Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 6 REPORT

CELAL CAN KAYA
161044014

Course Assistant: Fatma Nur Esirci

1 Q1

1.1 Problem Solution Approach

Graphlar, vertex, edge ve weight olmak üzere 3 parçadan oluşur.Edgeler 2 vertexi birbirine bağlar.Weight ise bu edgelerin uzunluğuna verilen isimdir.Graph oluşturmak için List Graph yapısını kullandım.Insert metoduna Source Vertex ve Destination Vertex olmak üzere 2 parametre (Eğer weighted graph yapmak istiyorsak 3. Parametre olarakta weight gönderiyoruz) gönderiyoruz.Insert metodu bu 2 vertexi bir edge ile birbirine bağlar.

Graph üzerinde bir vertexten başka bir vertexe en kısa yolu bulmak için Djikstra Algoritması veya Prim's Algoritmasını kullanabiliriz.En kısa yolu bulurken kitapta verilen Djikstra Algoritmasını kullanarak buldum.Djikstra Algoritması için 2 Set, 2 Array'e ihtiyacımız var.Setlerden birinde en kısa yolu hesapladığımız vertexleri, diğerinde hesaplamadığımız vertexleri tutuyoruz.Arraylerden birinde predecessor'u diğerinde ise en kısa yolu tutuyoruz.En kısa yolları tutan arrayimizin tüm indexlerini ∞ ile ilklendiriyoruz.Daha sonra başlangıç verteximize bağlı olan tüm vertexlere olan uzaklığımız sonsuzdan daha az olacağı için yeni değerlerle güncelliyoruz ve aynı zamanda predecessorleride güncelliyoruz.Bu şekilde edgeler üzerinde ilerleyerek daha kısa bir yol bulduğumuzda o değeri ve predecessoru güncelleyerek ilerliyoruz.İşlem bittiğinde başlangıç vertexinden diğer tüm vertexlere olan en kısa uzaklığı bulmuş olacağız.

Kitaptaki kodlardan Djikstra Algorithm dışında bir değişiklik yapmadım. Djikstra Algoritmasında parametrelere birbirine ulaşılması mümkün olmayan 2 nokta koyduğumuz zaman Arraye index olarak -1 gönderiyordu ve ArrayIndexOutOfBounds Exceptionuyla karşılaşıyordum.Bunu engellemek için bir if statement'ı koyup index==-1 olduğu zaman return; yapıp fonksiyonu sonlandırıyorum.

Plot_Graph fonksiyonu için iç içe 2 tane for döngüsüyle vertexleri geziyorum ve isEdge fonksiyonuyla aralarında edge olup olmadığını kontrol ediyorum.Eğer edge varsa uygun formatta ekrana bastırıyorum.List Graph yapısını kullandığım için ekrana Linked List şeklinde bastırıyorum.

Is_Undirected fonksiyonu için iç içe 2 tane for döngüsüyle vertexleri geziyorum ve isEdge fonksiyonuyla aralarında edge olup olmadığını kontrol ediyorum.Eğer edge varsa parametrelerin yerini değiştirip zıt yönde bir edge'nin olup olmadığını kontrol ediyorum.Zıt yönde bir edge bulamadığı anda graph'ın directed olduğu anlaşılacağı için false return edip fonksiyonu sonlandırıyorum.

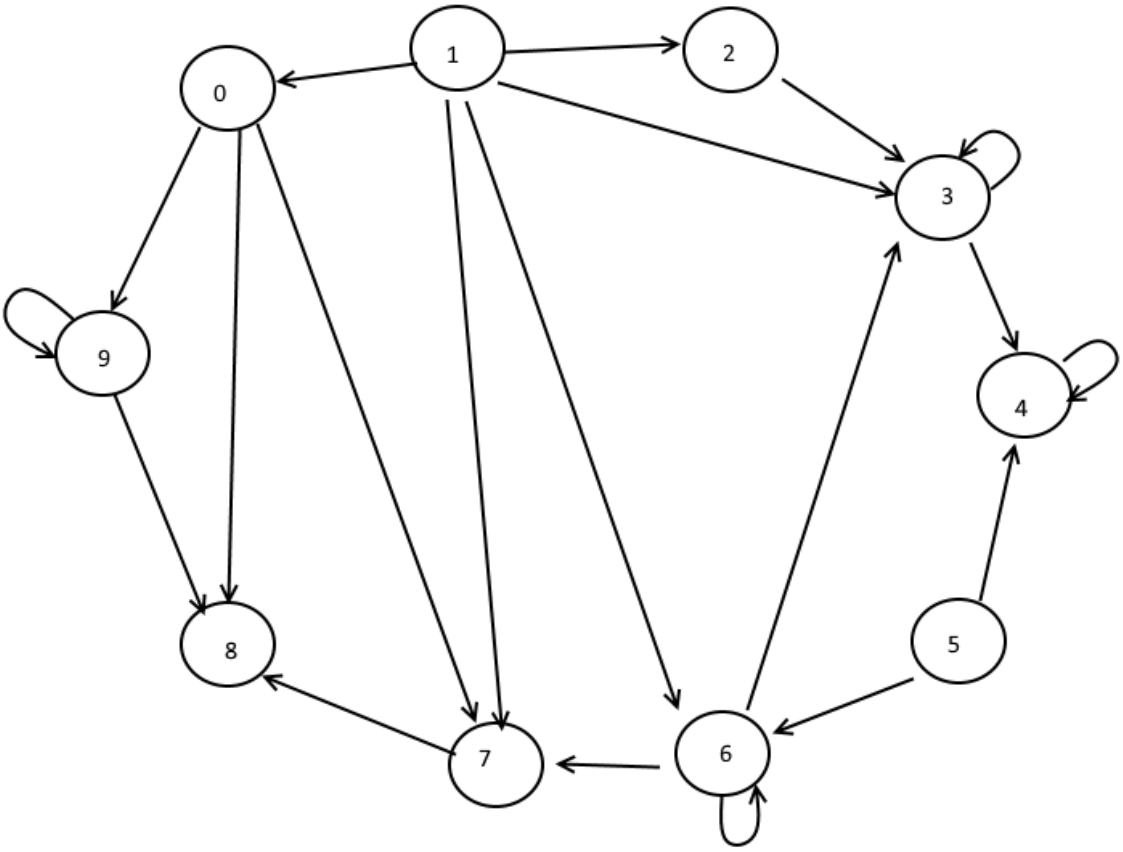
Is_Acyclic_Graph fonksiyonunda

Is_Connected fonksiyonunda verilen iki vertex arasında bir path olup olmadığını kontrol ediyorum.Kontrol etmek için BFS algoritmasını çağırıp return değeri olan parent arrayini bir değişkene atıyorum.Parent arrayi üzerinde v2==-1 olduğunda sonlanacak bir while döngüsüyle path üzerinde ilerliyorum.Eğer ilerleme esnasında v1 vertexime ulaşırsam ilerlediğim elemanları attığım Stackteki elemanları pop ederek ekrana bastırıyorum.Eğer v1 vertexime ulaşamazsam v1 ve v2 vertexi arasında bir path olmadığı için false return ediyorum.

Shortest_Path fonksiyonunda Djikstra Algoritmasını kullanarak Predecessor arrayi ve en kısa yolların olduğu arrayi elde ediyorum.Predecessor arrayini kullanarak en kısa yolda ki vertexleri Vector'e atıyorum.Path tamamlandığında ise ters bir şekilde olduğundan Vector'u tersine çevirip ekrana bastırıyorum.

1.2 Test Cases

TEST 1



- Plot_graph

```
PLOT_GRAPH

0-(1.0)->7-(2.0)->8-(4.0)->9
1-(6.0)->0-(8.0)->2-(3.0)->3-(2.0)->6-(6.0)->7
2-(1.0)->3
3-(2.0)->3-(8.0)->4
4-(3.0)->4
5-(4.0)->4-(5.0)->6
6-(5.0)->3-(1.0)->6-(6.0)->7
7-(10.0)->8
8
9-(2.0)->8-(3.0)->9
```

Graph'ın linked list şeklinde gösterilmiş hali.

- Is_UnDirected

```
IS_UNDIRECTED

Directed - Çünkü (0,7) Edgesi Mevcutken (7,0) Edgesi Mevcut Değil.
```

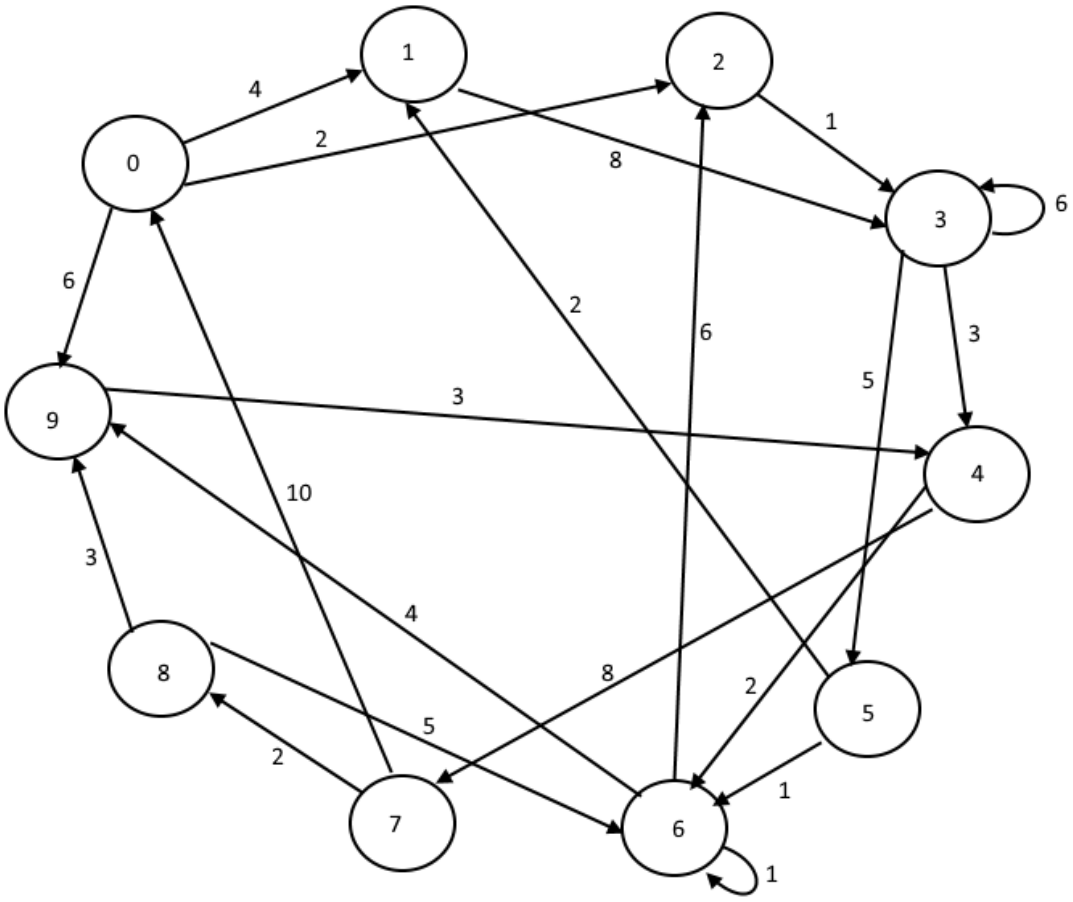
- Is_Acyclic

```
IS_ACYCLIC

Bu Bir Acyclic Graphtır.
```

- Shortest_Path

```
SHORTEST_PATH


1'Den 8'Ye Giden En Kısa Yol : [1, 0, 8]      Path Weight : 8.0
5'Den 3'Ye Giden En Kısa Yol : [5, 6, 3]      Path Weight : 10.0
1'Den 5'Ye Giden En Kısa Yol : []      Path Weight : Infinity
```

TEST 2



Yukarıda görselleştirdiğim ağacı oluşturdum. Ödev PDF'inde Acyclic oluşturmamız istendiğini bu testi yaptıktan sonradan gördüm.O nedenle bu testi silmek yerine **ekstra** olarak ekledim.

- Plot_graph

```
PLOT_GRAPH

0-(4.0)->1-(2.0)->2-(6.0)->9
1-(8.0)->3
2-(1.0)->3
3-(6.0)->3-(3.0)->4-(5.0)->5
4-(2.0)->6-(8.0)->7
5-(2.0)->1-(1.0)->6
6-(6.0)->2-(1.0)->6-(4.0)->9
7-(10.0)->0-(2.0)->8
8-(5.0)->6-(3.0)->9
9-(3.0)->4
```

Graph'ın linked list şeklinde gösterilmiş hali.

- Is_UnDirected

```
IS_UNDIRECTED

Directed - Çünkü (0,1) Edgesi Mevcutken (1,0) Edgesi Mevcut Değil.
```

- Is_Acyclic

```
IS_ACYCLIC

Bu Bir Cyclic Graphtır.
```

- Shortest_Path

```
SHORTEST_PATH


3'Den 6'Ye Giden En Kısa Yol : [3, 4, 6]      Path Weight : 5.0
5'Den 8'Ye Giden En Kısa Yol : [5, 6, 9, 4, 7, 8]      Path Weight : 18.0
0'Den 4'Ye Giden En Kısa Yol : [0, 2, 3, 4]      Path Weight : 6.0
```

2 Q2

2.1 Problem Solution Approach

Part 1'de açıklanması gereken her şeyi açıkladığım için aynısını bu kısma kopyalıyorum. Part 1'i okuduysanız bu kısmı atlayabilirsiniz.

Graphlar, vertex, edge ve weight olmak üzere 3 parçadan oluşur.Edgeler 2 vertexi birbirine bağlar.Weight ise bu edgelerin uzunluğuna verilen isimdir.Graph oluşturmak için List Graph yapısını kullandım.Insert metoduna Source Vertex ve Destination Vertex olmak üzere 2 parametre (Eğer weighted graph yapmak istiyorsak 3. Parametre olarakta weight gönderiyoruz) gönderiyoruz.Insert metodu bu 2 vertexi bir edge ile birbirine bağlar.

Graph üzerinde bir vertexten başka bir vertexe en kısa yolu bulmak için Djikstra Algoritması veya Prim's Algoritmasını kullanabiliriz.En kısa yolu bulurken kitapta verilen Djikstra Algoritmasını kullanarak buldum.Djikstra Algoritması için 2 Set, 2 Array'e ihtiyacımız var.Setlerden birinde en kısa yolu hesapladığımız vertexleri, diğerinde hesaplamadığımız vertexleri tutuyoruz.Arraylerden birinde predecessor'u diğerinde ise en kısa yolu tutuyoruz.En kısa yolları tutan arrayimizin tüm indexlerini ∞ ile ilklendiriyoruz.Daha sonra başlangıç verteximize bağlı olan tüm vertexlere olan uzaklığımız sonsuzdan daha az olacağı için yeni değerlerle güncelliyoruz ve aynı zamanda predecessorleride güncelliyoruz.Bu şekilde edgeler üzerinde ilerleyerek daha kısa bir yol bulduğumuzda o değeri ve predecessoru güncelleyerek ilerliyoruz.İşlem bittiğinde başlangıç vertexinden diğer tüm vertexlere olan en kısa uzaklığı bulmuş olacağız.

Kitaptaki kodlardan Djikstra Algorithm dışında bir değişiklik yapmadım. Djikstra Algoritmasında parametrelere birbirine ulaşılması mümkün olmayan 2 nokta koyduğumuz zaman Arraye index olarak -1 gönderiyordu ve ArrayIndexOutOfBounds Exceptionuyla karşılaşıyordum.Bunu engellemek için bir if statement'ı koyup index==-1 olduğu zaman return; yapıp fonksiyonu sonlandırıyorum.

Plot_Graph fonksiyonu için iç içe 2 tane for döngüsüyle vertexleri geziyorum ve isEdge fonksiyonuyla aralarında edge olup olmadığını kontrol ediyorum.Eğer edge varsa uygun formatta ekrana bastırıyorum.List Graph yapısını kullandığım için ekrana Linked List şeklinde bastırıyorum.

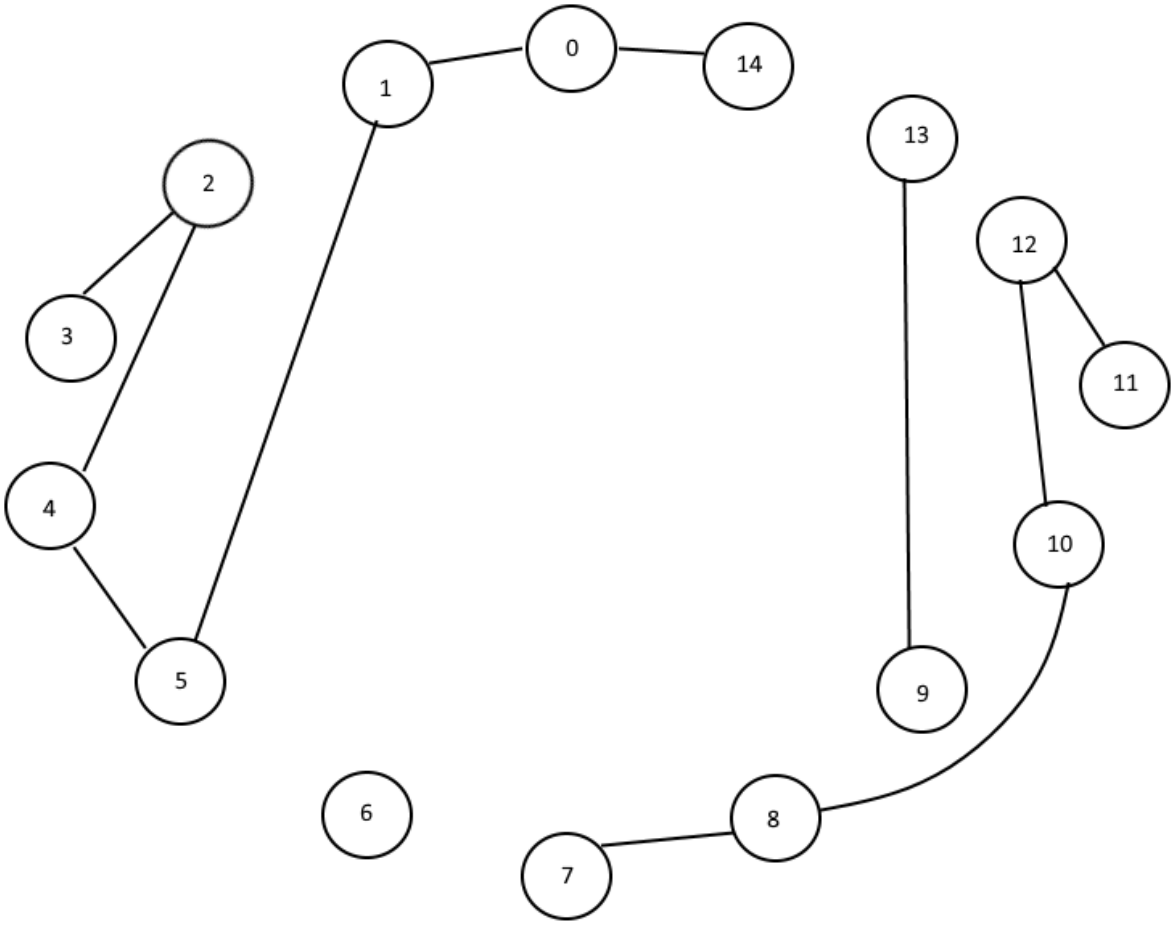
Is_Undirected fonksiyonu için iç içe 2 tane for döngüsüyle vertexleri geziyorum ve isEdge fonksiyonuyla aralarında edge olup olmadığını kontrol ediyorum.Eğer edge varsa parametrelerin yerini değiştirip zıt yönde bir edge'nin olup olmadığını kontrol ediyorum.Zıt yönde bir edge bulamadığı anda graph'ın directed olduğu anlaşılacağı için false return edip fonksiyonu sonlandırıyorum.

Is_Acyclic_Graph fonksiyonunda

Is_Connected fonksiyonunda verilen iki vertex arasında bir path olup olmadığını kontrol ediyorum.Kontrol etmek için BFS algoritmasını çağırıp return değeri olan parent arrayini bir değişkene atıyorum.Parent arrayi üzerinde v2==-1 olduğunda sonlanacak bir while döngüsüyle path üzerinde ilerliyorum.Eğer ilerleme esnasında v1 verteximе ulaşırsam ilerlediğim elemanları attığım Stackteki elemanları pop ederek ekrana bastırıyorum.Eğer v1 verteximе ulaşamazsam v1 ve v2 vertexi arasında bir path olmadığı için false return ediyorum.

Shortest_Path fonksiyonunda Djikstra Algoritmasını kullanarak Predecessor arrayi ve en kısa yolların olduğu arrayi elde ediyorum.Predecessor arrayini kullanarak en kısa yolda ki vertexleri Vector'e atıyorum.Path tamamlandığında ise ters bir şekilde olduğundan Vector'u tersine çevirip ekrana bastırıyorum.

2.2 Test Cases



- Plot_graph

```
PLOT_GRAPH

0-(1.0)->1-(1.0)->14
1-(1.0)->0-(1.0)->5
2-(1.0)->3-(1.0)->4
3-(1.0)->2
4-(1.0)->2-(1.0)->5
5-(1.0)->1-(1.0)->4
6
7-(1.0)->8
8-(1.0)->7-(1.0)->10
9-(1.0)->13
10-(1.0)->8-(1.0)->12
11-(1.0)->12
12-(1.0)->10-(1.0)->11
13-(1.0)->9
14-(1.0)->0
```

Graph'ın linked list şeklinde gösterilmiş hali.

- Is_UnDirected

```
IS_UNDIRECTED

Undirected
```

- Is_Acyclic

```
IS_ACYCLIC

Bu Bir Acyclic Graphtır.
```

- Is_Connected

```
IS_CONNECTED

0->1->5->4->2->3
7->8->10->12->11
(5 13) Bu iki vertex arasında bir path yok.
```

3 Q3

3.1 Problem Solution Approach

Hemen hemen diğer partlarla aynı şeyler açıklandığı için gereken açıklamaları bu kısıma kopyalıyorum.Eğer diğer partları okuduysanız bu kısmın en altındaki DFS ve BFS metodlarını açıkladığım kısıma atlayabilirsiniz.

Graphlar, vertex, edge ve weight olmak üzere 3 parçadan oluşur.Edgeler 2 vertexi birbirine bağlar.Weight ise bu edgelerin uzunluğuna verilen isimdir.Graph oluşturmak için List Graph yapısını kullandım.Insert metoduna Source Vertex ve Destination Vertex olmak üzere 2 parametre (Eğer weighted graph yapmak istiyorsak 3. Parametre olarakta weight gönderiyoruz) gönderiyoruz.Insert metodu bu 2 vertexi bir edge ile birbirine bağlar.

Graph üzerinde bir vertexten başka bir vertexe en kısa yolu bulmak için Djikstra Algoritması veya Prim's Algoritmasını kullanabiliriz.En kısa yolu bulurken kitapta verilen Djikstra Algoritmasını kullanarak buldum.Djikstra Algoritması için 2 Set, 2 Array'e ihtiyacımız var.Setlerden birinde en kısa yolu hesapladığımız vertexleri, diğerinde hesaplamadığımız vertexleri tutuyoruz.Arraylerden birinde predecessor'u diğerinde ise en kısa yolu tutuyoruz.En kısa yolları tutan arrayimizin tüm indexlerini ∞ ile ilklendiriyoruz.Daha sonra başlangıç verteximize bağlı olan tüm vertexlere olan uzaklığımız sonsuzdan daha az olacağı için yeni değerlerle güncelliyoruz ve aynı zamanda predecessorleride güncelliyoruz.Bu şekilde edgeler üzerinde ilerleyerek daha kısa bir yol bulduğumuzda o değeri ve predecessoru güncelleyerek ilerliyoruz.İşlem bittiğinde başlangıç vertexinden diğer tüm vertexlere olan en kısa uzaklığı bulmuş olacağız.

Plot_Graph metodu için iç içe 2 tane for döngüsüyle vertexleri geziyorum ve isEdge fonksiyonuyla aralarında edge olup olmadığını kontrol ediyorum.Eğer edge varsa uygun formatta ekrana bastırıyorum.List Graph yapısını kullandığım için ekrana Linked List şeklinde bastırıyorum.

Is_Undirected metodu için iç içe 2 tane for döngüsüyle vertexleri geziyorum ve isEdge fonksiyonuyla aralarında edge olup olmadığını kontrol ediyorum.Eğer edge varsa parametrelerin

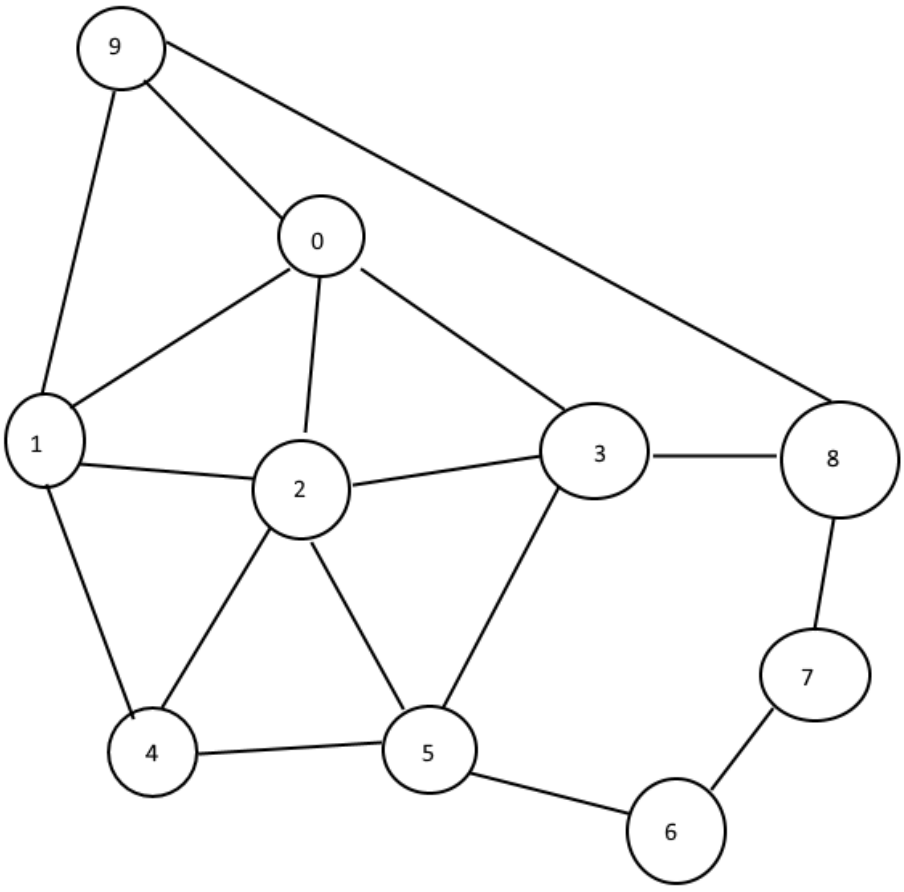
yerini değiştirip zıt yönde bir edge'nin olup olmadığını kontrol ediyorum.Zıt yönde bir edge bulamadığı anda graph'ın directed olduğu anlaşılacağı için false return edip fonksiyonu sonlandırıyorum.

Is_Acyclic_Graph metodunda

DepthFirstSearch, metodunda ekstra olarak bir Edge arrayi ve getter'ını oluşturdum. Graph'ı traverse ederken koşula bağlı olarak Edge arrayini dolduruyor.Daha sonra Edge Arrayini plot_graph metoduyla yeni bir Graph ile ekrana bastırıyorum. Bunları yaparken yardımcı olarak DFSSpan isimli metodu yazdım.

BreadthFirstSearch, metodunda ekstra olarak bir Edge ArrayList'i ve vertexlerin bulunduğu bir array oluşturdum. Graph'ı traverse ederken vertexlerin bulunduğu array koşuluna bağlı olarak Edge ArrayList'ini dolduruyorum.Normalde return edilen parent arrayini işime yaramadığı için tamamen kaldırıp metodun return değerinide Edge tipinde bir ArrayList döndürmesi şeklinde düzenledim.Bu sayede metod sonlandığı zaman yeni ağacın Edgelerini tutan Edge arrayi return edilmiş oluyor.Daha sonra Edge Arrayini plot_graph metoduyla yeni bir Graph ile ekrana bastırıyorum. Bunları yaparken yardımcı olarak BFSSpan isimli metodu yazdım.

3.2 Test Cases



- Plot_graph

```
PLOT_GRAPH

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

- Is_Undirected

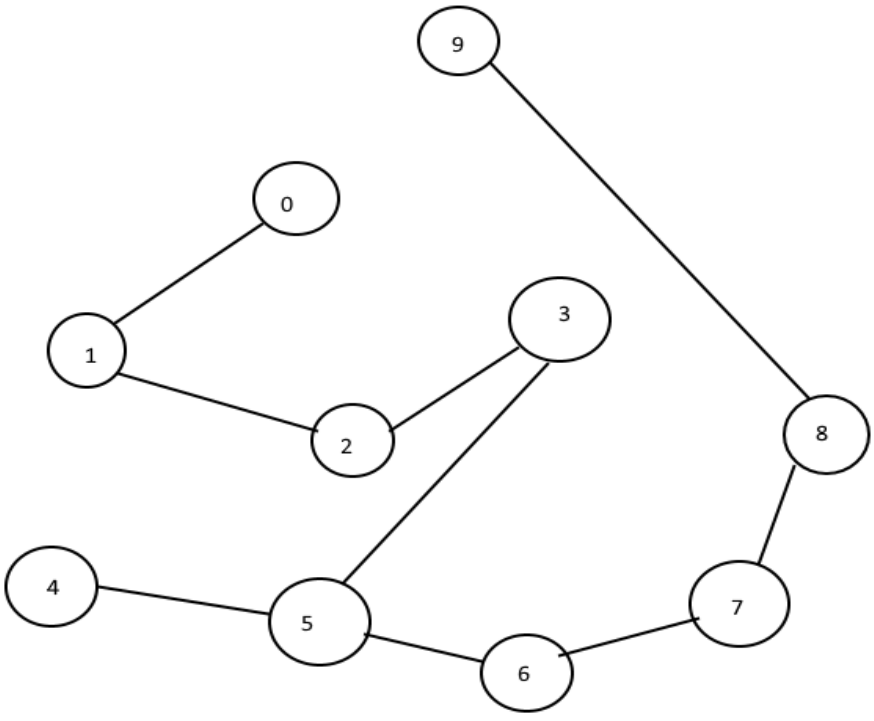
```
IS_UNDIRECTED

Undirected
```

- Is_Acyclic

```
IS_ACYCLIC

Bu Bir Cyclic Graphtır.
```

- DFS SPAN TREE

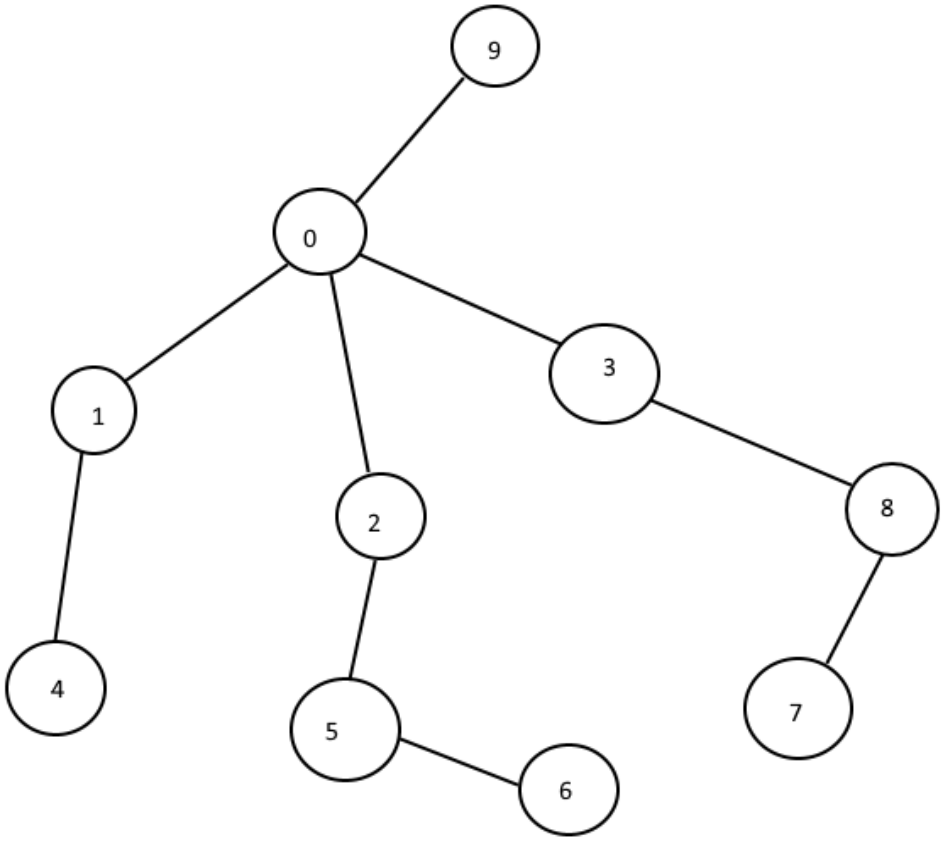
```
DFS SPAN TREE

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



Amacımız en sayıda edge kullanarak herhangi bir vertexten diğer bütün vertexlere ulaşabileceğimiz bir ağaç yapmaktı.DFS sonucunda solda görülen output'u aldım.Sağ tarafta ise onun graph şeklinde görselleştirilmiş hali bulunmaktadır.Sağ taraftanda rahatça görebileceğimiz gibi her vertexten diğer vertexlere path mevcuttur.

- BFS SPAN TREE

```
BFS SPAN TREE

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



Amacımız en sayıda edge kullanarak herhangi bir vertexten diğer bütün vertexlere ulaşabileceğimiz bir ağaç yapmaktı.BFS sonucunda solda görülen output'u aldım.Sağ tarafta ise onun graph şeklinde görselleştirilmiş hali bulunmaktadır.Sağ taraftanda rahatça görebileceğimiz gibi her vertexten diğer vertexlere path mevcuttur.

4 Q4

BFS, Level By Level Traverse yaparken DFS Derinliğe göre yapar.DFS bir vertex'te işlem yaparken daha önce ziyaret edilmemiş başka bir vertex bulursa yeni vertexte işlemine devam eder.

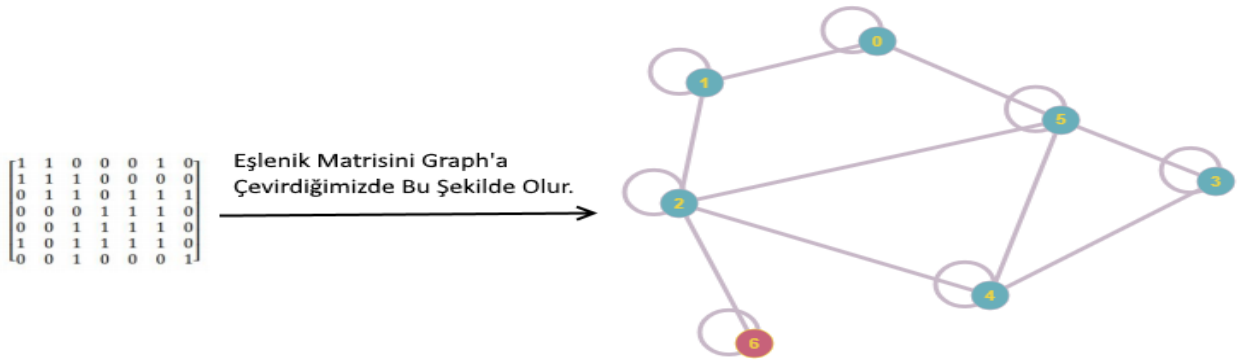
BFS, Ziyaret edilmemiş vertexleri tutarken Queue kullanır. DFS ise Stack kullanır.

BFS, DFS'ye göre daha yavaştır.

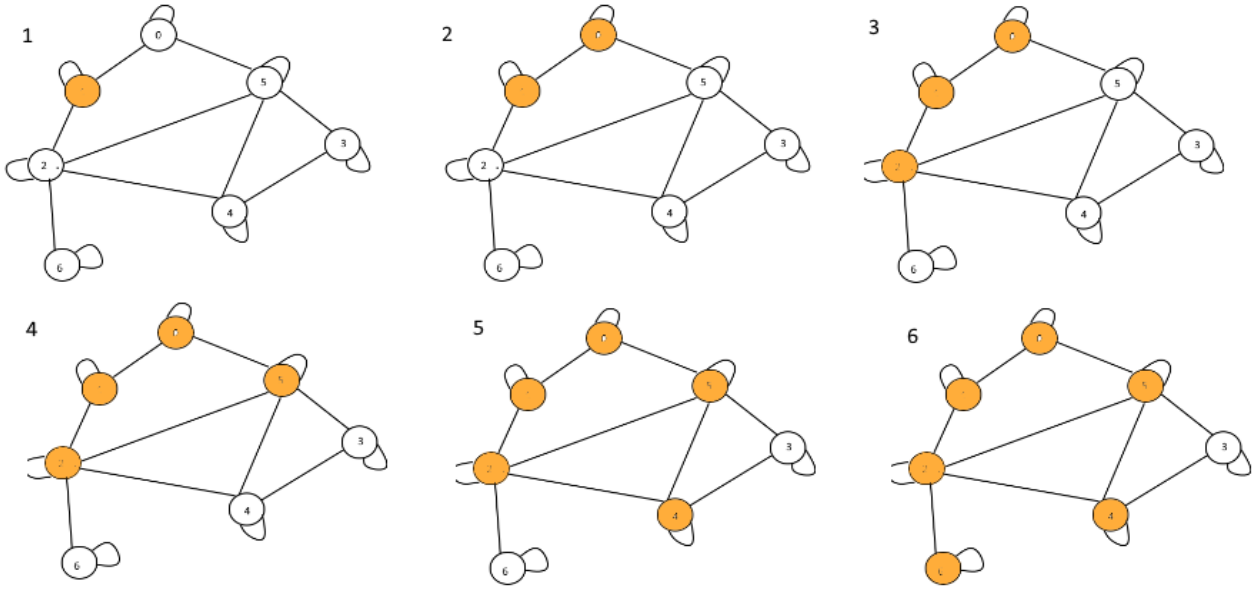
BFS, DFS'ye göre daha fazla memory tüketir.

BFS, 2 vertex arasındaki en kısa yolu bulmakta iyidir.Aynı zamanda birbirine bağlı bütün vertexleri bulma gibi işlemlerde daha avantajlıdır.

DFS, Cycle bulma, span tree bulma, labirent çözme gibi işlemlerde daha avantajlıdır.



BFS Traversal – Queueların durumunu gösteremedim fakat vertex olarak bu sıralya traverse edilecek.



DFS Traversal – Finish Order Sıralaması