CSE222 DATA STRUCTURES AND ALGORITHMS

HOMEWORK 8

HATİCE SEVRA GENÇ

1801042611

PART 2

```
public ArrayList<Integer> connectCounterBFS(int start) {
```

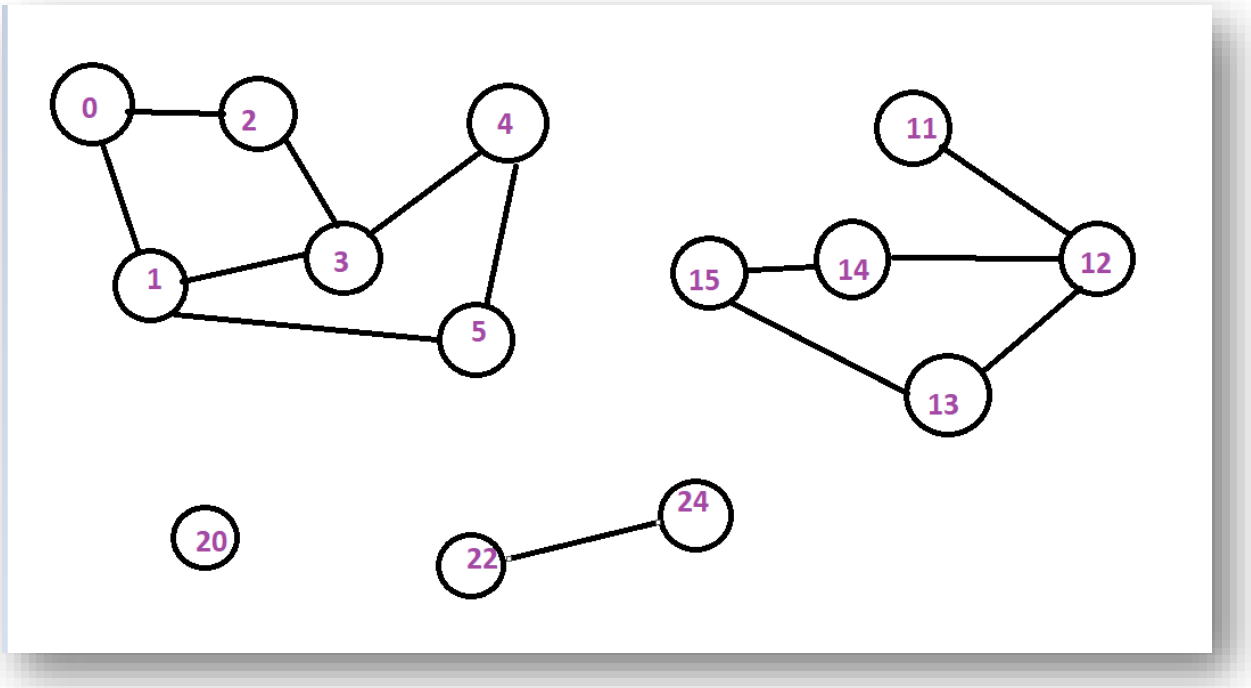
```
// DFSUtil()
public ArrayList<Integer> connectCounterDFS(int v) {
```

I implement those functions to do the operations in this part. Functions returns the connected components as ArrayList.

I use **Graph** class in this part. When the code is run, firstly **sample** of this part is printed.

```
private void part2() {
    System.out.println("********** PART2 ************");

    sample();
```

I draw the graph in the sample function:



This graph have multiple connections

```
SAMPLE FUNCTION
ALL VERTEXES IN THE GRAPH = [0, 1, 2, 3, 4, 5, 11, 12, 13, 14, 15, 20, 22, 24]
Connected Components Part 1 with BREADTH FIRST TRAVERSAL
[0, 2, 1, 3, 5, 4]
Number Of Connected Components: 6

Connected Components Part 1 with DEPTH FIRST TRAVERSAL
[0, 2, 3, 4, 1, 5]
Number Of Connected Components: 6
```

As you see in this photo, the function detects connected components then traverse it BFS and DFS methods.

Then do the same thing rest of the graph.

```
Connected Components Part 2 with BREADTH FIRST TRAVERSAL
[11, 12, 13, 14, 15]
Number Of Connected Components: 5

Connected Components Part 2 with DEPTH FIRST TRAVERSAL
[11, 12, 13, 15, 14]
Number Of Connected Components: 5

Connected Components Part 3 with BREADTH FIRST TRAVERSAL
[20]
Number Of Connected Components: 1

Connected Components Part 3 with DEPTH FIRST TRAVERSAL
[20]
Number Of Connected Components: 1

Connected Components Part 4 with BREADTH FIRST TRAVERSAL
[22, 24]
Number Of Connected Components: 2

Connected Components Part 4 with DEPTH FIRST TRAVERSAL
[22, 24]
Number Of Connected Components: 2
```

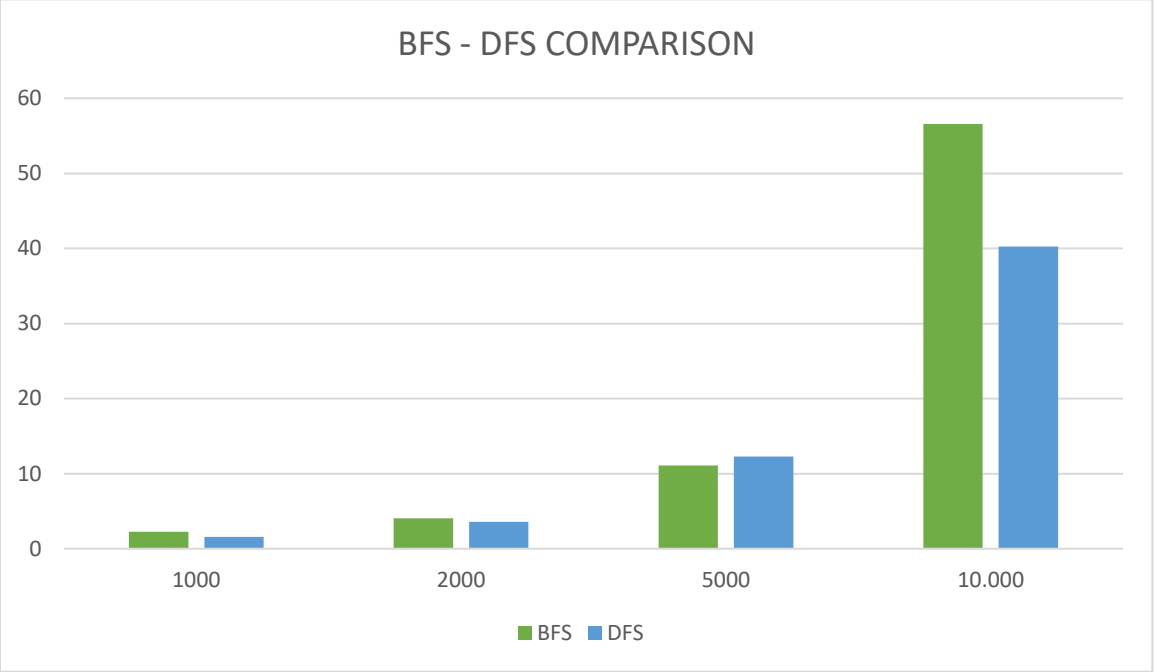
- ## RUNNING TIME RESULTS

I create graphs by generating random numbers. Then connect some of them randomly to traverse BFS and DFS. I did this process for 4 size * 10 times * 2 methods. After that calculate the average running time for each size:

```
------------------------------------ RANDOM GRAPH 1000 ----------------------------------------

Average of 10 times BFS (size 1000) ->  2,2727 ms
Average of 10 times DFS (size 1000) ->  1,6007 ms

------------------------------------ RANDOM GRAPH 2000 ----------------------------------------

Average of 10 times BFS (size 2000) ->  4,0410 ms
Average of 10 times DFS (size 2000) ->  3,5844 ms

------------------------------------ RANDOM GRAPH 5000 ----------------------------------------

Average of 10 times BFS (size 5000) ->  11,0984 ms
Average of 10 times DFS (size 5000) ->  12,2902 ms

------------------------------------ RANDOM GRAPH 10.000 ----------------------------------------

Average of 10 times BFS (size 10.000) ->  56,5995 ms
Average of 10 times DFS (size 10.000) ->  40,2538 ms
Process finished with exit code 0
```

- **COMPARISON OF RESULTS**

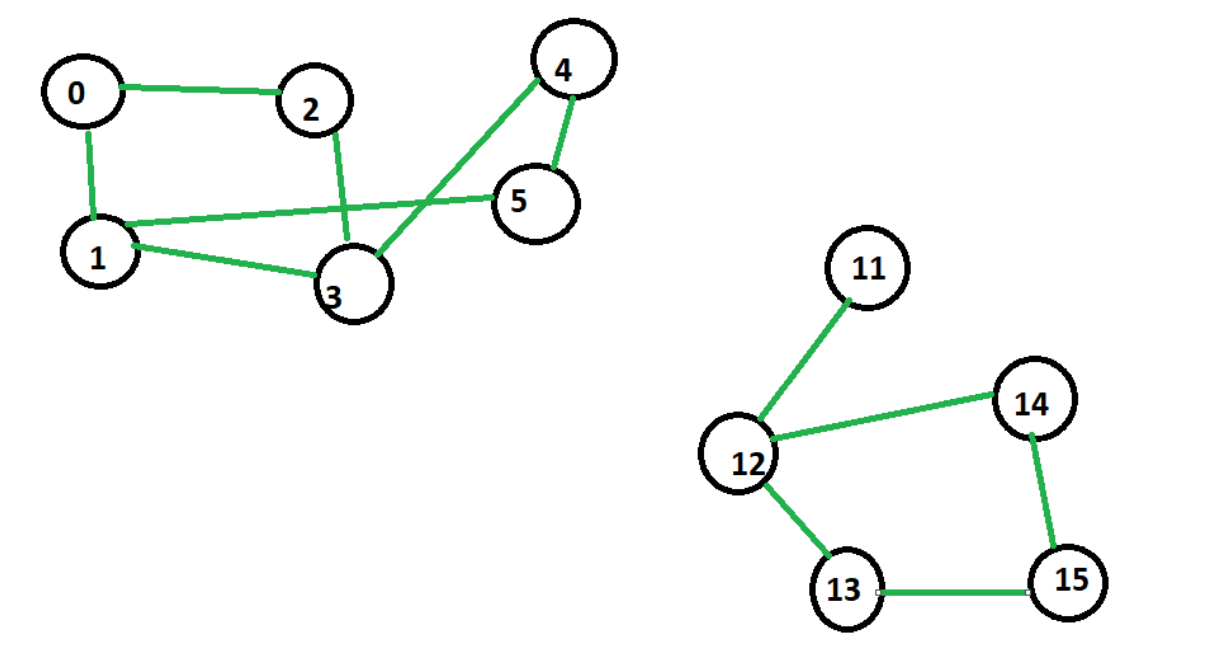
	BFS	DFS
1000	2,2727	1,6007
2000	4,0410	3,5844
5000	11,0984	12,2902
10.000	56,5965	40,2538



The time spent increases linearly for all sizes. BFS seems to be more efficient than DFS. As the size of the processed data increases, the efficiency of BFS also increases.

PART 3

In this part I use this graph:



This graph has 2 different connected component groups: (0,1,2,3,4,5) and (11,12,13,14,15). I detect this groups using functions in Part 2. Then I assign u,w and v values in order.

For example when **u=11, w=15 and v=14**. My shortest paths are:

[11, 12 ,13, 15]

[11, 12 ,14, 15]

So **sigma_uw=2.** Because of one of the paths not passes v=14 vertex **sigma_uw(v)=1.** I calculate sigma_uw(v)/sigma_uw value for v=14. Then do the same thing for every v value for each (u,w) pair.

I print only non-zero values:

```
********** PART3 ************
All vertexes = [0, 1, 2, 3, 4, 5, 11, 12, 13, 14, 15]

Connected Vertices: 0 , 2 , 1 , 3 , 5 , 4 ,
u-> 0  w-> 3  v-> 2
Sigma uw(v) / sigma uw:  0.5
Current Total:  0.5

u-> 0  w-> 4  v-> 2
Sigma uw(v) / sigma uw:  0.3333333333333333
Current Total:  0.8333333333333333

u-> 0  w-> 3  v-> 1
Sigma uw(v) / sigma uw:  0.5
Current Total:  0.5

u-> 0  w-> 5  v-> 1
Sigma uw(v) / sigma uw:  1.0
Current Total:  1.5
```

Continuing like this, it calculates the importance of all connected vertices and adds them all. Then we divide this value to square of num of connected components (equals 6*6 here).

```
TOTAL : 6,0000
NUM OF CONNECTED VERTICES: 6
FAIR IMPORTANCE VALUE :  0,1667
```

We found **fair importance value** of this connected component group.

```
Connected Vertices: 11 , 12 , 13 , 14 , 15 ,
u-> 11  w-> 13  v-> 12
Sigma uw(v) / sigma uw:  1.0
Current Total:  1.0

u-> 11  w-> 14  v-> 12
Sigma uw(v) / sigma uw:  1.0
Current Total:  2.0
```

We do same thing to other group of connected components

And the result is:

```
TOTAL : 5,0000
NUM OF CONNECTED VERTICES: 5
FAIR IMPORTANCE VALUE :  0,2000
```

So we find **‘normalized importance value of each vertex’** and **‘fair importance value of each connected component group’**.

All results can see in the driver code

[Hocam bu kısmı doğru anlamak için çok uğraştım. Neyin fair importance olduğu, nasıl bir graf üzerinden işlem yapılacağı, bulunan değerin neyin karesine bölüneceği gibi konularda birçok soru işaretim oldu. Veriler elimdeydi ama kavramlarla eşleştirmekte biraz zorlandım. Sizden aldığım cevaplar ve kendi araştırmalarım sonucunda istenilen değerlerin bunlar olduğuna karar verdim. Umarım hata çıkmaz 😊]

- ## CLASS DIAGRAM