Program 3 – Marvel Universe Graph

Jordan Stein

Introduction

Using the “six degrees of separation” concept from the six degrees of Kevin Bacon game, I have constructed two different Java programs to calculate the separation degree of SPIDERMAN for the entire Marvel Universe. The data used to find these values comes from “porgat.txt”, which contains every Marvel character and an association of every comic book they appear in. The Spiderman number is calculated by comparing the comics of every character against each other to find their proper association to a Spiderman comic. I have also constructed a collaboration matrix between all marvel characters to determine the number of instances where each character has been in a comic with another. A total collaboration and number of collaborating pairs is outputted within the first program. The two programs operate nearly the same, the only difference is that one program represents the Marvel graph in a matrix and the other program uses an array of lists to represent the data.

Here is what the input file, “porgat.txt” looks like.

```
*Vertices 19428 6486
1 "24-HOUR MAN/EMMANUEL"
2 "3-D MAN/CHARLES CHAN"
3 "4-D MAN/MERCURIO"
4 "8-BALL/"
5 "A"
6 "A'YIN"
7 "ABBOTT, JACK"
8 "ABCISSA"
9 "ABEL"
10 "ABOMINATION/EMIL BLO"
…
6484 "STORMER"
6485 "TIGER WYLDE"
6486 "ZONE"
6487 "AA2 35"
6488 "M/PRM 35"
6489 "M/PRM 36"
6490 "M/PRM 37"
6491 "WI? 9"
6492 "AVF 4"
6493 "AVF 5"
…
19426 "AA2 30"
19427 "AA2 20"
19428 "AA2 38"
*Edgeslist
1 6487
2 6488 6489 6490 6491 6492 6493 6494 6495 6496
3 6497 6498 6499 6500 6501 6502 6503 6504 6505
4 6506 6507 6508
5 6509 6510 6511
6 6512 6513 6514 6515
7 6516
8 6517 6518
9 6519 6520
10 6521 6522 6523 6524 6525 6526 6527 6528 6529 6530 6531 6532 6533 6534 6535
10 6536 6537 6538 6539 6540 6541 6542 6543 6544 6545 6546 6547 6548 6549 6550
10 6551 6552 6553 6554 6555 6556 6557 6558 6559 6560 6561 6562 6563 6564 6565
…
6484 18709
6485 15336
6486 15336
```

Storage of Comic Information

I had created a class named Characters to represent each character. MarvelGraph uses the 2D matrix of Booleans to determine who is in what comic book, so the Characters class for MarvelGraph contains members for their name, file number, a 2D array of Booleans to determine if they are in a certain comic, and their spider number. MarvelGraph2's character class is nearly identical however it uses an array of integer lists that store the comics for each character.

Below is my class for MarvelGraph (2D Boolean storage)

```
import java.util.ArrayList;

public class Characters {
	private String name="";
	private String num;
	public Boolean[] comics = new Boolean[19429];
	public int spiderNum=-1; // anyone without a spiderman number will be listed as -1

	public Characters(String name, String num){
		this.name = name;
		this.num = num;

		for (int i=0; i < comics.length; i++)
		{
			comics[i] = false; // initializes all comics to false when character is
created
		}
	}

	public String getName(){
		return this.name;
	}

	public String getNum(){
		return this.num;
	}
}
```

Below is the Characters class for MarvelGraph2 (ArrayList of integers containing comic book numbers)

```
import java.util.ArrayList;

public class Characters {
	private String name="";
	private String num;
	public ArrayList<Integer> comicsList = new ArrayList<Integer>(); // list of all comics
this character is in.
	public int spiderNum=-1; // anyone without a spiderman number will be listed as -1

	public Characters(String name, String num){
		this.name = name;
		this.num = num;
	}

	public String getName(){
		return this.name;
	}

	public String getNum(){
		return this.num;
	}
}
```

I made a function named readData that returns an array of Characters with all comic book data calculated and stored. Both programs store the data identically. When a comic book is found in MarvelGraph, the Boolean for that index is flipped to True. When a comic book is found in MarvelGraph2, the comic's integer value is pushed onto the ArrayList.

Below is the readData function for MarvelGraph

```
public static Characters[] readData(){
		File file = new File("porgat.txt");
		Characters[] readIn = new Characters[6487];
		try{
			Scanner sc = new Scanner(file);
			int i = 0;
			while (sc.hasNextLine() && i < 6487)
			{
				String data = sc.nextLine();

				char[] dataCh = data.toCharArray();

				int j=0;
				while (j < dataCh.length) // calculates the offset for inserting name string
into array.
				{
					if (!Character.isDigit(dataCh[j]))
						break;
					j++; // j stores that offset
				}

				if (i > 0)
				{					// reads that offset into the substring of
what was read in from the file.
					readIn[i-1] = new Characters(data.substring(j+2, data.length()-1),
data.substring(0,j));
				}					// creates new character, first field is
the name, second field is their number.
				i++;
			}
			while (i < 19430)
			{
				String comicVerticies = sc.nextLine();
				i++;
			}
			// now the scanner is at the edgelist.

			String edgeList[] = new String[30520-19430];
			i=0;
			while(sc.hasNextLine())
			{
				edgeList[i] = sc.nextLine(); // stores all edges into the array.
				i++;
			}

			i=0;
			while (i < edgeList.length)
			{
				String[] edges = edgeList[i].split(" "); // creates a string array, splitting
all edge values.
				for (int j = 1; j < edges.length; j++)
				{
					int edgeNum = Integer.parseInt(edges[0]);
					edgeNum--; //decrement because edge list starts at 1.
					readIn[edgeNum].comics[Integer.parseInt(edges[j])] = true; // stores
the edge value at the edgeNum index of the array.
				}
				i++;
			}

			sc.close();
		}
		catch (FileNotFoundException e)
		{
			e.printStackTrace();
		}
		return readIn;
	}
```

Collaboration Matrix

In the first program, MarvelGraph, I have constructed a collaboration matrix using a 2D integer array that counts the amount of occurrences of shared comics between every character. To access any count, you can index the ith and the jth character using collab[i][j]. After the matrix was created, I parsed through the matrix to keep a count of the total amount of collaborations and the total collaborating pairs. After these counts were made, I divided the total collaborating pairs by two to prevent myself from double counting and outputted the two values. The algorithm I used to make the matrix is very slow and has a O(N^3) time complexity because we are comparing the entire comics book list for every character against each other using three for loops.

The function I wrote for this is below.

```
public static void collaborationMatrix(Characters[] ch){

	int[][] collab = new int[ch.length][ch.length]; // collaboration matrix

	for (int i=0; i < ch.length-1; i++)
	{
		int count = 0; // reset count for next character

		for(int j=1; j < ch.length-2; j++)
		{
			for (int k=0; k < ch[i].comics.length; k++)
			{
				if (ch[i].comics[k] == ch[j].comics[k] && i != j)
				count++; // increment count if their comics are the same
			}
		collab[i][j] = count; // store that count into the collab matrix
		}
	}

	int totalCollab = 0;
	int totalCollabPairs = 0;

	for(int i=0; i < ch.length; i++)
	{
		for(int j=1; j < ch.length-1; j++) // iterate through every element in
the collaboration matrix
		{
			if (collab[i][j] > 0) // if there is a collaboration,
			{
				totalCollab++;    // increment the total collab count
				totalCollabPairs += collab[i][j]; // add that collab value
to the total pairs amount.
			}
		}
	}

	totalCollabPairs /= 2; // divide total collab pairs by 2 because we double
counted in the calculation.

	System.out.println("Total number of collaborations = " + totalCollab);
	System.out.println("Total number of collaborating pairs of characters = " +
totalCollabPairs);
	}
```

Six degrees of Spider Man

To calculate the Spiderman number for each character, I took the array of characters and compared them against each other to compute the separation degree from Spiderman. I first parsed through the array until Spiderman was found, in which I stored his index and stopped parsing. After I obtained Spiderman's index, I compared the comics for every character to Spiderman's comic list. If they had appeared in the same comic as Spiderman, I assigned their spider number to 1 and pushed that character into a temporary list named numberOners to represent everybody who had a spider man number of one. I then re-iterated through the array. If the characters spider number was still at the default value of -1, I would compare their comics to everybody in the numberOners list. If there was an associated comic, I would assign their spider number to 2 and push all of those characters into a new list, numberTwoers. I continued to do this until every associated character was assigned a spider number. An accesses count kept track of the total amount of vertiсy accesses for each algorithm. I will compare these numbers in the Analysis section.

The largest separation from spider man in the entire Marvel Universe is 3. Any character who was not assigned a spider number has no association to any character who does have a spider number. Those characters retain the default value of -1 to represent this.

Below is the code for calculating the spider numbers in MarvelGraph (2D Boolean array)

```
public static int computeSpiderNum(Characters[] ch){

		int accesses = 0;

		int spidermanIndex = 0;
		Boolean foundSpidey = false;

		int i=0;
		while (!foundSpidey) // set spiderman's spider number to zero.
		{
			for (int j=0; j < ch[i].comics.length; j++)
			{
				if (ch[i].getName().equals("SPIDER-MAN/PETER PAR"))
				{
					ch[i].spiderNum = 0; // set spiderNum to zero for spider
man himself.
					spidermanIndex = i; // stores spider man's index
					foundSpidey = true;
					break;
				}
			}
			i++;
		}

		if (spidermanIndex == 0)
			spidermanIndex = 5305; // hard coded his index in case user didn't input
enough characters to get to spider man.


		//System.out.println("Amount whose spiderNum is 0 = " + 1);

		ArrayList<Characters> numberOners = new ArrayList<Characters>(); // stores
everyone with a spiderman number of 1


		for (i = 0; i < ch.length-1; i++) // calculate everyone who has a spiderman number
of 1.
		{
```

```
                        for (int j=0; j < ch[i].comics.length; j++)
                        {
                                if (ch[i].comics[j] == true && ch[spidermanIndex].comics[j] == true
&& i != j)
                                {
                                        ch[i].spiderNum = 1; // this character is in a comic with
spiderman.
                                        numberOners.add(ch[i]);
                                        break;
                                }
                                accesses++;
                        }
                }

                //System.out.println("Amount whose spiderNum is 1 = " + numberOners.size());

                ArrayList<Characters> numberTwoers = new ArrayList<Characters>(); // stores
everyone with a spiderman number of 2
                for (i = 0; i < ch.length-1; i++)
                {
                        if (ch[i].spiderNum == -1)
                        {
                                for (int j = 0; j < ch[i].comics.length; j++)
                                {
                                        if (ch[i].comics[j] == true && i != j)
                                        {
                                                for (int k=0; k < numberOners.size(); k++)
                                                {
                                                        if (numberOners.get(k).comics[j] == true)
                                                        {
                                                                ch[i].spiderNum = 2;
                                                                numberTwoers.add(ch[i]);
                                                                break;
                                                        }
                                                }
                                        }
                                        accesses++;
                                }
                        }
                }

                //System.out.println("Amount whose spiderNum is 2 = " + numberTwoers.size());

                ArrayList<Characters> numberThreeers = new ArrayList<Characters>(); // stores
everyone with a spiderman number of 3
                for (i = 0; i < ch.length-1; i++)
                {
                        if (ch[i].spiderNum == -1)
                        {
                                for (int j = 0; j < ch[i].comics.length; j++)
                                {
                                        if (ch[i].comics[j] == true  && i != j)
                                        {
                                                for (int k=0; k < numberTwoers.size(); k++)
                                                {
                                                        if (numberTwoers.get(k).comics[j] == true)
                                                        {
                                                                ch[i].spiderNum = 3;
                                                                numberThreeers.add(ch[i]);
                                                                break;
                                                        }
                                                }
                                        }
                                        accesses++;
                                }
                        }
                }

                //System.out.println("Amount whose spiderNum is 3 = " + numberThreeers.size());
```

```
		ArrayList<Characters> numberFourers = new ArrayList<Characters>(); // stores
everyone with a spiderman number of 3
		for (i = 0; i < ch.length-1; i++)
		{
			if (ch[i].spiderNum == -1)
			{
				for (int j = 0; j < ch[i].comics.length; j++)
				{
					if (ch[i].comics[j] == true  && i != j)
					{
						for (int k=0; k < numberThreeers.size(); k++)
						{
							if (numberThreeers.get(k).comics[j] == true)
							{
								ch[i].spiderNum = 4;
								numberFourers.add(ch[i]);
								break;
							}
						}
					}
					accesses++;
				}
			}
		}

		return accesses;
	}
```

MarvelGraph2 (edge list) uses the same algorithm to compute the spider numbers, however it only iterates through the comic list for each character, rather than iterating though the entire list of Booleans. Below is an excerpt of the edge list computation code. (Since code is similar to MarvelGraph's)

```
		ArrayList<Characters> numberTwoers = new ArrayList<Characters>(); // stores
everyone with a spiderman number of 2
		for (i = 0; i < ch.length-1; i++) // calculate everyone who has a spiderman number
of 1.
		{
			innerLoop:
			if (ch[i].spiderNum == -1)
			{
				for (int j=0; j < ch[i].comicsList.size(); j++)
				{
					int comic = ch[i].comicsList.get(j);
					for (int k=0; k < numberOners.size(); k++)
					{
						for (int l=0; l <
numberOners.get(k).comicsList.size(); l++)
						{
							if (comic ==
numberOners.get(k).comicsList.get(l))
							{
								ch[i].spiderNum = 2; // this
character is in a comic with spiderman.
								numberTwoers.add(ch[i]);
								break innerLoop;
							}
						}
					}
					accesses++;
				}
			}
		}
```

Output

When either program is run, it prompts the user to input the amount of marvel characters they wish to view. You can input as many marvel characters that are in the progat.txt file as you'd like. It then prints each character and their associated spider man number. If their Spiderman number was -1, they do not have any correlation with any character that does have a spider man number. Afterword's, the total amount of vertices accesses for computing the spider number for each representation is outputted.

Output for MarvelGraph: (2D Boolean matrix)

```
How many marvel characters would you like to view?
10
24-HOUR MAN/EMMANUEL has spider man number of 3
3-D MAN/CHARLES CHAN has spider man number of 1
4-D MAN/MERCURIO has spider man number of 2
8-BALL/ has spider man number of 2
A has spider man number of 2
A'YIN has spider man number of 2
ABBOTT, JACK has spider man number of 1
ABCISSA has spider man number of 2
ABEL has spider man number of 2
ABOMINATION/EMIL BLO has spider man number of 1

It took 204602224 vertiсy accesses to compute every spiderman number for the entire matrix.
```

Output for MarvelGraph2: (Edge list for comics)

```
How many marvel characters would you like to view?
10
24-HOUR MAN/EMMANUEL has spider man number of 3
3-D MAN/CHARLES CHAN has spider man number of 1
4-D MAN/MERCURIO has spider man number of 2
8-BALL/ has spider man number of 2
A has spider man number of 2
A'YIN has spider man number of 2
ABBOTT, JACK has spider man number of 1
ABCISSA has spider man number of 2
ABEL has spider man number of 2
ABOMINATION/EMIL BLO has spider man number of 1

It took 34532 vertiсy accesses to compute every spiderman number for every comic list.
```

Output for collaboration matrix in MarvelGraph:

```
How many marvel characters would you like to view?
1
Total number of collaborations = 42055223
Total number of collaborating pairs of characters = 612401820
24-HOUR MAN/EMMANUEL has spider man number of 3

It took 204602224 vertiсy accesses to compute every spiderman number for the entire matrix.
```

Analysis

Comparing the outputs of the two programs against each other, you can clearly determine that the edge list representation is far superior to the 2D Boolean matrix representation. The edge list representation only required 34532 vertices accesses while the matrix representation required 204602224 accesses, even though the same algorithm format was used in both calculations. This difference makes sense because the 2D Boolean matrix representation had to iterate through the entire Boolean array every time it was trying to compute a Spiderman number. The edge list only needed to compare the comics list for each character to compute the Spiderman numbers. Because these comics lists only included the comics the characters were in, rather than storing a true or false value for every comic, many less comparisons were required for that algorithm. Correspondingly, the run-time for matrix is slower than the run-time for the edge list representation.

The time complexity for computing the Spiderman numbers for the matrix is O(N^3). This is because we must iterate through the Booleans array for each character and compare that against every other character. I have three for loops in the code for this, one to iterate through the array of characters, another to compare each character against each other, and a final one to access every comic for each character.

The space complexity for computing the Spiderman numbers for the matrix is O(1). This is because the amount of array lists created to compare the Spiderman numbers against each other is constant regardless of the input.

The time complexity for computing the Spiderman numbers for the edge list is O(N^4). I used four for loops to calculate the numbers. The first for loop iterates for each character, the second loop iterates for the entire comic list length for each character, the third loop iterates through the list of every character with a certain Spiderman number, and the forth loop iterates through those characters comics list. I have short circuited the algorithm wherever possible.

Although this implementation has a higher worst-case time complexity, the run-time for this data is much faster because it iterates less times than the matrix representation (since the Boolean array is so large).

The space complexity for computing the Spiderman numbers for the edge list is also O(1) because the amount of array lists created is also constant regardless of the input.

Conclusion

Both programs generate the same output as expected. I wasn't surprised that the matrix representation was much slower than the edge list representation due to the matrix being so large and spacious. Neither program takes too long to complete however because the input file is rather short. The most challenging part of these programs was calculating the spider man numbers; however it was not too difficult. I enjoyed writing both programs.