Extended Essay

Computer Science

Depth first Search vs. Breadth first Search

To what extent would Depth first search or Breadth first search be suitable for search in graph data structures used by social networks, taking time and memory as determining factors, in the java virtual machine?

Student Id: glx661

Supervisor: *Kept Anonymous by CS EE World*

Word count: 4047

TABLE OF CONTENTS

Heading	Page
Introduction	3
Big data with reference to social networks	4
Why is data from Social networks classified as Big data?	4
Data Structures	6
Social Network Graphs	9
Methodology and Experimental Set-up	12
Analysis and Evaluation	16
References	
	21
Appendix	
	23

1. Introduction

The digital era in computing today, has been greatly affected by a huge desire to protect, store and access data due to the fact that the usage of computer systems have quadrupled over the years and as a result, given birth to ***Big Data analysis***. Data is growing faster than ever before and by the year 2020, about 1.7 megabytes of new information will be created every second for every human being on the planet .[1] The number of social media users worldwide from 2010 to 2016 with projections until 2020 has exponentially increased, and in 2018 alone consisted of an estimated 2.67 billion social media users around the globe, up from 2.34 billion in 2016[2]

As a result, the research question **"To what extent would Depth first search or Breadth first search be suitable for search in graph data structures used by social networks, taking time and memory as determining factors in java"**, seeks to investigate which algorithms used in graph data structures would be better suited for search in data structures implemented by social networking sites like Facebook, Twitter, Google plus etc. as there are more users on these platforms every day. This research is quite important because, users of such social networking platforms, perform searches to connect with more people on such networks daily. Thus, the speed of the search as well as the manageability of the search algorithms that make these features available, is very important to the user and provider respectively.

Data management interventions and speed to access data has been craved for in recent years, and in a bid to efficiently help manage all and any type of data (ranging from numerical values to Objects) being produced, computer scientists produced the concepts of data structures such as arrays, lists, linked lists, queues, stacks,

[1] Marr, Bernard. "Big Data: 20 Mind-Boggling Facts Everyone Must Read." *Forbes*, Forbes Magazine, 19 Nov. 2015, www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-everyone-must-read/#62e17e3c17b1.

[2] * All products require an annual contract. Prices do not include sales tax (New York residents only). "Number of Social Media Users Worldwide 2010-2021." *Statista*, www.statista.com/statistics/278414/number-of-worldwide-social-network-users/.

trees and graphs. Computer scientists also came up with sequential 'step by step' ways through which such data can be processed, managed and then churned into information while stored in such data structures.

1.1 Big Data with Reference to Social Networks

The word "Big Data" comes from the concept that describes one dataset whose size exceeds the typical database software acquisition and storage, management and analysis. There are comprehensively three broad categories of Big data that is Structured data, Semi-structured Data and Unstructured data. Structured data is data that is contained in a field and saved in a record or file. A common example of this is data that is stored in a relational database or a log file. Conversely, unstructured data can be understood as not being organised in a particular way. An important example of this will be content on a social media site, such as Facebook. This content would include images, videos, text and advertisements to name a few, but the content will usually be too difficult to organise to be stored in a database or log file.[3] The data used by any Social network architecture, could fall into any of such three categories depending on the module being used by the organization needing the data and for their own purpose.

1.2 Why is data from Social networks classified as Big data?

There are many common features of Big Data, otherwise known as "V" features coined by different researchers which stand for Volume, Variety, Velocity, Veracity.

Volume: The amount of data being generated by social networks is growing every day. Examples include, new friendships between users as in Facebook and new face book groups created by users (known as clusters in the

[3] Beal, Vangie. "Structured Data." *What Is Structured Data? Webopedia Definition*, www.webopedia.com/TERM/S/structured_data.html.

face book graph), new followers and retweets as in twitter and finally new profile accounts being created by people joining these social networks to mention a few. Such logistical processes generate large gamut of data about human beings who patronize such social networks.

Variety: In a bid to provide the best form of socialization and entertainment, social networks have integrated variety of data - both generated by their users and the organization itself - forms for their users to enjoy. These different types of data include: image, audio, video, diagram and others. They are derived from different sources like GPS signals, sensors, ad- hoc network, social networks and many more that capture data and information updates.

Velocity: For Social networks, data is continuously generated at every time but only useful data are needed for the processing to give effective information. 'Velocity' refers to the increasing speed at which this data is created, and the increasing speed at which the data can be processed, stored and analysed by relational databases. The possibilities of processing data in real-time is an area of particular interest, which allows companies to do things like display personalised ads based on the on the web pages you visit, based on your recent search, viewing and purchase history as in face book where videos shown on your timeline are based on your recent account activities.

Veracity: Although there's widespread agreement about the potential value of Big Data, data is virtually worthless if it's not accurate. This is particularly true in programs that involve automated decision-making, or feeding the data into an unsupervised machine learning algorithm. The results of such programs are only as good as the data they're working with. Social Networks produce Big data due to the fact that majority of the data provided by users is almost always authentic as these users are looking forward to reap the full benefits from registering in these networks. [4]

[4] McNulty, Eileen, et al. "Understanding Big Data: The Seven V's." *Dataconomy*, 8 May 2017, dataconomy.com/2014/05/seven-vs-big-data/.

2. Data structures

A Data Structure is a way of collecting and organising data in such a way that we can perform operations on these data in an effective way – It is a storage place for data. The concept of Data Structures is about rendering data elements in terms of some relationship, for better organization and storage. Data Structures are structures programmed to store ordered or unordered data, so that various operations can be performed on it easily in order to churn such data into information. It is normally designed and implemented in such a way that it reduces the complexity and increases the efficiency.

Anything that can store data can be called as a data structure. Hence, Integer, Float, Boolean, Char etc., all are data structures. They are known as **Primitive Data Structures**. However, there are also some complex Data Structures called Abstract Data structures, which are used to store large and connected data. Examples are:

- Linked List
- Tree
- Graph
- Stack, Queue etc.

All these data structures allow us to perform different operations on data. Data structure selection is normally based on what type of operation is required for what kind of data being generated.

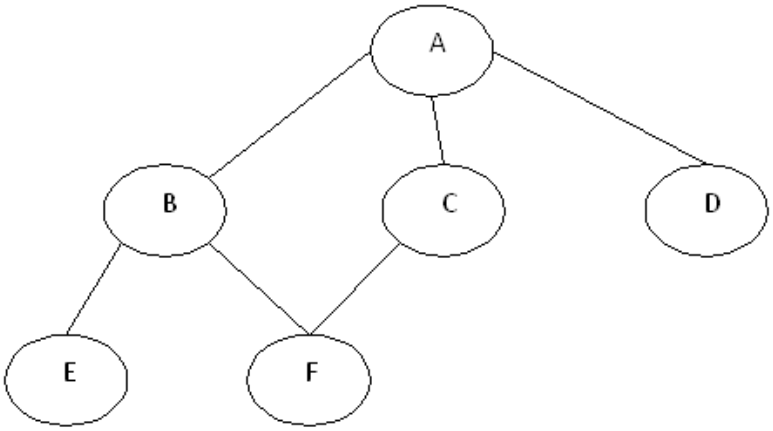
2.1 Graph data structures

Graphs are one of the most interesting data structures in computer science. Graphs and the trees are somewhat similar by their structure as they are both considered abstract data structures. The tree data structure is derived from the graph data structure. However, there are two important differences between trees and graphs:

1. Unlike trees, in graphs, a node can have many parents.

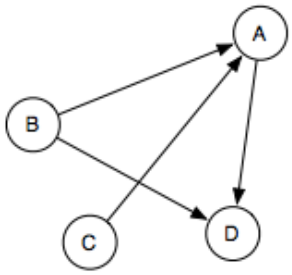
2. The link between the nodes may have values or weights.

The following example shows a very simple graph that can be abstractly represented on the computer:

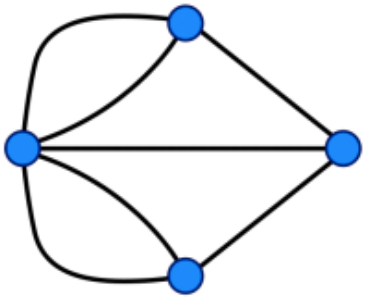


In the above graph, A, B, C, D, E, F are called nodes and the connecting lines between these nodes are called edges. The edges can be directed edges which are shown by arrows; they can also be weighted edges in which some numbers are assigned to them. Hence, a graph can be a directed/undirected and weighted/un-weighted graph. [5]

[5] Bijulsoni. "Introduction to Graph with Breadth First Search(BFS) and Depth First Search(DFS) Traversal Implemented in JAVA." *Introduction to Graph with Breadth First Search(BFS) and Depth First Search(DFS) Traversal Implemented in JAVA - CodeProject*, www.codeproject.com/Articles/32212/Introduction-to-Graph-with-Breadth-First-Search-BF.



An example of a directed graph on 4 vertices.



An undirected graph on 4 vertices[1]

Since they are powerful abstractions, graphs can be very important in modelling data. In fact, many problems can be reduced to known graph problems considering the fact that the nodes in the graphs could be represented as objects, and the edges as relationship objects.

Some real-world solutions that use graph data structures include:

1. Social network graphs: Here, we find Graphs that represent who knows whom and who communicates with whom. An example is the twitter graph of who follows whom. Such graphs can be used to determine how information flows, how topics trend etc.

2. Transportation networks. In road networks vertices are intersections and edges are the road segments between them, and for public transportation networks vertices are stops and edges are the links between them. Such networks are used by many map programs such as Google maps and Bing maps. They are used to find the best routes between locations, and also used for studying traffic patterns, traffic light timings, and many aspects of transportation.

3. Social network Graphs

A social graph is a diagram that illustrates interconnections among people, groups and organizations in a social network. The term refers to both the social network itself and a diagram representing the network. Individuals and organizations, called actors, are nodes on the graph[6] For the purposes of this research, I am going to discuss graph structures used by social networks and then analyse two algorithms that provide information from the data stored in the graph.

3.1 Why do social networks use Graph data structures?

What makes graphs special is that they represent relationships between things from the most abstract to the most concrete e.g., mathematical objects, things, events. A social network is an umbrella with nodes of individuals, groups, organizations and related systems that tie in one or more types of interdependencies.

However, the social network and type of graph entirely depends on the architecture that is being used by the company providing the online social network services – That is, for a social network that is going to be bidirectional in terms of the relationship between nodes, an undirected graph structure is going to be used as in the Face book graph. Conversely, if the relationship between nodes is going to be one directional, then a directional graph is used. Social network analysis is focused on uncovering the patterning of people's interaction. A major habit of most users on social network involves the feature of Search.

[6] "What Is Social Graph? - Definition from WhatIs.com." *WhatIs.com*, whatis.techtarget.com/definition/social-graph.

3.2 Graph Traversals

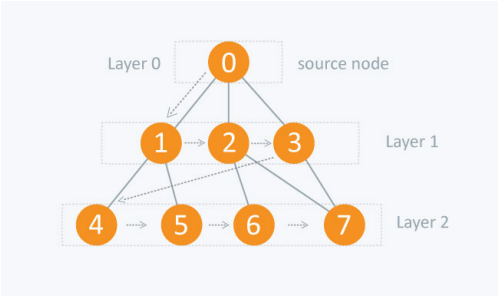
In the last years, huge graphs with billions of vertices and edges have become very common because accounts are being created every day, more relationships are growing between users and finally, more data is being produced by each node in the network.

Search in graph terminology is known as graph traversal, and is a means of visiting every vertex and edge exactly once in a well-defined order. There are two basic graph traversals which are Breadth first search (BFS)and Depth first search (DFS). These algorithms as their names denote, are coined from the manner in which both traverse(search) a graph.

Breadth first search

Breadth first search is an algorithm used in traversing graphs where traversal is done from a selected node called the source node and then layer wise thus exploring the neighbour nodes. The algorithm then moves towards the next-level neighbour nodes. In other words, it explores the neighbours of the neighbours of a particular node while moving towards the next-level nodes.

So, first move horizontally and visit all the nodes of the current layer, then move to the next layer.



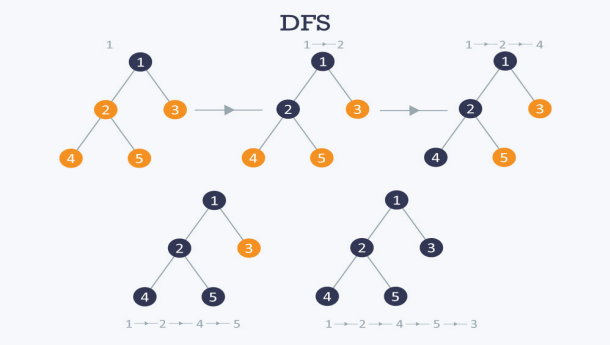
Looking at the graph above, bfs first visits the source node which is 0, and immediately visits its immediate neighbours all in the same layer 1, 2 , and 3 before moving to the next layer to explore the others. The distance

between the nodes in layer 1 is comparatively lesser than the distance between the nodes in layer 2. Therefore, in BFS, you must traverse all the nodes in layer 1 before you move to the nodes in layer 2.

BFS uses a queue used to store a node and mark it as 'visited' until all its neighbours (vertices that are directly connected to it) are marked. The queue follows the First In First Out (FIFO) queuing method, and therefore, the neighbours of the node will be visited in the order in which they were inserted in the node.

Depth first Search

Depth first search algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking.[7] All the nodes will be visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected.



DFS is implemented using stacks. A starting node is picked and all its adjacent nodes are pushed into a stack. A node is then popped from stack to select the next node to visit and all its adjacent nodes are pushed into a

[7] Here, the word backtrack means that when you are moving forward and there are no more nodes along the current path, you move backwards on the same path to find nodes to traverse.

stack. This process is repeated until the stack is empty. However, the nodes that are visited have to be marked else a node might be visited more than once, and an infinite loop would occur.[8]

Methodology and Experimental Setup

Concerning the memory used by both algorithms, I hypothesize that BFS would use significantly more memory than DFS because in its execution, it selects a vertex, inserts its entire adjacent vertices in a queue, then takes another vertex from memory and also inserts its entire adjacent vertices into the same queue without deleting any. However, DFS does the same insertion and deletion as BFS, but it removes a node from memory once its descendants have been expanded. I also hypothesize that there might not be a dominance of either algorithm with regards to time as the search might depend on other factors that might be beyond the scope of this paper.

I plan to conduct an experiment in java using an external library called "Jgrapht" - a popular library used for implementing graph data structures and algorithms. I am going to be implementing a social network graph in java. This graph is extracted from Facebook and consists of people (nodes) with edges representing friendship ties, based on students from Colleges in the United States of America. These Colleges Include California Institute of Technology, Reed University, Haverford College, Swarthmore College, Middlebury College, Bucknell College, John Hopkins University and Massachusetts Institute of Technology. The data for these colleges can be found in the appendix.

[8] "Breadth First Search Tutorials & Notes | Algorithms." *HackerEarth*, www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/.

I plan to use a mac book air 13' whose operating system is MAC OS X 10.12.5, has a total of 4 processors and a total physical memory of 8 Gigabytes. My Integrated development environment (IDE) will be the Eclipse neon.3 because that is what I am most proficient in using as well as the fact that it is a suitable IDE for java programming.

First off, I plan to implement a model graph that has the same number of nodes and relationships as the dataset I obtained but with randomized relationships between the nodes. I am using the same number of nodes and edges, but having randomized edges (relationships between the nodes) because it will be quite cumbersome to implement, for example, a total of 6400 nodes and 251,200 edges(relationships) "manually".

Nevertheless, because of the randomised nature of the graph I will initially produce, the relationships between nodes might change every time the program is run thus, I plan to write the generated graph as a serialised object to a file and then search for a particular node in each graph. I plan to do this in order to ensure that the graph both algorithms traverse is constant to eliminate any disparities that may arise while collecting data.

I will then implement both DFS and BFS libraries using the Jgrapht library and then measure the time and memory used by both algorithms by finding initial and final time and memory space, and then subtracting to find the difference. I plan to use inbuilt java functions to find out the memory and time statistics of both algorithms.

Figure 1

```
public static class RandomGraphCreator1 {

    /**
     * A simple class to create a random UndirectGraph
     *
     * @param numVertices
     * @param numEdges
     * @return
     */
    public Graph<Integer, DefaultEdge> createRandomUndirectGraph(int numVertices, int numEdges) {

        GnmRandomGraphGenerator<Integer, DefaultEdge> randomGraphGenerator = new GnmRandomGraphGenerator<Integer, Default
                numVertices, numEdges);
        Graph<Integer, DefaultEdge> sourceGraph = new SimpleGraph<>(DefaultEdge.class);

        randomGraphGenerator.generateGraph(sourceGraph, new IntVertexFactory(), null);
        return sourceGraph;
    }

    /**
     * Integer vertex factory
     */
    public class IntVertexFactory implements VertexFactory<Integer> {

        private int nextVertex = 1;

        @Override
        public Integer createVertex() {
```

The method createRandomUndirectGraph () graph creates an undirected graph that has its nodes as Int objects who are randomly connected. However, the number of edges and nodes as parameters for the method are derived from the real-world dataset of colleges in Table 1.

Name of College	|V| (No. of nodes in Graph)	|E| (No. of edges in Graph)
California Institute of Technology	769	17000
Reed University	962	19000
Swarthmore College	2000	61000
Middlebury	3000	125000
Bucknell College	4000	159000
John Hopkins University	5000	187000
Massachusetts Institute of Technology	6000	251000

The dfsSearch() and bfsSearch()methods – shown in figures 2 and 3 - were implemented with the help of Jgrapht and traversed the graph to find a specific node which was randomly generated and recorded as 745.

```
private static <Int> void bfsSearch(Graph<Int, DefaultEdge> graph, Int search) {
    GraphIterator<Int, DefaultEdge> iterator = new DepthFirstIterator<Int, DefaultEdge>(graph);

    try {
        while (iterator.hasNext()) {

            if (!iterator.next().equals(search)) {

                System.err.println("Not what you are looking for ");
            } else {
                System.out.println("Found search in graph: " + search);

                break;
            }

        }
    } catch (Exception e) {
        System.err.println("Error in program ");
    }
}
```

```
private static <Int> void dfsSearch(Graph<Int, DefaultEdge> graph, Int search) {
    GraphIterator<Int, DefaultEdge> iterator = new DepthFirstIterator<Int, DefaultEdge>(graph);

    try {
        while (iterator.hasNext()) {

            // System.out.println(iterator.next());

            if (!iterator.next().equals(search)) {

                System.err.println("Not what you are looking for ");
            } else {
                System.out.println("Found search in graph: " + search);
                break;
            }

        }
    } catch (Exception e) {
        System.err.println("Error in program ");
    }
}
```

In order to get data to do with the time and memory used during the execution of a particular search, the java code in figure 4 below was implemented.

```
30      Graph = ReadGraph(Graph);
31
32      System.gc();
33
34      Thread.sleep(5000);
35
36      long InitialMemory = runtime.totalMemory();
37      long begin = System.currentTimeMillis();
38
39      bfsSearch(Graph, 745);
40
41      long endd = System.currentTimeMillis();
42      long freeMemory = runtime.freeMemory();
43
44      long usedMemory = (InitialMemory - freeMemory);
45
46      long time = endd - begin;
47
48      System.out.println((usedMemory) / 1024 * 1024 + "mb");
49
50      System.out.println(time + "millisecs");
51
52      }
```

fig. 4

I decided that after Reading the graph from the file, I needed to ensure that all unused objects generated from the reading of the file were deleted from memory, so that it doesn't affect the memory measurements I get. Thus, I run the Garbage collector explicitly. I then took measurements for memory and time by wrapping them around the search method as shown in figure 4 above. The memory and runtime of both algorithms time were found with the help of the java bench marking class Runtime.

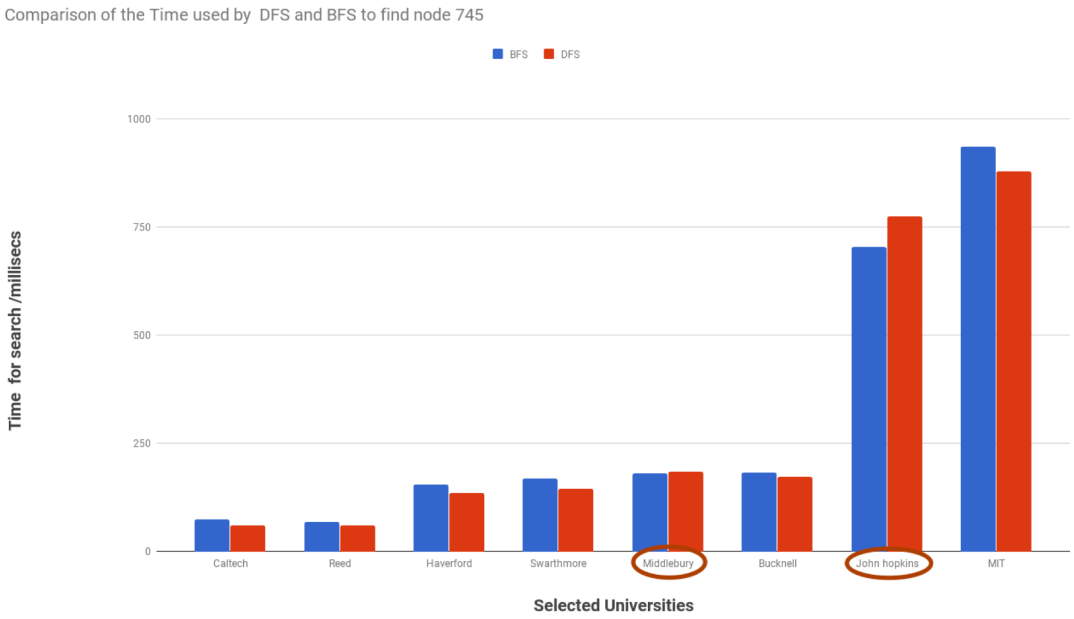
An example of the whole Program – showing both writing and reading of the Caltech Graph - was executed as shown below for all the Colleges and their respective number of nodes and edges (relationships).

```
1  package ee;
2
3  import java.io.FileInputStream;
18
19 public class jgraphtest {
20
21     public static void main(String[] args) throws IOException, ClassNotFoundException {
22
23         RandomGraphCreator1 RnmGraph = new RandomGraphCreator1();
24
25         System.out.println(" Making a Caltech Graph : 769 nodes , 17K edges ");
26
27         Graph<Integer, DefaultEdge> Graph = RnmGraph.createRandomUndirectGraph(769, 17000);
28
29          WriteGraph(Graph);
30
31          System.out.println(Graph.toString());
32
33         System.out.println(" done writing Caltech Graph to file ");
34
35          System.exit(0);
36
37
38     }
39
```

```
20
21
22 public class jgraphtest {
23
24     public static void main(String[] args) throws IOException, ClassNotFoundExcept
25     Runtime runtime = Runtime.getRuntime();
26         Graph<Integer, DefaultEdge> Graph = null;
27
28     System.out.println("The node that will be looked for is " + "745");
29
30     Graph = ReadGraph(Graph);
31
32     System.gc();
33
34     Thread.sleep(5000);
35
36     long InitialMemory = runtime.totalMemory();
37     long begin = System.currentTimeMillis();
38
39     bfsSearch(Graph, 745);
40
41     long endd = System.currentTimeMillis();
42     long freeMemory = runtime.freeMemory();
43
44     long usedMemory = (InitialMemory - freeMemory);
45
46     long time = endd - begin;
47
48     System.out.println((usedMemory) / 1024 * 1024 + "mb");
49
50     System.out.println(time + "millisecs");
51
52     }
53
Markers  Properties  Servers  Data Source Explorer  Snippets  Console
```

Analysis and Evaluation

Despite the fact that the repository that provided me with information about the Facebook relationships of students in many universities in the United States, I chose these Universities based on whether they were co-educational, in order to alleviate any gender inconsistencies, and most importantly based on how much each university's nodes



Graph 1.

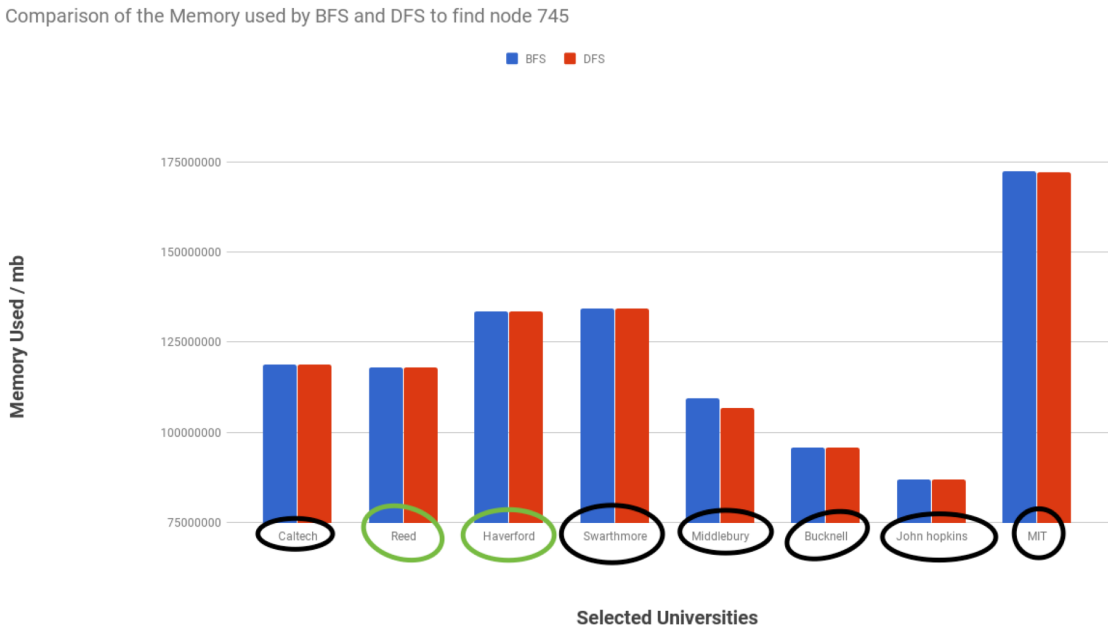
and edges varied from each other in order to establish a huge difference to have significant effects.

Graph 1 shows that DFS used less time to find node 745 in Caltech, Reed, Haverford, Swarthmore, Bucknell and MIT. However, BFS used less time to find node 745 in Middlebury and John Hopkins. Therefore, overall, DFS used less time to search for node 745. However, with BFS finding node 745 in the graph structure of 2 schools in lesser time compared to DFS, I was forced to explore other factors aside the design of both algorithms such as the structure of the graph, the position of the node being searched for in the graph, or anomalies. I retook data for the Middlebury and John Hopkins graphs to investigate whether I had an anomaly. However, the data still remained consistent in terms of BFS using less time. Due to Jgrapht's limitations and my programming skills, there was no way I could determine the structure of the tree or the position of the node in the implemented graphs of both

schools. Hence, I found out the number of edges node 745 had in the graph of Middlebury and John Hopkins, and Node 745 had 85 and 66 edges respectively.

From this information, I could not find any reason why BFS had lesser time in finding node 745 however, thinking about the approach both algorithms use – i.e. exploring the widest nodes first (as in BFS) and exploring the deepest nodes first (as in DFS) – I am hypothesizing that perhaps node 745 was wider in the graphs of both Schools, hence BFS took less time.

Graph 2.



The memory used by both algorithms during the search for node 745 was very close and sometimes even had the same value. In Graph 2, the universities labelled with the green circle had BFS and DFS use the same amount of memory in the execution of the search. However, all the universities in the black ellipses, had DFS use less memory compared to BFS. The closeness of the memory used in Reed and Haverford – if we relate it back to design of both algorithms – made me speculate that perhaps both BFS and DFS explored the same number of nodes.

Figure 5

```
105 |   private static <Int> void dfsTraverse(Graph<Int, DefaultEdge> graph) {
106         GraphIterator<Int, DefaultEdge> iterator = new DepthFirstIterator<Int, DefaultEdge>(graph);
107         try {
108             while (iterator.hasNext()) {
109
110                 if (!iterator.next().equals(745)) {
111
112                     System.err.println(iterator.next().toString());
113                 } else {
114                     System.out.println("Found search in graph: " + "745");
115
116                     break;
117                 }
118
119             }
120         } catch (Exception e) {
121             System.err.println("Error in program ");
122         }
123
124     }
```

```
127 private static <Int> void bfsTraverse(Graph<Int, DefaultEdge> graph) {
128     GraphIterator<Int, DefaultEdge> iterator = new BreadthFirstIterator<Int, DefaultEdge>(graph);
129     try {
130         while (iterator.hasNext()) {
131
132             if (!iterator.next().equals(745)) {
133
134                 System.err.println(iterator.next().toString());
135             } else {
136                 System.out.println("Found search in graph: " + "745");
137
138                 break;
139             }
140
141         }
142     } catch (Exception e) {
143         System.err.println("Error in program ");
```

However, after implementing a DFS and BFS traverse methods shown in fig. 5, I realised that in the graph for Reed, BFS explored 481 nodes whereas DFS explored only 133 nodes.

Surprisingly, in the graph of Haverford, both BFS and DFS algorithms explored 500 nodes each before finding node 745. Hence making my speculations not fairly accurate. To answer why both algorithms used up the same memory, I think that perhaps memory profiler applications could be used to investigate this.

Nevertheless, out of curiosity, I decided to find out the number of nodes explored by each search algorithm in the graphs of the universities in Graph 2 that had an "explicitly" significant difference when the graph is looked at (i.e. Middlebury and MIT). I found out that in the Middlebury graph, DFS explored 304 nodes before finding node 745; however, BFS explored an outstanding 1500 nodes before finding node 745. In the graph of MIT, DFS explored 869 nodes and BFS explored a massive 3000 nodes before finding node 745. *Therefore, the number of nodes explored could be a reason for the memory size.*

University	(BFS) No. of nodes explored	(DFS) No. of nodes explored
Caltech	63	262
Reed	481	133
Haverford	500	500
Swarthmore	1000	1000
Middlebury	1500	304
Bucknell	2000	2000
John Hopkins	2500	2500
MIT	3000	869

However, after finding out the number of nodes explored in all graphs by both algorithms, I realised that my speculation of memory size and nodes explored was quite inconsistent with John Hopkins, Haverford, Swarthmore and Bucknell, as both algorithms had explored the same number of nodes, but still had DFS use less memory. In the case of Caltech, DFS explored more nodes than BFS, however, DFS still used lesser memory compared to BFS according to the data shown in Graph 2.

Even though the number of nodes explored did not have a direct relationship with the memory used by both algorithms, it gave me a sense of the position of node 745 in the graphs where the number of nodes explored were not same. For example, in the graph of Middlebury, I could predict that node 745 is deeper in the graph because it took DFS less memory and time to get to it than it took BFS because BFS was exploring breadthwise whereas DFS was exploring depth wise; Therefore, the DFS got to node 745 quicker using less memory than the BFS. In the case of Caltech, due to the difference in the number of nodes explored by both algorithms, I could predict that in its graph, node 745 was in a much wider position thus, BFS explored less nodes and got to it. Intuitively, if BFS explored less nodes, then it should be faster than DFS in finding node 745. However, according to graph 1, DFS took less time to find node 745.

Due to inconsistencies both in Graph 1 and Graph 2, I think a major flaw in my design is that fact that I did not have any method that could provide information on the position of node 745 or the structure of the graph implemented. Such an inclusion could have given better information to draw suitable conclusions from.

Possible extensions of this paper could include implementing one graph structure with a set number of nodes and randomized edges, and finding randomized nodes in the graph using DFS or BFS. Another extension of this paper could be the implementation of a graph that models an actual real-world data set of people's relationships, and finding out whether there might be values to do with time and memory that are more consistent.

Conclusions

According to my data, Depth first Search can generally be used for search in social graph data structures when the priorities of the organization implementing the system has to do with conserving memory and time. However, breadth first search can be used for search when such metrics are not priorities. Concerning the number of nodes explored and memory used, depth first search is mostly efficient. However, it might not always be the case as reflected by my data. Nevertheless, even though the time used by both search algorithms is dependent on the path they take –that is, whether deep first or wide first- there might still be other factors that are beyond the scope of this paper that are responsible for determining how much time and memory can be used by DFS or BFS.

References

Marr, Bernard. “Big Data: 20 Mind-Boggling Facts Everyone Must Read.” *Forbes*, Forbes Magazine, 19 Nov. 2015, www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-everyone-must-read/#62e17e3c17b1.

* All products require an annual contract. Prices do not include sales tax (New York residents only). “Number of Social Media Users Worldwide 2010-2021.” *Statista*, www.statista.com/statistics/278414/number-of-worldwide-social-network-users/.

Beal, Vangie. “Structured Data.” *What Is Structured Data? Webopedia Definition*, www.webopedia.com/TERM/S/structured_data.html.

McNulty, Eileen, et al. “Understanding Big Data: The Seven V's.” *Dataconomy*, 8 May 2017, dataconomy.com/2014/05/seven-vs-big-data/.

Bijulsoni. “Introduction to Graph with Breadth First Search(BFS) and Depth First Search(DFS) Traversal Implemented in JAVA.” *Introduction to Graph with Breadth First Search(BFS) and Depth First Search(DFS) Traversal Implemented in JAVA - CodeProject*, www.codeproject.com/Articles/32212/Introduction-to-Graph-with-Breadth-First-Search-BF.

"What Is Social Graph? - Definition from WhatIs.com." *WhatIs.com*, whatis.techtarget.com/definition/social-graph.

"Breadth First Search Tutorials & Notes | Algorithms." *HackerEarth*, www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/.

"Time and Space Complexity Tutorials & Notes | Basic Programming." *HackerEarth*, www.hackerearth.com/practice/basic-programming/complexity-analysis/time-and-space-complexity/tutorial/.

Appendices

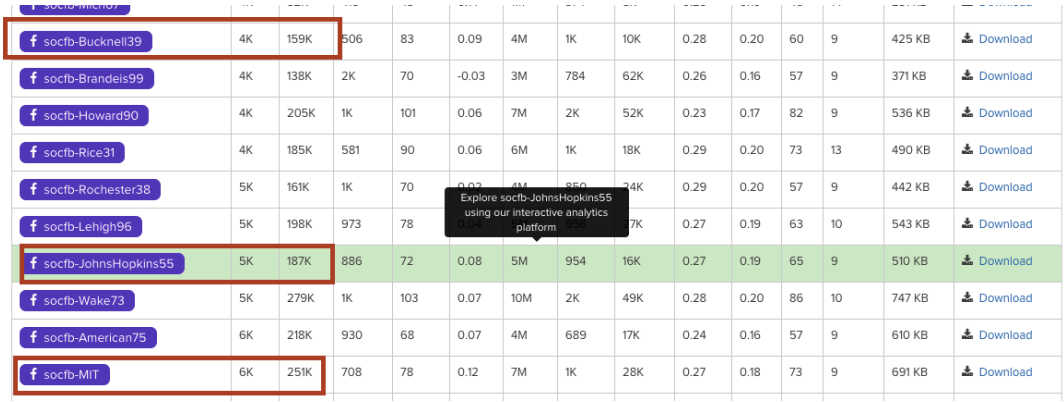
Appendix 1

			Breadth first search					Depth first search				
	No. of Nodes	NO. of Edges	memory/mb	Time /ms				memory/mb	Time /ms			
745				trial 1	trial 2	trial 3	avg.time		trial 1	trial 2	trial 3	avg. Time
University												
Caltech	769	17,000	118717440	77	78	66	73.67	118716416	64	60	59	61.00
Reed	962	19,000	117965824	58	78	71	69.00	117965824	58	68	56	60.67
Haverford	1000	60,000	134845440	146	184	135	155.00	132119552	136	139	131	135.33
			134845440					132119552				
Swarthmore	2000	61,000	134251520	166	166	177	169.67	134250496	158	111	169	146.00
Middlebury	3000	125,000	109386752	164	176	167	169.00	106659840	195	171	188	184.67
			109387776	199	179	164	180.67	106659840	213	178	166	185.67
Bucknell	4000	159,000	94415872	184	186	176	182.00	97141760	165	181	176	174.00
			94415872					97141760				
John hopkins	5000	187,000	87044096	861	774	718	784.33	84320256	1840	717	761	1106.00
			87044096	698	700	716	704.67	87042048	748	788	789	775.00
MIT	6000	251,000	173066240	877	1052	882	937.00	173041664	909	856	876	880.33
			173066240					173041664				

this shows the data table that was used in plotting Graph 1 and Graph 2.

Appendix 2

Graph Name	|V|	|E|	d_{max}	d_{avg}	r	|T|	T_{avg}	T_{max}	κ_{avg}	κ	K	ω_{heu}	Size	Download
socfb-Caltech36	769	17K	248	43	-0.07	359K	466	4K	0.41	0.29	36	12	46 KB	Download
socfb-Reed98	962	19K	313	39	0.02	291K	302	4K	0.32	0.22	35	10	52 KB	Download
socfb-Haverford76	1K	60K	375	82	0.07	2M	1K	13K	0.32	0.25	64	17	152 KB	Download
socfb-Simmons81	2K	33K	300	43	-0.06	506K	333	5K	0.31	0.21	35	10	91 KB	Download
socfb-Swarthmore42	2K	61K	577	73	0.06	2M	999	20K	0.30	0.23	61	11	157 KB	Download
socfb-Amherst41	2K	91K	467	81	0.06	3M	1K	15K	0.31	0.23	64	10	235 KB	Download
socfb-Bowdoin47	2K	84K	670	74	0.06	2M	946	22K	0.29	0.22	59	9	219 KB	Download
socfb-Hamilton46	2K	96K	602	83	0.03	3M	1K	19K	0.30	0.22	64	15	249 KB	Download
socfb-Trinity100	3K	112K	404	85	0.07	3M	1K	12K	0.29	0.23	68	11	290 KB	Download
socfb-USFCA72	3K	65K	405	48	0.09	1M	415	7K	0.27	0.19	44	10	179 KB	Download
socfb-Williams40	3K	113K	610	80	0.04	3M	1K	19K	0.29	0.21	63	14	295 KB	Download
socfb-nips-ego	3K	3K	769	2	-0.67	273	-	52	0.03	0.00	4	4	12 KB	Download
socfb-Oberlin44	3K	90K	478	61	0.05	2M	570	11K	0.26	0.17	50	11	241 KB	Download
socfb-Smith60	3K	97K	349	65	0.04	2M	641	7K	0.28	0.19	49	9	261 KB	Download
socfb-Wellesley22	3K	95K	746	63	0.06	2M	613	20K	0.26	0.17	52	10	254 KB	Download
socfb-Vassar85	3K	119K	482	77	0.10	3M	830	12K	0.25	0.18	61	10	315 KB	Download
socfb-Middlebury45	3K	125K	473	81	0.08	3M	1K	13K	0.28	0.21	63	13	327 KB	Download



socfb-Bucknell39	4K	159K	506	83	0.09	4M	1K	10K	0.28	0.20	60	9	425 KB	Download
socfb-Brandeis99	4K	138K	2K	70	-0.03	3M	784	62K	0.26	0.16	57	9	371 KB	Download
socfb-Howard90	4K	205K	1K	101	0.06	7M	2K	52K	0.23	0.17	82	9	536 KB	Download
socfb-Rice31	4K	185K	581	90	0.06	6M	1K	18K	0.29	0.20	73	13	490 KB	Download
socfb-Rochester38	5K	161K	1K	70				24K	0.29	0.20	57	9	442 KB	Download
socfb-Lehigh96	5K	198K	973	78				7K	0.27	0.19	63	10	543 KB	Download
socfb-JohnsHopkins55	5K	187K	886	72	0.08	5M	954	16K	0.27	0.19	65	9	510 KB	Download
socfb-Wake73	5K	279K	1K	103	0.07	10M	2K	49K	0.28	0.20	86	10	747 KB	Download
socfb-American75	6K	218K	930	68	0.07	4M	689	17K	0.24	0.16	57	9	610 KB	Download
socfb-MIT	6K	251K	708	78	0.12	7M	1K	28K	0.27	0.18	73	9	691 KB	Download

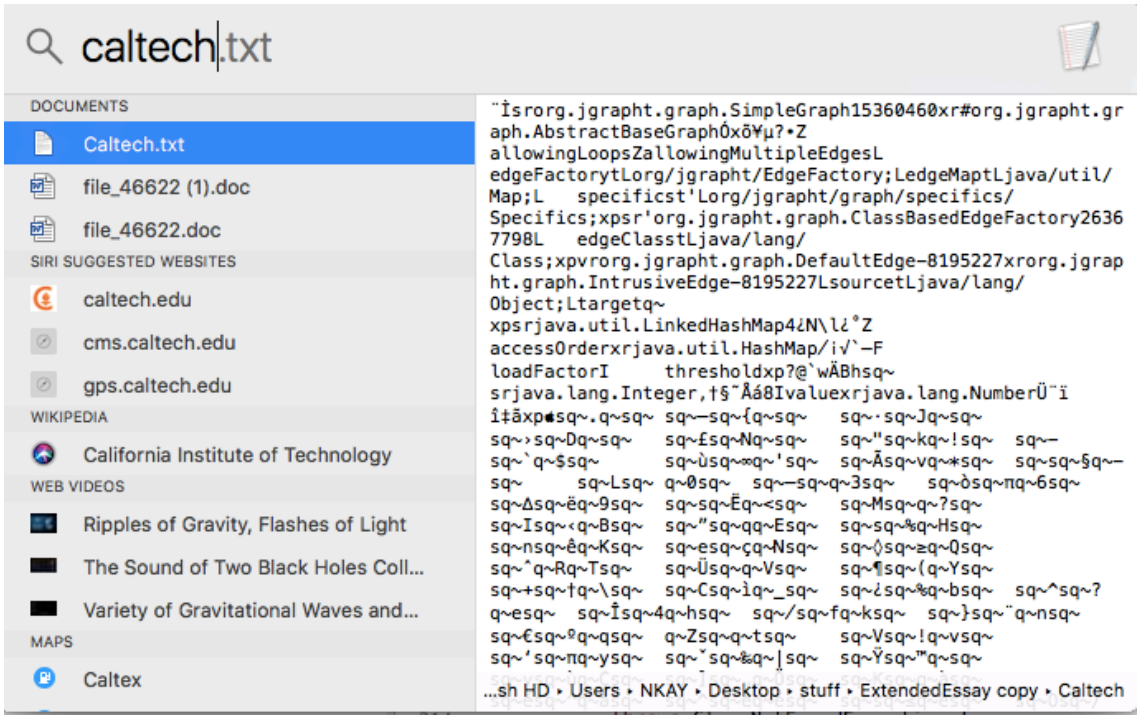
This shows the repository from which I took the number of nodes and edges from according per university Facebook data.

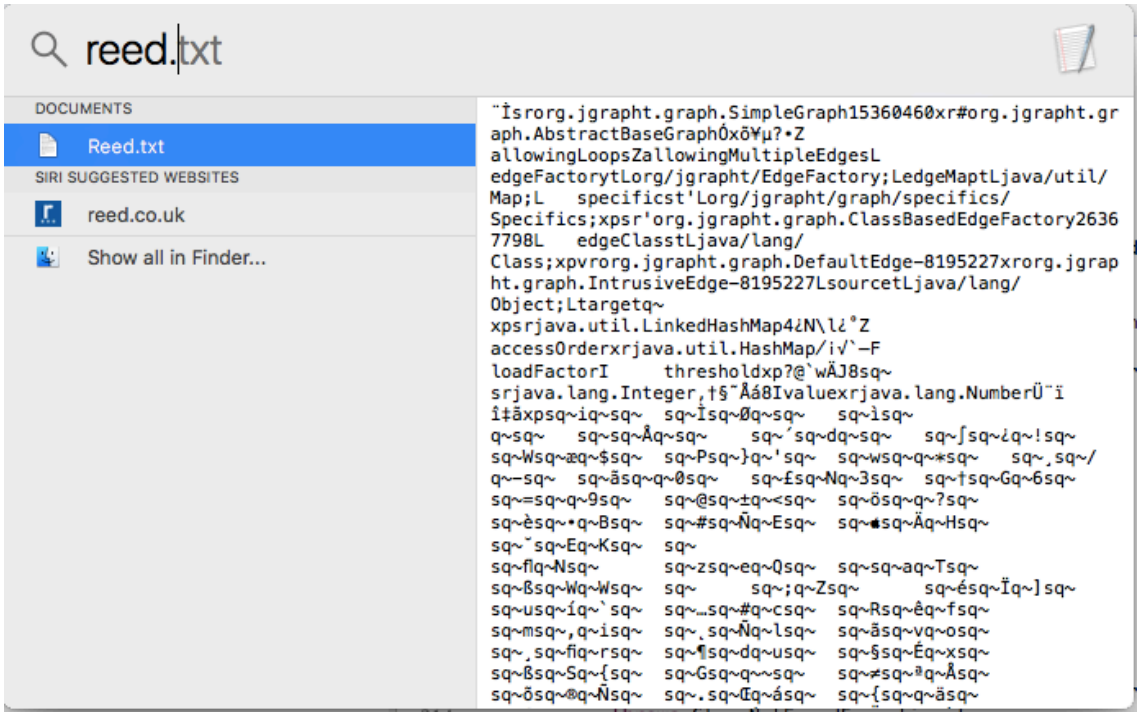
Appendix 3.

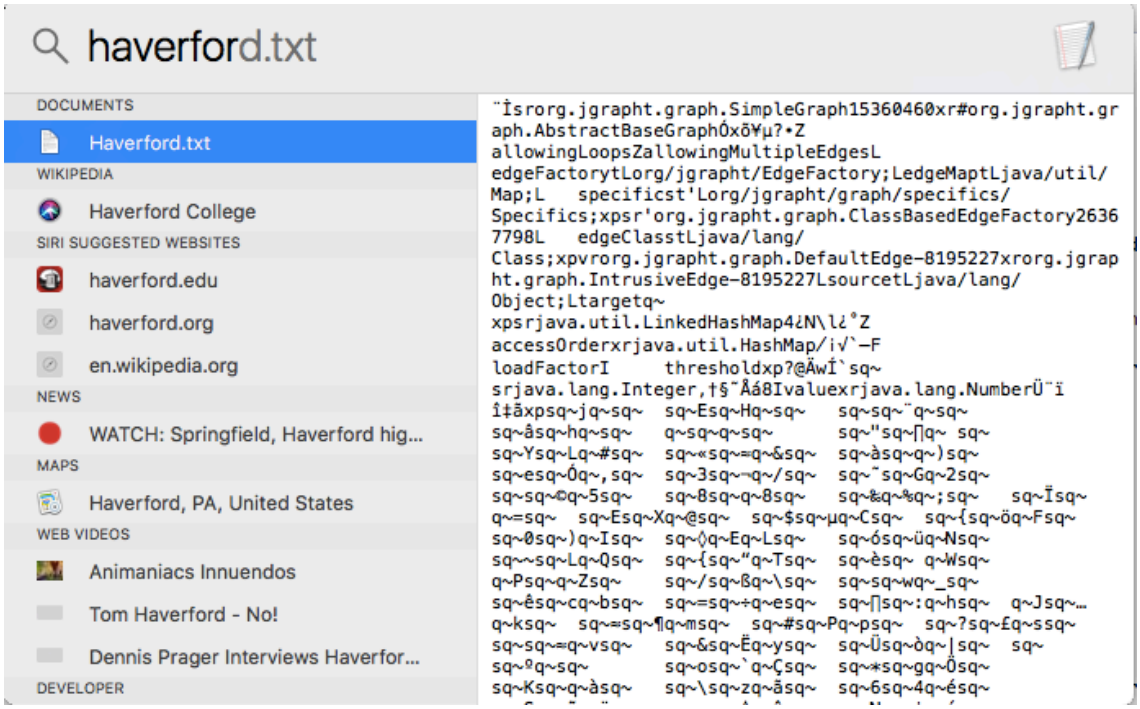
```
    public static void WriteGraph(Graph<Integer, DefaultEdge> Graph, String name) {

        try {
            FileOutputStream file = new FileOutputStream(name+".txt");

            ObjectOutputStream write = new ObjectOutputStream(file);

            write.writeObject(Graph);

            write.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

    @SuppressWarnings("unchecked")
    public static Graph<Integer, DefaultEdge> ReadGraph(Graph<Integer, DefaultEdge> Graph , String name)
            throws ClassNotFoundException {
        try {
            FileInputStream fileIn = new FileInputStream(name+".txt");

            ObjectInputStream read = new ObjectInputStream(fileIn);

            Graph = (org.jgrapht.Graph<Integer, DefaultEdge>) read.readObject();

            read.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        return Graph;

    }

}
```

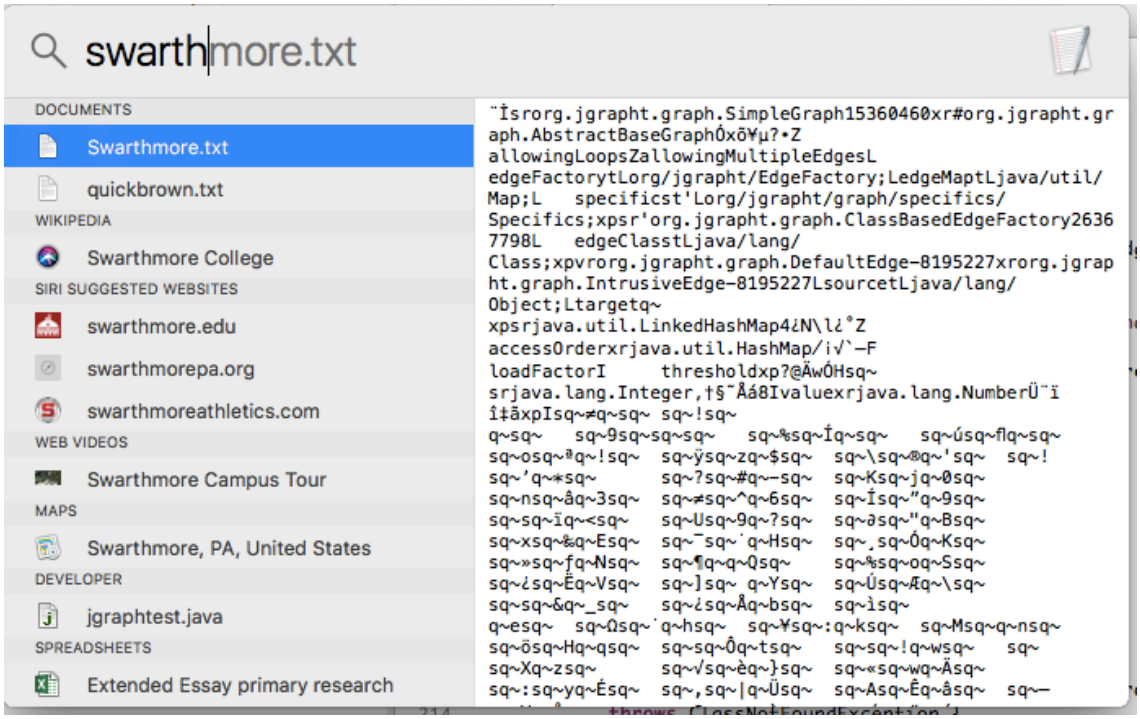
this shows my write graph and read graph method used in my program.

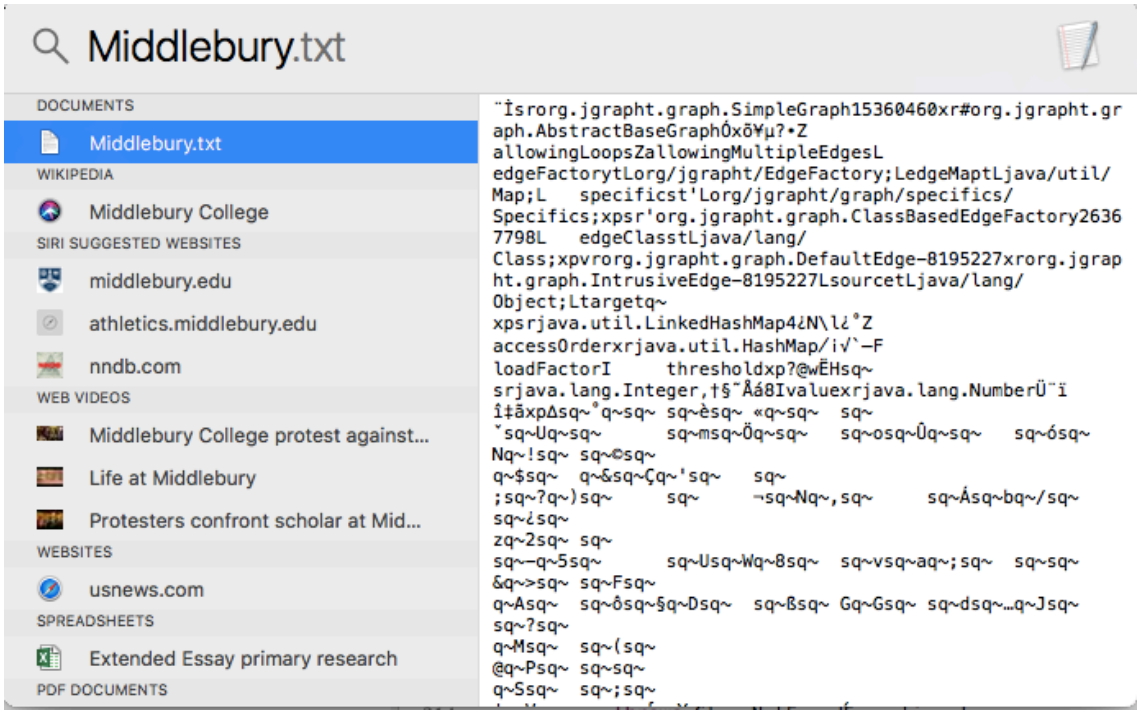
Appendix 4.

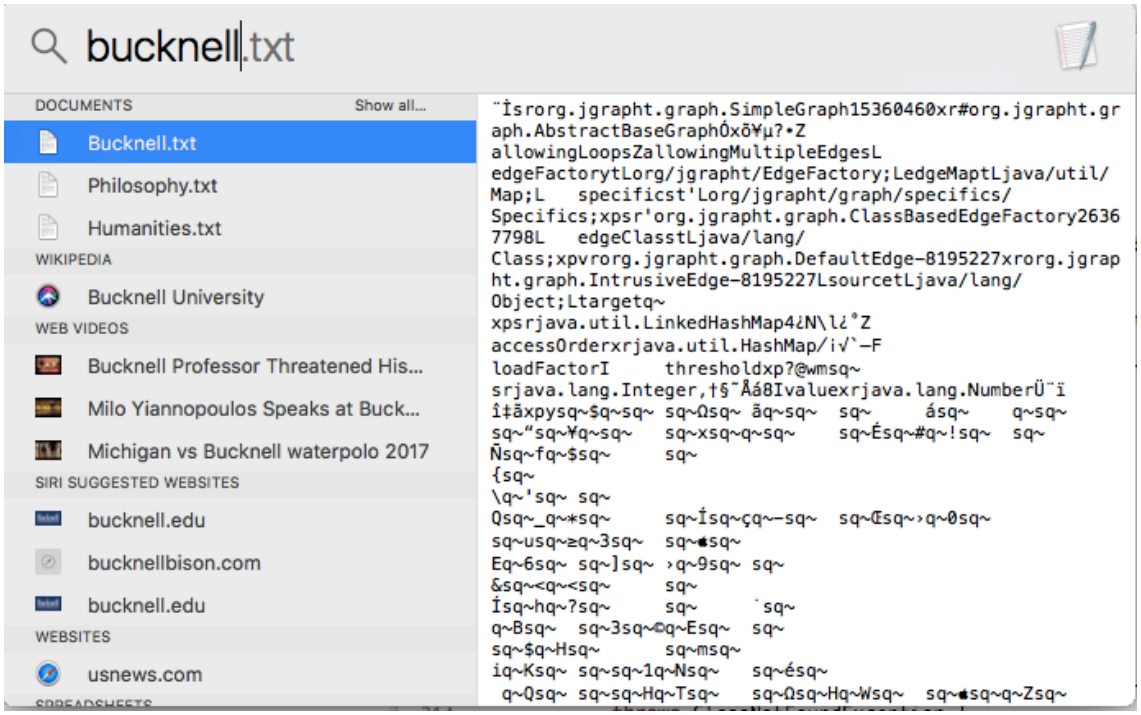

caltech.txt
DOCUMENTS
Caltech.txt
file_46622 (1).doc
file_46622.doc
SIRI SUGGESTED WEBSITES
caltech.edu
cms.caltech.edu
gps.caltech.edu
WIKIPEDIA
California Institute of Technology
WEB VIDEOS
Ripples of Gravity, Flashes of Light
The Sound of Two Black Holes Coll...
Variety of Gravitational Waves and...
MAPS
Caltex
¨Ìsrorg.jgrapht.graph.SimpleGraph15360460xr#org.jgrapht.graph.AbstractBaseGraphÓxõ¥µ?•Z
allowingLoopsZallowingMultipleEdgesL
edgeFactorytLorg/jgrapht/EdgeFactory;LedgeMaptLjava/util/Map;L specificst'Lorg/jgrapht/graph/specifics/Specifics;xpsr'org.jgrapht.graph.ClassBasedEdgeFactory26367798L edgeClasstLjava/lang/Class;xpvrorg.jgrapht.graph.DefaultEdge-8195227xrorg.jgrapht.graph.IntrusiveEdge-8195227LsourcetLjava/lang/Object;Ltargetq~
xpsrjava.util.LinkedHashMap4¿N\l¿°Z
accessOrderxrjava.util.HashMap/¡√`−F
loadFactorI thresholdxp?@`wÄBhsq~
srjava.lang.Integer,†§˜Åá8Ivaluexrjava.lang.NumberÜ¨ï
Î‡ãxp◆sq~.q~sq~ sq~—sq~{q~sq~ sq~·sq~Jq~sq~
sq~›sq~Dq~sq~ sq~£sq~Nq~sq~ sq~"sq~kq~!sq~ sq~—
sq~`q~$sq~ sq~ùsq~∞q~'sq~ sq~Ãsq~vq~*sq~ sq~sq~§q~—
sq~ sq~Lsq~ q~0sq~ sq~—sq~q~3sq~ sq~òsq~πq~6sq~
sq~Δsq~ëq~9sq~ sq~sq~Êq~<sq~ sq~Msq~q~?sq~
sq~Isq~‹q~Bsq~ sq~"sq~qq~Esq~ sq~sq~%q~Hsq~
sq~nsq~êq~Ksq~ sq~esq~çq~Nsq~ sq~◊sq~≥q~Qsq~
sq~^q~Rq~Tsq~ sq~Üsq~q~Vsq~ sq~¶sq~(q~Ysq~
sq~+sq~†q~\sq~ sq~Csq~ìq~_sq~ sq~¿sq~%q~bsq~ sq~^sq~?
q~esq~ sq~Îsq~4q~hsq~ sq~/sq~fq~ksq~ sq~}sq~¨q~nsq~
sq~€sq~ºq~qsq~ q~Zsq~q~tsq~ sq~Vsq~!q~vsq~
sq~'sq~πq~ysq~ sq~ˇsq~‰q~|sq~ sq~Ÿsq~™q~sq~
...sh HD ▸ Users ▸ NKAY ▸ Desktop ▸ stuff ▸ ExtendedEssay copy ▸ Caltech

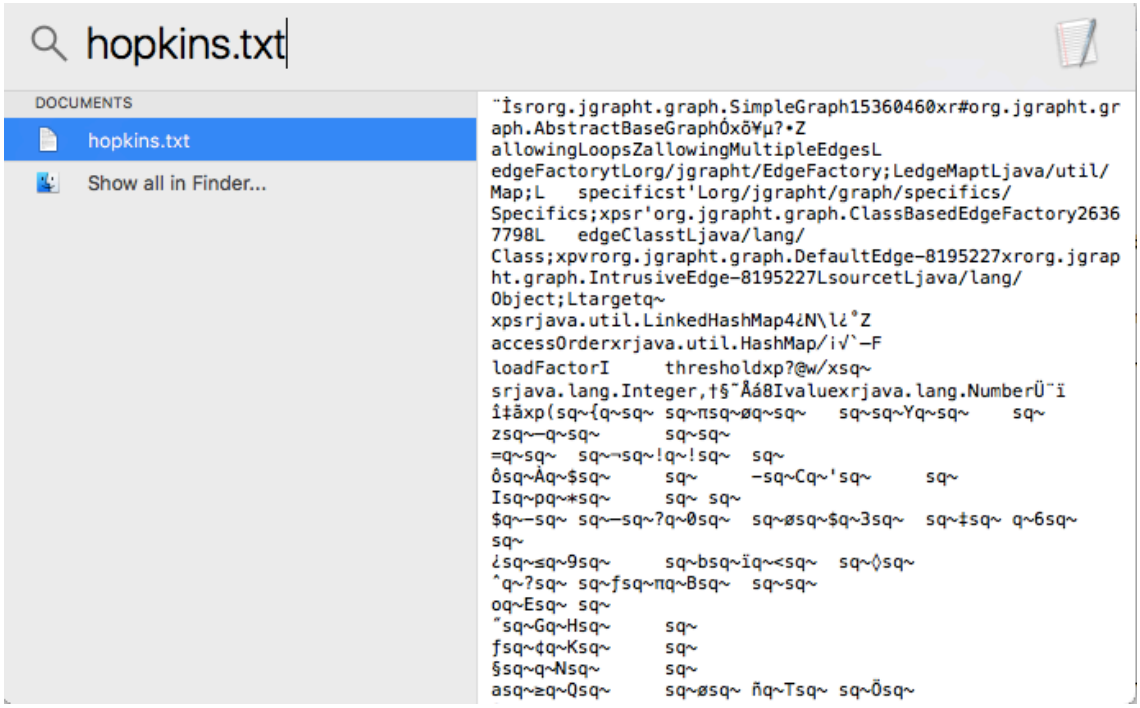

reed.txt
DOCUMENTS
Reed.txt
SIRI SUGGESTED WEBSITES
reed.co.uk
Show all in Finder...
¨Ìsrorg.jgrapht.graph.SimpleGraph15360460xr#org.jgrapht.graph.AbstractBaseGraphÓxõ¥µ?•Z
allowingLoopsZallowingMultipleEdgesL
edgeFactorytLorg/jgrapht/EdgeFactory;LedgeMaptLjava/util/Map;L specificst'Lorg/jgrapht/graph/specifics/Specifics;xpsr'org.jgrapht.graph.ClassBasedEdgeFactory26367798L edgeClasstLjava/lang/Class;xpvrorg.jgrapht.graph.DefaultEdge-8195227xrorg.jgrapht.graph.IntrusiveEdge-8195227LsourcetLjava/lang/Object;Ltargetq~
xpsrjava.util.LinkedHashMap4¿N\l¿°Z
accessOrderxrjava.util.HashMap/¡√`−F
loadFactorI thresholdxp?@`wÄJ8sq~
srjava.lang.Integer,†§˜Åá8Ivaluexrjava.lang.NumberÜ¨ï
Î‡ãxpsq~iq~sq~ sq~Ìsq~Øq~sq~ sq~ìsq~
q~sq~ sq~sq~Åq~sq~ sq~´sq~dq~sq~ sq~∫sq~¿q~!sq~
sq~Wsq~æq~$sq~ sq~Psq~}q~'sq~ sq~wsq~q~*sq~ sq~¸sq~/
q~−sq~ sq~ãsq~q~Øsq~ sq~£sq~Nq~3sq~ sq~†sq~Gq~6sq~
sq~=sq~q~9sq~ sq~@sq~±q~<sq~ sq~ösq~q~?sq~
sq~èsq~•q~Bsq~ sq~#sq~Ñq~Esq~ sq~◆sq~Äq~Hsq~
sq~ˇsq~Eq~Ksq~ sq~
sq~flq~Nsq~ sq~zsq~eq~Qsq~ sq~sq~aq~Tsq~
sq~ßsq~Wq~Wsq~ sq~ sq~;q~Zsq~ sq~ésq~Ïq~]sq~
sq~usq~íq~ˇsq~ sq~…sq~#q~csq~ sq~Rsq~êq~fsq~
sq~msq~,q~isq~ sq~¸sq~Ñq~lsq~ sq~ãsq~vq~osq~
sq~¸sq~fiq~rsq~ sq~¶sq~dq~usq~ sq~§sq~Êq~xsq~
sq~ßsq~Sq~{sq~ sq~Gsq~q~~sq~ sq~≠sq~ªq~Åsq~
sq~õsq~®q~Ñsq~ sq~.sq~Œq~ásq~ sq~{sq~q~äsq~

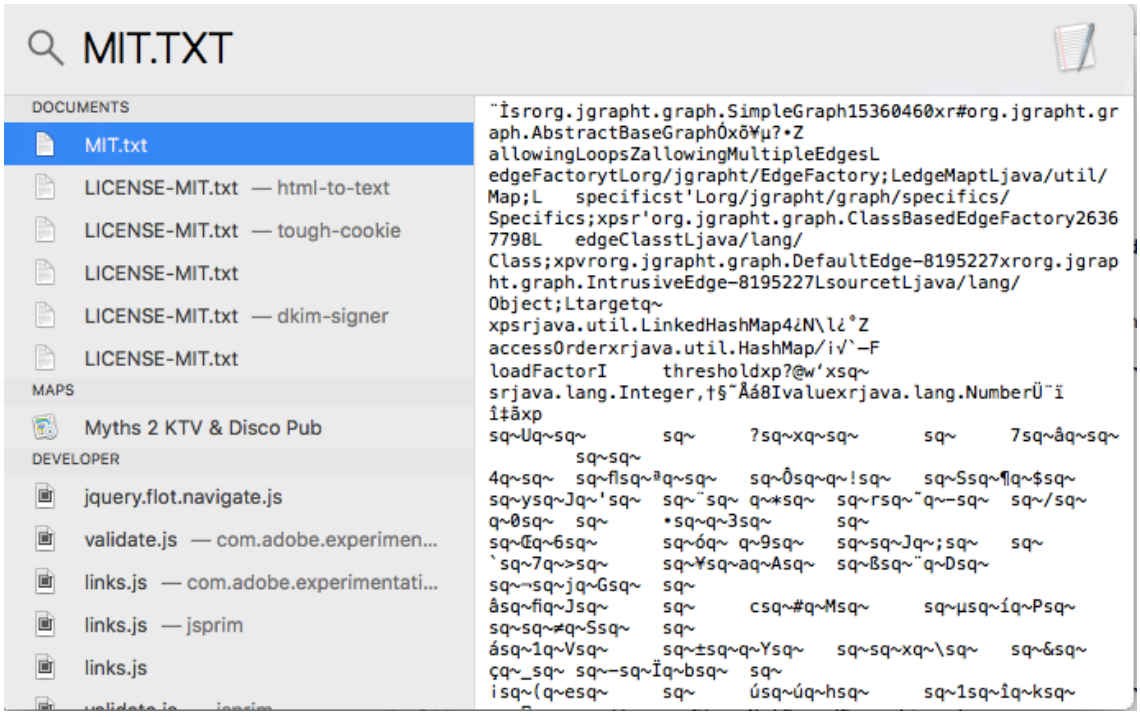

haverford.txt
DOCUMENTS
Haverford.txt
WIKIPEDIA
Haverford College
SIRI SUGGESTED WEBSITES
haverford.edu
haverford.org
en.wikipedia.org
NEWS
WATCH: Springfield, Haverford hig...
MAPS
Haverford, PA, United States
WEB VIDEOS
Animaniacs Innuendos
Tom Haverford - No!
Dennis Prager Interviews Haverfor...
DEVELOPER


swarthmore.txt
DOCUMENTS
Swarthmore.txt
quickbrown.txt
WIKIPEDIA
Swarthmore College
SIRI SUGGESTED WEBSITES
swarthmore.edu
swarthmorepa.org
swarthmoreathletics.com
WEB VIDEOS
Swarthmore Campus Tour
MAPS
Swarthmore, PA, United States
DEVELOPER
jgraphtest.java
SPREADSHEETS
Extended Essay primary research


Middlebury.txt
DOCUMENTS
Middlebury.txt
WIKIPEDIA
Middlebury College
SIRI SUGGESTED WEBSITES
middlebury.edu
athletics.middlebury.edu
nndb.com
WEB VIDEOS
Middlebury College protest against...
Life at Middlebury
Protesters confront scholar at Mid...
WEBSITES
usnews.com
SPREADSHEETS
Extended Essay primary research
PDF DOCUMENTS


bucknell.txt
DOCUMENTS
Show all...
Bucknell.txt
Philosophy.txt
Humanities.txt
WIKIPEDIA
Bucknell University
WEB VIDEOS
Bucknell Professor Threatened His...
Milo Yiannopoulos Speaks at Buck...
Michigan vs Bucknell waterpolo 2017
SIRI SUGGESTED WEBSITES
bucknell.edu
bucknellbison.com
bucknell.edu
WEBSITES
usnews.com





These show all the files generated.