Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 6 REPORT

BATUHAN TOPALOĞLU
151044026

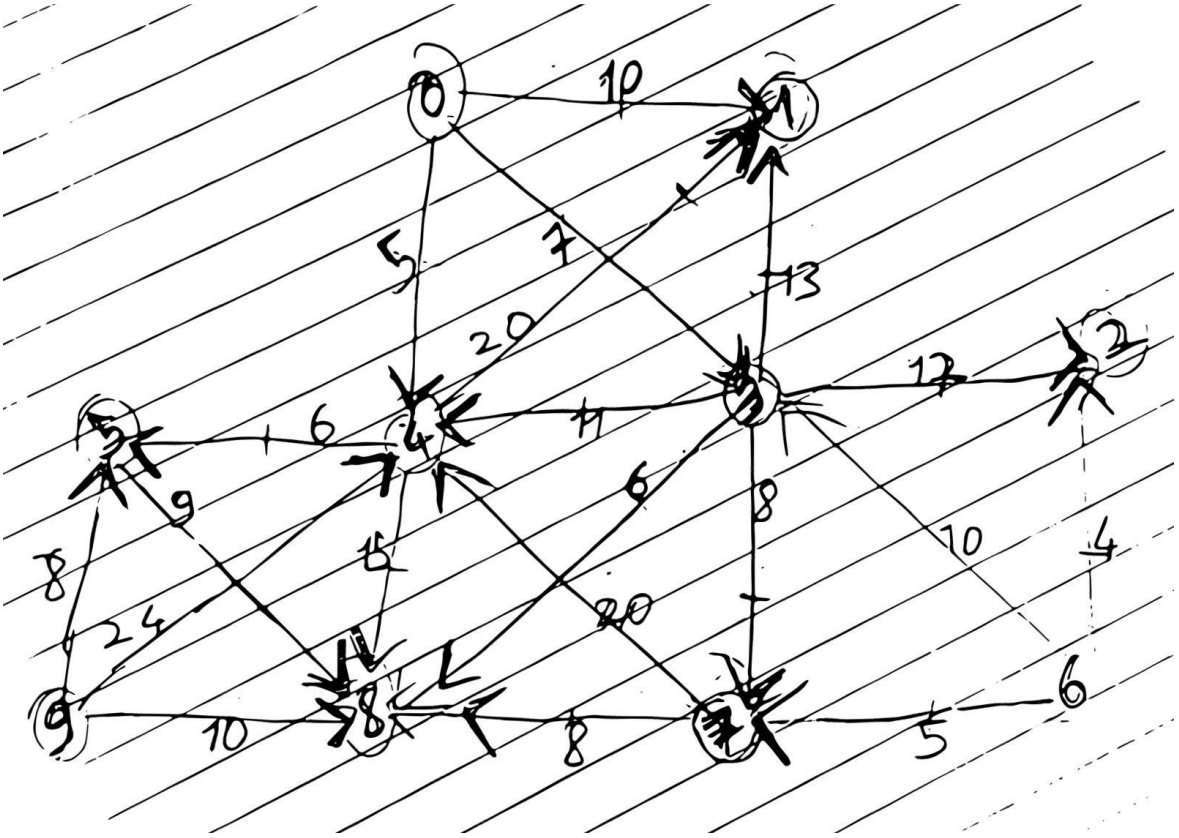
Course Assistant: Fatma Nur Esirci

1 Q1

*Methodlar ile ilgili açıklamalar rapor formatında yer almadığı için shortest path methodu dışındakiler ile ilgili açıklama eklemedim ancak gerekli açıklamalar javadoc'larında mevcut.
Part 1 için main de yapılan işlemlerin aynıları çıktıları ekrana basmak yerine assert macroları kullanarak beklenen outputlar ile kontrol ettiğim bir test kodu da yazdım ancak diğer partlar için yapmaya fırsatım olmadı , onların method çıktılarını ekrana bastım .Gözle kontrol edilebilirler.
Kitapın kaynak kodlarında bazı hatalar aldım o yüzden çözüm için üzerlerinde küçük değişiklikler yaptım

1.1 Problem Solution Approach

10 vertex ve 20 edge kullanarak directed ve acycle weighted bir graph oluşturdum .Burada weight değerleri gelişi güzel şekilde verildi . Graph ' plot eden method'un çıktısı her vertex'ten çıkan edge'leri göstermek üzerinde kurulu olduğundan Graph'ı aklımızda daha iyi canlandirmak için elimle çizdiğim halini ekliyorum .



(Part 1 için kullandığım Graph v : 10 e: 20)

->Shortest path'i arka planda dijkstrasAlgorithm 'i kullanarak buluyorum. Kabaca açıklayacak olursak , shortest_path(Graph graph, int V1 , int V2 , double[] distance) de V1 olarak gelen parh'in başlangıç vertex'ini , graph 'ı ve dijkstrasAlgorithm 'in parametre olarak aldığı boş int arr

predParam ve boş double distParam ‘ı dijkstrasAlgoriyhm’ e gönderiyorum . Method içeride v1 in graph üzerindeki bütün vertexlere olan bulup sırası ile distParam’ a ekler. Method çalıştıktan sonra distParam’ın v2. indexinde ki değer v1 ve v2 arasındaki en kısa mesafeye eşittir .Shortest_path’in parametre olarak aldığı distance değerine bu değer atanır. En kısa path’ i veren vertexlerin hangileri olduğunu bulmaya gelecek olursak ise onu dijkstrasAlgoriyhm’ in doldurduğu predParam array’i üzerinden buluyorum , predParam içerisinde v1 den giderken her bir vertex’e eşirmek için geçilecek bir önceki vertex’i içerisinde tutar. Ben de v2’ye erişmek için gelinmesi gereken vertex’i alıyorum ve o vertex’ v1 olana kadar her vertex’in bir önceki göstericisine geçiyorum bu sayede v1 ve v2 arasında ki shortestPathi bulmuş oluyorum.

1.2 Test Cases

Show that this func results >>>

- `plot_graph(listGraph);`

  Satırının çıktısı >>

```
'0' vertex'inden çıkan bağlantılar :
0 : ------10.0----->  : 1
0 : ---5.0-->  : 4
0 : ----7.0--->  : 3

'3' vertex'inden çıkan bağlantılar :
3 : ------11.0----->  : 4
3 : ----6.0-->  : 8
3 : -----8.0--->  : 7
3 : -------12.0----->  : 2
3 : -------13.0------>  : 1

'4' vertex'inden çıkan bağlantılar :
4 : --------15.0------->  : 8
4 : ----6.0-->  : 5
4 : -----------20.0--------->  : 1

'5' vertex'inden çıkan bağlantılar :
5 : -----9.0---->  : 8

'6' vertex'inden çıkan bağlantılar :
6 : ---4.0->  : 2
6 : ------10.0----->  : 3
6 : ---5.0-->  : 7

'7' vertex'inden çıkan bağlantılar :
7 : ----7.0--->  : 8
7 : -----------20.0--------->  : 4

'9' vertex'inden çıkan bağlantılar :
9 : -----8.0--->  : 5
9 : -------------24.0----------->  : 4
9 : ------10.0----->  : 8
```

< - (ekran resmi)

- *is_undirected*(listGraph:

```
System.out.println("\nreturn value of 'is_undirected(listGraph)' : " +
is_undirected(listGraph)+"\n");
```

Satırının çıktısı >>

```
return value of 'is_undirected(listGraph)' : false
```

(ekran resmi)

- *is_acylcle_graph*(listGraph):

```
  System.out.println("return value of 'is_acylcle_graph(listGraph)'"+
is_acylcle_graph(listGraph)+"\n");
```

Satırının çıktısı >>

```
return value of 'is_acylcle_graph(listGraph)'true
```

(ekran resmi)

- *shortest_path*(listGraph,X,Y,distance):

```
System.out.println("0 ile 2 vertxler arası path : "+
        shortest_path(listGraph,0,2,distance)+ " distance : "+distance[0]
);

System.out.println("6 ile 4 vertxler arası path : "+
        shortest_path(listGraph,6,4,distance)+ " distance : "+distance[0]
);

System.out.println("0 ile 9 vertxler arası path : "+
        shortest_path(listGraph,0,9,distance)+ " distance : "+distance[0]
);
```

0 ile 2 , 6 ile 4 ve 0 ile 9 vertexleri arası shortest pathleri bulmak için method çağrılınca. Yukarıda ki satırların çıktısı >>

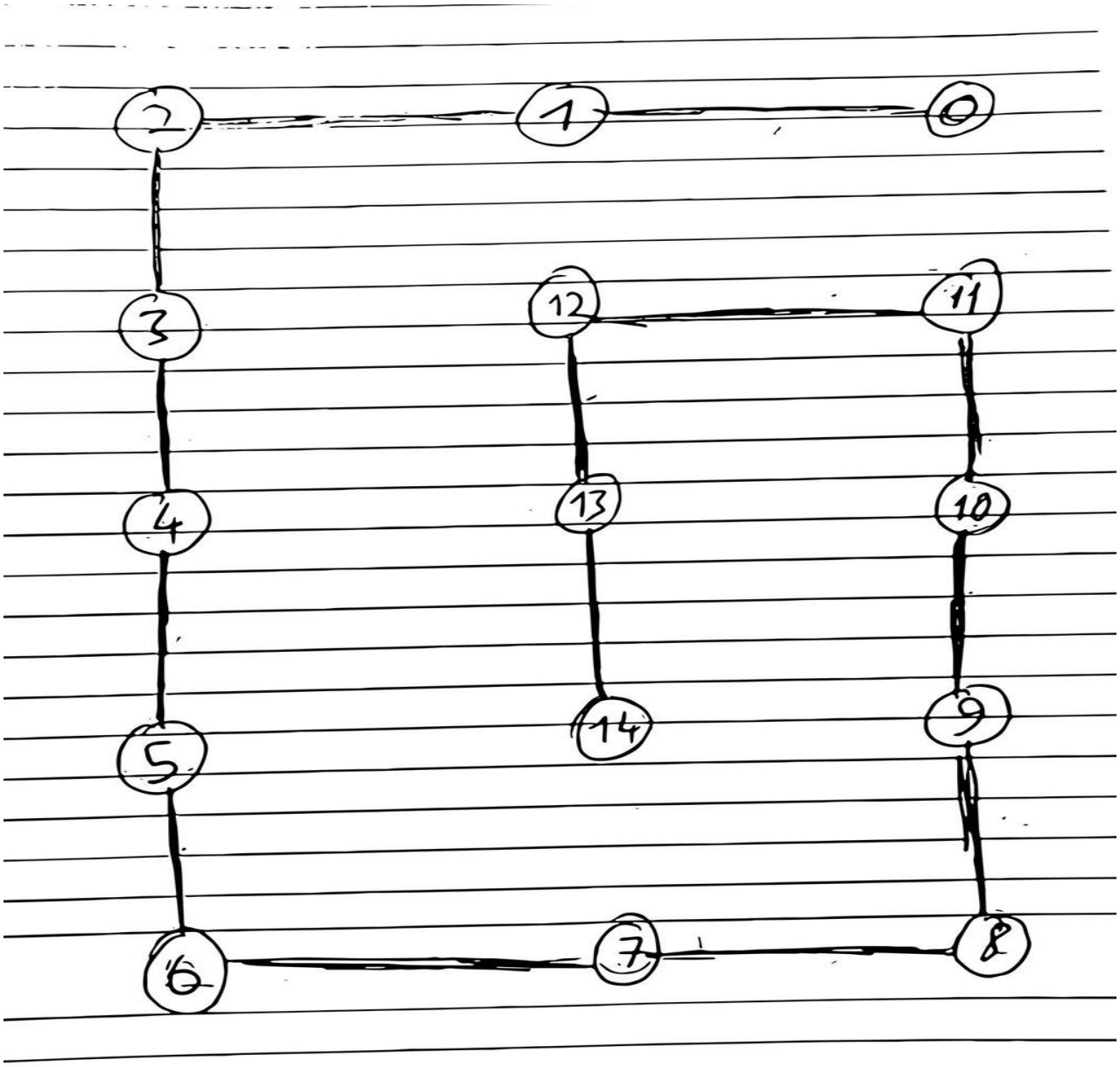
```
0 ile 2 vertxler arası path : [0, 3, 2] distance : 19.0
6 ile 4 vertxler arası path : [6, 3, 4] distance : 21.0
0 ile 9 vertxler arası path : null distance : Infinity
```

(ekran resmi)

2 Q2

2.1 Problem Solution Approach

Create undirected and acyclic graph have no weight (v=15),
15 vertex kullanarak undirected ve acycle edge'lerin de agılık olmayan bir graph oluşturmamız istenmiş . Graph 'ın acycle ve undirected olması için bulduğum en kısa ne basit yöntem vertex'leri sırayla birbirlerine birer edge ile bağlamak oldu bu sayede undirected ve acycle olan 15 vertexli bir graph oluşturmuş oldum . Zaten edge'ler de yön olmadığı için bu tarz bir şey oluşturmak zorundaydımda. ShortestPath için aynı methodu kullanabiliriz.



(Part 2 Graph v: 15)

2.2 Test Cases

Show that this func results >>>

- `plot_graph(part2);`

  Satırının çıktısı >>

```
'0' vertex'inden çıkan bağlantılar :
0 :  : 1

'1' vertex'inden çıkan bağlantılar :
1 :  : 0
1 :  : 2

'2' vertex'inden çıkan bağlantılar :
2 :  : 1
2 :  : 3

'3' vertex'inden çıkan bağlantılar :
3 :  : 2
3 :  : 4

'4' vertex'inden çıkan bağlantılar :
4 :  : 3
4 :  : 5

'5' vertex'inden çıkan bağlantılar :
5 :  : 4
5 :  : 6

'6' vertex'inden çıkan bağlantılar :
6 :  : 5
6 :  : 7

'7' vertex'inden çıkan bağlantılar :
7 :  : 6
7 :  : 8

'8' vertex'inden çıkan bağlantılar :
8 :  : 7
8 :  : 9

'9' vertex'inden çıkan bağlantılar :
9 :  : 8
9 :  : 10

'10' vertex'inden çıkan bağlantılar :
10 :  : 9
10 :  : 11

'11' vertex'inden çıkan bağlantılar :
11 :  : 10
11 :  : 12

'12' vertex'inden çıkan bağlantılar :
12 :  : 11
12 :  : 13

'13' vertex'inden çıkan bağlantılar :
13 :  : 12
13 :  : 14

'14' vertex'inden çıkan bağlantılar :
14 :  : 13
```

← (ekran resmi)

- *is_undirected*(Graph):

```
System.out.println("\nreturn value of 'is_undirected(part2)' : " + is_undirected(part2));
```

Satırının çıktısı >>

```
return value of 'is_undirected(part2)' : true
```

(ekran resmi)

- *is_acylcle_graph*(part2):

```
 System.out.println("return value of 'is_acylcle_graph(part2)' "+ is_acylcle_graph(part2)+"\n");
```

Satırının çıktısı >>

```
return value of 'is_acylcle_graph(part2)'true
```

(ekran resmi)

- *is_connected*(graph, v1, v2):

```
try {
    System.out.println("return value of 'is_connected(part2,0,5)' : " + is_connected(part2, 0, 5));
    System.out.println("return value of 'is_connected(part2,7,1)' : " + is_connected(part2, 7, 1));
    System.out.println("return value of 'is_connected(part2,3,12)' : " + is_connected(part2, 3, 12));
}
```

(is_connected() methodu hatalı v1 , v2 girişlerinde exception fırlattığı için try bloğu içinde çalıştırıyorum.)

Satırının çıktısı >>

```
return value of 'is_connected(part2,0,5)' : true
return value of 'is_connected(part2,7,1)' : true
return value of 'is_connected(part2,3,12)' : true
```

(ekran resmi)

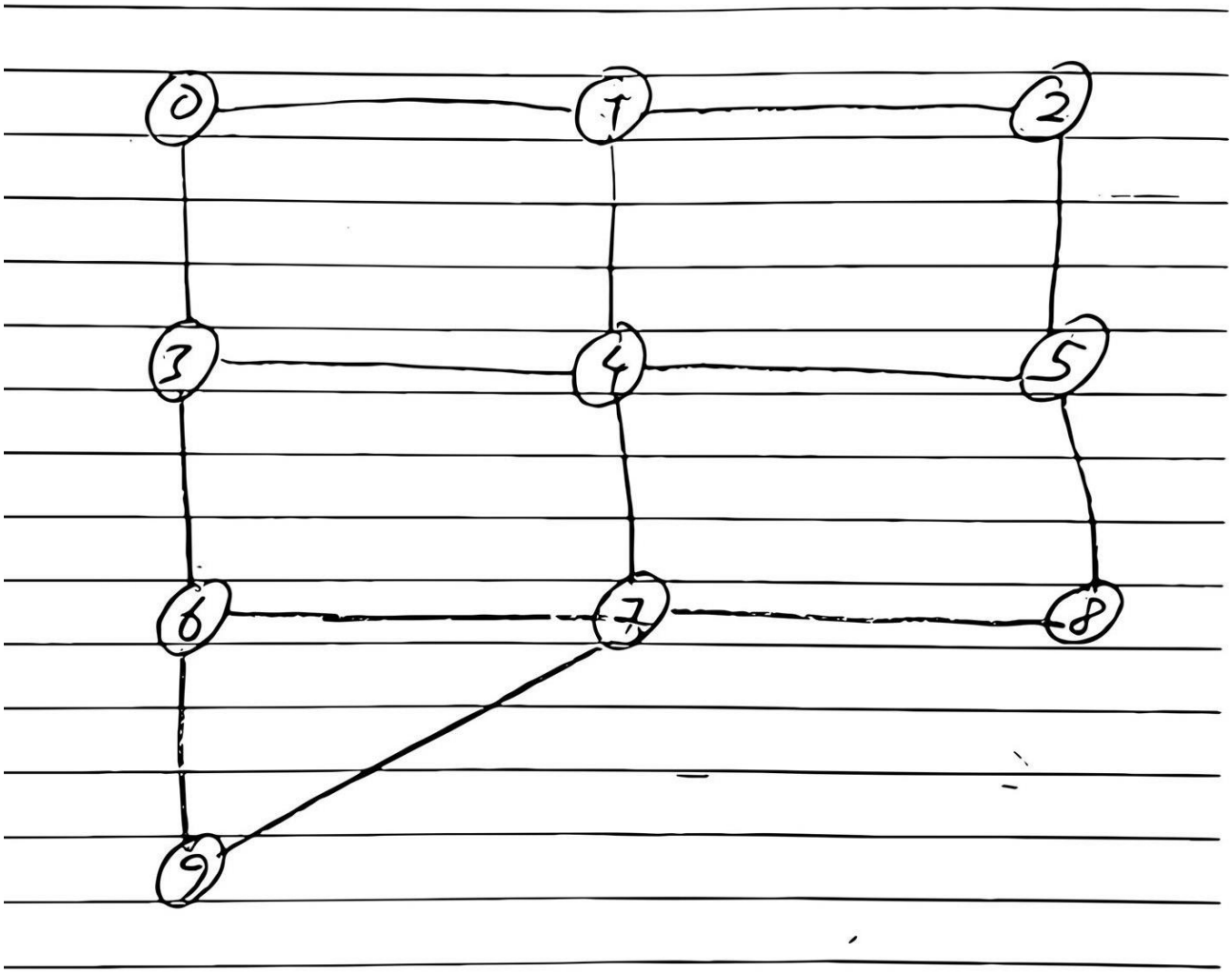
3 Q3

3.1 Problem Solution Approach

Create undirected and cyclic graph have no weight (v=10),

Bu partta undirected ve cyclic olucak şekilde edgelerinin ağırlığı olmayan 10 tane vertex'i olan bir

graph oluşturmamız istenmiş. Graph undirected ve cyclic olacağı için basit bir şelikde 14 adet vertex kullanarak aşşağıdaki gibi gözüken bir graph oluşturdum . Bu graph 10 vertex , cycle ve undirected olma özelliklerini sağlamaktadır. Bütün vertexler için cycle .Eğer bu graph üzerinde de önceki partlarda kullandığımız shortest path methodumuzu çalıştırsaydık doğru şekilde output alıcaktık , istenmediği için öyle bir işlem yapmıyorum.



(Part 3 graph v : 10)

3.2 Test Cases

Show that this func results >>>

- `plot_graph(part3):`

  Satırının çıktısı >>

```
'0' vertex'inden çıkan bağlantılar :
0 :  : 1
0 :  : 3

'1' vertex'inden çıkan bağlantılar :
1 :  : 0
1 :  : 2
1 :  : 4

'2' vertex'inden çıkan bağlantılar :
2 :  : 1
2 :  : 5

'3' vertex'inden çıkan bağlantılar :
3 :  : 0
3 :  : 6
3 :  : 4

'4' vertex'inden çıkan bağlantılar :
4 :  : 1
4 :  : 3
4 :  : 5
4 :  : 7

'5' vertex'inden çıkan bağlantılar :
5 :  : 2
5 :  : 4
5 :  : 8

'6' vertex'inden çıkan bağlantılar :
6 :  : 3
6 :  : 7
6 :  : 9

'7' vertex'inden çıkan bağlantılar :
7 :  : 4
7 :  : 6
7 :  : 8
7 :  : 9

'8' vertex'inden çıkan bağlantılar :
8 :  : 5
8 :  : 7

'9' vertex'inden çıkan bağlantılar :
9 :  : 6
9 :  : 7
```

(ekran resmi)

- *is_undirected*(part3):

```
System.out.println("\nreturn value of 'is_undirected(listGraph)' : " +
is_undirected(part3));
```

Satırının çıktısı >>

```
return value of 'is_undirected(part3)' : true
```

(ekran resmi)

- *is acylcle graph*(part3): ***

```
 System.out.println("return value of 'is_acylcle_graph(part3)'"+
is_acylcle_graph(part3)+"\n");
```

Satırının çıktısı >>

```
return value of 'is_acylcle_graph(part3)'true
```

(ekran resmi)

*** Görüldüğü gibi graph cycle olmasına rağmen method acycle dedi . Farklı koşullarda yapmış olduğum denemelerden şunu gördüm; method directed graph'larda doğru çalışmakta fakat undirected graphlar yanlış çalışmaktadır. Düzeltmeye ne yazık ki fırsatım olmadı belirtmek istedim.

- DepthFirstSearch (Show that spanning tree) // Boş
- BreathFirstSearch (Show that spanning tree) // Boş

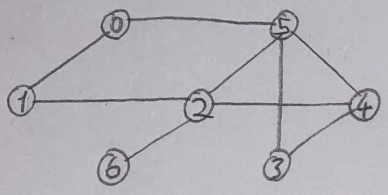
4 Q4

Raporun devamına eklendi.

Q4- Depth-First Search ve Breadth-First Search arasındaki farklılıkları ele alalım. DFS arka planda keşfetmediği nodeları tutmak için Stack yapısı kullanır, BFS ise arka planda Queue yapısı kullanarak çalışır. DFS daha hızlıdır ve daha az hafıza tüketimine sahiptir BFS'e göre. Kullanım alanlarına örnek ise BFS için Shortest path bulma, DFS için Topological Sorting'i verebiliriz.

Matrisimiz:

Vertex isimleri	0	1	2	3	4	5	6
0	1	1	0	0	0	1	0
1	1	1	1	0	0	0	0
2	0	1	1	0	1	1	1
3	0	0	0	1	1	1	0
4	0	0	1	1	1	1	0
5	1	0	1	1	1	1	0
6	0	0	1	0	0	0	1



Traverse edilecek graph.
(Vertex isimlerini 0'dan başlattım.)

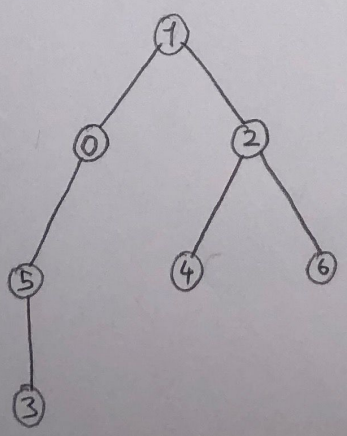
a-) DFS algoritmasını vertex 1'le başlatırsak:
Kitaptaki tabloya benzer bir çalışma tablosu yaptım.

Operation	Adjacent Vert.	Discovery Order	Finish Ord.
Visit 1	0,2	1	
Visit 0	1,5	1,0	
Visit 5	0,2,3,4	1,0,5	
Visit 2	1,4,5,6	1,0,5,2	
Visit 4	2,3,5	1,0,5,2,4	
Visit 3	4,5	1,0,5,2,4,3	
Finish 3			3
Finish 4			3,4
Visit 6	2	1,0,5,2,4,3,6	
Finish 6			3,4,6
Finish 2			3,4,6,2
Finish 5			3,4,6,2,5
Finish 0			3,4,6,2,5,0
Finish 1			3,4,6,2,5,0,1

b-) BFS algoritması, vertex 1'le başlatırsak:

Vertex Being Visited	Queue Contens after Visit	Visit Sequence
1	0,2	1
0	2,5	1,0
2	5,4,6	1,0,2
5	4,6,3	1,0,2,5
4	6,3	1,0,2,5,4
6	3	1,0,2,5,4,6
3	Boş	1,0,2,5,4,6,3

Breadth-First Search Tree



Depth-First Search Tree