Name: Naga Venkata V Vinnakota
Netid:nvv13
Course: ECE 16:332:573 DSA

EXAM-2

Question1:

Compute the
(i) **number of connected components**
(ii) **size of the largest component,**
(iii) **number of components of size less than 10 for data provided at:**
https://algs4.cs.princeton.edu/41graph/movies.txt .
Estimate the runtime of your algorithm.

Solution:

I have tried implementing the graph as a dictionary rather than creating a class for graphs and then using the methods. As soon as the program can read a line, it divides all the strings separated by '/' and tries to add all the strings as the vertices of the graph. By doing so I was able to build a graph by linking all the vertices provided in the graph. I assumed that a movie and a performer are linked if the line starting with movie name has that performer.

I used DFS to traverse through the graph. By using DFS I was able to reach all the vertices from a given vertex and thereby making them a component. I used the fields count, id_count and id_ to store the number of connected components, a list giving the component id for each vertex and a dictionary to map each vertex with its component id.

From my implementation I observed that there are nearly 37 components present in the data set provided.

I created a Dictionary, components_count, that holds the component id and size of each component, with component id as the key. I calculated the size of the largest component using the components_count dictionary. Using the same dictionary, I calculated the number of components that have size less than 10.

The results for the same are as below:

Results:

Importing data from movies.txt

Graph created successfully for nearly 4189 movies and the performers of each movie

Starting the timer in nano seconds

Number of vertices visited: 121317

The total number of connected components in the given graph is 37

Time taken to run DFS on the graph is 441342200 ns

The size of the largest component is 120651

A total of 9 components have size less than 10

The total runtime after the graph is created is 76616445700 ns

Observation:

From the above results of the program I created, we can understand that the algorithm I created can find the connected components if any and prints the number of components available in the graph, Prints the size of the largest component in the graph and the number of components that have the size less than 10 in around 76 seconds.

But we can see that DFS is executed for such a large graph in just 441342200 ns i.e. nearly 0.4413 seconds. By using the length of the list id_, I got the total number of vertices in the graph as 121317. This gives us a clear idea DFS consumes as less time as possible. I got the order of runtime for DFS nearly as O(V), where V is the number of vertices present in the graph.

But the total run time is of order nearly O(V^2), for the given data set.

Conclusion:

We can conclude from the result of this experiment that the run time of DFS on a graph is a linear function of the number of vertices present in the graph. The total runtime of the program after creating the graph is nearly of order O(V^2) i.e. it is quadratic function of the number of vertices present in the graph.

Question2:

Write a program that computes the "diameter" of a directed graph which is defined as the maximum-length shortest path connecting any two vertices. Estimate the runtime of your algorithm. Feel free to use the following datasets:
https://algs4.cs.princeton.edu/44sp/tinyEWD.txt
https://algs4.cs.princeton.edu/44sp/mediumEWD.txt
https://algs4.cs.princeton.edu/44sp/1000EWD.txt
https://algs4.cs.princeton.edu/44sp/10000EWD.txt

Solution:

After observing all the data sets, we can see that all the edge weights are positive. We know that Dijkstra's algorithm can compute the shortest path from a source. I have used the same fact and created

an algorithm that tries to compute the maximum shortest path a vertex can achieve using Dijkstra's algorithm.

This algorithm I made will have to be run on all the vertices as the graph is a digraph. By doing so we can compare the maximum length shortest path i.e. diameter of the digraph possible from each vertex. The maximum of all these would be the diameter of the digraph we require.

I have used a function FindPath() , which computes the shortest path from the source to destination. The function takes the list which contains the path from s to the destination and the destination as the arguments. This function will be called in dijkstra() function so that we can directly use the result we obtained in the function dijkstra(). The path returned by the FindPath() is compared with the maximum shortest path the source has and updates it accordingly. The function diameter () takes the distance list as parameter and returns the maximum value of the list.

We must run the Dijkstra's algorithm on each vertex. So, the expected time complexity is v*time complexity of Dijkstra's algorithm which is of order O(v^2) in general, where v is the number of vertices present in the graph.

From the above discussion the expected time complexity can be of order O(V^3)

The result of the program is as below:

Results:

For data set: tinyEWD.txt

Importing the data

Graph created successfully

Starting the timer in ns

The results of the algorithm are as below:

The maximum shortest path in the graph is as below:

[3, 6, 2, 7, 5, 1]

The Diameter ends are as follows:

The starting point is 3 and the end point is 1

The length of diameter is **1.86**

Run time of the algorithm is: **0** ns

For data set: mediumEWD.txt

Importing the data

Graph created successfully

Starting the timer in ns

The results of the algorithm are as below:

The maximum shortest path in the graph is as below:

[123, 246, 8, 212, 156, 196, 181, 71, 62, 128, 136, 234, 61, 111, 25, 63, 237]

The Diameter ends are as follows:

The starting point is 123 and the end point is 237

The length of diameter is **1.37466**

Run time of the algorithm is: **2911290800** ns

For data set: 1000EWD.txt

Importing the data

Graph created successfully

Starting the timer in ns

The results of the algorithm are as below:

The maximum shortest path in the graph is as below:

[13, 662, 90, 155, 509, 531, 947, 497, 264, 670, 148, 195, 518, 447, 402, 987, 214, 788, 219, 577, 346, 167, 620, 366, 697, 92]

The Diameter ends are as follows:

The starting point is 13 and the end point is 92

The length of diameter is **1.39863**

Run time of the algorithm is: **190919194200** ns

Observations and analysis:

From the above run times calculated, we can check if the run times of order O(V^3)

For v=8:

O(V^3)=512

For most of the runs, I got run time as 0 ns but for few runs, I observed the value to be nearly 996700.

Therefore, I considered run time as 0

For v=250:

O(V^3)=15325000

Run time =2911290800

i.e. f(n)= 2911290800

c=f(n)/O(f(n))=186.32

For v=1000:

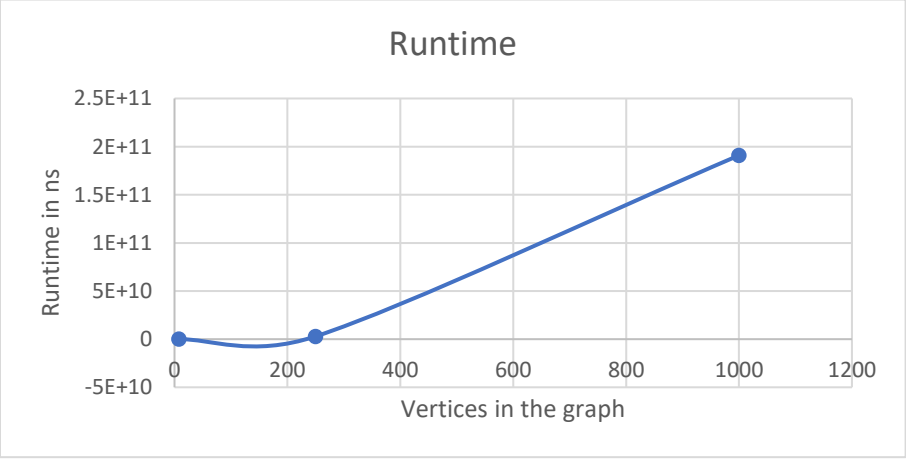
O(V^3)=1000000000

Run time = 190919194200

i.e. f(10000=190919194200

c=f(n)/O(f(n))=190.9

The below graph shows the run time of each data set used.



We can see that the increase is exponential as the number of vertices and edges increase.

Conclusion:

The algorithm takes exponential time to compute the diameter. The runtime is of order O(V^3) for V vertices in the graph.