Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 7 REPORT

STUDENT NAME: YAĞMUR KARAMAN
STUDENT NUMBER: 141044016

Course Assistant: Fatma Nur Esirci

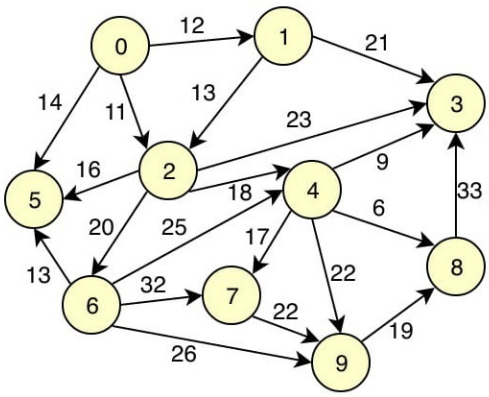
1 Q1

This part about Question1 in HW7

1.1 Problem Solution Approach

Toplam 10 vertex ve 20 edge'e sahip weighted bir graph oluşturdum. Tüm bu bilgileri test_part1.txt dosyası içerisine yazdım ve oradan okuma yaparak graph create ettim.

shortest_path bulurken parametre olarak gelen düğümlerin sahip olduğu tüm pathleri buldum ve ağırlıklarını hesapladım. Ağırlığı en az tutan path'i vector aracılığıyla tuttum ve return ettim.



-V=10
-E=15
-Directed
-Random weighted
-Acyclic

1.2 Test Cases

-Öncelikle dosyadan okunan vertex ve weight bilgilerine göre graph oluşturuldu.
-plot_graph methodu ile tüm edgeler print edildi.
-is_undirected methodu ile graph'ın directed veya undirected olduğu kontrol edildi.
-is_acyclic_graph methodu ile graph'ın herhangi bir cyclic'e sahip olup olmadığı kontrol edildi.
-shortest_path methodu 3 farklı source, destination kombinasyonu ile test edildi. İlk olarak (0,3), ikinci olarak (1,5) ve son olarak (4,8) düğümleri test edildi.

```
BufferedReader bf = new BufferedReader(new FileReader( fileName: "./src/files/test_part1.txt"));
Scanner scan = new Scanner(bf);
ListGraph listGraph = (ListGraph) AbstractGraph.createGraph(scan, isDirected: true, type: "List");
GraphMethodsHomework hwGraph = new GraphMethodsHomework();
System.out.println("plot_graph method:");
System.out.println("--------------");
hwGraph.plot_graph(listGraph);

System.out.print("\nis_undirected method: ");
if(hwGraph.is_undirected(listGraph)) {
    System.out.println("This graph is undirected");
}
else
    System.out.println("This graph is directed");

System.out.print("\nis_acyclic_graph method: ");
hwGraph.fillListD(listGraph.getNumV());
for(int i=0; i<listGraph.getNumV(); ++i) {
    for(int j=0; j<listGraph.getNumV(); ++j) {
        if(listGraph.isEdge(i,j))
            hwGraph.addEdgeD(i,j);
    }
}
if(hwGraph.is_acyclic_graph(listGraph)) {
    System.out.println("This graph is cyclic");
}
else
    System.out.println("This method is acyclic");
```

```
        System.out.print("\nshortest_path for (0, 3): ");
        Vector<Integer> vect= hwGraph.shortest_path(listGraph, i: 0, j: 3);
        for(int i=0; i<vect.size(); ++i)
            System.out.print(vect.get(i) + " ");

        System.out.print("\nshortest_path for (1, 5): ");
        Vector<Integer> vect2= hwGraph.shortest_path(listGraph, i: 1, j: 5);
        for(int i=0; i<vect2.size(); ++i)
            System.out.print(vect2.get(i) + " ");

        System.out.print("\nshortest_path for (4, 8): ");
        Vector<Integer> vect3= hwGraph.shortest_path(listGraph, i: 4, j: 8);
        for(int i=0; i<vect3.size(); ++i)
            System.out.print(vect3.get(i) + " ");
    }
}
```

Output:

```
MainTest (1)
[(7, 9): 22.0]
[(8, 0): Infinity]
[(8, 1): Infinity]
[(8, 2): Infinity]
[(8, 3): 33.0]
[(8, 4): Infinity]
[(8, 5): Infinity]
[(8, 6): Infinity]
[(8, 7): Infinity]
[(8, 8): Infinity]
[(8, 9): Infinity]
[(9, 0): Infinity]
[(9, 1): Infinity]
[(9, 2): Infinity]
[(9, 3): Infinity]
[(9, 4): Infinity]
[(9, 5): Infinity]
[(9, 6): Infinity]
[(9, 7): Infinity]
[(9, 8): 19.0]
[(9, 9): Infinity]

is_undirected method: This graph is directed

is_acyclic_graph method: This method is acyclic

shortest_path for (0, 3): 0 1 3
shortest_path for (1, 5): 1 2 5
shortest_path for (4, 8): 4 8
Process finished with exit code 0
```

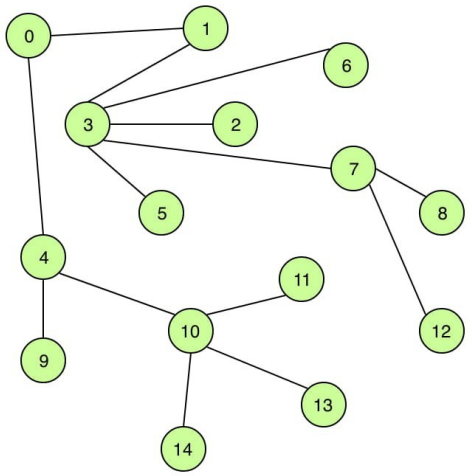
-Oluşturulan graph directed, acyclic bir graphtı. Bu yüzden doğru sonuçlar print edildi.
-0,3 arasında 2 farklı path bulunmaktadır. Birincisi 0->2->3 path'i, bu path'in toplam ağırlığı 34, diğer path ise 0->1>3 path'i ve bu path'in ağırlığı da 33. Bu yüzden en kısa olan 0->1->3, yani doğru sonuç print edildi. Diğer 2 test de aynı şekilde bakıldığında en kısa yol olduğu görülmektedir.

2 Q2

This part about Question2 in HW7

2.1 Problem Solution Approach

Graph oluşturulurken ödev pdf'inde de istenildiği gibi V=15, undirected, acyclic ve no weighted olacak şekilde test_part2.txt dosyasından okunarak oluşturuldu.



-V=15
-Undirected
-Acyclic
-No weighted

2.2 Test Cases

-Öncelikle dosyadan okunan vertex bilgilerine göre graph oluşturuldu.
-plot_graph methodu ile tüm edgeler print edildi.
-is_undirected methodu ile graph'ın directed veya undirected olduğu kontrol edildi.
-is_acyclic_graph methodu ile graph'ın herhangi bir cyclic'e sahip olup olmadığı kontrol edildi.
-is_connected methodu 3 farklı source, destination kombinasyonu ile test edildi. İlk olarak (0,3), ikinci olarak (1,5) ve son olarak (4,8) düğümleri test edildi.

```
BufferedReader bf = new BufferedReader(new FileReader( fileName: "./src/files/test_part2.txt"));
Scanner scan = new Scanner(bf);
ListGraph listGraph = (ListGraph) AbstractGraph.createGraph(scan, isDirected: false, type: "List");
GraphMethodsHomework hwGraph = new GraphMethodsHomework();
System.out.println("plot_graph method:");
System.out.println("--------------");
hwGraph.plot_graph(listGraph);

System.out.print("\nis_undirected method: ");
if(hwGraph.is_undirected(listGraph)) {
    System.out.println("This graph is undirected");
}
else
    System.out.println("This graph is directed");

System.out.print("\nis_acyclic_graph method: ");
hwGraph.fillListU(listGraph.getNumV());
for(int i=0; i<listGraph.getNumV(); ++i) {
    for(int j=0; j<listGraph.getNumV(); ++j) {
        if(listGraph.isEdge(i,j))
            hwGraph.addEdgeU(i,j);
    }
}
if(!hwGraph.is_acyclic_graph(listGraph)) {
    System.out.println("This graph is cyclic");
}
else
    System.out.println("This method is acyclic");
```

```java
/*************************************************************/
/*********** is_connected 1 ***********************************/
/*************************************************************/
System.out.print("\nis_connected for (0, 13): ");
if(hwGraph.is_connected(listGraph, v1: 0, v2: 13))
    System.out.println("This vertices are connected");

else
    System.out.println("This vertices are not connected");

/*************************************************************/
/*********** is_connected 2 ***********************************/
/*************************************************************/
System.out.print("is_connected for (2, 9): ");
if(hwGraph.is_connected(listGraph, v1: 2, v2: 9))
    System.out.println("This vertices are connected");

else
    System.out.println("This vertices are not connected");

/*************************************************************/
/*********** is_connected 3 ***********************************/
/*************************************************************/
System.out.print("is_connected for (5, 7): ");
if(hwGraph.is_connected(listGraph, v1: 5, v2: 7))
    System.out.println("This vertices are connected");

else
    System.out.println("This vertices are not connected");
```

Output:

```
MainTest (2)
[(13, 10): 1.0]
[(13, 11): Infinity]
[(13, 12): Infinity]
[(13, 13): Infinity]
[(13, 14): Infinity]
[(14, 0): Infinity]
[(14, 1): Infinity]
[(14, 2): Infinity]
[(14, 3): Infinity]
[(14, 4): Infinity]
[(14, 5): Infinity]
[(14, 6): Infinity]
[(14, 7): Infinity]
[(14, 8): Infinity]
[(14, 9): Infinity]
[(14, 10): 1.0]
[(14, 11): Infinity]
[(14, 12): Infinity]
[(14, 13): Infinity]
[(14, 14): Infinity]

is_undirected method: This graph is undirected

is_acyclic_graph method: This method is acyclic

is_connected for (0, 13): This vertices are connected
is_connected for (2, 9): This vertices are connected
is_connected for (5, 7): This vertices are connected

Process finished with exit code 0
```

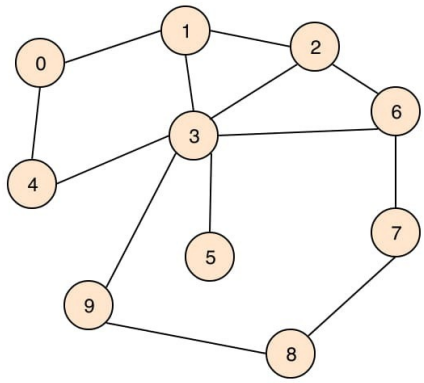
-Oluşturulan graph, undirected ve acyclic bir graphtı. Bu yüzden doğru print edildi.
-is_connected methodu için (0,13), (2,9) ve (5,7) düğümleri test edildi. Graph yapısına baktığımızda tüm bu düğümlerin connected olduğu görülmektedi. Bu yüden doğru print edilmiştir.

3 Q3

This part about Question3 in HW7

3.1 Problem Solution Approach

Graph oluşturulurken ödev pdf'inde de istenildiği gibi V=10, undirected, cyclic ve no weighted olacak şekilde test_part3.txt dosyasından okunarak oluşturuldu.



-V=10
-Cyclic
-No weighted
-Undirected

3.2 Test Cases

-Öncelikle dosyadan okunan vertex bilgilerine göre graph oluşturuldu.
-plot_graph methodu ile tüm edgeler print edildi.
-is_undirected methodu ile graph'ın directed veya undirected olduğu kontrol edildi.
-is_acyclic_graph methodu ile graph'ın herhangi bir cyclic'e sahip olup olmadığı kontrol edildi.
-BreadthFirstSearch ve DepthFirstSearch classları ile oluşturulan graplar üzerinde traverse edildi.

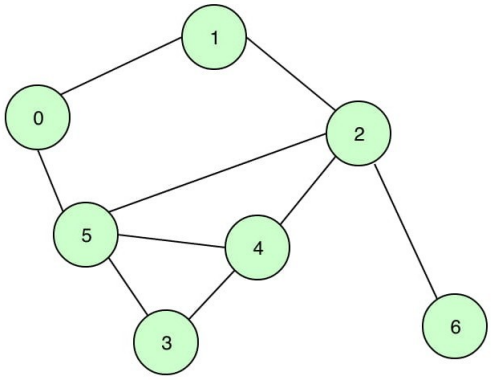
```
    BufferedReader bf = new BufferedReader(new FileReader( fileName: "./src/files/test_part3.txt"));
    Scanner scan = new Scanner(bf);
    ListGraph listGraph = (ListGraph) AbstractGraph.createGraph(scan, isDirected: false, type: "List");
    GraphMethodsHomework hwGraph = new GraphMethodsHomework();
    System.out.println("plot_graph method:");
    System.out.println("---------------");
    hwGraph.plot_graph(listGraph);

    System.out.print("\nis_undirected method: ");
    if(hwGraph.is_undirected(listGraph)) {
        System.out.println("This graph is undirected");
    }
    else
        System.out.println("This graph is directed");

    System.out.print("\nis_acyclic_graph method: ");
    hwGraph.fillListU(listGraph.getNumV());
    for(int i=0; i<listGraph.getNumV(); ++i) {
        for(int j=0; j<listGraph.getNumV(); ++j) {
            if(listGraph.isEdge(i,j))
                hwGraph.addEdgeU(i,j);
        }
    }
    if(!hwGraph.is_acyclic_graph(listGraph)) {
        System.out.println("This graph is acyclic");
    }
    else
        System.out.println("This method is cyclic");
```

```
        System.out.println("\n*********************************************");
        System.out.println("********         BFS               ***********");
        System.out.println("*********************************************");
        BreadthFirstSearch bfs = new BreadthFirstSearch();
        bfs.breadthFirstSearch(listGraph, start: 1);

        System.out.println("\n\n*********************************************");
        System.out.println("********         DFS               ***********");
        System.out.println("*********************************************");
        DepthFirstSearch dfs = new DepthFirstSearch(listGraph);
        int i;
        for(i=0; i<dfs.discoveryOrder.length; ++i) {
            System.out.print(dfs.discoveryOrder[i]);
            System.out.print(" -> ");
        }
    }
}
```

Output:

```
MainTest
[(8, 5): Infinity]
[(8, 6): Infinity]
[(8, 7): 1.0]
[(8, 8): Infinity]
[(8, 9): 1.0]
[(9, 0): Infinity]
[(9, 1): Infinity]
[(9, 2): Infinity]
[(9, 3): 1.0]
[(9, 4): Infinity]
[(9, 5): Infinity]
[(9, 6): Infinity]
[(9, 7): Infinity]
[(9, 8): 1.0]
[(9, 9): Infinity]

is_undirected method: This graph is undirected

is_acyclic_graph method: This method is cyclic

*********************************************
********         BFS               ***********
*********************************************
1 -> 0 -> 3 -> 2 -> 4 -> 5 -> 9 -> 6 -> 8 -> 7 ->

*********************************************
********         DFS               ***********
*********************************************
0 -> 1 -> 3 -> 4 -> 5 -> 9 -> 8 -> 7 -> 6 -> 2 ->
Process finished with exit code 0
```

4 Q4



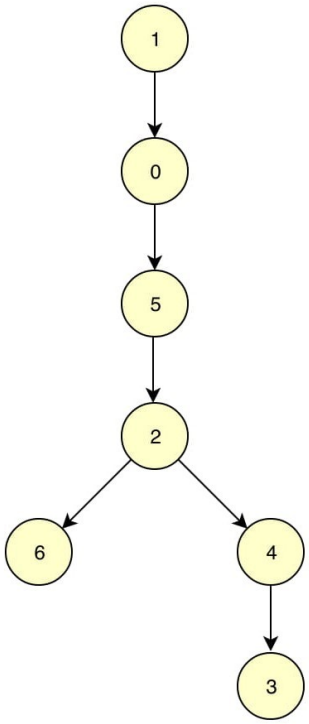
Verilen matrix ile oluşan graph yandaki gibidir.

1 numaralı vertexten başlarsak;

DFS: 1 -> 0 -> 5 -> 2 -> 6 -> 4 -> 3 şeklinde olabilir.

BFS: 1 -> 0 -> 2 -> 5 -> 6 -> 4 -> 3 şeklinde olabilir.

DFS Tree:



BFS Tree: