Computer Science 331 – Assignment 4

Methodology:

I divided my project into thee methods. The getGraph method reads the csv file specified by getFileName() and interprets characters in the string until two numbers are discerned. This is done via the two while loops. The method then takes these two numbers and adds the appropriately to the matrix and adjacency list. The next line of the file is then read, and the file continues to be read until all lines have been incremented through. My calcMpv and calcLpv methods create arrays that contain the vertices with the most or least neighbors respectively. It does this by checking every vertices one by one through the adjacency list created earlier and compares the size of each list corresponding to the vertices with the current max or min. If a new minimum or maximum is found the array containing these vertices is erased and new values are added. Upon termination the array will contain all vertices with the max or min number of neighbors. I created an addition initializeMatrix method that simply inputs default values for the matrix, assigning zero to all edges.

Worst Case Cost:

**initializeMatrix Method:**

The test has 2 comparisons (> or =) and 1 method call, it is checked p + 1 times so there is max 3(p + 1) instructions

Worst case cost for execution of loop body: (1 + inner while loop)

Upper bound on worst case cost to execute loop: 3(p + 1) + (1 + inner while loop)p

Total cost of initializeMatrix method: 3(p + 1) + (1 + inner while loop)p + 2

**Inner while loop:**

The test has 2 comparisons (> or =) and 1 method call, it is checked q + 1 times so there is max 3(q + 1) instructions

Worst case cost for execution of loop body: (12)

Upper bound on worst case cost to execute loop: 3(q + 1) + (12)q

*Total cost of initializeMatrix method: 3(p + 1) + (1 + 3(q + 1) + (12)q)p + 2*

**calcMpv Method:**

The test has 1 comparison and 1 method call, it is checked n + 1 times so there is max 2(n + 1) instructions

Worst case cost for execution of loop body: (15)

Upper bound on worst case cost to execute loop: 2(n + 1) + (15)n

*Total cost of calcMpv method: 2(n + 1) + (15)n + 1*

**calcLpv Method:**

The test has 1 comparison and 1 method call, it is checked m + 1 times so there is max 2(m + 1) instructions

Worst case cost for execution of loop body: (15)

Upper bound on worst case cost to execute loop: 2(m + 1) + (15)m

Total cost of calcLpv method: 2(m + 1) + (15)m + 1

**getGraph Method:**

The test has 1 method call, it is checked r + 1 times so there is max (r + 1) instructions

Worst case cost for execution of loop body: (30 + innerloop1 + innerloop2)

Upper bound on worst case cost to execute loop: (r + 1) + (30 + innerloop1 + innerloop2)r

Total cost of getGraph method: (r + 1) + (30 + innerloop1 + innerloop2)r + 3

**Inner while loop 1:**

The test has 1 comparison and 1 method call, it is checked d + 1 times so there is max 2(d + 1) instructions

Worst case cost for execution of loop body: (5)

Upper bound on worst case cost to execute loop: 2(d + 1) + (5)d

**Inner while loop 2:**

The test has 1 comparison and 1 method call, it is checked t + 1 times so there is max 2(t + 1) instructions

Worst case cost for execution of loop body: (5)

Upper bound on worst case cost to execute loop: 2(t + 1) + (5)t

*Total cost of getGraph method:* (r + 1) + (30 + 2(d + 1) + (5)d + 2(t + 1) + (5)t)r + 3

Calculating Big O:

**Proof of O(n^2) for initializeMatrix method:**

*3(p + 1) + (1 + 3(q + 1) + (12)q)p + 2 = 3p + 3 + q + 3p(q + 1) + 12qp + 2 = 6p + 15qp + 3 + q*

*In this case p = q thus 6p + 15p^2 + 3 + p = 15p^2 + 7p + 3 < 15p^2 + 7p^2 + 3p^2 = 25p^2*

*Thus O(n^2) for c = 25 and n >= 0*

**Proof of O(n) for calcMpv method:**

*2(n + 1) + (15)n + 1 = 2n + 1 + 15n + 1 = 17n + 2 < 17n + 2n = 19n thus O(n) for c = 17 and n >= 0*

**Proof of O(n) for calcLpv method:**

*2(m + 1) + (15)m + 1 = 2m + 1 + 15m + 1 = 17m + 2 < 17m + 2m = 19m thus O(n) for c = 17 and n >= 0*

**Proof of O(n^2) for getGraph method:**

(r + 1) + (30 + 2(d + 1) + (5)d + 2(t + 1) + (5)t)r + 3 = (r + 1) + (34 + 7d + 7t)r + 3 = r + 1 + 34r + 7dr + 7tr + 3

= 35r + 4 + 7dr + 7tr thus *O(n^2)*

Differences in file size, and memory usage:

The adjacency matrix is much larger than the adjacency list because the graph is sparse and the matrix must assign a 0 to all pairs of vertices that are not neighbors whereas the adjacency list only focuses on those that are neighbors.

The adjacency matrix uses more ram in this implementation because it requires we acknowledge every possible pair of vertices whereas the list only requires we focus on those neighbors that do exist.

Output in command Line:

Enter number of vertices

256

Enter filename

/CPSC331/GraphEdges\_256x3000.csv

Number of neighbors for MPV: 35

MPV neighbors:

21, 11, 16, 22, 24, 38, 49, 56, 60, 79, 81, 83, 88, 95, 98, 121, 125, 128, 135, 136, 143, 146, 149, 155, 162, 167, 177, 184, 186, 192, 202, 211, 215, 219, 232, 253

Number of neighbors for LPV: 12

LPV neighbors:

29, 15, 67, 68, 71, 88, 116, 144, 163, 173, 193, 206, 240

118, 1, 18, 20, 72, 82, 100, 117, 132, 173, 177, 232, 235