Vaishak Menon

# CS303 Lab 11 – Breadth First Search

## Problem Specification:

For this lab we have to use LinkedLists and Queues to create a Breadth First Search program. The program takes in two input files, mediumG.txt and largeG.txt and runs breadth first search on it. We are also given the task of printing the path from the source to any given vertex.

## Program Design:

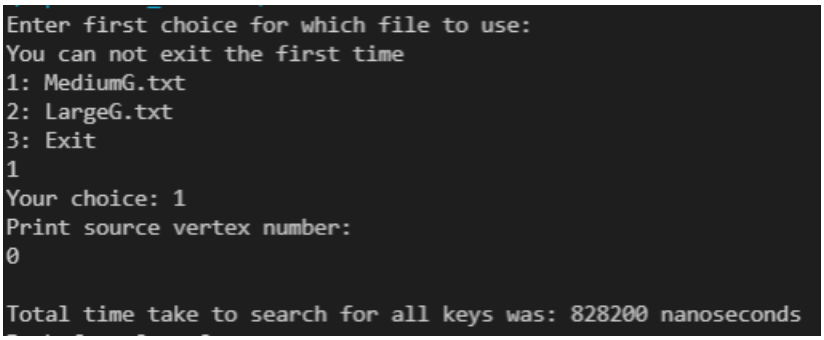
For this program, we had two functions. The first was the BFSearch, which is what I named the function, the idea is that it is the BreadthFirstSearch function. This function takes in a graph which I translated to a LinkedList of LinkedList of vertices and it also takes in a source vertex. The goal of the function is to essentially go through each vertex and its neighbors set its values, such as the parent, distance, and color. The second function is to print out the vertices and all of its connected neighbors/vertices. When writing my code, I eliminated the color and distance and replaced it with a Boolean value and a integer variable called accessed. Personally, I chose to use a Boolean value because it eliminated all of the need to check for color and it made it simpler for me to understand. I created the accessed variable because I needed to check how many times each vertex had been accessed to make sure that after it had been accessed once, it wouldn't get changed ever again.

## Testing Plan:

To test this, I read in both files, mediumG.txt and largeG.txt and create a vertex out of every number in the file and add it to a LinkedList array. This array is an array of LinkedLists and each index represents one of the vertices that is found in the text file. After adding all connected edges to a vertex to its corresponding array index, I run Breadth First Search with that array and a user input integer which is considered the source.

## Test Cases:

For test cases, I have put in a screen shot of a result of how long it took for BFSearch to go through the LinkedList.



```
Path from 0 to 230
0: 68 165 172 197 230
Path from 0 to 231
0: 44 168 208 231
Path from 0 to 232
0: 44 144 201 217 232
Path from 0 to 233
No path from 0 to 233 exists
Path from 0 to 235
0: 68 165 171 235 235
Path from 0 to 238
0: 68 165 171 238 238
Path from 0 to 239
No path from 0 to 239 exists
Path from 0 to 245
0: 68 165 171 238 245
Path from 0 to 248
0: 44 144 232 248
```

```
Enter first choice for which file to use:
You can not exit the first time
1: MediumG.txt
2: LargeG.txt
3: Exit
2
Your choice: 2
Print source vertex number:
0

Total time take to search for all keys was: 1835400 nanoseconds
```

←This is a sample path screen shot, it shows the path from the source which is the number on the left of the colon and the edges which are the numbers to the right of the colon.

## Analysis/Conclusion:

I am confused with my results because somehow, the time that it took for the largeG text file was greater, but not by so much that it would make sense. The mediumG file took approx. 0.0008 seconds while the largeG file took approx. 0.001 seconds. Technically this is a big difference, but I expected the difference to be much greater when looking at the size difference between each file.

## References:

The only references I must make are to the pseudocode and my previous lab in which I used a similar format for writing this report. Below I have added images of all my code for this lab.

```
import java.util.LinkedList;

public class BFS
{
    int origin;
    LinkedList<Vertex> input[];
    LinkedList<Vertex> reInput;

    public void BFSearch3(LinkedList<Vertex> V[], int source)
    {
        origin = source;
        LinkedList<Vertex> queue = new LinkedList<Vertex>();

        if(!(V[source].isEmpty()))
        {
            queue.add(V[source].removeFirst());
            while (queue.size() != 0)
            {
                reInput = new LinkedList<Vertex>();
                reInput.add(queue.get(0));
                Vertex num = queue.poll();

                int initialSize = V[num.id].size();
                while ( !(V[num.id].isEmpty()) && num.accessed <= initialSize)
                {
                    Vertex n = V[num.id].poll();

                    //System.out.println(n.id);
                    if (!n.checked)
                    {
                        n.checked = true;
                        n.p = num;
                        queue.add(n);
                        reInput.add(n);
                        num.accessed += 1;
                    }
                }

                if(reInput.size() != 0)
                {
                    V[num.id] = reInput;
                }
            }
        }
        input = V;
    }
```

```
public void printPath(LinkedList<Vertex> input[], Vertex s, Vertex v)
{
    if(s == v)
    {
        System.out.print(s.id + ": ");
    }
    else if(v.p == null)
    {
        System.out.print("No path from " + s.id + " to " + v.id + " exists");
    }
    else
    {
        printPath(input, s, v.p);
        System.out.print(v.id + " ");
    }

}
```

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.LinkedList;
import java.util.Scanner;

public class Driver
{
    Run | Debug
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter first choice for which file to use:");
        System.out.println("You can not exit the first time");
        String[] choices = new String[]{"MediumG.txt", "LargeG.txt", "Exit"};
        int i = 1;
        for(String s : choices)
        {
            System.out.println(i + ": " + s);
            i++;
        }

        int choice = sc.nextInt();
        System.out.println("Your choice: " + choice);
        File f;

        do
        {
            switch (choice)
            {
                case 1:
                BFS bfs = new BFS();
                f = new File("mediumG.txt");
                LinkedList<Vertex> V[];
                try
                {
                    Scanner sc2 = new Scanner(f);
                    int[] specs = new int[2];

                    int a = 0;
                    String[] s = sc2.nextLine().split(" ");
                    while (s.length == 1)
                    {
                        specs[a] = Integer.parseInt(s[0]);
                        s = sc2.nextLine().split(" ");
                        a++;
                    }

                    V = new LinkedList[specs[0]];
```

```java
case 2:
f = new File("largeG.txt");
try
{
    Scanner sc2 = new Scanner(f);
    int[] specs = new int[2];

    int a = 0;
    String[] s = sc2.nextLine().split(" ");
    while (s.length == 1)
    {
        specs[a] = Integer.parseInt(s[0]);
        s = sc2.nextLine().split(" ");
        a++;
    }

    V = new LinkedList[specs[0]];

    for(int b = 0; b < specs[0]; b++)
    {
        V[b] = new LinkedList<Vertex>();
    }

    while(sc2.hasNextLine())
    {
        s = sc2.nextLine().split(" ");
        int vID = Integer.parseInt(s[0]);
        Vertex eID = new Vertex(Integer.parseInt(s[1]));
        if(V[vID].size() == 0)
        {
            Vertex firstV = new Vertex(vID);
            V[vID].add(0,firstV);
        }
        V[vID].add(eID);
    }

    bfs = new BFS();

    System.out.println("Print source vertex number: ");
    int source = sc.nextInt();

    System.out.println();

    long bTime = System.nanoTime();
    bfs.BFSearch3(V, source);
    long ebTime = System.nanoTime() - bTime;
    System.out.println("Total time take to search for all keys was: " + ebTime + " nanoseconds");
```

```java
    for(int b = 0; b < specs[0]; b++)
    {
        V[b] = new LinkedList<Vertex>();
    }

    while(sc2.hasNextLine())
    {
        s = sc2.nextLine().split(" ");
        int vID = Integer.parseInt(s[0]);
        Vertex eID = new Vertex(Integer.parseInt(s[1]));
        if(V[vID].size() == 0)
        {
            Vertex firstV = new Vertex(vID);
            V[vID].add(0,firstV);
        }
        V[vID].add(eID);
    }

    bfs = new BFS();

    System.out.println("Print source vertex number: ");
    int source = sc.nextInt();

    System.out.println();

    long bTime = System.nanoTime();
    bfs.BFSearch3(V, source);
    long ebTime = System.nanoTime() - bTime;
    System.out.println("Total time take to search for all keys was: " + ebTime + " nanoseconds");

    for(int b = 0; b < V.length; b++)
    {
        if(V[b].size() != 0)
        {
            System.out.println("Path from " + source + " to " + b);
            bfs.printPath(V, V[source].getFirst(), V[b].getFirst());
            System.out.println();
        }
    }

    sc2.close();
}
catch (FileNotFoundException e)
{
    System.out.println(e);
}
break;
```

```java
                    /*
                    for(int b = 0; b < V.length; b++)
                    {
                        if(V[b].size() != 0)
                        {
                            System.out.println("Path from " + source + " to " + b);
                            bfs.printPath(V, V[source].getFirst(), V[b].getFirst());
                            System.out.println();
                        }
                    }
                    */

                    sc2.close();
                }
                catch (FileNotFoundException e)
                {
                    System.out.println(e);
                }
                break;
            }
            System.out.println();

            i = 1;
            for (String s : choices)
            {
                System.out.println(i + ": " + s);
                i++;
            }
            System.out.println("Enter file number from menu: ");
            choice = sc.nextInt();
            System.out.println("Your choice: " + choice);
        }
        while(choice != 3);

        sc.close();
    }
}
```

```java
public class Vertex
{
    boolean checked = false;
    int id;
    Vertex p = null;
    int accessed = 0;

    public Vertex(int ID)
    {
        id = ID;
    }
}
```