Internet Weakness Analysis

New Mexico Supercomputing Challenge

Final Report

April 4th, 2007

Team 53

Los Alamos High School

Team Members:

Kirat Pandya

Calvin Loncaric

Paco Venneri

Gabe Vigio

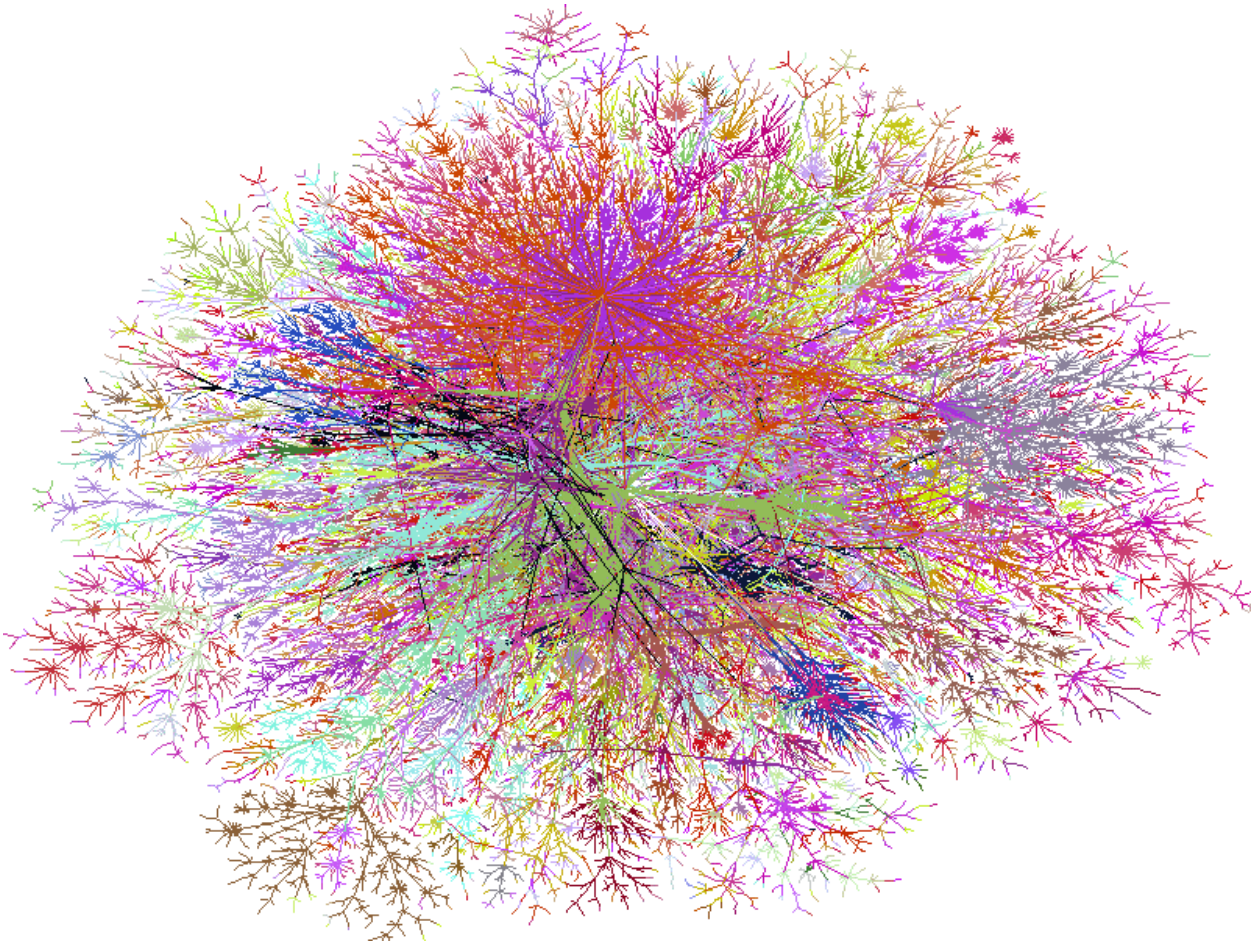
Teacher:

Mrs. Diane Medford

Mentors:

Elena Georgi,

Matthew Sottile



Map of the internet created by Bell Laboratories, based on the same data used in this project

Table of Contents:

6.3. **Executive Summary**

The internet has grown enourmously and most of today's worldwide commerce is based on it. Terrorist of the future will be "cyber terrorists", taking down important nodes (i.e. interconnecting points) of the internet, thereby debilitating large sections of the network, imacting the security and stability of our entire world

As the internet spreads, network density end up being unevenly distributed, with some areas of the network having a high density of links and other areas of the network sparsely connected. This leads to the creation of critical network nodes that handle the traffic between these unevenly distributed, large subsections of the internet. The decentralized structure of the internet makes this extremely difficult to manage.

To protect ourselves from such terrorist attacks, we need to know which such critical nodes of the internet we need to protect.

"Internet Weakness Analysis" is a project that attempts to locate these nodes.

Our data comes from the Internet Mapping Project (http://www.cheswick.com/ches/map/index.html). It started at Bell Labs in the summer of 1998. Its goal is to acquire and save internet topological data over a long period of time. This data has been used in the study of routing problems and changes, Distributed Denial of Service (DDoS) attacks, and graph theory. The project provides us with an edge-connectivity list, i.e. with information on which computer has a direct link to which other computers/network centers.

Our program performs a thorough weight (i.e. importance) analysis of the data to compute the exact weight of each node on the internet, thereby providing a ranking, by importance, of nodes. The resultant data can be further process, and by applying the Max Flow, Min Cut Theorem, we can also figure out the maximum flow between any two major chunks in a given network, which in turn has countless applications; for example, finding the maximum flow between two major networks.

1. Introduction

The internet has grown enourmously and most of today's worldwide commerce is based on it. Terrorist of the future will be "cyber terrorists", physically taking down (i.e. attacking and damaging) important nodes on the internet, thereby debilitating large sections of the network. It is time we take a deeper interest in protecting mankind’s greatest creation to date

1.1. Purpose

The purpose of this project is to locate the nodes of the internet, which if disconnected, would cause the greatest disruption of service across the network. The program utilizes established principles of Graph Theory to compute this information.

1.2. Significance

The internet is a godsend for those who are involved in intelligence activity. But the expansion of information technology also poses a threat. Terrorist networks, such as Al-Qaida, can easily get hold of detailed information. The Internet also increases the vulnerability of authorities, organizations and companies. The ability of terrorists to carry out attacks on the critical infrastructure of society may increase. The cause is the way society is developing in the area of information technology and also the generally available knowledge of basic technology and methods which exists in Computer Network Intelligence (CNI). CNI here means intelligence activity where the gathering of open or secret information takes place from computer networks using computers or computer networks.

As our day-to-day life becomes more and more dependant on the internet, it has become a possession the damage to which means damage to our national security. Phone systems, banking, transport, and most importantly, the intelligence agencies and other national defense organizations rely heavily, if not completely, on the internet. The destruction of a few critical nodes would result in the overload of the rest of the infrastructure, thereby bringing our country to a standstill.

1.3. Background

The idea of "cyber-terrorism" has been around for a while, but "physical cyber terrorism" has been given little thought, compared to "virtual cyber terrorism" (as in hacking). The concept has been around for a while, but it has been believed that terrorists would rather perform a "clean", virtual attack and gather information, rather that physically destroy networking centers. However, with our economic system so dependant on the internet, and the internet so under-protected, it would make it easier for a terrorist to cause nationwide disruption by destroying important internet nodes, rather that virtually targeting public services.

2. Description

2.1. Scope

The project aims at locating the critical nodes of the internet using a mathematical and graphical analysis of the topology of the internet. Although the main concept behind the project was the internet , our computational model is designed to perform weight analysis on any form of network topology. Hence, it can be applied to other computer, telecommunication and transportation networks also.

2.2. Materials

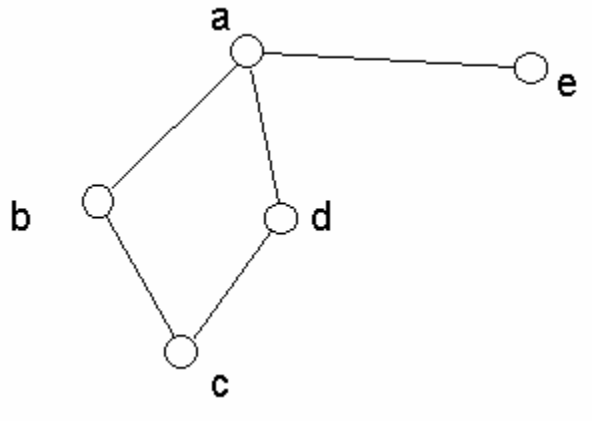
The program has been written for being used on a supercomputing cluster. The memory and processing power requirements for the computing the data of even a network with just a few thousand nodes, is too much for any modern workstation or server-grade computer alone to handle. Also, since a computational cluster is an expensive and rare tool to have access too, the program has been written in such a way that any regular network of computers, including those which may be at the other end of the internet, can be grouped together and used as a cluster, without the need for any additional software or other modifications.

2.3. Methods

The internet can be represented by Graph Theory. Graph theory is a branch of mathematics which allows for easy analysis of large networks. Unlike Euclidean geometry, in graph theory each vertex is defined only by the other vertices it is connected to, instead of by its absolute coordinates. The position of a vertex does not matter, only its relation to other vertices. Connections between vertices are called edges. Edges may have weights, representing their importance. A higher weight means a higher importance.

The problem can best be approached using the max-flow min-cut theorem, derived from Menger's theorem. The max-flow min-cut theorem states that the maximal amount of flow is equal to the capacity of a minimal cut. In layman terms, the theorem states that the maximum flow in a network is dictated by its bottleneck. Between any two nodes, the quantity of material flowing from one to the other cannot be greater than the weakest set of links somewhere between the two nodes. Therefore, if the most important links of the internet are disconnected for whatever reason, then the maximum capacity of flow will be restricted to a crawl, thereby making the internet practically useless.

2.3.1. Mathematical Model



Above is a sample undirected graph, consisting of nodes a, b, c, and d. Let us assume that each node is a computer system that generates its own traffic of, on average, '1' along each of its links.

Let W(x, y) represent the weight of the undirected edge between the nodes x and y,
Therefore,
W(a, b) = W(b, c) = W(c, d) = W(d, a) = W(a, e) = 1

Here, the weight of an edge represents the average amount of traffic flowing through it at any given point in time.
Now, since each node is also interconnects each of the edges connected to it, therefore, the weight of each edge is the sum of its own weight and the weights of the other edges connected to the nodes at either end of that edge. The weights do not add progressively, but rather two versions of the graph are maintained, initial and final.

Therefore, calculating the weights gives us,
W(a, b) = 4
W(b, c) = 3
W(c, d) = 3
W(d, a) = 4
W(a, e) = 3

In the case of this project the weights of each edge are unimportant. What is needed is the weight of each node. The weight of each node can be obtained by the sum of the weights of the edges connected to it.

Let 'W_x' be the weight of the node x.

Therefore,

$W_a = 3 + 4 + 4 = 11$

$W_b = 4 + 3 = 7$

$W_c = 3 + 3 = 6$

$W_d = 3 + 4 = 7$

$W_e = 3$

Based on the above analysis, we can rank the importance of the nodes for the sanctity of the network as follows:

$a > (b = d) > c > e$

This project applies the above mathematical model to a similar graph of the internet, and ranks the nodes in the order of importance based on the average traffic they handle at any point of time

2.3.2. Computational Model

The mathematical model has been implemented using the Java programming language. The program uses Remote Method Invocation (RMI) to operate simultaneously across all the computers of a cluster. One computer is set up as the "server" and all the other nodes function as "clients" (The terms are reversed as we made a major decision change midway during development). The server passes each client a path to a data file and the number of to ranked nodes it needs returned. Each client performs its calculations and returns the result to the server for final compilation.

The client side of the program has been written with newer multi-core processors in mind, and the program dynamically creates the same number of threads as the number of processors available in each client, thereby taking advantage of the latest technology to achieve the greatest possible performance

2.3.2.1. Raw Data

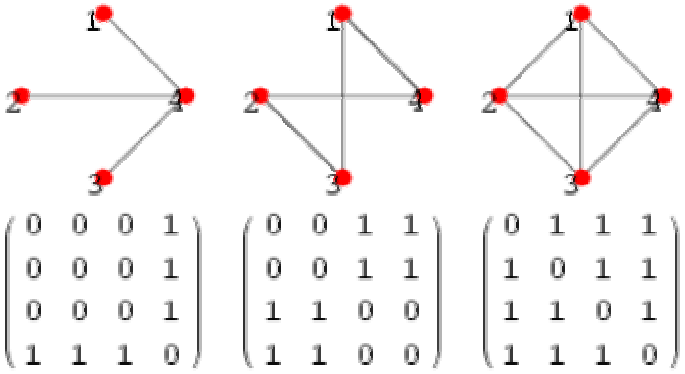
The raw data file obtained from the Internet Mapping Project is structured as follows:

<IP address 1> <IP address 2>

This indicates that IP1 is connected to IP 2 directly. The data file has about 300,000 such records.

2.3.2.2. Graph representation

The program reads through the data file and counts the number of records. It then creates and adjacency matrix. "The adjacency matrix of a simple graph is a matrix with rows and columns labeled by graph vertices, with a 1 or 0 in position (v_i, v_j) according to whether v_i and v_j are adjacent or not. For a simple graph with no self-loops, the adjacency matrix must have 0s on the diagonal. " (Wolfram Mathworld)



Although a computer does have a "loopback" address through which it can communicate with itself, this loop connection bears no importance in this mathematical model since it does not affect the weight of any node.

For an undirected graph, the adjacency matrix is symmetric. However, if a traditional adjacency matrix were to be created for our data, its initialization size would be:

Number of columns = Number of rows = 2 * <no of lines in data file> – (Equation 1)

Although this size could later be cut down (as for a network to exist, it is not possible to have two unique IPs on every line of the data file), It would have to be initialized to its maximum required size.

So, in order to reduce the size of the adjacency matrix after optimization, the computer program does the following

- It counts the number of records (lines) in the data file
- It creates a two dimensional array in the computer's memory of
  size = (2 * <no of lines in data file>) * (2 * <no of lines in data file>)
- The program then goes through each line
  - Reads the two IP addresses in each line
  - Passes them to a function that converts the IP into a base 255 number (a serialized IP)
  - The serial for <IP address 1> is added to the y axis, and the serial for <IP address 2> is added to the x axis, and at (<index of IP address 1>, <index IP address 1>) in the adjacency matrix, it sets the weight to 1

- The serialized IP is actually a hash of the IP and cannot be converted back to the IP. Hence, an "IP to Serialized IP" lookup table is also maintained in memory

The item of note here is that each on each line is not added to both x and y axis at the same time. This causes the adjacency matrix to be non-symmetric, and invalidates most of its properties. However, it also has a major advantage.

Let us assume that there are *n* lines in the data file

Therefore, according to equation 1, N_{max} (maximum possible no of IPS) = n*2,

Now if each IP were to be added to each axis, the number of rows and columns used of the adjacency matrix would be close to $(3/4)^{th}$ of N_{max}.(Experimentally derived value)

However, if the algorithm implemented in our program were to be used, then, the number of rows and columns used of the adjacency matrix would be close to $(1/2)^{th}$ of N_{max}.(Experimentally derived value). Therefore, equation 1 can be safely optimized to

Number of columns = Number of rows = <no of lines in data file> – (Equation 2)

This algorithm, as mentioned above, invalidates most of the established properties of a symmetric adjacency matrix, and hence, we had do create our own algorithm for calculating the weight of each edge.

2.3.2.3. Time – Memory tradeoff

The size of the available data file is 300,000 lines. Hence, according to equation 2, the size of the adjacency matrix required to be initialized at first (although it can be optimized to some extent after being filled with connection data) would be:

Size = 300,000 * 300,000 = 90,000,000,000 records

Now, in order to hold the serialized IPs, the array (i.e. adjacency matrix) needs to composed of integers, each of which takes 4 bytes memory. Although many IP serials overflow (into a negative number) the signed limit of an integer, +2147483647, the overflow never hits the lowest negative value, -2147483648, and hence there is no need to have an array of double values. Even then, the size of the array would be,

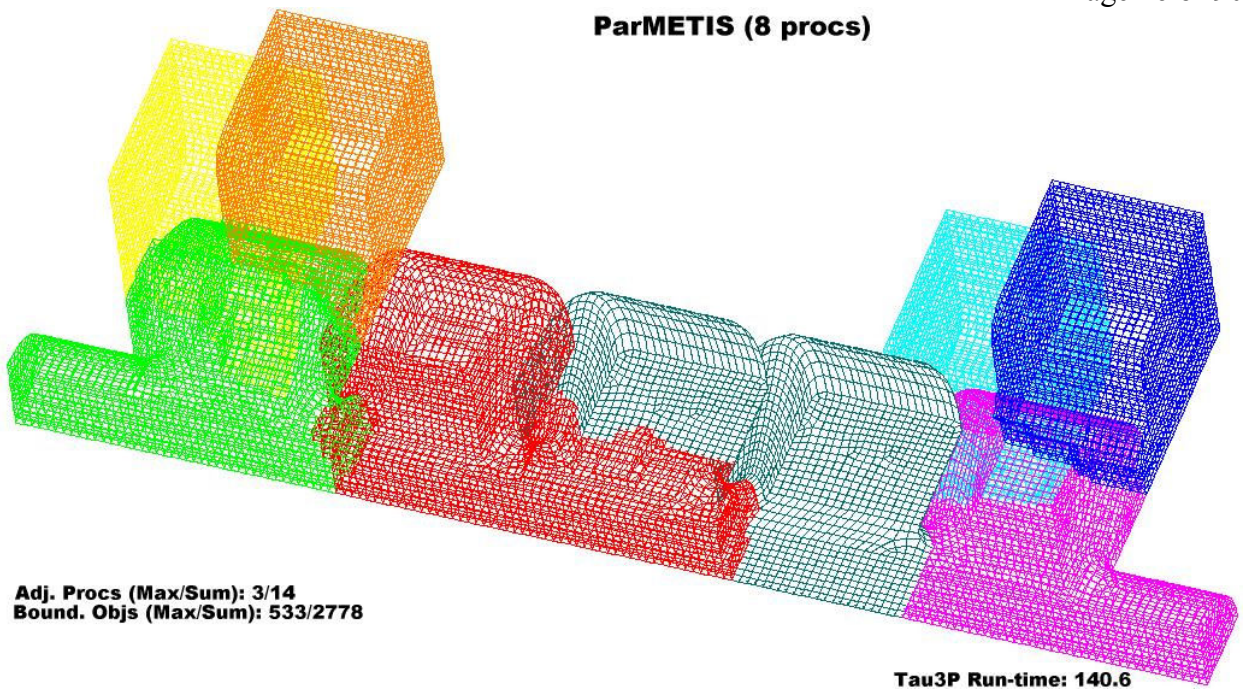
Size (in memory) = 90,000,000,000 * 4 bytes = 360000000000 Bytes
= 360000000 KiloBytes
= 360000 MegaBytes
= 360 GigaBytes (More than the hard drive size of most computeres!)

We had to create the program for a supercomputing cluster. However, if we were just cut the data file into 25 pieces directly, for a 25 node cluster, then there was no guarantee that the data for any node would represent a proper network, rather than a bunch of disconnected nodes.

If we were to break the adjacency matrix into pieces and place a piece in the memory of each node of the cluster, then the amount of network traffic generated would bring the cluster to almost a standstill, since for the calculation of the weight of each edge, the program has to cycle through an entire column and row of the adjacency matrix 4 times. (Algorithm explained in the next section. This ruled out the use of existing technologies such as MPI (Message Passing Interface).

Hence, we had to make a reverse Time-Memory tradeoff. Mr. Sottile, our mentor, suggested that we look into METIS.

METIS is a family of programs for partitioning unstructured graphs and hypergraphs and computing fill-reducing orderings of sparse matrices. The underlying algorithms used by METIS are based on the state-of-the-art multilevel paradigm that has been shown to produce high quality results and scale to very large problems.



The above picture represents a 3D mesh. The coloring indicates the eight pieces that METIS has cut the graph into. In this same way, we used METIS to divide up our graph into smaller chunks so that each client in the cluster could work on a manageable chunk. The most important function of METIS is that it cuts the graph in such a way as to minimize the cut (i.e. the number of connections cut between one chunk and others around it). METIS also marks each interconnection between chunks with a unique number, but because of the way METIS partitions graphs, the most important nodes of each chunk are closer to the center of the chunk and hence these interconnection points can be safely ignored

So, if the program is to be run on a 25 node cluster, METIS is first used to partition the main data file into 25 pieces. This pre-partitioning process takes time, but by trading the time, we were reducing the memory requirements of the program to a manageable limit.

2.3.2.4. Calculation Algorithm

The program implements the mathematical model discussed previously as follows For each edge (x, y)

- The program runs a loop from i = 1 to i = the number of columns of the array, adding the weight of each edge (i, y) to a temporary variable, except the weight of the edge (x, y). This step adds the weight of all the edges connected to node y.

- The program runs a loop from j = 1 to j = the number of rows of the array, adding the weight of each edge (x, j) to the same temporary variable, except the weight of the edge (x, y). This step adds the weight of all the edges connected to node y.
- Now, since the graph is not symmetric, node x may or may not reappear along the y axis. Hence, the program looks through the y axis for node x
- If node x is found, the index of that location on the x axis is saved to a variable and then The program runs a loop from i = 1 to i = the number of columns of the array, adding the weight of each edge (i, <index on node x on the y axis>) to the temporary variable, except the weight of the edge (x, y). This step adds the weight of all the edges connected to node y.
- Finally, the weight of the edge (x, y) is added to the temporary variable
- The value of the temporary variable is then set as the weight of the edge (x, y) in a copy of the adjacency matrix. A second copy of the matrix is maintained so that the weights do not add progressively, skewing the calculations

After the weight calculations are finished for all edges, the program creates a table of the structure
<serialized IP> <weight>

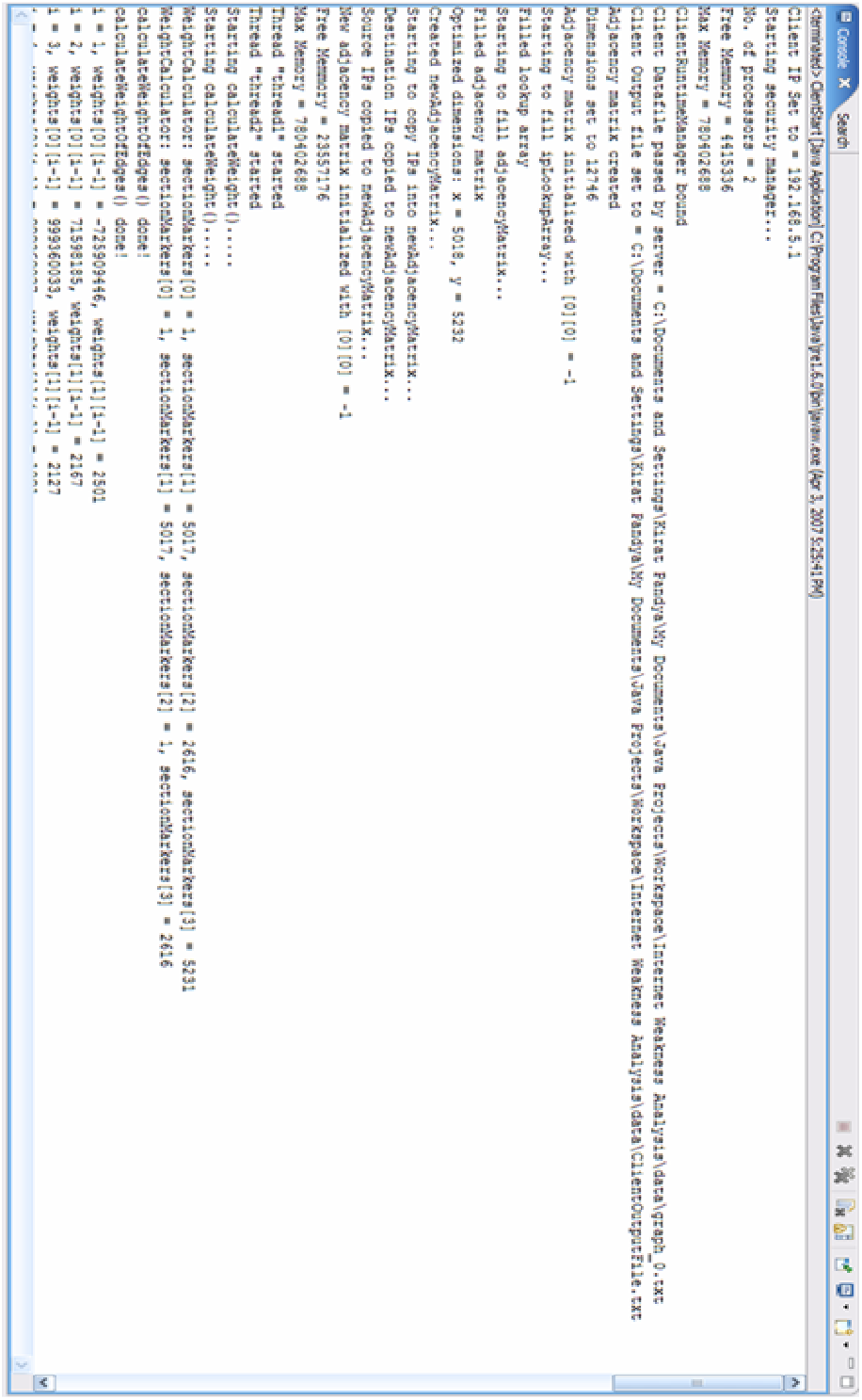
- The programs loops through all the nodes along the y axis of the new adjacency matrix, adding all the weights for each node (i.e. sum all the values along each row is the weight of that node) and then adding the node and its weight to the array.
- loops through all the nodes along the x axis of the new adjacency matrix, adding all the weights for each node (i.e. sum all the values along each column is the weight of that node) and then adding the node and its weight to the array.
- The table is then arranged in descending order based on weight
- The client then returns, to the server, a table with the requested number of most weighted nodes in descending order.

The server then goes through all such tables it receives from each of the nodes, compiles a large table of the same structure as the client tables, with all the data of the received tables in it. The server then sorts this table in descending order, based on weight, and outputs the requested number of most important nodes to the screen and an output file.

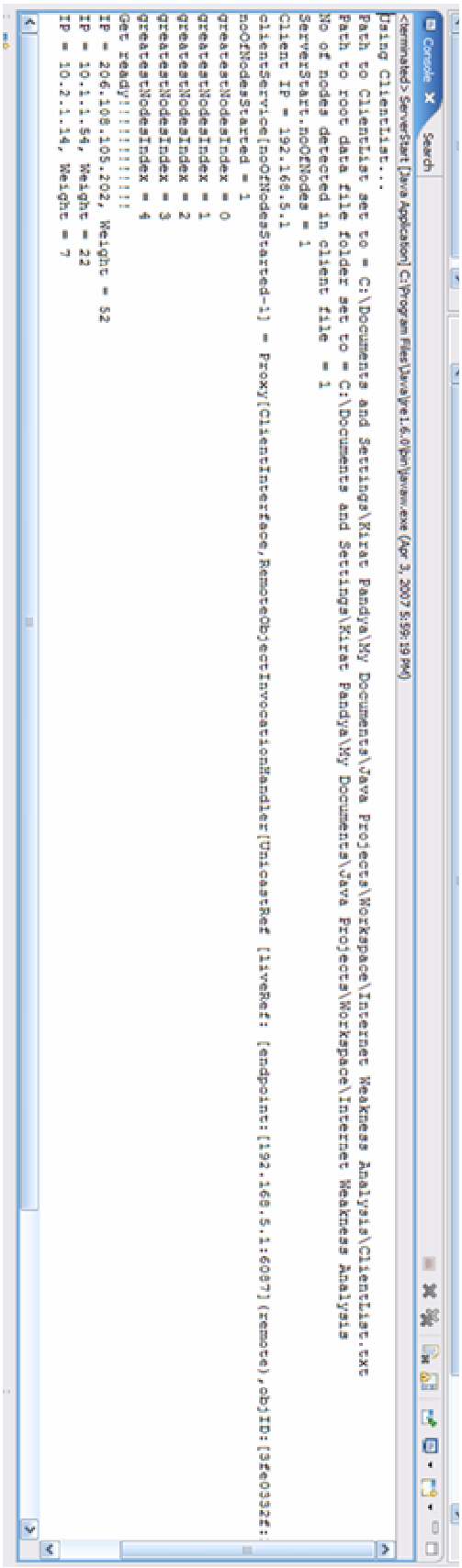
Results

The program was run on a 25 node cluster of linux computers with 512 MB of RAM, AMD 3200 processors, and 5GB swap (i.e. hard drivs space that complements the RAM) partitions. Below is a screenshot of a test run of the client. Final run output is too large to be effectively displayed here.

On-screen output



from the server during a test run with a small file. Final run output is too large to be effectively displayed here.



ClientOutputFile.txt - Notepad

File Edit Format View Help

```
Adjacency Matrix:
-1              -831755893      -831755830      -831757818      1029574637
167838006               1               1               1               1
174361602               0               1               0               0
167903502               0               1               0               0
167903690               0               1               0               0
167903698               0               1               0               0
167903722               0               1               0               0
167903554               0               1               0               0

IP Lookup Table:
1                       1
10.1.1.54                       167838006
206.108.105.139                 -831755893
206.108.105.202                 -831755830
206.108.98.6                    -831757818
61.94.15.237                    1029574637
10.100.140.2                    174361602
10.2.1.14                       167903502
10.2.1.202                      167903690
10.2.1.210                      167903698
10.2.1.234                      167903722

New adjacency matrix:
-1              -831755893      -831755830      -831757818      1029574637
167838006               4               10              4               4
174361602               0               7               0               0
167903502               0               7               0               0
167903690               0               7               0               0
167903698               0               7               0               0
167903722               0               7               0               0
167903554               0               7               0               0
```

Above is the output from a client during a test run with a small graph. This output is disabled when running large files since the time taken to write a huge adjacency file to disk is too long.

To the right is the screen shot of the output file from the server, listing the top IP addresses and their weights. This is the final output of our project

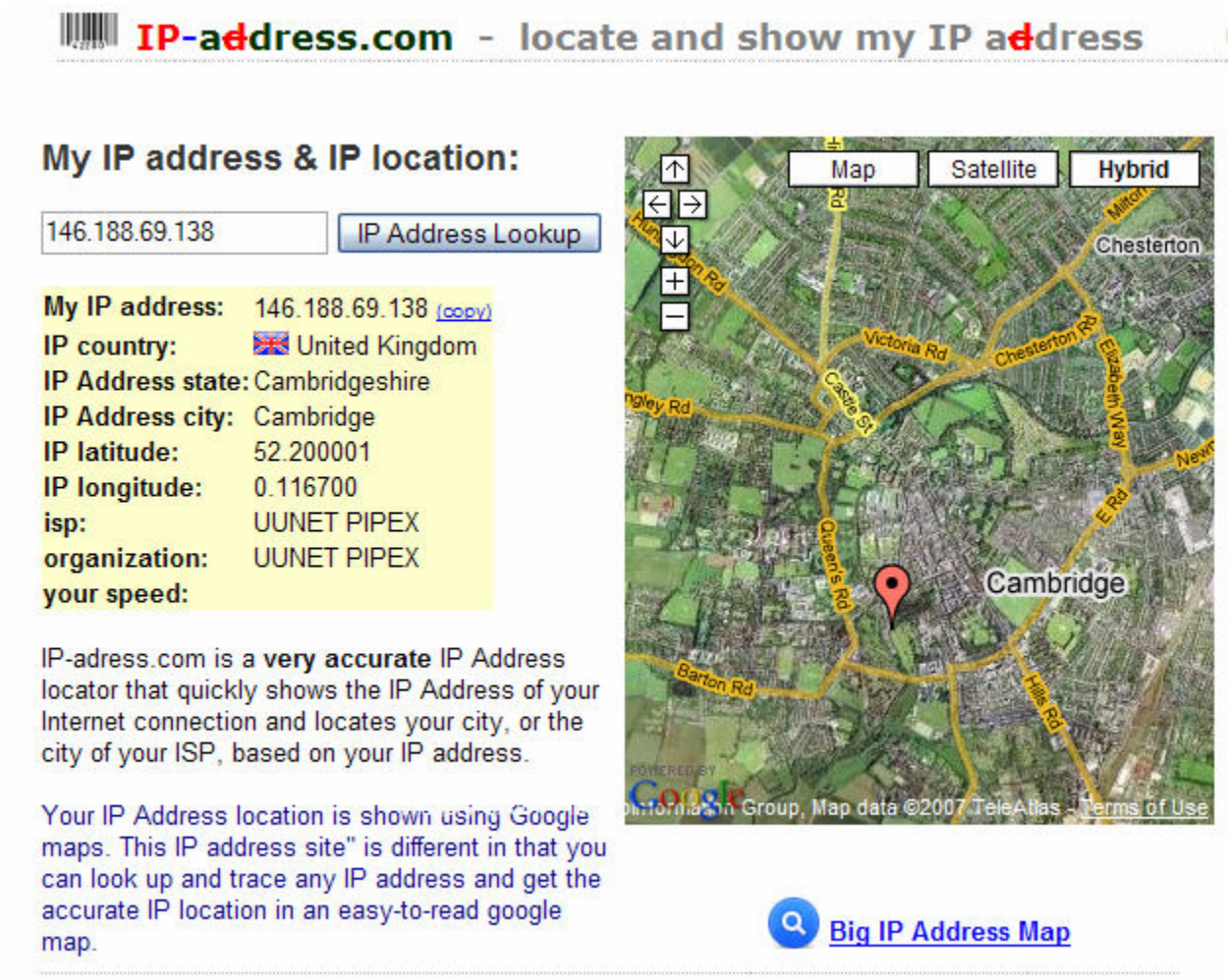
ServerOutputFile.txt - Notepad

File Edit Format View Help

```
IP = 146.188.69.138, weight = 26569
IP = 139.4.101.2, weight = 3721
IP = 212.187.128.58, weight = 2501
IP = 193.19.194.129, weight = 2304
IP = 206.24.169.138, weight = 2209
IP = 4.68.128.105, weight = 2167
IP = 59.145.6.33, weight = 2127
IP = 204.255.169.114, weight = 2116
IP = 59.145.6.37, weight = 1891
IP = 151.164.191.26, weight = 1600
IP = 59.145.11.70, weight = 1462
IP = 152.63.19.33, weight = 1444
IP = 213.248.64.109, weight = 1369
IP = 59.145.11.74, weight = 1311
IP = 209.247.9.125, weight = 1215
IP = 59.145.6.30, weight = 1162
IP = 208.51.239.86, weight = 1156
IP = 151.164.190.202, weight = 1156
IP = 129.250.11.158, weight = 1156
IP = 195.215.109.70, weight = 1225
IP = 64.159.1.126, weight = 1090
IP = 206.108.105.207, weight = 1089
IP = 64.159.1.157, weight = 1086
IP = 172.16.0.6, weight = 1024
IP = 195.215.109.66, weight = 961
```

A free IP geolocation service at http://www.ip-adress.com/ can be used obtain the geographical location of an IP.

For example,



The results of this project can now be used to figure out bottlenecks in the internet For example, the maximal flow between the US and China will be equal to the minimum cut between the IP ranges of USA and China.

Suppose $G(V,E)$ is a finite directed graph and every edge (u,v) has a capacity $c(u,v)$ (a non-negative real number). Further assume two vertices, the source s and the sink t, have been distinguished.

A **cut** is a split of the nodes into two sets *S* and *T*, such that *s* is in *S* and *t* is in *T*. Hence there are

$$2^{|V|-1} - 1$$

possible cuts in a graph. The capacity of a cut (*S*,*T*) is

$$c(S,T) = \sum_{u \in S, v \in T | (u,v) \in E} c(u,v)$$

,

the sum of the capacity of all the edges crossing the cut, from the region *S* to the region *T*.

This calculation was a part of our project, but for a graph the size of the internet, figuring out all the possible cuts between two chunks of the internet (USA and China, for example) is such a time consuming process, that we did not have access to our school's Linux computer cluster for enough time to even finish this calculation once.

3. Conclusion

Our project has successfully pointed out the most important nodes of the internet (Ref. pg 14). It is now in the hands of the government and organizations like the Los Alamos National Laboratory and Sandia National Laboratories to take action to look into the situation and devise ways to protect these critical interconnection points through military power, or to add more redundancy to the network, so that in case of a strike against these nodes, the internet does not crumble.

The code used for this project has been written in a modular, extensible structure. Hence this same code can be easily extended to add more factors in the calculations, or to specialize it for networks with different topologies, such as telecommunications, road, and rail systems, and also human organization structures.

This project does not end here. Research and development will continue to make this program an open-source tool that can be used to check networks for vulnerabilities, and prepare for the worst.

4. Citations

[1] Wikipedia, “Graph Theory”, http://en.wikipedia.org/wiki/Graph_theory

[2] Wikipedia, “Max-flow, min-cut theorem”, http://en.wikipedia.org/wiki/Max-flow_min-cut_theorem

[3] Wolfram Mathworld, “Adjacency Matrix” http://mathworld.wolfram.com/AdjacencyMatrix.html

5. Appendix A – Code

5.1. Server Code

5.1.1. Package main

5.1.1.1. Class ServerStart

```
package main;

import client.ClientStart;

public class ServerStart {
	public static int noOfNodes = 0;
	static int[] ipSubnet = new int[4];
	static String ipSubnetString;
	static String pathToClientList = "C:\\Documents and Settings\\Kirat
Pandya\\My Documents\\Java Projects\\Workspace\\Internet Weakness
Analysis\\ClientList.txt";
	static String pathToClientDataFile = "";
	public static String pathToOutputFile = "";
	static boolean clientList = false;
	static boolean passDatafilePathToClients = false;

	public static void main(String[] args) {
		if(args.length == 0){
			System.out.println("Usage: ");
			System.out.println("java main.ServerStart <no. Of Nodes>
<IP subnet> <use ClientList> ");
			System.out.println("IP subnet should be as follows: 192 168
5 0");
			System.out.println("    OR    ");
			System.out.println("java main.ServerStart");
			System.out.println("The above will result in the assumption
that use of ClientList was intended");
			System.out.println("");
			System.out.println("java main.ServerStart <path to
ClientList> <Path to folder of Client DataFiles>");
			System.out.println("<Path to folder of Client DataFiles> is
optional if set in clients - graph_<no>.txt will be appended");
			System.exit(0);
		}

		if(args.length >= 1 && args.length <=2){
			System.out.println("Using ClientList...");
			pathToClientList = args[0];
			System.out.println("Path to ClientList set to = " +
pathToClientList);

			if(args.length == 2){
				pathToClientDataFile = args[1];
				System.out.println("Path to root data file folder set
to = " + pathToClientDataFile);
				passDatafilePathToClients = true;
			}
			else
				System.out.println("No path to root folder of Client
Datfiles specified...");
```

```
                  for(int i = args[0].length()-1; i >= 0; i--){
                        if (args[0].charAt(i) == '\\' || args[0].charAt(i) ==
'/')
                              ServerStart.pathToOutputFile =
args[0].substring(0, i) + "ServerOutputFile.txt";
                  }
                  clientList = true;
            }

            if(args.length > 2 && args.length !=4){
                  System.out.println("Usage: ");
                  System.out.println("java main.ServerStart <no. Of Nodes>
<IP subnet> <use ClientList> ");
                  System.out.println("IP subnet should be as follows: 192 168
5 0");
                  System.out.println("    OR    ");
                  System.out.println("java main.ServerStart");
                  System.out.println("The above will result in the assumption
that use of ClientList was intended");
                  System.out.println(" ");
                  System.exit(0);
            }

            if(!clientList)
                  new ServerStart().setupIPSubnetArray(args);
            else{
                  Initializer init = new Initializer();
                  init.start();
            }
      }


      void setupIPSubnetArray(String args[]){
            noOfNodes = Integer.parseInt(args[0]);
            System.out.println("Set number Of nodes to " + noOfNodes);

            for(int i = 0; i <= 3; i++)
                  ipSubnet[i] = Integer.parseInt(args[i+1]);

            ipSubnetString = ipSubnet[0] + "." + ipSubnet[1] + "." +
ipSubnet[2] + "." + ipSubnet[3];

            System.out.println("IP subnet of cluster set to " +
ipSubnetString);

            Initializer init = new Initializer();
            init.start();
      }
}
```

5.1.1.2.Class Initializer

```
package main;

class Initializer {
            RuntimeManager runMan;

            Initializer(){
                  //add any initialization commands here
                  //RMIConnectionManager rmiConnM = new
RMIConnectionManager();
            }

            void start(){
                  runMan = new RuntimeManager();
                  runMan.startExecution();
            }
}
```

5.1.1.3. Class RuntimeManager

```
package main;

import gui.GuiManager;
import calc.FinalCalculations;

public class RuntimeManager {

	static String[][] greatestNodes;
	int noOfGreatestNodesOverall = 3;//overall
	int accuracyLevel = 1; //set this number for max accuracy level for weight calculation - 0 for max - BUG PRESENT - DO NOT CHANGE FROM 1
	int noOfGreatestNodes = 5;//this is per node

	void startExecution(){
		GuiManager guiM = new GuiManager();
		guiM.startGUI();

		ClientPropogator clientProp = new ClientPropogator();
		clientProp.propogateAcrossCluster();

		ClientController clientController = new ClientController();
		clientController.startCalculations(noOfGreatestNodes, accuracyLevel);

		manageExecution();
	}

	void manageExecution(){
		FinalCalculations finCalc = new FinalCalculations();
		//make greatest
```

```
		String[][] OverallMostImportantNodes =
finCalc.findGreatesWeightedNodes(greatestNodes, noOfGreatestNodesOverall);

		System.out.println("Get ready!!!!!!!!!!!!");

		for(int i = 0; i < OverallMostImportantNodes[0].length; i++)
			System.out.println("IP = " + OverallMostImportantNodes[0][i] + ",
Weight = " + OverallMostImportantNodes[1][i]);

		for(int i = 0; i < OverallMostImportantNodes[0].length; i++)
			ServerOutputFileWriter.updateWorkfile("IP = " +
OverallMostImportantNodes[0][i] + ", Weight = " + OverallMostImportantNodes[1][i]);

		beginCleanup();
	}

	void beginCleanup(){
		Cleanup cleanup = new Cleanup();
	}
}
```

5.1.1.4.Class ClientPropagator

```
package main;

class ClientPropogator {
	void propogateAcrossCluster(){
		//never implemented as permissions on most machines would not
allow remote initiation
	}
}
```

5.1.1.5.Class ClientController

```
package main;

import client.*;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.rmi.*;

import fileIO.FileIOManager;

class ClientController {
	static ClientInterface[] clientService;
	static int noOfNodesStarted = 0;
	BufferedReader reader;
	String str = "";
	int[] subnets = {0,0,0,0};
	static int i = 0;
	int noOfGreatestNodes = 0;
	int accuracyLevel = 0;
	Thread t;
	static ThreadGroup threadGroup;

	ClientController(){
		try{//creates new BufferedReader to restart reading of database file so that
adjacencyMatrix can be updated
			reader = new BufferedReader(new
FileReader(ServerStart.pathToClientList));
		}catch (FileNotFoundException fnfe){
```

```
                System.out.println("ClientList file not found");
        }

        threadGroup = new ThreadGroup("threadGroup");

        if(ServerStart.clientList){
                while(str!=null){
                        try{
                                readClientListFile();
                                if(str!=null)
                                        ServerStart.noOfNodes++;
                        } catch(Exception e){
                                System.out.println("Error while counting no of nodes from
ClientList");
                                //e.printStackTrace();
                        }
                }

                System.out.println("No of nodes detected in client file  = " +
ServerStart.noOfNodes);

                try{//creates new BufferedReader to restart reading of database file so that
adjacencyMatrix can be updated
                        reader = new BufferedReader(new
FileReader(ServerStart.pathToClientList));
                }catch (FileNotFoundException fnfe){
                        System.out.println("ClientList file not found");
                }
        }

        clientService = new ClientInterface[ServerStart.noOfNodes];
```

```
            str = "";

            if(System.getSecurityManager()==null)
                System.setSecurityManager(new RMISecurityManager());

        }

        void getNextIP(){
            int tempLocationCounter = 0;
            String subnetHolder;
            int oldTempLocation = 0;
            int newTempLocation = 0;

            //System.out.println("str = " + str);

            for(int i = 0; i < 4 ; i++){
                if(i > 0)
                    oldTempLocation = newTempLocation+1;
                    tempLocationCounter = newTempLocation+1;

                while(tempLocationCounter < str.length() &&
(str.charAt(tempLocationCounter)!= '.' && str.charAt(tempLocationCounter) !=' ')){
                    tempLocationCounter++;
                }

                newTempLocation = tempLocationCounter ;

                //System.out.println("OTL = " + oldTempLocation);
                //System.out.println("NTL = " + newTempLocation);
                subnetHolder = str.substring(oldTempLocation, newTempLocation);

                //System.out.println(subnetHolder);
```

```
                subnets[i] = Integer.parseInt(subnetHolder);
            }
        }

        protected void readClientListFile(){
            try{
                str=reader.readLine();
            }catch (FileNotFoundException fnfe){
                System.out.println("ClientList file not found");
            }catch (IOException ioe) {
                System.out.println("Error reading from ClientList file");
            }
        }

        void connectUsingClientList(){
            str = " ";

            try{//creates new BufferedReader to restart reading of database file so that
adjacencyMatrix can be updated
                reader = new BufferedReader(new
FileReader(ServerStart.pathToClientList));
            }catch (FileNotFoundException fnfe){
                System.out.println("ClientList file not found");
            }

            while(str!=null){
                try{
                    readClientListFile();
                    if(str!=null){
                        i++;
                        System.out.println("Client IP = " + str);
```

```
                                        getNextIP();
                                        //System.out.println("i (in connectUsingClientList() = " +
i);
                                        clientService[noOfNodesStarted] = (ClientInterface)
Naming.lookup("//" + subnets[0] + "." + subnets[1] + "." + subnets[2] + "." + subnets[3] +
"/ClientInterface");
                                        //System.out.println(clientService[noOfNodesStarted]);
                                        t = new Thread(threadGroup, threadService);//creates
thread with name as thread<i> - using "t" to perform check in CalcManager
                                        t.start();
                                        noOfNodesStarted++;
                                }
                        } catch(Exception e){
                                System.out.println("Error while trying to perform registry lookup
of ClientInterface on clients");
                                e.printStackTrace();
                                System.out.println("    //" + subnets[0] + "." + subnets[1] + "." +
subnets[2] + "." + subnets[3] + "/ClientInterface");
                        }
                }
        }

        void connectUsingBruteForce(){
                //following code is for bruteforcing ips
                while(noOfNodesStarted < ServerStart.noOfNodes){//creats array of
ClientService skeleon addresses - use clientService[i].<whatever> to call functions on client no. i
                        try{
                                i++;
                                clientService[noOfNodesStarted] = (ClientInterface)
Naming.lookup("rmi://" + ServerStart.ipSubnet[0] + "." + ServerStart.ipSubnet[1] + "." +
ServerStart.ipSubnet[2] + "." + i + "/ClientInterface");
                                noOfNodesStarted++;
```

```
                        t = new Thread(threadGroup, threadService);//creates thread with
name as thread<i> - using "t" to perform check in CalcManager
                        t.start();
                    } catch(Exception e){
                        System.out.println("Error while trying to perform registry lookup
of ClientInterface on clients");
                        //e.printStackTrace();
                        System.out.println("    rmi://" + ServerStart.ipSubnet[0] + "." +
ServerStart.ipSubnet[1] + "." + ServerStart.ipSubnet[2] + "." + i + "/ClientInterface");
                    } finally{
                        if(i == 254){
                            ServerStart.ipSubnet[2]++;
                            i = 1;
                        }

                        else
                            i++;

                        connectUsingBruteForce();
                    }
                }
            }

            Runnable threadService = new Runnable(){
                public void run(){
                    int noOfNodesStarted = ClientController.noOfNodesStarted;
                    String temp[][];
                    try{
                        temp = new String[2][noOfGreatestNodes];
                        //System.out.println("i (in thread) = " + i);
                        System.out.println("clientService[noOfNodesStarted-1] = " +
clientService[noOfNodesStarted-1]);
```

```
                        if(ServerStart.passDatafilePathToClients){
                                //String path = ServerStart.pathToClientDataFile +
"\\graph_" + (noOfNodesStarted-1) + ".txt";
                                String path = ServerStart.pathToClientDataFile +
"\\Datafile.txt";
                                temp = clientService[noOfNodesStarted-
1].startCalculation(path, accuracyLevel, noOfGreatestNodes);
                        }
                        else
                                temp = clientService[noOfNodesStarted-
1].startCalculation(null, accuracyLevel, noOfGreatestNodes);

                        System.out.println("noOfNodesStarted = " + noOfNodesStarted);
                        int greatestNodesIndex = (noOfNodesStarted-1) *
noOfGreatestNodes;

                        for(int k = 0; k < noOfGreatestNodes; k++){
                                System.out.println("greatestNodesIndex = " +
greatestNodesIndex);

                                RuntimeManager.greatestNodes[0][greatestNodesIndex] =
temp[0][k];
                                RuntimeManager.greatestNodes[1][greatestNodesIndex] =
temp[1][k];

        //System.out.println("RuntimeManager.greatestNodes[0][greatestNodesIndex] = " +
RuntimeManager.greatestNodes[0][greatestNodesIndex]);

        //System.out.println("RuntimeManager.greatestNodes[1][greatestNodesIndex] = " +
RuntimeManager.greatestNodes[1][greatestNodesIndex]);
```

```
                                        greatestNodesIndex++;
                                }
                        } catch(Exception e){
                                System.out.println("Error in thread for startCalculation() on client
" + i);
                                e.printStackTrace();
                                System.out.println("    //" + subnets[0] + "." + subnets[1] + "." +
subnets[2] + "." + subnets[3] + "/ClientInterface");
                        }
                }
        };

        void startCalculations(int noOfGreatestNodes, int accuracyLevel){
                try{//creates new BufferedReader to restart reading of database file so that
adjacencyMatrix can be updated
                        reader = new BufferedReader(new
FileReader(ServerStart.pathToClientList));
                }catch (FileNotFoundException fnfe){
                        System.out.println("ClientList file not found");
                }

                int lineCounter = 0;
                while(str != null){
                        readClientListFile();
                        if(str != null)
                                lineCounter++;
                }

                str = "";

                ServerStart.noOfNodes = lineCounter;
```

```
		System.out.println("ServerStart.noOfNodes = " + ServerStart.noOfNodes);

		RuntimeManager.greatestNodes = new
String[2][ServerStart.noOfNodes*noOfGreatestNodes];// node, x [for (ip,weight)],

		this.noOfGreatestNodes = noOfGreatestNodes;
		this.accuracyLevel = accuracyLevel;

		if(ServerStart.clientList)
			connectUsingClientList();
		else
			connectUsingBruteForce();

			while(threadGroup.activeCount() != 0){
			}

			/*System.out.println("Node = " + i);
			for(int j = 0; j < ServerStart.noOfNodes*noOfGreatestNodes; j++)
				System.out.println("    Serialized IP = " +
RuntimeManager.greatestNodes[0][j] + ", Weight = " + RuntimeManager.greatestNodes[1][j]);*/
		}
}
```

5.1.1.6.Class ServerOutputFileWriter

```
package main;

import java.io.*;

class ServerOutputFileWriter {
	protected static synchronized void updateWorkfile(String str){
		try {
			BufferedWriter writer = new BufferedWriter(new
FileWriter(ServerStart.pathToOutputFile, true));

			writer.write(str);

			writer.newLine();
```

```
                writer.close();

            } catch (IOException ioe) {
                System.out.println("Error writing to Workfile");
            }
        }
}
```

5.1.1.7.Class Cleanup

```
package main;

class Cleanup {
      Cleanup(){//constructor to start defulat cleanup cycle
            Runtime.getRuntime().gc(); // call JVM's garbace collector for
the current runtime object
      }
}
```

5.1.2. Package gui

5.1.2.1.Class GuiManager

```
package gui;

public class GuiManager {
	public void startGUI(){
		//gui not implemented due to networking restrictions
	}
}
```

5.1.2.2.Class SetupWindow

```
package gui;

class SetupWindow {
		//gui not implemented due to networking restrictions
}
```

5.1.3. Package calc

5.1.3.1. Class FinalCalculations

```
package calc;


public class FinalCalculations {
	String[][] a;

	public String[][] findGreatesWeightedNodes(String[][] arrayFromNodes,
int noOfNodes){//overall max nodes calculated
		String[][] nodes = new String[2][noOfNodes];
		a = new String[2][arrayFromNodes[0].length];

		/*for(int j = 0; j < arrayFromNodes[0].length; j++)
			System.out.println("		Serialized IP = " +
arrayFromNodes[0][j] + ", Weight = " + arrayFromNodes[1][j]);*/

		for(int i = 0; i < a[0].length; i++){
			a[0][i] = arrayFromNodes[0][i];
			a[1][i] = arrayFromNodes[1][i];
		}

		/*for(int j = 0; j < a[0].length; j++)
			System.out.println("		Serialized IP = " + a[0][j] +
", Weight = " + a[1][j]);*/


		sort();//at this point, we have a sorted array

		/*System.out.println("Final sorted array:");
		for(int j = 0; j < a[0].length; j++)
			System.out.println("		Serialized IP = " + a[0][j] +
", Weight = " + a[1][j]);*/

		for(int i = 0; i < noOfNodes; i++){
			nodes[0][i] = a[0][i];
			nodes[1][i] = a[1][i];
		}

		return nodes;
	}

	void sort(){
		String tempIP = "";
		String tempWeight = "";
		for(int i = 0; i < a[0].length; i++){
			for(int j = i; j < a[0].length; j++){
				if(Integer.parseInt(a[1][i]) <
Integer.parseInt(a[1][j])){
					tempIP = a[0][i];
					tempWeight = a[1][i];

					a[0][i] = a[0][j];
					a[1][i] = a[1][j];
```

```
                                    a[0][j] = tempIP;
                                    a[1][j] = tempWeight;
                                }
                            }
                        }
                    }
                }
```

5.2. Client Code

5.2.1. Package client

5.2.1.1.Class ClientStart

```
package client;

import java.rmi.Naming;
import java.rmi.RMISecurityManager;
import java.rmi.registry.Registry;



public class ClientStart{

    public static String clientIP = "";
    public static String pathToClientDatafile = "";
    public static String pathToClientOutputfile = "";

    public static void main(String[] args) throws Exception{
            if(args.length == 0 || args.length > 2){
                    System.out.println("Usage: ");
                    System.out.println("java client.ClientStart <client IP> <path to data
file>");
                    System.out.println("Example: java client.ClientStart 192.168.5.5
\\home\\kpandya\\Datafile.txt");
                    System.out.println("<path to data file> is optional if it will be passed from
server");
                    System.exit(0);
            }

            ClientStart.clientIP = args[0];
            System.out.println("Client IP Set to = "+ ClientStart.clientIP);

            if(args.length== 2){
```

```
                ClientStart.pathToClientDatafile = args[1];
                System.out.println("Client Datafile set to = " +
ClientStart.pathToClientDatafile);

                for(int i = args[1].length()-1; i >= 0; i--){
                        if (args[1].charAt(i) == '\\' || args[1].charAt(i) == '/')
                                ClientStart.pathToClientOutputfile = args[1].substring(0, i)
+ "ClientOutputFile.txt";
                }
        }

        Registry registry;
        ClientInterface stub;
        //String name = "//"+ ClientStart.clientIP + "/iwa/bin/client/ClientInterface";
        String name = "ClientInterface";

        System.out.println("Starting security manager...");
        if(System.getSecurityManager()==null)
                System.setSecurityManager(new RMISecurityManager());

        try {
                //LocateRegistry.createRegistry(1099);
                //stub = (ClientInterface) UnicastRemoteObject.exportObject(this, 5555);
                //registry = LocateRegistry.getRegistry();
                //registry.rebind(name, stub);
                stub = (ClientInterface) new ClientRuntimeManager();
                Naming.rebind(name, stub);
                System.out.println("ClientRuntimeManager bound");
        } catch (Exception e) {
                System.err.println("ClientRuntimeManager exception:");
                e.printStackTrace();
        }
```

```
            /*ClientInterface clientInt = new ClientRuntimeManager();

            if(System.getSecurityManager()==null)
                    System.setSecurityManager(new RMISecurityManager());

            try {
                    //LocateRegistry.createRegistry(1099);
          String name = "ClientInterface";
          ClientInterface stub = (ClientInterface) UnicastRemoteObject.exportObject(clientInt,
8095);
          Registry registry = LocateRegistry.getRegistry(9999);
          registry.rebind(name, stub);
          //Naming.bind(name, stub);
          System.out.println("ClientRuntimeManager bound");
       } catch (Exception e) {
          System.err.println("ClientRuntimeManager exception:");
          e.printStackTrace();
       }*/

            //Naming.bind("IWAClientService", clientRuntimeM);
    }
}
```

5.2.1.2.Interface ClientInterface

```
package client;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ClientInterface extends Remote{
    public String[][] startCalculation(String pathToDataFile, int accuracyLevel, int noOfNodes) throws RemoteException;// starts calculation weight to the given level within data chunk

    //public void closeClientProcess();

    //public int[][] getGreatestWeights(int noOfNodes) throws RemoteException;//returns the top weights
}
```

5.2.1.3.Class ClientRuntimeManager

```
package client;

import java.rmi.Naming;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

import calc.CalcManager;

public class ClientRuntimeManager extends UnicastRemoteObject implements
ClientInterface{
    private static int noOfProcessors = 0;
    private static long freeMemory = 0; //current available RAM in bytes
    private static long maxMemory = 0; //system assigned value in bytes, given that
maxMemory>=freeMemory
    static CalcManager calcM = new CalcManager();//global init since calcM needs to
remain constant

    public ClientRuntimeManager() throws RemoteException{
            autosetNoOfProcessors();
            autosetFreeMemory();
            autosetMaxMemory();
    }

    ClientRuntimeManager(int noOfProcessors) throws RemoteException{
            forcesetNoOfProcessors(noOfProcessors);
            autosetFreeMemory();
            autosetMaxMemory();
    }
```

```
    protected static void autosetNoOfProcessors(){
            noOfProcessors = Runtime.getRuntime().availableProcessors();
            System.out.println("No. of processors = " + noOfProcessors);
    }

    protected static void forcesetNoOfProcessors(int noOfProcessors){ // method to be used
in case there is a mismatch between the number of processors physically available and that
returned by autosetNoOfProcessors()
            ClientRuntimeManager.noOfProcessors = noOfProcessors;
            System.out.println("No. of processors = " + noOfProcessors);
    }

    public static int getNoOfProcessors(){
            return noOfProcessors;
    }

    static void autosetFreeMemory(){
            freeMemory = Runtime.getRuntime().freeMemory();
            System.out.println("Free Memmory = "+ freeMemory);
    }

    public static long getFreeMomory(){
            autosetFreeMemory();
            return freeMemory;
    }

    static void autosetMaxMemory(){
            maxMemory = Runtime.getRuntime().maxMemory();
            System.out.println("Max Memory = " + maxMemory);
    }
```

```
    public static long getMaxMemory(){
            autosetMaxMemory();
            return maxMemory;
    }

    //================================================================
============================

    public String[][] startCalculation(String pathToDataFile, int accuracyLevel, int
noOFNodes)throws RemoteException{
            if(pathToDataFile!=null){
                    ClientStart.pathToClientDatafile = pathToDataFile;

                    for(int i = ClientStart.pathToClientDatafile.length()-1; i >= 0; i--){
                            if (ClientStart.pathToClientDatafile.charAt(i) == '\\' ||
ClientStart.pathToClientDatafile.charAt(i) == '/'){
                                    ClientStart.pathToClientOutputfile =
ClientStart.pathToClientDatafile.substring(0, i) + "\\ClientOutputFile.txt";
                                    break;
                            }
                    }

                    System.out.println("Client Datafile passed by server = " +
ClientStart.pathToClientDatafile);
                    System.out.println("Client Output file set to = " +
ClientStart.pathToClientOutputfile);
            }
            return calcM.startCalculation(accuracyLevel, noOFNodes);
    }

    /*public void closeClientProcess(){
            System.out.println("Exiting...");
```

```
        try{
                new ClientRuntimeManager(false);
        } catch (Exception e) {
                System.err.println("ClientRuntimeManager exception:");
                e.printStackTrace();
        }
    }*/

    /*public int[][] getGreatestWeights(int noOfNodes){
        return calcM.getGreatestWeightedNodes(noOfNodes);
    }*/
}
```

Package fileIO

5.2.1.4.Class FileIOManager

```
package fileIO;

import client.ClientStart;

public class FileIOManager {
	//use either workfile or an array in memmory
	//current support for Workfile is rudimentary and has not been included
in this version
	//MemmoryArrayCreator memArrayCreator;
	//WorkfileReader wfReader;
	//WorkfileWriter wfWriter;
	static MemoryArrayContainer memArrayContainer;
	static MemoryArrayCreator memArrayCreator;
	static MemoryArrayReader memArrayReader;
	static MemoryArrayWriter memArrayWriter;
	static ArrayPrinter arrayPrinter;
	static String pathToDataFile =ClientStart.pathToClientDatafile;
	static String pathToOutputFile = ClientStart.pathToClientOutputfile;

	public FileIOManager(){

	}

	public FileIOManager(boolean initialize){
		//Why can't these reference variables be private?

		//use either workfile or an array in memmory
		//wrokfile implementation abandoned since METIS will provide for
partitioning
		if(initialize){
			memArrayContainer = new MemoryArrayContainer();
			System.out.println("Adjacency matrix created");
			initializeAdjacencyMatrix();
			System.out.println("Adjacency matrix initialized with
[0][0] = -1");

			memArrayCreator = new MemoryArrayCreator();
			memArrayCreator.AddDatabaseToMemmory();
			//wfCreator = new WorkfileCreator();
			//wfCreator.createWorkfile();

			//Use only one pair of the following objects
			//wfReader = new WorkfileReader();
			//wfWriter = new WorkfileWriter();
			memArrayReader = new MemoryArrayReader();
			memArrayWriter = new MemoryArrayWriter();
			arrayPrinter = new ArrayPrinter();

			arrayPrinter.printAdjacencyMatrix();
			arrayPrinter.printIPLookupTable();
		}
	}
	//use either Array or Workfile

	public int getArrayDimensions(){
```

```
        return MemoryArrayContainer.getArrayDimensions();
}

protected void initializeAdjacencyMatrix(){
        memArrayContainer.initializeAdjacencyMatrix();
}

public void setNewMatrixAsCurrent(){
        memArrayContainer.setNewMatrixAsCurrent();
}

public void writeToArray(int x, int y, int weight){
        memArrayWriter.writeToArray(x, y, weight);
}

public void writeAdditinalWeightToNewArray(int x, int y, int weight){
        memArrayWriter.writeAdditionalWeightToNewArray(x, y, weight);
}

public int readWeightsFromArray(int x, int y){
        return MemoryArrayContainer.adjacencyMatrix[x][y];
}

public int readWeightsFromNewArray(int x, int y){
        return MemoryArrayContainer.newAdjacencyMatrix[x][y];
}

public int readSerializedIPsFromArray(int y){
        return MemoryArrayContainer.adjacencyMatrix[0][y];
}

public synchronized int readSerializedIPsFromNewArray(int y){
        return MemoryArrayContainer.newAdjacencyMatrix[0][y];
}

public void printAdjacencyMatrix(){
        arrayPrinter.printAdjacencyMatrix();
}

public void printNewAdjacencyMatrix(){
        arrayPrinter.printNewAdjacencyMatrix();
}

public int getNewXDimension(){
        return MemoryArrayContainer.xDimensions;
}

public int getNewYDimension(){
        return MemoryArrayContainer.yDimensions;
}

public String getSerializedIPFromLookupTable(int y){
        return MemoryArrayContainer.ipLookupTable[1][y];
}

public String getIPFromLookupTable(int y){
        return MemoryArrayContainer.ipLookupTable[0][y];
}
```

```
        /*void writeToWorkfile(){

        }

        void readWeightsFromWorkfile(int serializedIP){

        }

        void readIPsFromWeightsFromWrokfile(int searchWeight){

        }*/

}
```

5.2.1.5.Class ArrayPrinter

```
package fileIO;

class ArrayPrinter {
	FileIOManager fileIOMan = new FileIOManager();

	void printAdjacencyMatrix(){
		String str = "";
		String temp = "";

		FileIOManager.memArrayWriter.writeToOutputFile(" ");
		FileIOManager.memArrayWriter.writeToOutputFile("Adjacency
Matrix:");

		for(int j = 0; j < MemoryArrayContainer.getArrayDimensions();
j++){
			str =
Integer.toString(fileIOMan.readSerializedIPsFromArray(j));

			if(Integer.parseInt(str)==0)
				break;

			for (int i = 1; i <
MemoryArrayContainer.getArrayDimensions(); i++){
				if (j == 0){
					if(i == 1)
						temp = "          " +
Integer.toString(fileIOMan.readWeightsFromArray(i, j));
					else
						temp = "    " +
Integer.toString(fileIOMan.readWeightsFromArray(i, j));
				}
				else
					temp = "          " +
Integer.toString(fileIOMan.readWeightsFromArray(i, j));
				str += temp;
			}

			FileIOManager.memArrayWriter.writeToOutputFile(str);
		}
	}

	void printAdjacenyMatrixMidExecution(){
		String str = "";
		String temp = "";

		FileIOManager.memArrayWriter.writeToOutputFile(" ");
		FileIOManager.memArrayWriter.writeToOutputFile("Adjacency Matrix
(mid-execution):");

		for(int j = 0; j < fileIOMan.getNewYDimension(); j++){
			str =
Integer.toString(fileIOMan.readSerializedIPsFromArray(j));

			if(Integer.parseInt(str)==0)
				break;
```

```
                  for (int i = 1; i < fileIOMan.getNewXDimension(); i++){
                        if (j == 0){
                              if(i == 1)
                                    temp = "          " +
Integer.toString(fileIOMan.readWeightsFromArray(i, j));
                              else
                                    temp = "    " +
Integer.toString(fileIOMan.readWeightsFromArray(i, j));
                        }
                        else
                              temp = "          " +
Integer.toString(fileIOMan.readWeightsFromArray(i, j));
                        str += temp;
                  }

                  FileIOManager.memArrayWriter.writeToOutputFile(str);
            }
      }

      void printNewAdjacencyMatrix(){
            String str = "";
            String temp = "";
            FileIOManager.memArrayWriter.writeToOutputFile(" ");
            FileIOManager.memArrayWriter.writeToOutputFile("New adjacency
matrix:");

            for(int j = 0; j < MemoryArrayContainer.yDimensions; j++){
                  str =
Integer.toString(fileIOMan.readSerializedIPsFromNewArray(j));

                  for (int i = 1; i < MemoryArrayContainer.xDimensions; i++){
                        if(j==0){
                              if(i==1)
                                    temp = "          " +
Integer.toString(fileIOMan.readWeightsFromNewArray(i, j));
                              else
                                    temp = "    " +
Integer.toString(fileIOMan.readWeightsFromNewArray(i, j));
                        }
                        else
                              temp = "          " +
Integer.toString(fileIOMan.readWeightsFromNewArray(i, j));
                        str += temp;
                  }

                  FileIOManager.memArrayWriter.writeToOutputFile(str);
            }
      }

      void printIPLookupTable(){
            String str = "";
            FileIOManager.memArrayWriter.writeToOutputFile(" ");
            FileIOManager.memArrayWriter.writeToOutputFile("IP Lookup
Table:");
            for(int i = 0; i <= fileIOMan.getArrayDimensions(); i++){
                  str = MemoryArrayContainer.ipLookupTable[0][i] + "
      "+ MemoryArrayContainer.ipLookupTable[1][i];
                  FileIOManager.memArrayWriter.writeToOutputFile(str);
```

```
            }
        }
}
```

5.2.1.6.Class MemoryArrayCreator

```
package fileIO;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import client.ClientRuntimeManager;

class MemoryArrayCreator{
    static String str = "";
    static int lookupArrayIndex = 1;
    BufferedReader reader;
    FileIOManager fileIOMan = new FileIOManager();

    protected MemoryArrayCreator(){
        try{//creates new BufferedReader to restart reading of database file so that
adjacencyMatrix can be updated
            reader = new BufferedReader(new
FileReader(FileIOManager.pathToDataFile));
        }catch (FileNotFoundException fnfe){
            System.out.println("Database file not found");
        }
    }

    protected void AddDatabaseToMemmory(){
        boolean checkForHole = false;
        int lineCounter = 1;
```

```
        System.out.println("Starting to fill ipLookupArray...");

        while(str!=null){
                readDatabaseFile();

                if (str!=null){
                        checkForHole = false;
                        for(int i = 0; i < str.length(); i++)
                                if(str.charAt(i) == 'H' || str.charAt(i) == 's'){
                                        checkForHole = true;
                                }

                        //System.out.println("lineCounter = " + lineCounter+ "        |
Str = " + str + "        |        checkForHole = " + checkForHole);

                        if(!checkForHole)
                                updateLookupArray(ipSerializer());

                        lineCounter++;
                }

                //System.out.println("Free Memory = " +
ClientRuntimeManager.getFreeMomory());
        }

        System.out.println("Filled lookup array");

        /*while(str!=null){//initializes lookup table
                readDatabaseFile();
                if (str!=null){
                        for(int i = 0; i < str.length(); i++)
                                if(str.charAt(i) == 'H')
```

```
                                        continue;

                        updateLookupArray(ipSerializer());
                        }
                }

                System.out.println("Filled lookup array");

                try{//creates new BufferedReader to restart reading of database file so that
adjacencyMatrix can be updated
                        reader = new BufferedReader(new
FileReader(FileIOManager.pathToDataFile));
                }catch (FileNotFoundException fnfe){
                        System.out.println("Database file not found");
                }*/

                str="";

                try{//creates new BufferedReader to restart reading of database file so that
adjacencyMatrix can be updated
                        reader = new BufferedReader(new
FileReader(FileIOManager.pathToDataFile));
                }catch (FileNotFoundException fnfe){
                        System.out.println("Database file not found");
                }

                checkForHole = false;
                //int lineCounter = 0;
                System.out.println("Starting to fill adjacencyMatrix...");
                while(str!=null){
                        readDatabaseFile();
```

```
                if (str!=null){
                        checkForHole = false;

                        for(int i = 0; i < str.length(); i++)
                                if(str.charAt(i) == 'H' || str.charAt(i) == 's'){
                                        checkForHole = true;
                                }

                        //System.out.println("lineCounter = " + lineCounter+ "        |
Str = " + str + "       |       checkForHole = " + checkForHole);

                        if(!checkForHole)
                                updateAdjacencyMatrix(ipSerializer());

                        //lineCounter++;
                }
        }

        System.out.println("Filled adjacency matrix");

        int tempXDimensions = 1, tempYDimensions = 1;

        for(int i = 1; i < MemoryArrayContainer.getArrayDimensions(); i++){
                if(MemoryArrayContainer.adjacencyMatrix[i][0] == 0){
                        break;
                }
                tempXDimensions++;
        }

        for(int j = 1; j< MemoryArrayContainer.getArrayDimensions(); j++){
                if(MemoryArrayContainer.adjacencyMatrix[0][j] == 0)
                        break;
```

```
                tempYDimensions++;
        }

        System.out.println("Optimized dimensions: x = "+ tempXDimensions + ", y = " +
tempYDimensions);
        MemoryArrayContainer.createNewAdjacencyMatrix(tempXDimensions,
tempYDimensions);

        System.out.println("Created newAdjacencyMatrix...");

        /*for(int i = 1; i< MemoryArrayContainer.xDimensions; i++){//intinalize
newAdjacencyMatrix to 1s to simplify further calculations
                for(int j = 1; j< MemoryArrayContainer.yDimensions; j++){
                        MemoryArrayContainer.newAdjacencyMatrix[i][j] = 0;
                }
        }*/
        System.out.println("Starting to copy IPs into newAdjacencyMatrix...");

        for(int i = 1; i < MemoryArrayContainer.xDimensions; i++){//copying ips along y
axis
                MemoryArrayContainer.newAdjacencyMatrix[i][0] =
MemoryArrayContainer.adjacencyMatrix[i][0];
        }
        System.out.println("Destination IPs copied to newAdjacencyMatrix...");

        for(int j = 1; j < MemoryArrayContainer.yDimensions; j++){//copying ips along x
axis
                MemoryArrayContainer.newAdjacencyMatrix[0][j] =
MemoryArrayContainer.adjacencyMatrix[0][j];
        }
        System.out.println("Source IPs copied to newAdjacencyMatrix...");
```

```
            MemoryArrayContainer.newAdjacencyMatrix[0][0] = -1;//setting error code to -1

            System.out.println("New adjacency matrix initialized with [0][0] = -1");
    }

    protected void readDatabaseFile(){
            try{
                    str=reader.readLine();
            }catch (FileNotFoundException fnfe){
                    System.out.println("Database file not found");
            }catch (IOException ioe) {
                    System.out.println("Error reading from Database file");
            }
    }

    //ipSerializer() is overloaded toeither work on static string str or a parameter passed to it
    protected int[] ipSerializer(){
            int[] originSubnets = {0,0,0,0}; //initialising subnet holding array - can't use short
as its range is from 0 to 254
            int[] destinationSubnets = {0,0,0,0};
            int tempLocationCounter = 0;
            String subnetHolder;
            int oldTempLocation = 0;
            int newTempLocation = 0;

            //System.out.println("str = " + str);

            for(int i = 0; i < 4 ; i++){
                    if(i > 0)
                            oldTempLocation = newTempLocation+1;
                    tempLocationCounter = newTempLocation+1;
```

```
                while(tempLocationCounter < str.length() &&
(str.charAt(tempLocationCounter)!= '.' && str.charAt(tempLocationCounter) !=' ')){
                        tempLocationCounter++;
                }

                newTempLocation = tempLocationCounter ;

                //System.out.println("OTL = " + oldTempLocation);
                //System.out.println("NTL = " + newTempLocation);
                subnetHolder = str.substring(oldTempLocation, newTempLocation);

                //System.out.println(subnetHolder);

                originSubnets[i] = Integer.parseInt(subnetHolder);
        }

        oldTempLocation = newTempLocation + 1;
        tempLocationCounter++;

        for(int i = 0; i < 4 ; i++){
                if(i > 0)
                        oldTempLocation = newTempLocation+1;
                        tempLocationCounter = newTempLocation+1;

                while(tempLocationCounter < str.length() &&
(str.charAt(tempLocationCounter)!= '.' && str.charAt(tempLocationCounter) !=' ')){
                        tempLocationCounter++;
                }

                newTempLocation = tempLocationCounter ;
```

```
                //System.out.println("OTL = " + oldTempLocation);
                //System.out.println("NTL = " + newTempLocation);
                subnetHolder = str.substring(oldTempLocation, newTempLocation);

                //System.out.println(subnetHolder);

                destinationSubnets[i] = Integer.parseInt(subnetHolder);
            }

            //moves counter to the destination IP
            //converts array of subnets to an integer number of the base 10 system
            //check for accuracy 256/255
            int serializedOrigin = (originSubnets[0]*256*256*256) +
(originSubnets[1]*256*256) + (originSubnets[2]*256) + originSubnets[3];
            int serializedDestination = (destinationSubnets[0]*256*256*256) +
(destinationSubnets[1]*256*256) + (destinationSubnets[2]*256) + (destinationSubnets[3]);

            int[] serializedIPs = {serializedOrigin, serializedDestination};
            //System.out.println("originSubnet = "+ originSubnets[0] + ":" + originSubnets[1]
+ ":" + originSubnets[2] + ":" + originSubnets[3] + "(Serialized = " + serializedIPs[0] + ")"+
" // destinationSubnet = "+ destinationSubnets[0] + ":" + destinationSubnets[1] + ":" +
destinationSubnets[2] + ":" + destinationSubnets[3] + "(Serialized = " + serializedIPs[1] +
")");
            return serializedIPs;

    }

    /*protected int[] ipSerializer(String str){
            char[] temp = str.toCharArray();//temporary char array for changing ips to
numbers

            //check - why can't typecast int to short?
```

```
        int[] originSubnets = {0,0,0,0}; //initialising subnet holding array - can't use short as its range is from 0 to 254
        int tempLocationCounter = 0;
        for(int i = 0; i < 4 ; i++){
            originSubnets[i] = temp[tempLocationCounter]*100 + temp[tempLocationCounter+1]*50 + temp[tempLocationCounter+2];//adds the subnets to the subnet array
            tempLocationCounter += 3;//moves counter to begining of next subnet
        }

        tempLocationCounter++; //moves counter to the destination IP
        int[] destinationSubnets = {0,0,0,0};
        for(int i = 0; i < 4 ; i++){
            destinationSubnets[i] = temp[tempLocationCounter]*100 + temp[tempLocationCounter+1]*50 + temp[tempLocationCounter+2];//adds the subnets to the subnet array
            tempLocationCounter += 3;//moves counter to begining of next subnet
        }

        //converts array of subnets to an integer number of the base 10 system
        //check for accuracy 256/255
        int serializedOrigin = (originSubnets[0]*256*256*256) + (originSubnets[1]*256*256) + (originSubnets[2]*256) + originSubnets[3];
        int serializedDestination = (destinationSubnets[0]*256*256*256) + (destinationSubnets[1]*256*256) + (destinationSubnets[2]*256) + (destinationSubnets[3]);

        int[] serializedIPs = {serializedOrigin, serializedDestination};
        return serializedIPs;
    }*/
```

```
    protected void updateLookupArray(int[] serializedIPs){//adds the passed serialzed IPs to serial number to serialized IP table
        //do not use MemoryArrayWriter or use FileIOManage.writeToArray();
        int[] originSubnets = {0,0,0,0}; //initialising subnet holding array - can't use short as its range is from 0 to 254
        int[] destinationSubnets = {0,0,0,0};
        int tempLocationCounter = 0;
        String subnetHolder;
        int oldTempLocation = 0;
        int newTempLocation = 0;

        //System.out.println("str = " + str);

        for(int i = 0; i < 4 ; i++){
            if(i > 0)
                oldTempLocation = newTempLocation+1;
            tempLocationCounter = newTempLocation+1;

            while(tempLocationCounter < str.length() && (str.charAt(tempLocationCounter)!= '.' && str.charAt(tempLocationCounter) !=' ')){
                tempLocationCounter++;
            }

            newTempLocation = tempLocationCounter ;

            //System.out.println("OTL = " + oldTempLocation);
            //System.out.println("NTL = " + newTempLocation);
            subnetHolder = str.substring(oldTempLocation, newTempLocation);

            //System.out.println(subnetHolder);

            originSubnets[i] = Integer.parseInt(subnetHolder);
```

```
		}

		oldTempLocation = newTempLocation + 1;
		tempLocationCounter++;

		for(int i = 0; i < 4 ; i++){
			if(i > 0)
				oldTempLocation = newTempLocation+1;
			tempLocationCounter = newTempLocation+1;

			while(tempLocationCounter < str.length() &&
(str.charAt(tempLocationCounter)!= '.' && str.charAt(tempLocationCounter) !=' ')){
				tempLocationCounter++;
			}

			newTempLocation = tempLocationCounter ;

			//System.out.println("OTL = " + oldTempLocation);
			//System.out.println("NTL = " + newTempLocation);
			subnetHolder = str.substring(oldTempLocation, newTempLocation);

			//System.out.println(subnetHolder);

			destinationSubnets[i] = Integer.parseInt(subnetHolder);
		}

		boolean sourceIPAlreadyInTable = false;
		boolean destinationIPAlreadyInTable = false;

		MemoryArrayContainer.ipLookupTable[0][0] = "1";
		MemoryArrayContainer.ipLookupTable[1][0] = "1";
```

```
		for(int i = 0; i < MemoryArrayContainer.getArrayDimensions(); i++){

			if(MemoryArrayContainer.ipLookupTable[1][i] != null){
				if(Integer.parseInt(MemoryArrayContainer.ipLookupTable[1][i])
== 0)
					break;
				if(Integer.parseInt(MemoryArrayContainer.ipLookupTable[1][i])
== serializedIPs[0])
					sourceIPAlreadyInTable = true;
				if(Integer.parseInt(MemoryArrayContainer.ipLookupTable[1][i])
== serializedIPs[1])
					destinationIPAlreadyInTable = true;
			}

			else
				break;
		}

		if(!sourceIPAlreadyInTable){
			MemoryArrayContainer.ipLookupTable[0][lookupArrayIndex]=
originSubnets[0] + "." + originSubnets[1] + "." + originSubnets[2] + "." + originSubnets[3];
			MemoryArrayContainer.ipLookupTable[1][lookupArrayIndex]=
Integer.toString(serializedIPs[0]);
			lookupArrayIndex++;
		}

		if(!destinationIPAlreadyInTable){
			MemoryArrayContainer.ipLookupTable[0][lookupArrayIndex]=
destinationSubnets[0] + "." + destinationSubnets[1] + "." + destinationSubnets[2] + "." +
destinationSubnets[3];
			MemoryArrayContainer.ipLookupTable[1][lookupArrayIndex]=
Integer.toString(serializedIPs[1]);
```

```
                lookupArrayIndex++;
            }
    }

    /*protected void updateAdjacencyMatrix(int[] serializedIPs){//commented as
ipLookupArray is not needed! Doh!
            ClientOutPutFileWriter.updateWorkfile("Source = " + serializedIPs[0] + ",
Destination = " + serializedIPs[1]);
            int tempIndex1 = 0;
            for(int j = 0; j < MemoryArrayContainer.getArrayDimensions(); j++){//lookup
serial of 1st serialized IP
                    if(MemoryArrayContainer.ipLookupTable[1][j] ==
serializedIPs[0]);
                            tempIndex1 =j;
            }

            if (tempIndex1==0){
                    for(int j = 0; j < MemoryArrayContainer.getArrayDimensions();
j++){//lookup serial of 1st serialized IP
                            if(MemoryArrayContainer.ipLookupTable[0][j] == 0);
                                    tempIndex1 = j;
                    }
            }

            MemoryArrayContainer.adjacencyMatrix[tempIndex1][0] =
MemoryArrayContainer.ipLookupTable[0][tempIndex1];//these two can be directly equated
to tempIndex1 also
            MemoryArrayContainer.adjacencyMatrix[0][tempIndex1] =
MemoryArrayContainer.ipLookupTable[0][tempIndex1];

            int tempIndex2 = 0;
```

```
            for(int j = 0; j < MemoryArrayContainer.getArrayDimensions(); j++){//lookup
serial of 1st serialized IP
                        if(MemoryArrayContainer.ipLookupTable[1][j] ==
serializedIPs[0]);
                                tempIndex2 = j;
            }

            if (tempIndex2==0){
                    for(int j = 0; j < MemoryArrayContainer.getArrayDimensions();
j++){//lookup serial of 1st serialized IP
                        if(MemoryArrayContainer.ipLookupTable[0][j] == 0);
                                tempIndex2 = j;
                    }
            }

            ClientOutPutFileWriter.updateWorkfile("tempIndex1 = " + tempIndex1 + ",
tempIndex2 = "+ tempIndex2);

            MemoryArrayContainer.adjacencyMatrix[tempIndex2][0] =
MemoryArrayContainer.ipLookupTable[0][tempIndex2];//these two can be directly equalted
to tempIndex2 also
            MemoryArrayContainer.adjacencyMatrix[0][tempIndex2] =
MemoryArrayContainer.ipLookupTable[0][tempIndex2];

            MemoryArrayContainer.adjacencyMatrix[tempIndex1][tempIndex2] = 1; //set the
weight to 1
            MemoryArrayContainer.adjacencyMatrix[tempIndex2][tempIndex1] = 1; //set the
weight to 1
    }*/

    protected void updateAdjacencyMatrix(int[] serializedIPs){
```

```
		//ClientOutPutFileWriter.updateWorkfile("Source = " + serializedIPs[0] + ",
Destination = " + serializedIPs[1]);
		//System.out.println("Source = " + serializedIPs[0] + ", Destination = " +
serializedIPs[1]);

		int tempSourceIndex = 0;
		for(int j = 0; j < MemoryArrayContainer.getArrayDimensions(); j++){//lookup
serial of 1st serialized IP
				if(MemoryArrayContainer.adjacencyMatrix[0][j] ==
serializedIPs[0]){
					tempSourceIndex = j;
					break;
				}
		}

		if (tempSourceIndex==0){//if IP not found in previous loop, find end of current
dataset in array
			//ClientOutPutFileWriter.updateWorkfile("Starting failsafe for
tempSourceIndex");
			for(int j = 1; j < MemoryArrayContainer.getArrayDimensions(); j++){
				if(MemoryArrayContainer.adjacencyMatrix[0][j] == 0){
					tempSourceIndex = j;
					break;
				}
			}
		}

		int tempDestinationIndex = 0;
		for(int i = 1; i < MemoryArrayContainer.getArrayDimensions(); i++){//lookup
serial of 2nd serialized IP
				if(MemoryArrayContainer.adjacencyMatrix[i][0] ==
serializedIPs[1]){
```

```
                        tempDestinationIndex = i;
                        break;
                    }
            }

            if (tempDestinationIndex==0){//if IP not found in previous loop, find end of
current dataset in array
                    //ClientOutPutFileWriter.updateWorkfile("Starting failsafe for
tempDestinationIndex");
                    for(int i = 0; i < MemoryArrayContainer.getArrayDimensions(); i++){
                        if(MemoryArrayContainer.adjacencyMatrix[i][0] == 0){
                            tempDestinationIndex = i;
                            break;
                        }
                    }
            }

            //ClientOutPutFileWriter.updateWorkfile("tempSourceIndex = " +
tempSourceIndex + ", tempDestinationIndex = "+ tempDestinationIndex);
            //ClientOutPutFileWriter.updateWorkfile(" ");

            MemoryArrayContainer.adjacencyMatrix[0][tempSourceIndex] =
serializedIPs[0];
            MemoryArrayContainer.adjacencyMatrix[tempDestinationIndex][0] =
serializedIPs[1];

        MemoryArrayContainer.adjacencyMatrix[tempDestinationIndex][tempSourceIndex] +=
1;
    }
}
```

5.2.1.7.Class MemoryArrayContainer

```
package fileIO;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

class MemoryArrayContainer {
    //assuming 300,000 records for adjacency matrinx
    //fact that matrix reflects along unique diagonal doesn't help
    //first index is for x, second is for y
    static private int dimensions;
    static protected int[][] adjacencyMatrix;
    static protected int[][] newAdjacencyMatrix;//two arrays to avoid exponential calulations
    static protected String[][] ipLookupTable;
    static private boolean initialized = false;
    private static String str = "";
    private static BufferedReader reader;
    static protected int xDimensions, yDimensions;

    protected void initializeAdjacencyMatrix(){
        if(!initialized){
            setArrayDimensions();
            adjacencyMatrix = new int[dimensions][dimensions];
            ipLookupTable = new String[2][dimensions*2];
            MemoryArrayContainer.adjacencyMatrix[0][0] = -1; //error code in case
requested serialized ips are not found by any method
        }

        initialized = true;
    }
```

```
    protected static void createNewAdjacencyMatrix(int xDimensions, int
yDimensions){//intitializes newAdjacencyMatrix with y=parameter
            MemoryArrayContainer.xDimensions = xDimensions;
            MemoryArrayContainer.yDimensions = yDimensions;
            newAdjacencyMatrix = new int[xDimensions][yDimensions];//two arrays to
avoid exponential calulations
    }

    protected synchronized void setNewMatrixAsCurrent(){
            System.out.println("Starting setNewMatrixAsCurrent()...");
            int[][] tempAdjacencyMatrix = adjacencyMatrix;
            adjacencyMatrix = new
int[newAdjacencyMatrix.length][newAdjacencyMatrix[0].length];
            //now copy newAdjacencyMatrix into AdjacencyMatrix

            for(int i = 0; i < newAdjacencyMatrix.length; i++){
                    for(int j = 0; j < newAdjacencyMatrix[0].length; j++){
                            adjacencyMatrix[i][j] = newAdjacencyMatrix[i][j];
                    }
            }

            ArrayPrinter arrayPrinter = new ArrayPrinter();

            arrayPrinter.printAdjacenyMatrixMidExecution();
            arrayPrinter.printNewAdjacencyMatrix();

            System.out.println("setNewMatrixAsCurrent() done...");
            /*newAdjacencyMatrix = tempAdjacencyMatrix;

            //next loop is to trim the new adjacency matrix
            tempAdjacencyMatrix = newAdjacencyMatrix;
```

```
            newAdjacencyMatrix = new int[tempXDimensions][tempYDimensions];

            for(int i = 0; i < tempXDimensions; i++){
                    for(int j = 0; j < tempYDimensions; j++){
                            if(tempNewAdjacencyMatrix[i][j] == 0){
                                    break;
                            }
                            newAdjacencyMatrix[i][j] = tempNewAdjacencyMatrix[i][j];
                    }
            }



            /*for(int i = 1; i< MemoryArrayContainer.dimensions; i++){//intinalize
newAdjacencyMatrix to 0s to simplify further calculations
                    for(int j = 1; j< MemoryArrayContainer.dimensions; j++){
                            MemoryArrayContainer.newAdjacencyMatrix[i][j] = 0;
                    }
            }

            for(int i = 1; i < MemoryArrayContainer.dimensions; i++){
                            newAdjacencyMatrix[0][i] = adjacencyMatrix[0][1];
            }

            for(int j = 1; j < MemoryArrayContainer.dimensions; j++){
                    newAdjacencyMatrix[j][0] = adjacencyMatrix[0][1];
            }

            MemoryArrayContainer.newAdjacencyMatrix[0][0] = -1;*/
    }

    static protected int getArrayDimensions(){
```

```
        return dimensions;
    }

    static protected void setArrayDimensions(){
        try{
            reader = new BufferedReader(new
FileReader(FileIOManager.pathToDataFile));
        }catch (FileNotFoundException fnfe){
            System.out.println("Database file not found");
        }

        int lineCounter = 0;

        while(str!=null){
            readDatabaseFile();
            if (str!=null){
                for(int i = 0; i < str.length(); i++)
                    if(str.charAt(i) == 'H')
                        continue;
                lineCounter++;
            }
        }

        MemoryArrayContainer.dimensions = lineCounter;
        System.out.println("Dimensions set to " + dimensions);
    }

    private static void readDatabaseFile(){
        try{
            str=reader.readLine();
        }catch (FileNotFoundException fnfe){
            System.out.println("Database file not found");
```

```
            }catch (IOException ioe) {
                  System.out.println("Error reading from Database file");
            }
      }
   }
```

5.2.1.8.Class MemoryArrayReader

```
package fileIO;

class MemoryArrayReader{

      int getSumOFWeightsOfConnectedEdges(int serializedIP){
            int indexOfSerializedIP = 0;
            for (int i = 0; i < MemoryArrayContainer.adjacencyMatrix.length;
i++){
                  if(serializedIP ==
MemoryArrayContainer.adjacencyMatrix[0][i]){
                        indexOfSerializedIP = i;
                        break;
                  }
            }

            int sumOfWeights = 0;

            for (int i = 0; i < MemoryArrayContainer.getArrayDimensions();
i++){

      if(MemoryArrayContainer.adjacencyMatrix[i][indexOfSerializedIP] > 0){
                        sumOfWeights +=
MemoryArrayContainer.adjacencyMatrix[i][indexOfSerializedIP];
                        break;
                  }
            }

            return sumOfWeights;
      }

      int getWeightofEdgeBetweenNodes(int serializedOrigin, int
serializedDestination){
            int indexOfSerializedOrigin = 0;
            for (int i = 0; i < MemoryArrayContainer.adjacencyMatrix.length;
i++){
                  if(serializedOrigin ==
MemoryArrayContainer.adjacencyMatrix[0][i]){
                        indexOfSerializedOrigin = i;
                        break;
                  }
            }

            int indexOfSerializedDestination = 0;
            for (int i = 0; i < MemoryArrayContainer.adjacencyMatrix.length;
i++){
```

```
                    if(serializedDestination ==
MemoryArrayContainer.adjacencyMatrix[i][0]){
                            indexOfSerializedDestination = i;
                            break;
                    }
             }

             return
MemoryArrayContainer.adjacencyMatrix[indexOfSerializedDestination][indexOfSer
ializedOrigin];
       }
}
```

5.2.1.9.Class MemoryArrayWriter

```
package fileIO;

class MemoryArrayWriter {
	//use synchronized protected methods
	synchronized protected void writeToArray(int x, int y, int weight){
		//System.out.println("Writing: x = " + x + ", y = " + y + ",
weight = " + weight);
		MemoryArrayContainer.newAdjacencyMatrix[x][y] = (weight);
		//writeToOutputFile("Writing (current matrix): x = " + x + ", y =
" + y + ", weight = " + weight);
	}

	synchronized protected void writeAdditionalWeightToNewArray(int x, int
y, int weight){
		//System.out.println("Writing (additional): x = " + x + ", y = "
+ y + ", weight = " + weight);
		MemoryArrayContainer.newAdjacencyMatrix[x][y] = weight +
MemoryArrayContainer.adjacencyMatrix[x][y];
		//writeToOutputFile("Writing (new matrix): x = " + x + ", y = " +
y + ", weight = " + weight);
	}

	protected void writeToOutputFile(String str){
		ClientOutPutFileWriter.updateWorkfile(str);
	}
}
```

Package calc

5.2.1.10. Class CalcManager

```
package calc;

import client.ClientRuntimeManager;
import fileIO.FileIOManager;

public class CalcManager{
    static int availableProcessors = 0;
    static long freeMemory = 0; //current available RAM in bytes
    static long maxMemory = 0;
    FileIOManager fileIOMan;

    public CalcManager(){
        //updateResourceAvailability();
    }

    void updateResourceAvailability(){
        availableProcessors = ClientRuntimeManager.getNoOfProcessors();
        freeMemory = ClientRuntimeManager.getFreeMomory();
        maxMemory = ClientRuntimeManager.getMaxMemory();
    }

    public String[][] startCalculation(int accuracyLevel, int noOFNodes){
        fileIOMan = new FileIOManager(true);
        updateResourceAvailability();
        //ThreadManager threadM = new ThreadManager(accuracyLevel);
        boolean checkForEnd = false;
        for(int accuracyCounter = 1; accuracyCounter <= accuracyLevel;
accuracyCounter++){
            new ThreadManager(accuracyLevel).startCalculationEngine();
            if(accuracyLevel > 1)
```

```
                    do{//check if threads have completed execution
                        if(ThreadManager.threadGroup.activeCount() == 0){
                            fileIOMan.setNewMatrixAsCurrent();
                            checkForEnd = true;
                        }
                    }while(!checkForEnd);
            }

            while(true){//check if threads have completed execution
                if(ThreadManager.threadGroup.activeCount() == 0){
                    fileIOMan.printNewAdjacencyMatrix();
                    return getGreatestWeightedNodes(noOFNodes);
                }
            }
        }

        public String[][] getGreatestWeightedNodes(int noOfNodes){
            WeightCalculator weightCalc = new WeightCalculator();
            return weightCalc.getGreatestWeightedNodes(noOfNodes);
        }
    }
```

5.2.1.11. Class ThreadManager

```
package calc;

import fileIO.FileIOManager;

class ThreadManager extends CalcManager{
	WeightCalculator weightCalc;
	int accuracyLevel;
	static FileIOManager fileIOMan;
	static int threadDivisionID = 0;
	static int[][] threadDivisions  = new
int[availableProcessors][4];//might need to make this an instance when
implementing accuracyLevel
	//static int[] sectionMarkers = new int[4];//might need to make this an
instance when implementing accuracyLevel
	//Thread t;
	static ThreadGroup threadGroup;

	ThreadManager(int accuracyLevel){
//		will use accuracy level later
		this.accuracyLevel = accuracyLevel;
		setupThreading();
	}

	void setupThreading(){
		weightCalc = new WeightCalculator();
		//weightCalc.setStartWeights();//stupid idea!!!
		threadGroup = new ThreadGroup("threadGroup");
		//threadDivisions = new int[availableProcessors][4];//2 for
previous method
		//sectionMarkers = new int[4];

		int yDivisionPoint = calculateThreadDivisions();

		//System.out.println("divisionCounter[0] = " + divisionCounter[0]
+ ", divisionCounter[1] = " + divisionCounter[1]);

		for(int proc = 0; proc < availableProcessors; proc++){
			/*if(proc==0){
				threadDivisions[proc][0] = 1;
				threadDivisions[proc][2] = 1;
			}

			else{
				threadDivisions[proc][0] = xDivisionCounterTemp;//min
				threadDivisions[proc][2] = yDivisionCounterTemp;//min
			}*/

			/*if(fileIOMan.getNewXDimension()%2 == 0){
				if(proc == 0)
					threadDivisions[proc][0] = 1;//min
				else
					threadDivisions[proc][0] =
threadDivisions[proc-1][1];//min

				threadDivisions[proc][1] =threadDivisions[proc][0] +
divisionCounter[0];//max
```

```
                }
                else{
                        if(proc == 0)
                                threadDivisions[proc][0] = 1;//min
                        else
                                threadDivisions[proc][0] =
threadDivisions[proc-1][1];//min

                        threadDivisions[proc][1] = threadDivisions[proc][0] +
divisionCounter[0] - 1;//max
                }*/

                threadDivisions[proc][0] = 1;
                threadDivisions[proc][1] = fileIOMan.getNewXDimension()-1;

                if(fileIOMan.getNewYDimension()%2 == 0){
                        if(proc == 0){
                                threadDivisions[proc][2] = 1;//min
                        }

                        else{
                                threadDivisions[proc][2] =
threadDivisions[proc-1][3];//min
                        }

                        threadDivisions[proc][3] = threadDivisions[proc][2] +
yDivisionPoint;//divisionCounter[1];//max
                }
                else{
                        if(proc == 0){
                                threadDivisions[proc][2] = 1;//min
                        }

                        else{
                                threadDivisions[proc][2] =
threadDivisions[proc-1][3];//min
                        }

                        threadDivisions[proc][3] = threadDivisions[proc][2] +
yDivisionPoint - 1;//divisionCounter[1] - 1;//max
                }



                //System.out.println("threadDivisions[proc][0] = " +
threadDivisions[proc][0] + ", threadDivisions[proc][1] = " +
threadDivisions[proc][1] + ", threadDivisions[proc][2] = " +
threadDivisions[proc][2] + ", threadDivisions[proc][3] = " +
threadDivisions[proc][3]);
            }

            /*for(int proc = 0; proc < availableProcessors; proc++){//sets up
the divisions of the datafile according to the no of threads
                if(proc==0)
                        threadDivisions[proc][0] = 1;
                else
                        threadDivisions[proc][0] = divisionCounterTemp;
```

```
                  divisionCounterTemp += divisionCounter;

                  threadDivisions[proc][1] = divisionCounterTemp;
                  //System.out.println("threadDivisions[proc][0] = " +
threadDivisions[proc][0] + ", threadDivisions[proc][1] = " +
threadDivisions[proc][1]);
            }*/
      }

      void startCalculationEngine(){//begins the main threaded calculation
engine
            startThreading();
      }

      void startThreading(){
            String threadNameTemp = "thread";
            for(int i = 1; i <= availableProcessors; i++){//creates 1 thread
per processor
                  threadNameTemp += Integer.toString(i);
                  new Thread(threadGroup, threadService,
threadNameTemp).start();//creates thread with name as thread<i> - using "t"
to perform check in CalcManager
                  System.out.println("Thread \""+ threadNameTemp + "\"
started");
                  //need to implement ThreadGroup and long stacksize
                  threadNameTemp = "thread";
            }
      }

      Runnable threadService = new Runnable(){
            public void run(){
                  //check algorithm of distribution of Weight calculator
across threads
                  //also the main max-flow min-cut calculation class will be
called from here - also needs to be split

                  /*if(threadDivisionID > 0)//this wait insures that
threadDivisionID gets incremented before the next thread attempts to assign
sectionMarkers
                        try{
                        wait(500);
                        }catch(Exception e){
                              System.out.println("Thread " + threadDivisionID
+ " interrrupted while waiting for other threads initialize");
                        }*/
                  System.out.println("Starting calculateWeight().....");
                  weightCalc.calculateWeightOfEdges(setDivisionPoints(),
accuracyLevel);
            }
      };//defining inner Runnable object for access control - do not remove
semi-colon

      synchronized int[] setDivisionPoints(){
            int[] sectionMarkers = new int[4];
            sectionMarkers [0] = threadDivisions[threadDivisionID][0];
            sectionMarkers [1] = threadDivisions[threadDivisionID][1];
            sectionMarkers [2] = threadDivisions[threadDivisionID][2];
            sectionMarkers [3] = threadDivisions[threadDivisionID][3];
```

```
            //System.out.println("ThreadManager: sectionMarkers[0] = " +
sectionMarkers[0] + ", sectionMarkers[1] = " + sectionMarkers[1]+ ",
sectionMarkers[2] = " + sectionMarkers[2]+ ", sectionMarkers[3] = " +
sectionMarkers[3]);
            threadDivisionID++;
            return sectionMarkers;
      }

      int calculateThreadDivisions(){//method to distribute Weight calculator
across threads
            fileIOMan = new FileIOManager();

            /*int xDivisionPoint = 0;
            if(fileIOMan.getNewXDimension()%2 == 0)
                  xDivisionPoint = (fileIOMan.getNewXDimension()-
1)/availableProcessors;
            else
                  xDivisionPoint =
fileIOMan.getNewXDimension()/availableProcessors;*/

            int yDivisionPoint = 0;
            if(availableProcessors == 1){
                        yDivisionPoint = (fileIOMan.getNewYDimension()-2);
            }

            else{
                  if(fileIOMan.getNewYDimension()%2 == 0)
                        yDivisionPoint = (fileIOMan.getNewYDimension()-
1)/availableProcessors;
                  else
                        yDivisionPoint =
fileIOMan.getNewYDimension()/availableProcessors;
            }
            //int[] divisionPoints = {xDivisionPoint, yDivisionPoint};;

            return yDivisionPoint;
      }
}
```

5.2.1.12. Class WeightCalculator

```
package calc;

import fileIO.FileIOManager;

class WeightCalculator {
	FileIOManager fileIOMan = new FileIOManager();
	static int[] sectionMarkers = new int[4];

	/*void setStartWeights(){//set weights of all edges to 1
		for(int i = 1; i < fileIOMan.getArrayDimensions(); i++){
			for(int j = 1; i < fileIOMan.getArrayDimensions(); i++){
				//System.out.println("i = " + i + ", j = " + j);
				//System.out.println("Current value in newAdjacency
Matrix = " + fileIOMan.readWeightsFromNewArray(i, j));
				fileIOMan.writeToArray(i, j, 1);
			}
		}
	}*/

	void calculateWeightOfEdges(int[] sectionMarkers, int degree){ //takes
in serialized ip of node and computes weight to nth degree
		//sectionMarkers format : xmin(0), xmax(1), ymin(2), ymax(3)
		WeightCalculator.sectionMarkers = sectionMarkers;
		System.out.println("WeightCalculator: sectionMarkers[0] = " +
sectionMarkers[0] + ", sectionMarkers[1] = " + sectionMarkers[1]+ ",
sectionMarkers[2] = " + sectionMarkers[2]+ ", sectionMarkers[3] = " +
sectionMarkers[3]);
		for(int y = sectionMarkers[2]; y <= sectionMarkers[3]; y++){
			for(int x = sectionMarkers[0]; x <= sectionMarkers[1];
x++){
				//System.out.println("calculateWeight(): x = " + x +
", y = " + y);
				/*if(fileIOMan.readSerializedIPsFromArray(y) == 0)
					break;

				if(fileIOMan.readWeightsFromArray(x, 0) ==
0)//reading end of data on x axis
					break;*/

				if (fileIOMan.readWeightsFromNewArray(x,y) >
fileIOMan.readWeightsFromArray(x,y) || fileIOMan.readWeightsFromArray(x, y)
== 0){ // checks if new weights have been already calulated for the given
edge or if edge does not exist
					//System.out.println("continue for
sumWeightsForEdges(): x = " + x + ", y = " + y);
					continue;
				}

				else{
					//System.out.println("sumWeightsForEdges(): x =
" + x + ", y = " + y);
					sumWeightsForEdge(x,y);
				}
			}
		}
```

```
		System.out.println("calculateWeightOfEdges() done!");
	}

	void sumWeightsForEdge(int x, int y){
		int weightTemp = 0;

			for (int weightXIndex1 = 1; weightXIndex1 <
fileIOMan.getNewXDimension(); weightXIndex1++){
				if(weightXIndex1 == x){
					//System.out.println("continue1");
					continue;
				}
				else
					weightTemp +=
fileIOMan.readWeightsFromArray(weightXIndex1, y);
					//System.out.println("weightTemp = " +
weightTemp);
			}

			for (int weightYIndex1 = 1; weightYIndex1 <
fileIOMan.getNewYDimension(); weightYIndex1++){
				if(weightYIndex1 == y){
					//System.out.println("continue1");
					continue;
				}
				else
					weightTemp += fileIOMan.readWeightsFromArray(x,
weightYIndex1);
					//System.out.println("weightTemp = " +
weightTemp);
			}

			int indexOfXInY = 0;
			for (int weightYIndex2 = 1; weightYIndex2 <
fileIOMan.getNewYDimension(); weightYIndex2++){

		if(fileIOMan.readSerializedIPsFromArray(weightYIndex2) ==
fileIOMan.readWeightsFromArray(x, 0)){
					indexOfXInY = weightYIndex2;
					break;
				}
			}

			if(indexOfXInY != 0)
				for (int weightXIndex2 = 1; weightXIndex2 <
fileIOMan.getNewXDimension(); weightXIndex2++){
					if(weightXIndex2 == x){
						//System.out.println("continue1");
						continue;
					}

					else
						weightTemp +=
fileIOMan.readWeightsFromArray(weightXIndex2, indexOfXInY);
					//System.out.println("weightTemp = " +
weightTemp);
				}
```

```
                  writeAdditionalWeightToNewArray((short) x, (short) y,
weightTemp);
      }

      //check - will syncronization lock force caslculateWeight to lock too?
      void writeAdditionalWeightToNewArray(short x, short y, int weight){ //
calls on either MemmoryArrayWriter or WorkfileWriter
            fileIOMan.writeAdditinalWeightToNewArray(x, y, weight);
      }

      String[][] getGreatestWeightedNodes(int noOfNodes){
            int[][] weights =new int[2][noOfNodes];

            int[][] temp = new int[2][fileIOMan.getArrayDimensions()*2];//*2
since we will add ips from both x and y

            for(int y = 1; y < fileIOMan.getNewYDimension(); y++){//adding
ips along y
                  if (fileIOMan.readSerializedIPsFromNewArray(y) == 0)
                        break;

                  temp[0][y] = fileIOMan.readSerializedIPsFromNewArray(y);

                  //System.out.println("y = " + y);
                  for(int i = 1; i < fileIOMan.getNewXDimension(); i++){
                        temp[1][y] += fileIOMan.readWeightsFromNewArray(i,
y);
                        //System.out.println("  " + "i = " + i + ",
fileIOMan.readWeightsFromNewArray(i, y) = " +
fileIOMan.readWeightsFromNewArray(i, y));
                  }

                  //System.out.println("  temp[1][y] = " + temp[1][y]);

                  //System.out.println("temp (loop 1): y = " + y + ",
temp[0][y] = " + temp[0][y] + ", temp[1][y]  = " +  temp[1][y]);
            }

            for(int x = 1; x < fileIOMan.getNewXDimension(); x++){//adding
ips along x
                  if (fileIOMan.readWeightsFromNewArray(x, 0) == 0)//checks
for end of data along x axis
                        break;

                  temp[0][fileIOMan.getNewYDimension()-1+x] =
fileIOMan.readWeightsFromNewArray(x, 0);//add the ip to temo

                  //System.out.println("x = " + x + ",
fileIOMan.readWeightsFromNewArray(x, 0) = " +
fileIOMan.readWeightsFromNewArray(x, 0));

                  for(int j = 1; j < fileIOMan.getNewYDimension(); j++){
                        temp[1][fileIOMan.getNewYDimension()-1+x] +=
fileIOMan.readWeightsFromNewArray(x, j);
                  }

                  //System.out.println("temp (loop 2): x = :" + x + ",
fileIOMan.getNewYDimension()-1+x = " + (fileIOMan.getNewYDimension()-1+x) +
```

```
", temp[0][fileIOMan.getNewYDimension()-1+x] = " + temp[0][x] + ",
temp[1][fileIOMan.getNewYDimension()-1+x]  = " +  temp[1][x]);
            }

            for(int j = 1; j < fileIOMan.getArrayDimensions()*2; j++){
                  if (temp[0][j] == 0)
                        break;
                  //System.out.println("temp (before sort): IP = "+
temp[0][j] + ", Weight = " + temp[1][j]);
            }

            sort(temp);//sorting to bring up top weights

            for(int j = 1; j < fileIOMan.getArrayDimensions()*2; j++){
                  if (temp[0][j] == 0)
                        break;
                  //System.out.println("temp (after sort): IP = "+ temp[0][j]
+ ", Weight = " + temp[1][j]);
            }

            for(int i = 1; i <= noOfNodes; i++){
                  weights[0][i-1] = temp[0][i];
                  weights[1][i-1] = temp[1][i];

                  System.out.println("i = " + i + ", weights[0][i-1] = " +
weights[0][i-1] + ", weights[1][i-1] = " + weights[1][i-1]);
            }

            String[][] finalReturnArray = new String[2][noOfNodes];

            for(int i = 0; i < noOfNodes; i++){
                  finalReturnArray[0][i] = Integer.toString(weights[0][i]);
                  finalReturnArray[1][i] = Integer.toString(weights[1][i]);
            }

            /*for(int i = 0; i < noOfNodes; i++){
                  System.out.println("i = " + i + ", finalReturnArray[0][i] =
" + finalReturnArray[0][i] + ", finalReturnArray[1][i] = " +
finalReturnArray[1][i]);
            }*/

            for(int i = 0; i < noOfNodes; i++){
                  //System.out.println("finalReturnArray[0][i] = " +
finalReturnArray[0][i]);

                  for(int j = 0; j < fileIOMan.getArrayDimensions()*2; j++){
                        if(fileIOMan.getSerializedIPFromLookupTable(j) ==
null)
                              break;
                        //System.out.println("
      fileIOMan.getSerializedIPFromLookupTable(j) = " +
fileIOMan.getSerializedIPFromLookupTable(j));

                        if(Integer.parseInt(finalReturnArray[0][i]) ==
Integer.parseInt(fileIOMan.getSerializedIPFromLookupTable(j))){

      //System.out.println("fileIOMan.getSerializedIPFromLookupTable(j) = " +
fileIOMan.getSerializedIPFromLookupTable(j));
```

```
                              finalReturnArray[0][i] =
fileIOMan.getIPFromLookupTable(j);
                              break;
                        }
                  }
            }

            for(int i = 0; i < noOfNodes; i++){
                  System.out.println("i = " + i + ", finalReturnArray[0][i] =
" + finalReturnArray[0][i] + ", finalReturnArray[1][i] = " +
finalReturnArray[1][i]);
            }

            return finalReturnArray;
      }

      void sort(int[][] a){
            int tempIP = 0;
            int tempWeight = 0;

            for(int i = 1; i <= a[0].length/2; i++){
                  for(int j = i; j <= a[0].length/2; j++){
                        //System.out.println("i = " + i + ", j = " + j + ",
a[1][i] = " + a[1][i] +  ", a[1][j] = " + a[1][j]);
                        if(a[1][i] < a[1][j]){
                              //System.out.println("Switching...");

                              tempIP = a[0][i];
                              tempWeight = a[1][i];

                              a[0][i] = a[0][j];
                              a[1][i] = a[1][j];

                              a[0][j] = tempIP;
                              a[1][j] = tempWeight;
                        }
                  }
            }
      }
}
```