Project 3

Ayush Singh (G29283616) | ayushsingh@gwu.edu

# Problem Statement

Diameter of a graph is defined as the largest distance between any pair of vertices of G.  Give an efficient polynomial algorithm to find the diameter of the graph.

# Implementation Characteristics

Diameter of a graph, as described in the problem, is basically the largest distance between any 2 pair of points in an undirected but weighted graph. If we recall, we already know the Algorithm to find out the shortest path between the vertices of a graph. The algorithm is also known as Floyd Warshall’s Algorithm, but it does not work with negative weights. Although, we can easily apply that algorithm to find out the weights of all the pair of vertices between each pair, whether directly or through another vertex. As we find all the weights between each pair (cost to reach from v1 to v2), we just have to tweak the code a little bit. As we find the weights between each vertex, we select the largest weight (cost) and display it as the answer because the largest weight between 2 vertices will become the maximum cost paid to get from v1 to v2 or the largest distance between v1 and v2, which can be considered as the diameter of the graph. In the following code, which has been omitted due to spatial reasons, display function shows the Updated weight matrices/graphs and after every function calling in the code, also displays the largest number which is essentially the largest distance between any 2 vertices. Hence, the diameter of the graph is calculated after numerous attempts and updating the weight (cost) matrix multiple times. The time complexity of the algorithm is O(n3).

# Experimental Analysis

## Program Listing

void apsp::shortestpaths()

{ int i,j,k;

for(i=1;i <= vertices;i++)

{ for(j=1;j <= vertices;j++)

{ A[i][j]=weight[i][j]; }

}

for(k=1;k <= vertices;k++)

{ for(i=1;i <= vertices;i++)

{ for(j=1;j <= vertices;j++)

{

if(A[i][j] > (A[i][k]+A[k][j]))

{

A[i][j]=A[i][k]+A[k][j];

}

}

}

display();

}

}

Data Normalization Notes

We need to find the average of the theoretical values and experimental values. The Scaling constant, as always, refers to the ratio of the average values. In order to differentiate between the curves of theoretical and experimental values clearly, we need to adjust the theoretical values. The scaling constant is 2.09911 x 10-6.

## Output Numerical Data

|  |  |  |  |
| --- | --- | --- | --- |
| N | Experimental (ns) | Theoretical (n3) (ns) | Adjusted Theoretical Value (\*2.09911 x 10-6) |
| 1000 | 2930 | 1000000000 | 2099.10948 |
| 1259 | 4487 | 1995616979 | 4189.018519 |
| 1442 | 6373 | 2998442888 | 6294.059892 |
| 1587 | 8295 | 3996969003 | 8390.075526 |
| 1709 | 9894 | 4991443829 | 10477.58706 |
| 1817 | 12063 | 5998805513 | 12592.14952 |
| **Average** | **7340.34** | **3496879702** |  |
| **Ratio** |  | **2.09911 x 10-6** |  |

## Graph

## Graph Observations

The graph above shows that both the lines coincide very well but there are some value of n where the lines do not coincide. This shows that even though, the theoretical values seem very fitting, the experimental values still have some sort of deviation. But, the overall hue of both the graph lines indicate that the time complexity calculated must be similar to that which was calculated.

# Conclusions

The analysis of the algorithm as O(n3) seems to be supporting the graph. May be a better analysis of the scatterplots can be revealed if we change programming environments, but this graph seems to give the user a good idea that the complexity, nevertheless, is O(n3).