Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 7 REPORT

Vefik Fırat Akman
151044031

Course Assistant: Fatma Nur Esirci

1 Q1

Ön not: Q1-Q2-Q3 Folder olarak değil. Q1MainTest tarzı java sınıfları olarak böldüm.
Graph kodları ders kitabının kod havuzundan alınmıştır.

1.1 Problem Solution Approach

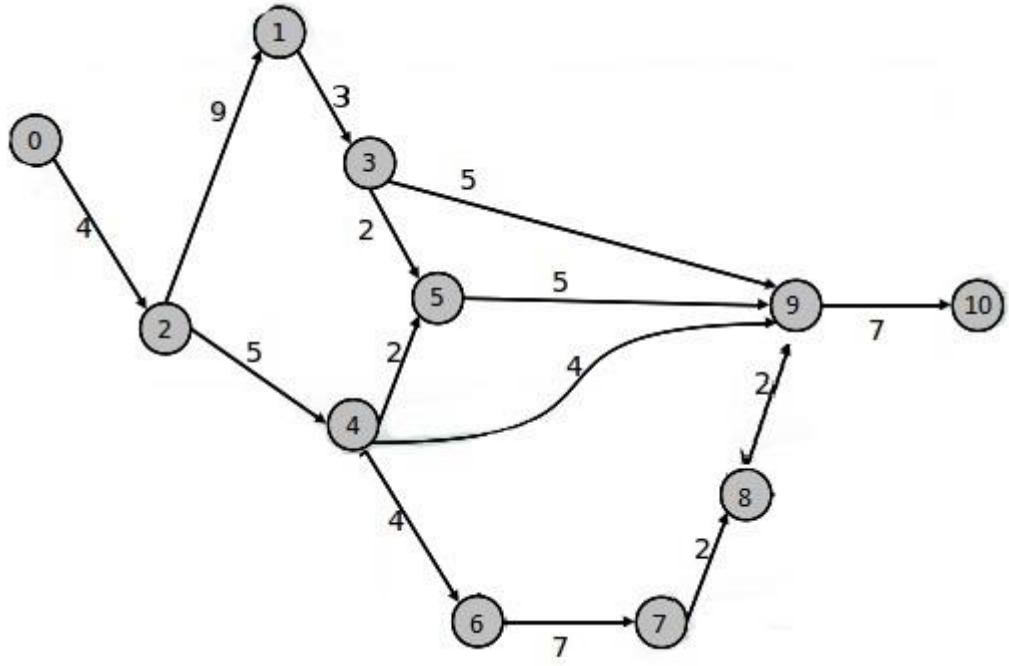
Öncelikle kitap kodlarının bulunduğu klasörden hangi sınıfların gerekli olduğunu araştırdım. Gerekli olanları aldım.
Ardından Q1 için 10 noktalı**(vertex)** bir çizge**(graph)** çizdim ve bunu gerçekledim**.(implement)**
Sonra hiçbir kaynaktan yardım almadan ve bakmadan kendim istenilen 3 metodu da yazdım.
3 metodumu da ListGraph sınıfına gömülü yaptım. Bu yüzden parametre olarak çizge**(graph)** almıyor. Lakin bunun sebebi "**edges**" dizisine **getter/setter** yazılamayacağı için.

Shortest_path(int v1,int v2): Doğru çalışıyor. **Path** yoksa **null** döndürüyor. "**Graph"** parametresi yok.
İs_undirected(): Doğru çalışıyor. **"directed"** değişkenini döndürmüyor. İstenildiği gibi **"Graph"** da gezerek **directed/undirected** olup olmadığını buluyor.

İs_acyclic(): Doğru çalışmıyor. Aslında **"cycle"** olduğunu doğru buluyor. Lakin **"cycle"** olmayan bir **"graph"** da aynı noktayı birkaç farklı nokta gösterirse onu da **"cycle"** sanıyor.

İs_undirected ve is_acyclic metotları diğer partlar içinde aynı olduğu için rapor da tekrar etmeyeceğim.

1.2 Test Cases




```
[[(0,2): 4.0]]
[[(1,3): 3.0]]
[[(2,1): 9.0], [(2,4): 5.0]]
[[(3,5): 2.0], [(3,9): 5.0]]
[[(4,5): 2.0], [(4,9): 4.0], [(4,6): 4.0]]
[[(5,9): 5.0]]
[[(6,7): 7.0]]
[[(7,8): 2.0]]
[[(8,9): 2.0]]
[[(9,10): 7.0]]

Graph undirected: false
Graph acyclic: true

0'dan 5'e shortest_path:
0->2->4->5

2'den 6'a shortest_path:
2->4->6

6'dan 2'e shortest_path:
path bulunamadı.
```

0’dan 5 e 0->1->3->5 yolu da vardır lakin daha uzundur. Yani fonksiyon doğru olan kısa yolu bulmuştur.
2’den 6’a yolu ilk noktadan başlamak zorunda olmadığını gösteren bir örnektir.
6’dan 2’e yolu ise olmayan yolu göstermek için bir örnektir.

Q2

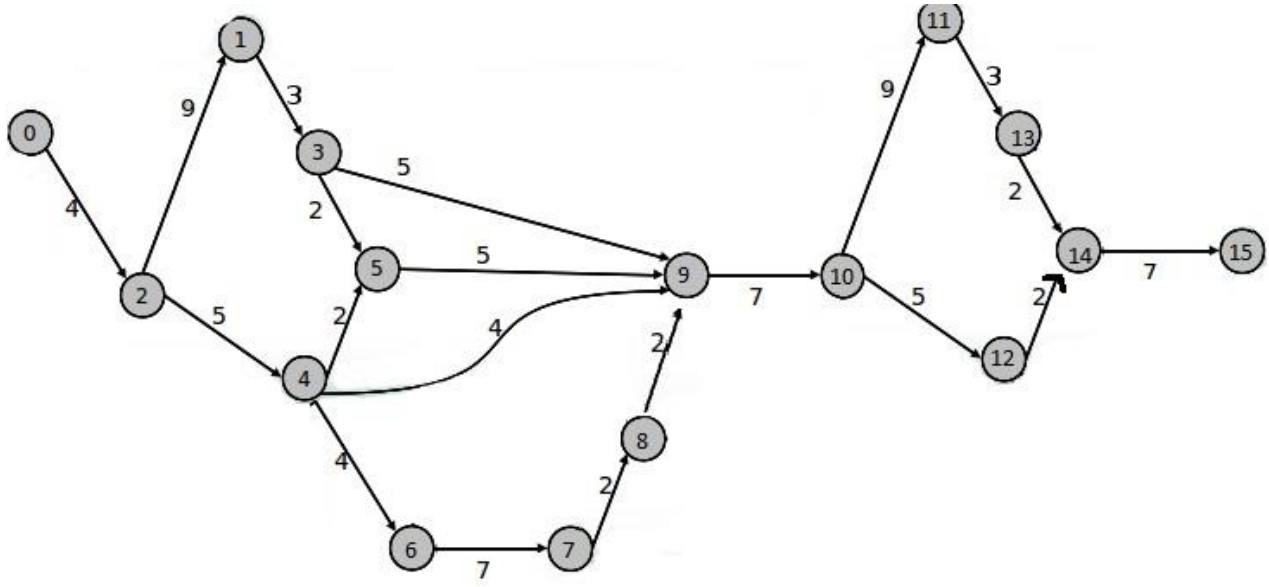
Q2 de Q1 de kullandığım “Graph”ı genişlettim. 15 noktalı bir duruma çevirdim.

1.3 Problem Solution Approach

İs_undirected ve is_acyclic metotları ilk partla aynı olduğu için tekrar etmeyeceğim.

İs_connected(int v1,int v2): Bu metot da Q1 de ki shortest_path metodumu kullandım. Şöyle ki eğer birbirlerine bağlılarsa bir yol**(path)** döndürecektir. Değiller ise **null** döndürecektir. Ben de gelen verinin **null** olup olmamasına göre **true/false** döndürdüm.
İs_connected metodu v1 veya v2 “**Graph**”da yoksa **Exception** fırlatır.

1.4 Test Cases




```
------------------------------------ QUESTION 2 ------------------------------------
[[(0,2): 4.0]]
[[(1,3): 3.0]]
[[(2,1): 9.0], [(2,4): 5.0]]
[[(3,5): 2.0], [(3,9): 5.0]]
[[(4,5): 2.0], [(4,9): 4.0], [(4,6): 4.0]]
[[(5,9): 5.0]]
[[(6,7): 7.0]]
[[(7,8): 2.0]]
[[(8,9): 2.0]]
[[(9,10): 7.0]]

Graph undirected: false
Graph acyclic: true
0 ve 1 bağlı mı?
is_connected fonksiyonu:  true
4 ve 10 bağlı mı?
is_connected fonksiyonu:  true
5 ve 8 bağlı mı?
is_connected fonksiyonu:  false
```

0 ve 1 bağlı mı? Basit bir örnek olarak arada 1 nokta daha olmasına rağmen doğru çalıştığını göstermek için bir örnek.
4 ve 10 bağlı mı? İlk noktadan başlamadan da çalıştığını göstermek ve arada 2 nokta olmasına rağmen doğru çalıştığını göstermek için bir örnek.
5 ve 8 bağlı mı? Bağlı olmadığını göstermek için bir örnek.

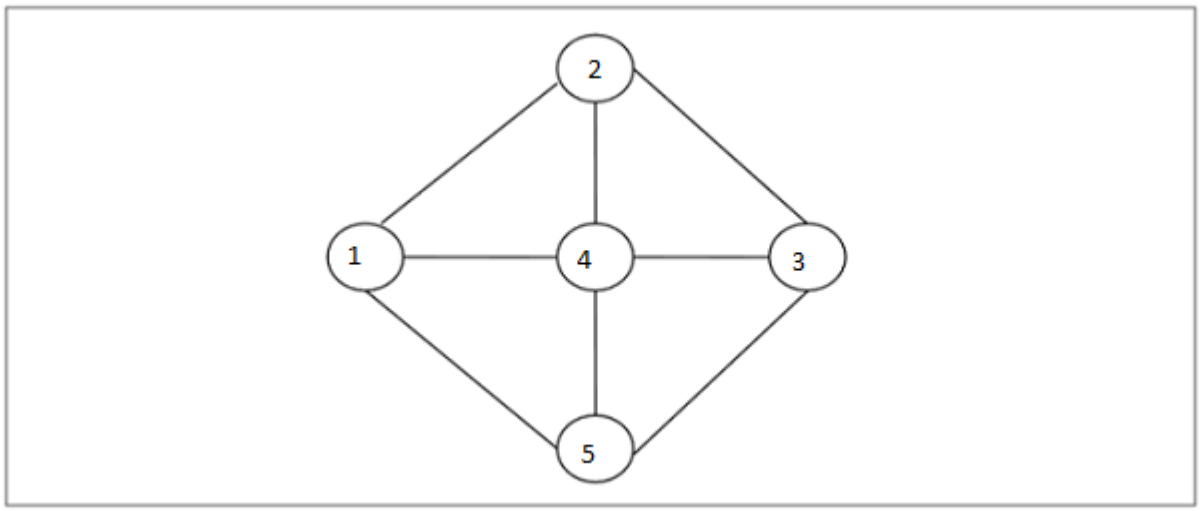
2 Q3

Q3 de geeksforgeeks'den bulduğum **DepthFirstSearch ve BreadthFirstSearch** kodlarını koduma uyarladım.

2.1 Problem Solution Approach

Geeksforgeeks de bulduğum kodları koduma uyarladım. Ardından bir spanning tree oluşturdum.
Not: Burada 10 noktalı bir spanning tree istenmiş lakin internette 10 noktalı bir görsel bulamadığım ve elle de çizemediğim için 5 noktalı kullandım.

2.2 Test Cases

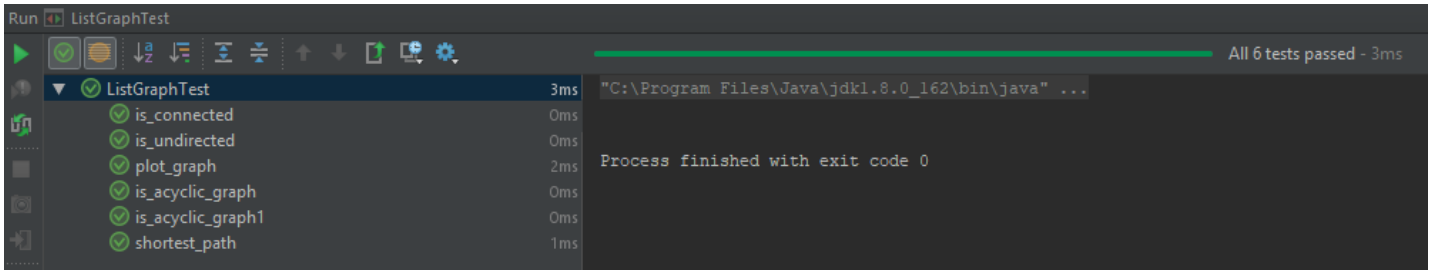



```
---------------------------------- QUESTION 3 ----------------------------------
[]
[[(1,2): 1.0], [(1,4): 1.0], [(1,5): 1.0]]
[[(2,1): 1.0], [(2,4): 1.0], [(2,3): 1.0]]
[[(3,4): 1.0], [(3,2): 1.0], [(3,5): 1.0]]
[[(4,1): 1.0], [(4,2): 1.0], [(4,3): 1.0], [(4,5): 1.0]]
[[(5,1): 1.0], [(5,4): 1.0], [(5,3): 1.0]]
[]

Graph undirected: true
Graph acyclic: true

BFS Result:
1 2 4 5 3
DFS Result:
1 2 4 3 5
Process finished with exit code 0
```

UNİT TESTİNG:

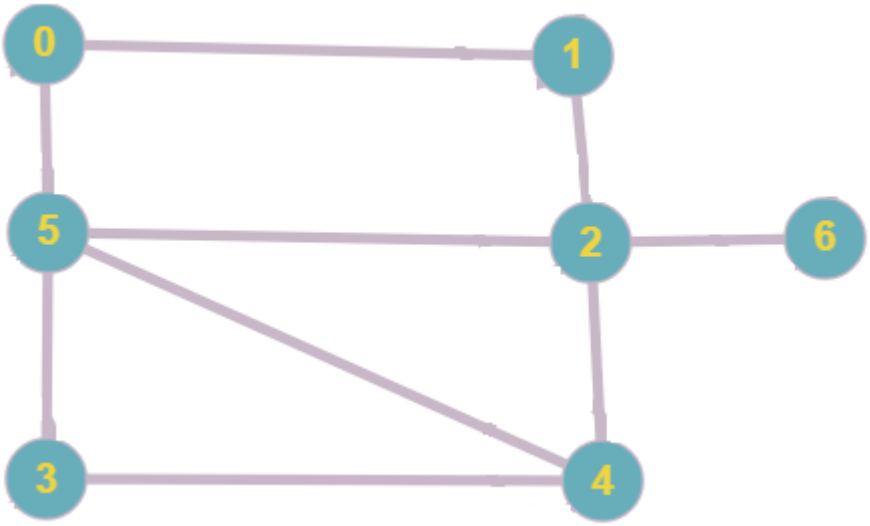


3 Q4

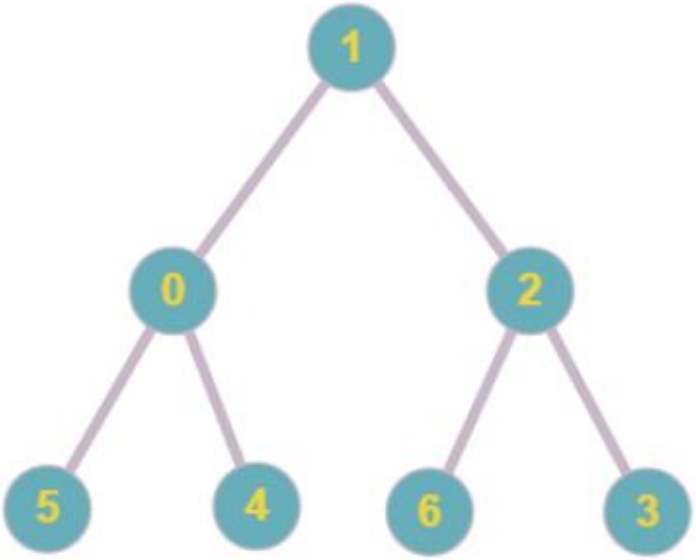
BreadthFirstSearch Ve DepthFirstSearch nedir? Farkları nelerdir?
İkisi de "**Graph**" da gezinmeyi sağlayan algoritmalardır. Ve ikisi de "**Graph**" da tüm noktaların gezileceğini garanti etmektedir. Yalnız şöyle farkları bulunmaktadır.

BreadthFirstSearch	**DepthFirstSearch**
Noktaları seviye seviye gezer.	En derinden başlayarak gezer.
Daha yavaştır ve daha fazla hafıza kullanır.	Daha hızlıdır ve daha az hafıza kullanır.
Kuyruk **(Queue)** kullanır. **FIFO**	Yığın **(Stack)** kullanır. **LIFO**
İki nokta arasında en kısa mesafe bulunabilir.	İki nokta arasında en kısa mesafe bulunamaz.
$O(b^d)$ süre *$O(b^d)$ hafıza* *d = derinlik*	*$O(b^h)$ süresi* *O(h) hafıza* *h = yükseklik*

Matrix To Graph:

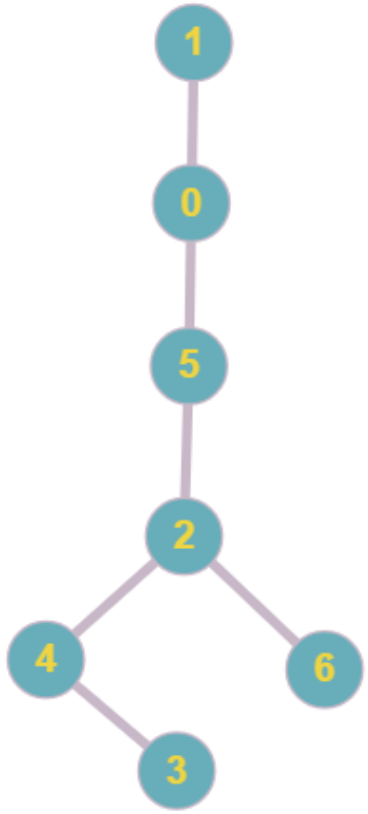


BFS:



1	0	2	5	4	6	3

DFS:



1	0	5	2	6	4	3