run through the file and declare all of the nodes with their given x and y values and their identifier (if there are n nodes, the identifier is a number between 0 and n - 1). With each node declared, I fill their list of differences with the other n - 1 nodes. This has a run time of $\theta(n^2)$.

I then create two 2-D matrices. One, chords[][], holds the third value of a triangle. For example, if there is a triangle between nodes 0, 1, 9, then chords[0][9] = 1. The other matrix, triSum[][], holds the perimeter value of the triangle held in the parallel matrix, chords[][]. For values when the column number is equal to the row number, I set the corresponding chords[][] point to -1. This denotes that there can't be a triangle between these points. The trivial triangles, where the difference between the column value and the row value are 2, are also filled in this step. This will have a run time of $\theta(n)$, as well, because all nodes will be traversed twice with no extra cost.

Now that the polygon is made, I test subproblems of the matrices. Running along the diagonals, I fill the two parallel matrices with the minimum triangle perimeter option. This has a run time of $\theta(n^3)$ because each node will need to test the other remaining nodes, as well as checking previous subproblems.

After I'm finished with this, I must print out the minimal perimeter value of the triangulation as well as print out the inner chords I've chosen for my polygon. These two steps are less than $\theta(n^2)$, because for each node, they only need to traverse less than n other options.

As you can see from the calculations above, this algorithm has a run time of $\theta(n^3)$, as expected.