

**Gebze Technical University
ComputerEngineering**

CSE 222 -2018 Spring

HOMEWORK 7 REPORT

**YUNUS ÇEVİK
141044080**

Course Assistant: Fatma Nur Esirci

1 Q1

1.1 Problem Solution Approach

Ağırlıkları rastgele verilen vertexlerin, directed olarak yaptıkları edgeler ile bir birlerine bağlanması sonucu directed acyclic graphımı oluşturdum. Bu graphın acyclic olduğunu bulmak için grap içerisinde gezinerek bir vertexden başlayıp diğer vertexler üzerinde gezindikten sonra tekrardan başladığı vertex değerine gelmiyorsa bu acyclic olayıdır. Bu partta bunu kanıtlamış oldum. Ayrıca Shortest Path metodunda Dijkstra Algoritmasını kullanarak yaptım.

1.2 Test Cases

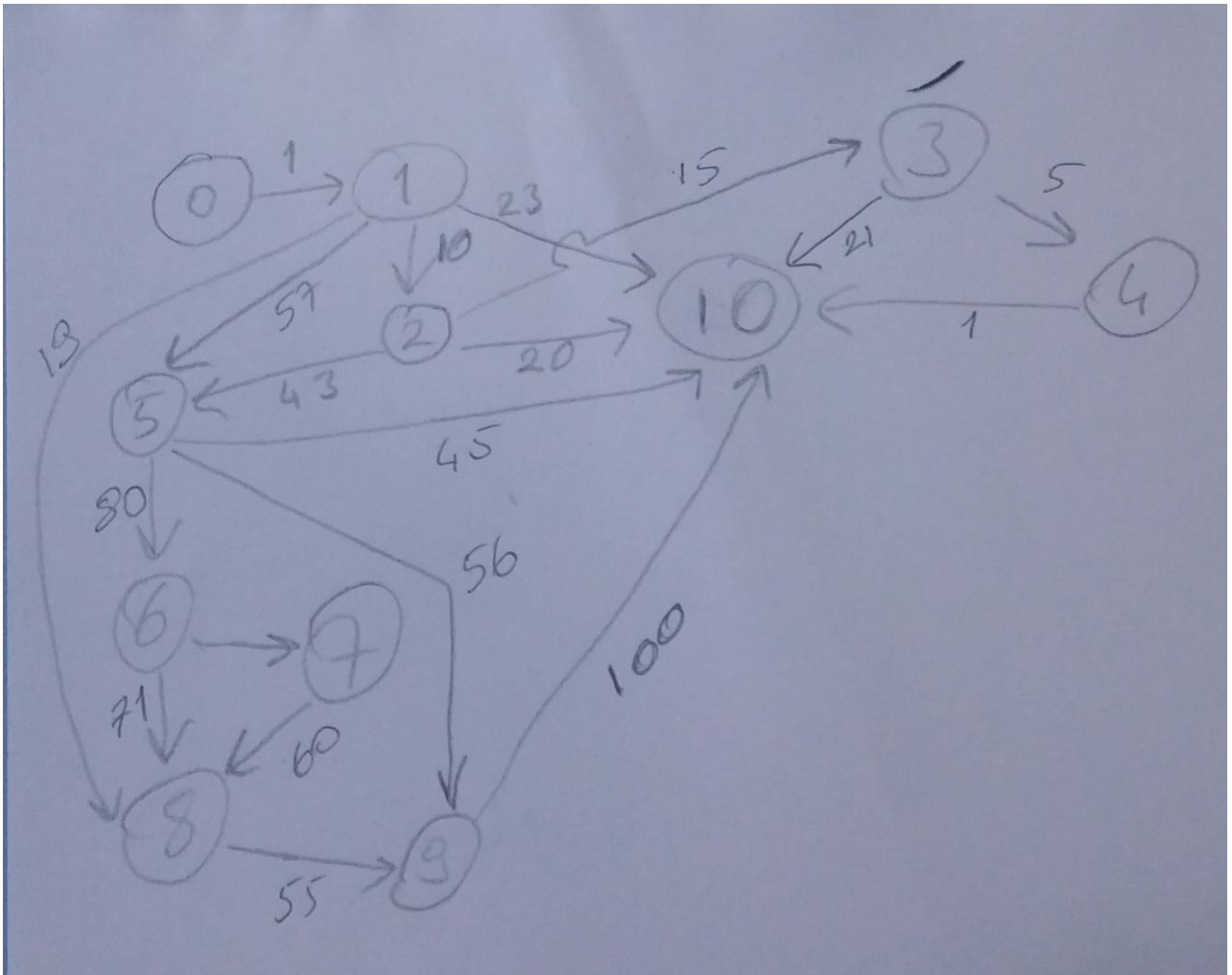
Vertex Input Değerleri

```
graph.insert(new Edge(0,1,1));
graph.insert(new Edge(1,2,10));
graph.insert(new Edge(2,3,15));
graph.insert(new Edge(2,10,20));
graph.insert(new Edge(2,5,43));
graph.insert(new Edge(6,8,71));
graph.insert(new Edge(1,8,19));
graph.insert(new Edge(3,4,5));
graph.insert(new Edge(4,5,30));
graph.insert(new Edge(5,10,45));
graph.insert(new Edge(5,6,80));
graph.insert(new Edge(5,9,56));
graph.insert(new Edge(6,7,3));
graph.insert(new Edge(8,9,55));
graph.insert(new Edge(1,10,23));
graph.insert(new Edge(3,10,21));
graph.insert(new Edge(7,8,60));
graph.insert(new Edge(4,10,1));
graph.insert(new Edge(1,5,57));
graph.insert(new Edge(9,10,100));
```

Output Sonucu

```
HW7Project [~/Desktop/intelijProject/HW7Project] - .../src/hw7/Q1/Q1_Main.java [HW7Project] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
HW7Project src hw7 Q1 Q1_Main
Run Q1_Main
/usr/lib/jvm/java-1.8.0-openjdk-1386/bin/java ...
ListGraph :
0 -> 1
1 -> 2 -> 8 -> 10 -> 5 -> 9
2 -> 3 -> 10 -> 5
3 -> 4 -> 10
4 -> 5 -> 10
5 -> 10 -> 6 -> 9
6 -> 8 -> 7
7 -> 8
8 -> 9
9 -> 10
Plot Graph
The graph is directed.
The graph is acyclic.
is_undirected ve is_acyclic
Shortest Path
The shortest path between 0 and 7: 0 -> 1 -> 2 -> 5 -> 6 -> 7
The shortest path between 2 and 10: 2 -> 10
The shortest path between 1 and 3: 1 -> 2 -> 3
There is no shortest path between 1 and 15
Process finished with exit code 0
0: Messages 4: Run 6: TODO Terminal
Compilation completed successfully in 4s 801ms (a minute ago)
22:1 CRLF: UTF-8:
[HW7Project] HW7Project [~/D...
```

Kağıt Üzerinde Graph Gösterimi



2 Q2

This part about Question 2 in HW7

2.1 Problem Solution Approach

Ağırlıkları önemsiz vertexlerin, undirected olarak yaptıkları edgeler ile bir birlerine bağlanması sonucu undirected cyclic graphımı oluşturdum. Bu graphın acyclic olduğunu bulmak için grap içerisinde gezinerek bir vertexden başlayıp diğer vertexler üzerinde gezindikten sonra tekrardan başladığı vertex değerine gelmiyorsa bu acyclic olayıdır. Bu partta bunu kanıtlamış oldum. Ayrıca oluşturulan graphın connected ya da unconnected olup olmadığını gösterdim.

2.2 Test Cases

Vertex Input Değerleri

```
graph.insert(new Edge(0,1));
graph.insert(new Edge(1,2));
graph.insert(new Edge(2,3));
graph.insert(new Edge(3,4));
graph.insert(new Edge(5,6));
graph.insert(new Edge(6,8));
graph.insert(new Edge(7,8));
graph.insert(new Edge(5,9));
graph.insert(new Edge(15,9));
graph.insert(new Edge(6,7));
graph.insert(new Edge(13,12));
graph.insert(new Edge(12,11));
graph.insert(new Edge(11,10));
graph.insert(new Edge(14,13));
graph.insert(new Edge(15,13));
```

Output Sonucu

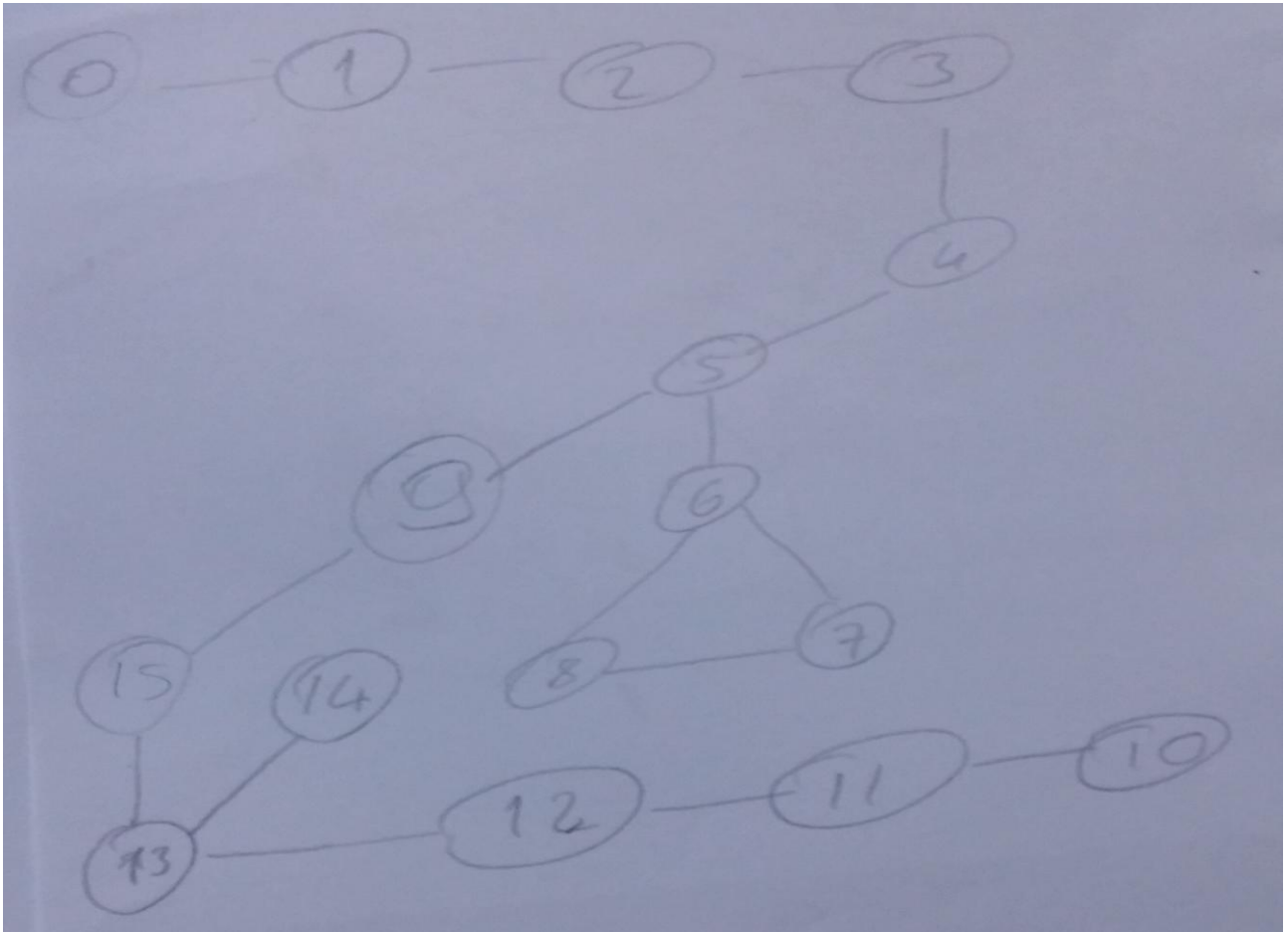
```
HW7Project [~/Desktop/IntelliJProject/HW7Project] - .../src/hw7/Q2/Q2_Main.java [HW7Project] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

Run Q2_Main
/usr/lib/jvm/java-1.8.0-openjdk-1386/bin/java ...
ListGraph :
0 -> 1
1 -> 0 -> 2
2 -> 1 -> 3
3 -> 2 -> 4
4 -> 3
5 -> 6 -> 9
6 -> 5 -> 8 -> 7
7 -> 8 -> 6
8 -> 6 -> 7
9 -> 5 -> 15
10 -> 11
11 -> 12 -> 10
12 -> 13 -> 11
13 -> 12 -> 14 -> 15
14 -> 13
15 -> 9 -> 13

The graph is undirected.
The graph is acyclic.
1 and 8 is not connected.
5 and 14 is connected.
1 and 18 is not connected.

Process finished with exit code 0
```

Kağıt Üzerinde Graph Gösterimi



3 Q3

3.1 Problem Solution Approach

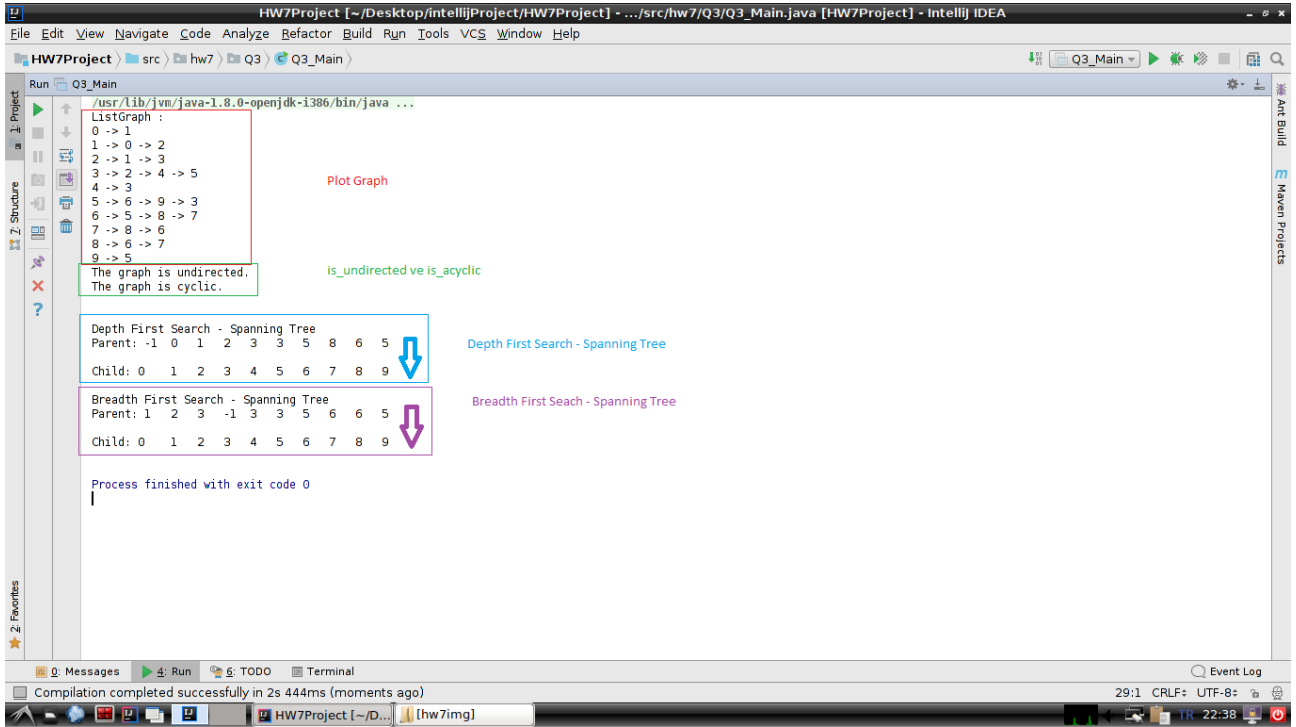
Ağırlıkları önemsiz vertexlerin, undirected olarak yaptıkları edgeler ile bir birlerine bağlanması sonucu undirected cyclic graphımı oluşturdum. Bu graphın acyclic olduğunu bulmak için grap içerisinde gezinerek bir vertexden başlayıp diğer vertexler üzerinde gezindikten sonra tekrardan başladığı vertex değerine gelmiyorsa bu acyclic olayıdır. Bu partta bunu kanıtlamış oldum. Ayrıca oluşturulan graphın da Depth First Search ve Breadth First Search algoritmaları ile Spanning Tree' leri oluşturdum.

3.2 Test Cases

Vertex Input Değerleri

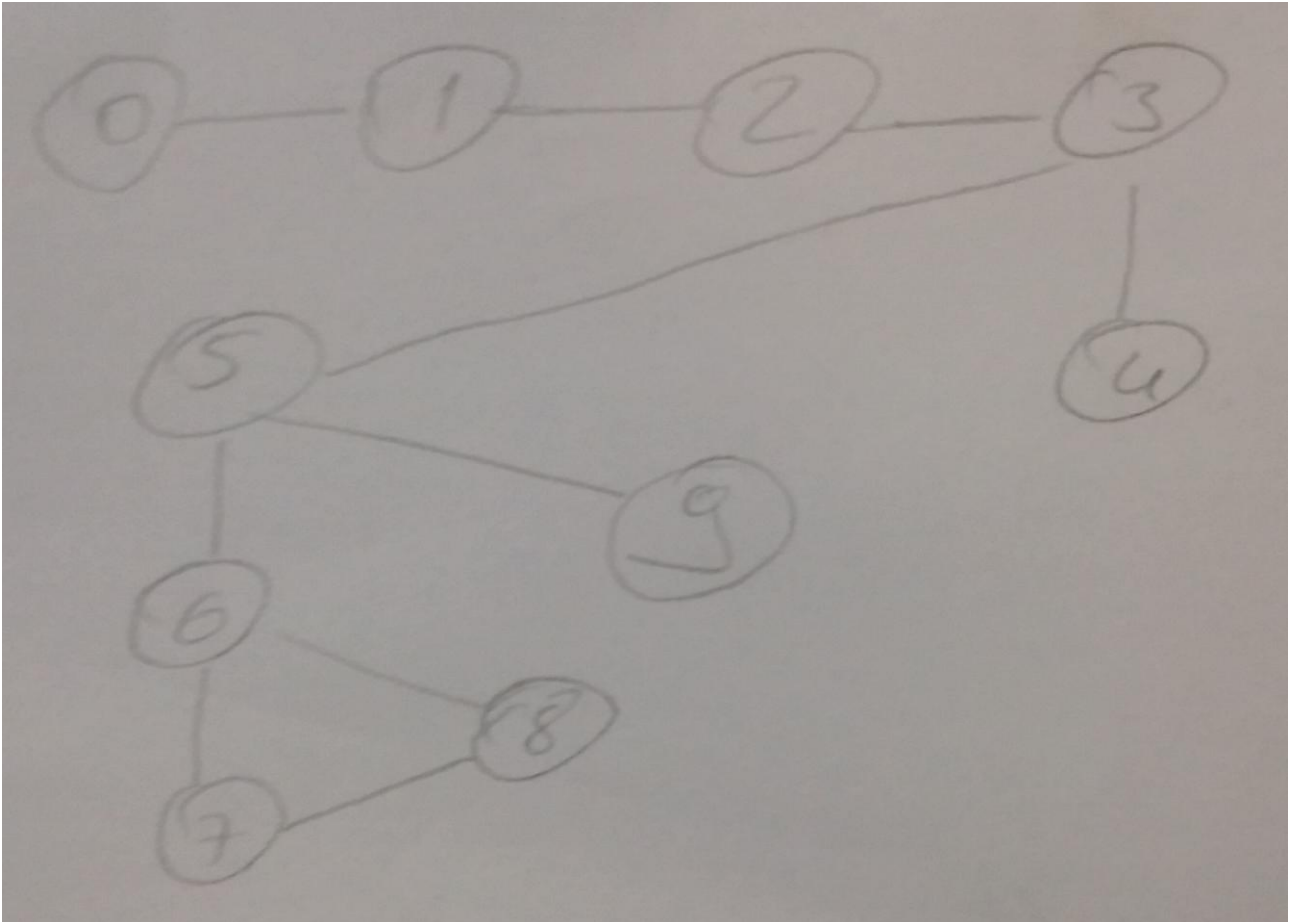
```
graph.insert(new Edge(0,1));  
graph.insert(new Edge(1,2));  
graph.insert(new Edge(2,3));  
graph.insert(new Edge(3,4));  
graph.insert(new Edge(5,6));  
graph.insert(new Edge(6,8));  
graph.insert(new Edge(7,8));  
graph.insert(new Edge(5,9));  
graph.insert(new Edge(6,7));  
graph.insert(new Edge(3,5));
```

Output Sonucu

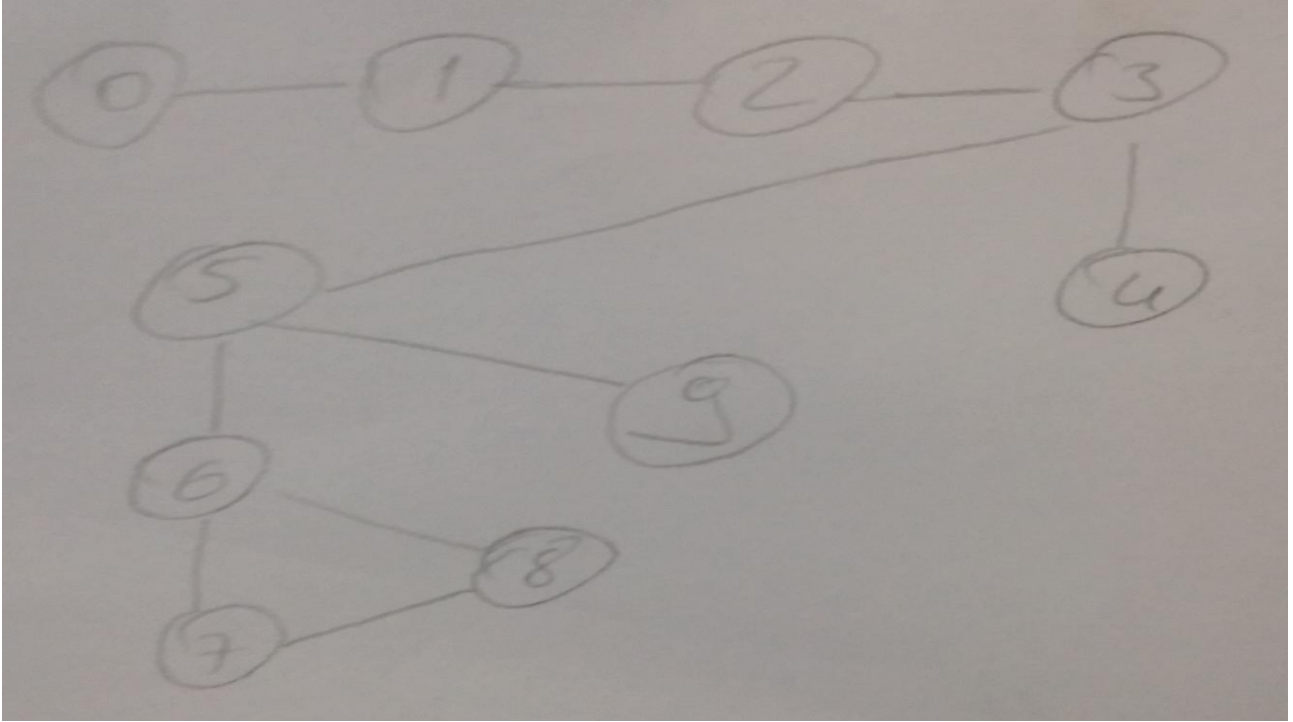


```
Run Q3_Main
/usr/lib/jvm/java-1.8.0-openjdk-1386/bin/java ...
ListGraph :
0 -> 1
1 -> 0 -> 2
2 -> 1 -> 3
3 -> 2 -> 4 -> 5
4 -> 3
5 -> 6 -> 9 -> 3
6 -> 5 -> 8 -> 7
7 -> 8 -> 6
8 -> 6 -> 7
9 -> 5
Plot Graph
The graph is undirected.
The graph is cyclic.
is_undirected ve is_is_cyclic
Depth First Search - Spanning Tree
Parent: -1 0 1 2 3 3 5 8 6 5
Child: 0 1 2 3 4 5 6 7 8 9
Breadth First Search - Spanning Tree
Parent: 1 2 3 -1 3 3 5 6 6 5
Child: 0 1 2 3 4 5 6 7 8 9
Process finished with exit code 0
```

Kağıt Üzerinde Graph Gösterimi

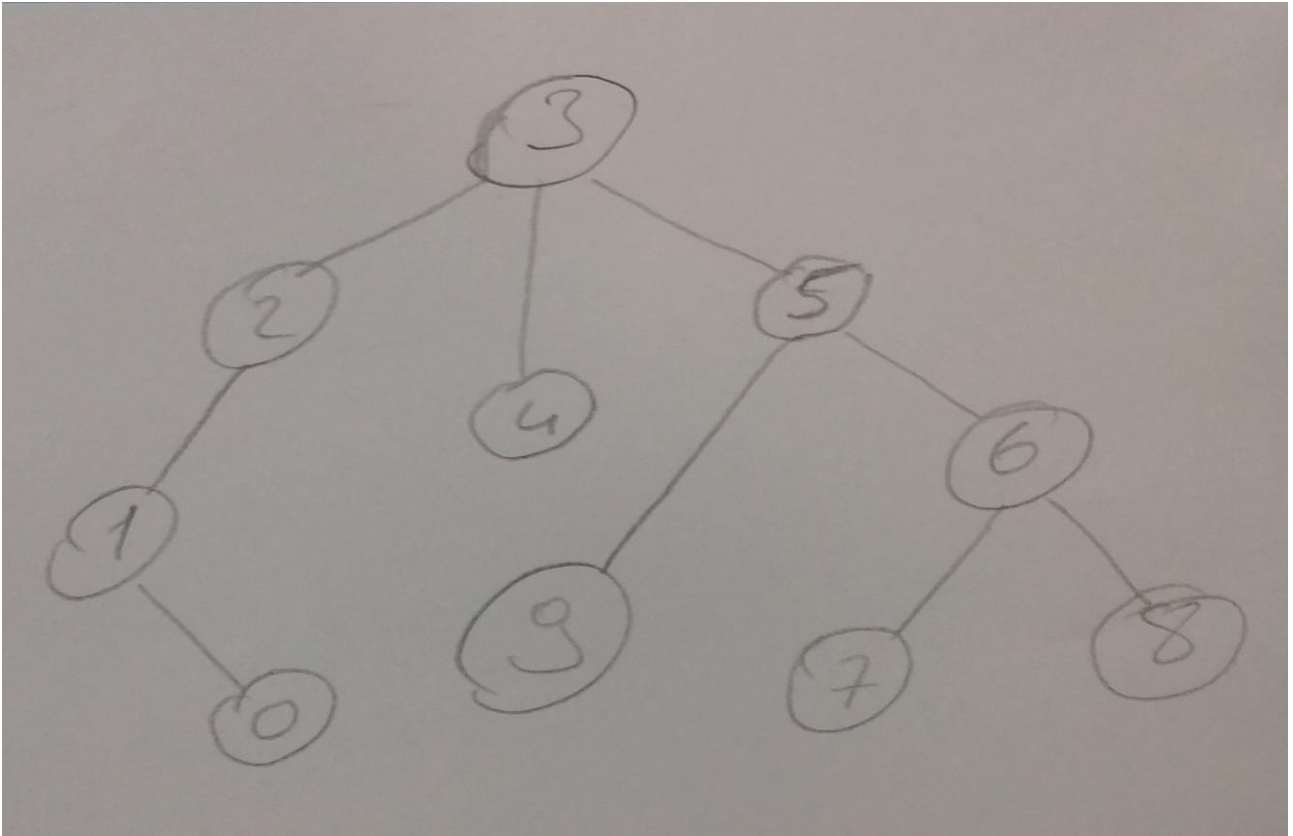


Depth First Search – Spanning Tree Kağıt Üzerinde Graph Gösterimi



Breadth First Search – Spanning Tree Kağıt Üzerinde Graph Gösterimi

Not: Başlangıç Vertex' i 3 tür.



4 Q4

Depth First Search (Derin Öncelikli Arama)

DFS (depth first search) algoritmasının çalışma mantığı şu şekildedir. Verilen graph daki tüm node lar gezilecek şekilde, sırayla ziyaret edilen bir nodun komşularına gidilir, gezilmemiş bir komşusu varsa ona gidilir ve daha sonra sıradaki kendi komşusundan önce komşusunun komşusuna gidilir. Böylece sürekli derine inilir. Daha sonra gezilmemiş node kalmayınca bir geri noda gelinir ve sıradaki komşu ve komşunun komşusu.... şeklinde devam eder. Bu algoritmada tüm graph gezilir(graphlarda bağlantısız node lar olsa da)

Algoritma

1. İlk olarak birinc node stack a konur.
2. Stack boşalincaya kadar döngü devam eder.
3. Stack dan node alınır.
4. Eğer alınan node gezilmemiş ise, komşularına bakılır, komşularından gezilmemiş olan varsa alınır ve stack a atılır.
5. Eğer gezilmemiş bir komşu kalmadıysa stackdan yeni node alınır.

Breadth First Search (Sığ Öncelikli Arama)

BFS (breadth first search) algoritmasının çalışma mantığı şu şekildedir. Verilen bir node dan başlayarak ilgili node un tüm komşuları gezilir. Daha sonra gezilen komşuların komşuları gezilerek verilen graph gezilmiş olur.

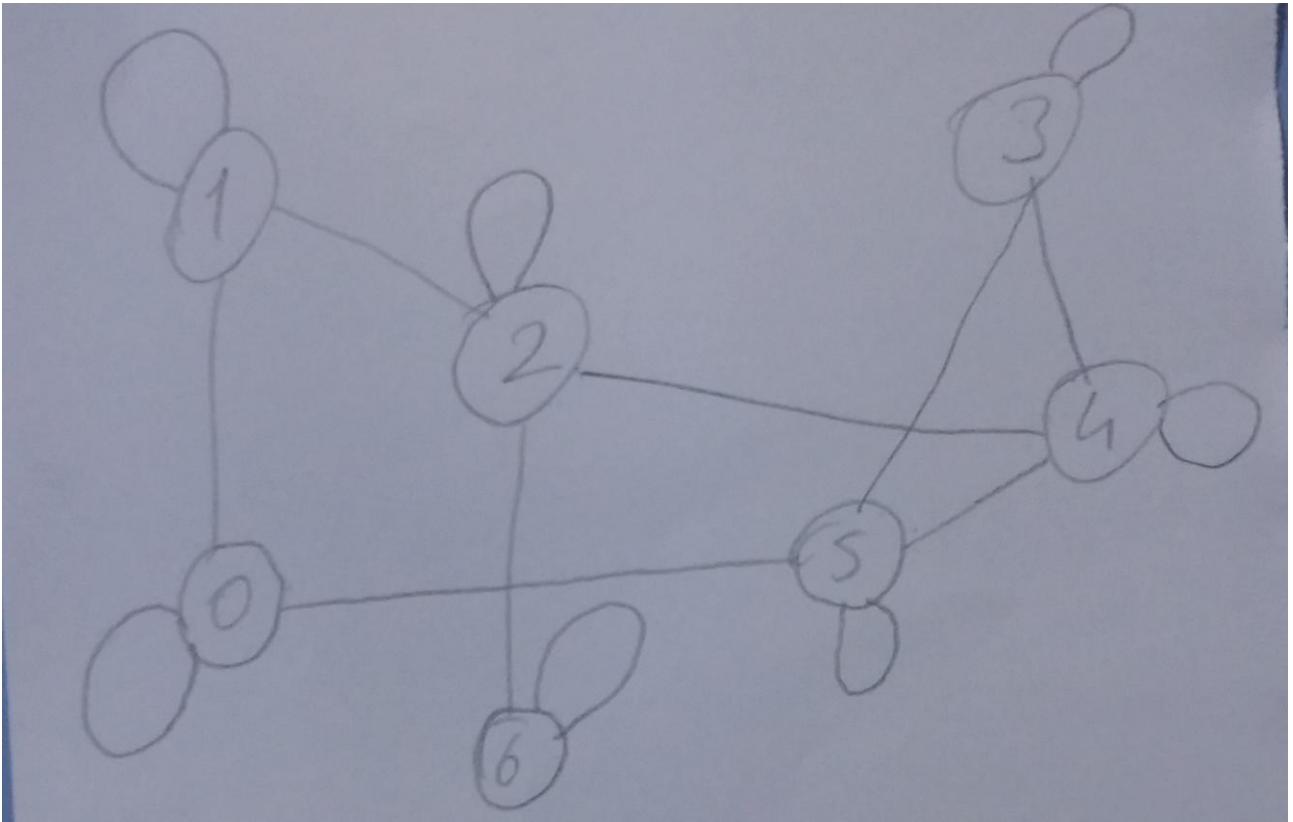
Algoritma

1. Fonksiyona gönderilen node kuyruğa atılır.
2. Kuyruk boşalincaya kadar döngü sürdürülür.
3. Sıradaki node kuyruktan çıkarılır.
4. Çıkarılan node daha önce gezilmemiş ise, gezildi işareti konur ve gezilmemiş komşuları kuyruğa konur.

Depth First Search ve Breadth First Search Arasındaki Farklar

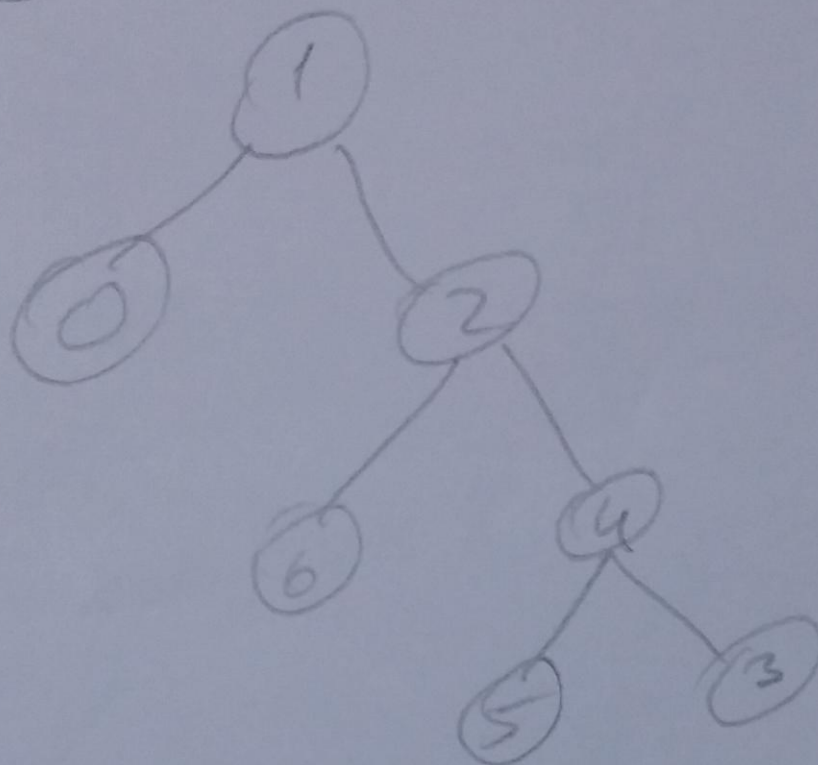
	Depth First Search	Breadth First Search
Algoritma Açıklaması	Bir arama yöntemidir. Root'tan başlar çıkmaz bir son bulana kadar mümkün olduğunca derinden arama yapar.	Bir arama yöntemidir. Root'tan başlar ve level level arama işlemi yapar.
Kullanılan Veri Yapısı	Stack kullanır.	Queue kullanır.
Search tipi	Pre Order Traversal	Level Order Traversal
Hız	DFS, BFS'den daha hızlıdır.	BFS, DFS'den daha yavaştır.
Avantajları	DFS en kısa yolu bulamayabilir.	BFS ağırlıksız graphlar için en kısa yoldan bulur.
Algoritma Tipi	Geriye Dönük bir algoritmadır.	Aç gözlü bir algoritmadır.

	1	2	3	4	5	6	7
1	1	1	0	0	0	1	0
2	1	1	1	0	0	0	0
3	0	1	1	0	1	1	1
4	0	0	0	1	1	1	0
5	0	0	1	1	1	1	0
6	1	0	1	1	1	1	0
7	0	0	1	0	0	0	1



DFS ve BFS Tree

DFS



BFS

