

**Gebze Technical University
Computer Engineering**

CSE 222 - 2018 Spring

HOMEWORK 6 REPORT

**AKIN ÇAM
151044007**

Course Assistant: Fatma Nur Esirci

1 Q1

Graph Yapısı ile ilgili istenen metotlar ListGraph ile implement edildi.
Shortest path için istenen dijsktra metodu kitap source kodu hatalı olduğu için yeniden yazıldı.
is_acycle metodu için breadthfirstsearch metodu(kitabın) kullanılmış ve değiştirilmiştir.

1.1 Problem Solution Approach

Burada weightler random olarak atanmıştır. Directed ve acyclic bir graph oluşturulmuştur. Cycle graph kabaca üçgen oluşturup oluşturmadığına bakmak gibi olabilir. Ya da bir yerden başlayınca kendisine dönüyor mu buna bakılır.

İs_connected metodu bir graph ve iki tane vertex alır.

Başlangıç vertexini seçer(ikisi arasından) ve orayı gezdi olarak işaretler.

Queue size 0 olana kadar devam eder.

Komşuları ile 2. Vertex i karşılaştırır eşleşme varsa true döndürür yoksa orayı queuedan siler ve bir sonraki ile devam eder. Her gezdiği yeri işaretler bulamazsa false return eder.

Çalışma zamanı için lineer ve $O(n)$ olduğunu söyleyebiliriz.

İs_shortest path metodu için dijsktra metodu tekrar yazılmıştır(kitap source koda bulunan metodu bende çalışmadığı için)

Bu metot şöyle çalışır:

Başlangıç indexini queue ya ekler.cost ve pred ilklendirilir. En küçük cost olan bulunur daha sonra komşuları arasında en düşük cost olan bulunur ve cost toplanıp ordan devam edilir. Kullanılan vertex silinir bir sonraki vertexten devam edilir queue.size 0 olana kadar. Çalışma zamanı için lineerden biraz daha yavaş olabilir diyebiliriz. Daha sonra shortest_path predecessorsleri ekler başlangıç indexinden başlayarak gidilmek istenen yere kadar ve vektöre yazılır. Eğer bağlantı yoksa boş vektör return edilir.

İs_undirected metodu bütün vertexleri gezer lineer zamanda çalışır. Bir vertexten başlar komşusuna bakar komşusu ile kendisi arasındada edge varsa ve costlar eşitse devam eder bunun dışında herhangi bir durumda false return eder çünkü bağlantı iki yönlü değildir ya da costlar eşit değildir. Lineer zamanda çalışır.

İs_acycle metodu breadthfirstsearch metodu kullanılarak üzerine ekleme yapılarak oluşturulmuştur. BreadthFirstSearch kitaptan source koddan alınmıştır. Komşulara bakarken directed olup olmadığına bakar directed değilse bu şu demektir başladığı noktaya döndüğünde parentı değişecektir. Bunu kontrol eder. Eğer directed ise üç sonraki komşusu kendisi mi diye bakar doğruysa cycledır false return eder. Çalışma zamanı $O(n^2)$ dir.

Plot_graph→

Bütün vertexleri dolaşır kendisini ve komşularını ekrana yazdırır. Lineer Zamanda $Teta(n)$

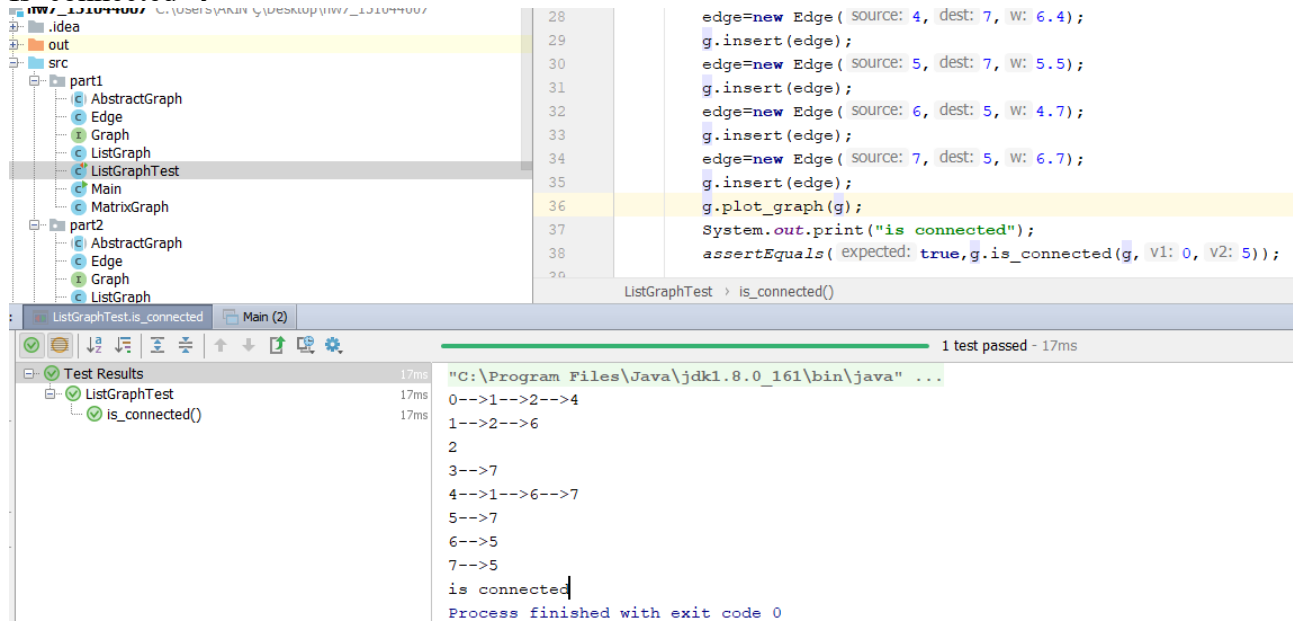
Spanning_method→

Bu metot part3 için yazılmıştır. Kitapta bulunan 585 teki pseduo kode dikkate alınarak yazılmıştır. Her metotta olduğu gibi gezilenler array liste kaydedilir. Ve vertexleri Queue yardımı ile gezilir. Vertex in komşusu gezilmedi ise kendisi yeni bir grapha eklenir. Daha sonra komşularına geçer onları queue ya ekler. eğer komşusu var ise queuedaki diğer eleman ile devam eder. Bu sayede gereksiz bağlantılar çıkarılır ve eski graph işlevini gören bir graph oluşturulur. Çünkü eğer komşusu varken eklenseydi fazladan bir yol eklenmiş olacaktır. Çalışma zamanı lineerdir.

1.2 Test Cases

Unit-Test:

is_connected→



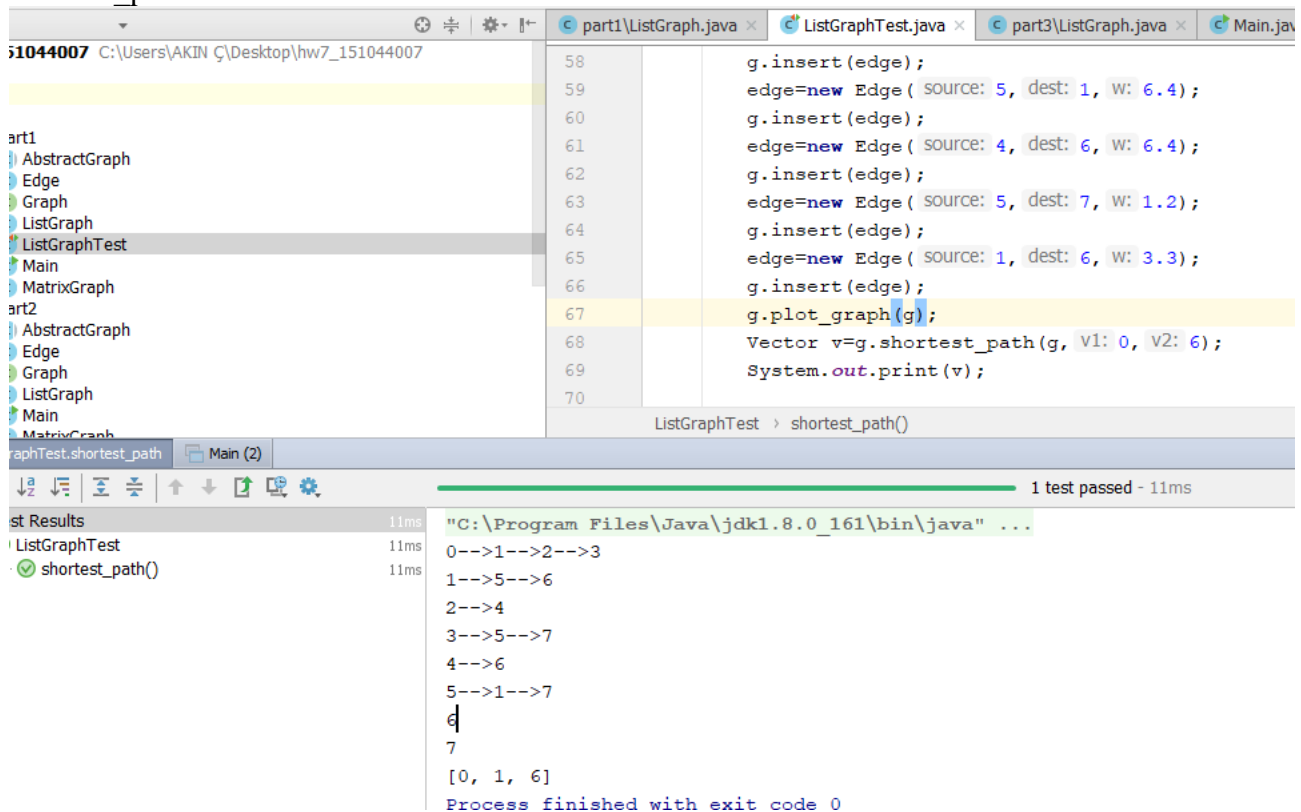
```
28 edge=new Edge( source: 4, dest: 7, w: 6.4);
29 g.insert(edge);
30 edge=new Edge( source: 5, dest: 7, w: 5.5);
31 g.insert(edge);
32 edge=new Edge( source: 6, dest: 5, w: 4.7);
33 g.insert(edge);
34 edge=new Edge( source: 7, dest: 5, w: 6.7);
35 g.insert(edge);
36 g.plot_graph(g);
37 System.out.print("is connected");
38 assertEquals( expected: true, g.is_connected(g, v1: 0, v2: 5));
```

Test Results: 1 test passed - 17ms

Test Results: ListGraphTest is_connected() 17ms

0-->1-->2-->4
1-->2-->6
2
3-->7
4-->1-->6-->7
5-->7
6-->5
7-->5
is connected
Process finished with exit code 0

Shortest_path→



```
58 g.insert(edge);
59 edge=new Edge( source: 5, dest: 1, w: 6.4);
60 g.insert(edge);
61 edge=new Edge( source: 4, dest: 6, w: 6.4);
62 g.insert(edge);
63 edge=new Edge( source: 5, dest: 7, w: 1.2);
64 g.insert(edge);
65 edge=new Edge( source: 1, dest: 6, w: 3.3);
66 g.insert(edge);
67 g.plot_graph(g);
68 Vector v=g.shortest_path(g, v1: 0, v2: 6);
69 System.out.print(v);
```

Test Results: 1 test passed - 11ms

Test Results: ListGraphTest shortest_path() 11ms

0-->1-->2-->3
1-->5-->6
2-->4
3-->5-->7
4-->6
5-->1-->7
[0, 1, 6]
Process finished with exit code 0

is_undirected→

```

7_151044007
91      edge=new Edge( source: 4, dest: 6, w: 6.4);
92      g.insert(edge);
93      edge=new Edge( source: 5, dest: 7, w: 1.2);
94      g.insert(edge);
95      edge=new Edge( source: 1, dest: 6, w: 3.3);
96      g.insert(edge);
97      g.plot_graph(g);
98      System.out.println("isnt is_undirected");
99      assertEquals( expected: false,g.is_undirected(g));
100  }
101  @org.junit.jupiter.api.Test
102  void is_acycle() {
103      ListGraph g=new ListGraph( numV: 8, directed: true);
ListGraphTest > is_undirected()

```

```

1 test passed - 12ms
12ms "C:\Program Files\Java\jdk1.8.0_161\bin\java" ...
12ms 0-->1-->2-->3
12ms 1-->5-->6
2-->4
3-->5-->7
4-->6
5-->1-->7
6
7
isnt is_undirected

Process finished with exit code 0

```

Is_acycle→

```

129      assertEquals( expected: true,g.is_acyclic_graph(g));
130      ListGraph graph=new ListGraph( numV: 4, directed: true);
131      edge=new Edge( source: 1, dest: 2, w: 1.2);
132      graph.insert(edge);
133      edge=new Edge( source: 2, dest: 3, w: 1.2);
134      graph.insert(edge);
135      edge=new Edge( source: 3, dest: 1, w: 1.2);
136      graph.insert(edge);
137      g.plot_graph(g);
138      System.out.print("is_acycle");
139      assertEquals( expected: false,g.is_acyclic_graph(graph));
140  }
141  @org.junit.jupiter.api.Test
142  void plot_graph() {
143      ListGraph g=new ListGraph( numV: 8, directed: true);
144      Edge edge=new Edge( source: 0, dest: 1, w: 3.0);
ListGraphTest > is_acycle()

```

```

1 test passed - 14ms
14ms "C:\Program Files\Java\jdk1.8.0_161\bin\java" ...
14ms isnt is_acycle0-->1-->2-->3
14ms 1-->5-->6
2-->4
3-->5-->7
4-->6
5-->1-->7
6
7
is_acycle

Process finished with exit code 0

```

Plot_graph→

```

1
Test
aph
Graph
1
aph
Graph
firstSearch
stSearch
1
aph
7.iml
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
}

g.insert(edge);
edge=new Edge( source: 4, dest: 6, w: 6.4);
g.insert(edge);
edge=new Edge( source: 5, dest: 7, w: 1.2);
g.insert(edge);
edge=new Edge( source: 1, dest: 6, w: 3.3);
g.insert(edge);

g.plot_graph(g);
ListGraph graph=new ListGraph( numV: 4, directed: true);
edge=new Edge( source: 1, dest: 2, w: 1.2);
graph.insert(edge);
edge=new Edge( source: 2, dest: 3, w: 1.2);
graph.insert(edge);
edge=new Edge( source: 3, dest: 1, w: 1.2);
graph.insert(edge);
System.out.println("-----");
graph.plot_graph(graph);
}

ListGraphTest > plot_graph()
t_graph Main (2)
1 test passed - 9ms
Test 9ms 2-->4
raph() 9ms 3-->5-->7
9ms 4-->6
5-->1-->7
6
7
-----
0
1-->2
2-->3
3-->1

```

Running Test Q1→

```

part2
- AbstractGraph
- Edge
- Graph
- ListGraph
- Main
- MatrixGraph
part3
- AbstractGraph
- BreadthFirstSearch
- DepthFirstSearch
- Edge
- Graph
Main Main (2)
31
32
33
34
35
36
37
38
39
}

graph.plot_graph(graph);
System.out.println("The graph is directed so is_undirected should be false");
System.out.println("The graph is acyclic so is_acyclic should be true");
System.out.println(graph.shortest_path(graph, v1: 0, v2: 5));
System.out.println(graph.shortest_path(graph, v1: 0, v2: 5));
System.out.println(graph.shortest_path(graph, v1: 1, v2: 5));
System.out.println(graph.shortest_path(graph, v1: 1, v2: 9));
}

Main > main()
Directed acyclic graph have random weight(v=10 e=20) w=random----->
0-->1-->3-->5
1-->2-->4-->5
2-->3-->5-->7
3-->4-->6
4-->5-->7-->8
5-->6-->8
6-->7-->9
7-->8
8-->9
9
The graph is directed so is_undirected should be false-->>false
The graph is acyclic so is_acyclic should be true-->true
[0, 1, 5]
[0, 1, 5]
[1, 5]
[1, 4, 7, 8, 9]

Process finished with exit code 0

```

2 Q2

This part about Question2 in HW7

2.1 Problem Solution Approach

Burada undirected ve acyclic bir graph oluşturulmuştur. Başlangıçta constuructra directed false yapılmış. Acycle için de eklenirken bir döngü oluşturulmamasına dikkat edilmiştir.

İs_connected metodu bir graph ve iki tane vertex alır.

Başlangıç vertexini seçer(ikisi arasından) ve orayı gezdi olarak işaretler.

Queue size 0 olana kadar devam eder.

Komşuları ile 2. Vertex i karşılaştırır eşleşme varsa true döndürür yoksa orayı queuedan siler ve bir sonraki ile devam eder. Her gezdiği yeri işaretler bulamazsa false return eder.

Çalışma zamanı için lineer ve $O(n)$ olduğunu söyleyebiliriz.

İs_shortest path metodu için dijsktra metodu tekrar yazılmıştır(kitap source koda bulunan metodu bende çalışmadığı için)

Bu metot şöyle çalışır:

Başlangıç indexini queue ya ekler.cost ve pred ilklendirilir. En küçük cost olan bulunur daha sonra komşuları arasında en düşük cost olan bulunur ve cost toplanıp ordan devam edilir. Kullanılan vertex silinir bir sonraki vertexten devam edilir queue.size 0 olana kadar. Çalışma zamanı için lineerden biraz daha yavaş olabilir diyebiliriz. Daha sonra shortest_path predeccesorsleri ekler başlangıç indexinden başlayarak gidilmek istenen yere kadar ve vectöre yazılır. Eğer bağlantı yoksa boş vektör return edilir.

İs_undirected metodu bütün vertexleri gezer lineer zamanda çalışır. Bir vertexten başlar komşusuna bakar komşusu ile kendisi arasındada edge varsa ve costlar eşitse devam eder bunun dışında herhangi bir durumda false return eder çünkü bağlantı iki yönlü değildir ya da costlar eşit değildir. Lineer zamanda çalışır.

İs_acycle metodu breadthfirstsearch metodu kullanılarak üzerine ekleme yapılarak oluşturulmuştur. BreadthFirstSearch kitaptan source koddan alınmıştır. Komşulara bakarken directed olup olmadığına bakar directed değilse bu şu demektir başladığı noktaya döndüğünde parentı değişecektir. Bunu kontrol eder. Eğer directed ise üç sonraki komşusu kendisi mi diye bakar doğruysa cycledır false return eder. Çalışma zamanı $O(n^2)$ dir.

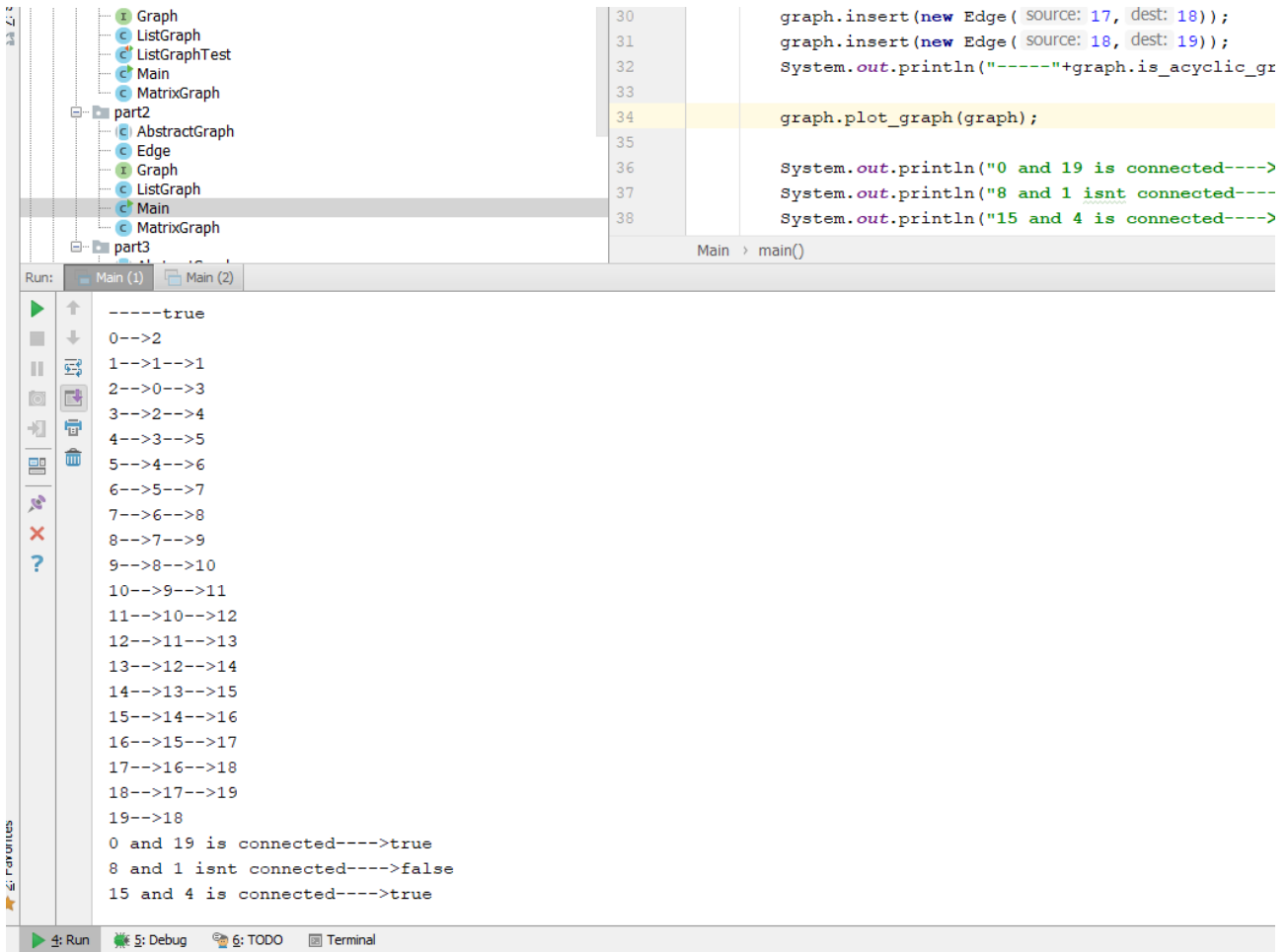
Plot_graph→

Bütün vertexleri dolaşır kendisini ve komşularını ekrana yazdırır. Lineer Zamanda Teta(n)

Spanning_method→

Bu metot part3 için yazılmıştır. Kitapta bulunan 585 teki pseduo kode dikkate alınarak yazılmıştır. Her metotta olduğu gibi gezilenler array liste kaydedilir. Ve vertexleri Queue yardımı ile gezilir. Vertex in komşusu gezilmedi ise kendisi yeni bir grapha eklenir. Daha sonra komşularına geçer onları queue ya ekler. eğer komşusu var ise queuedaki dğer eleman ile devam eder. Bu sayede gereksiz bağlantılar çıkarılır ve eski graph işlevini gören bir graph oluşturulur. Çünkü eğer komşusu varken eklenseydi fazladan bir yol eklenmiş olacaktır. Çalışma zamanı lineerdir.

2.2 Test Cases



The screenshot shows an IDE with a project structure on the left, source code in the center, and a run console at the bottom.

Project Structure:

- Graph
- ListGraph
- ListGraphTest
- Main
- MatrixGraph
- part2
 - AbstractGraph
 - Edge
 - Graph
 - ListGraph
 - Main
 - MatrixGraph
- part3

Source Code (Main.java):

```
30 graph.insert(new Edge( source: 17, dest: 18));
31 graph.insert(new Edge( source: 18, dest: 19));
32 System.out.println("-----"+graph.is_acyclic_gr
33
34 graph.plot_graph(graph);
35
36 System.out.println("0 and 19 is connected---->
37 System.out.println("8 and 1 isnt connected----
38 System.out.println("15 and 4 is connected---->
```

Run Console Output:

```
-----true
0-->2
1-->1-->1
2-->0-->3
3-->2-->4
4-->3-->5
5-->4-->6
6-->5-->7
7-->6-->8
8-->7-->9
9-->8-->10
10-->9-->11
11-->10-->12
12-->11-->13
13-->12-->14
14-->13-->15
15-->14-->16
16-->15-->17
17-->16-->18
18-->17-->19
19-->18
0 and 19 is connected---->true
8 and 1 isnt connected---->false
15 and 4 is connected---->true
```

3 Q3

This part about Question3 in HW7

3.1 Problem Solution Approach

Is_connected metodu bir graph ve iki tane vertex alır.

Başlangıç vertexini seçer(ikisi arasından) ve orayı gezdi olarak işaretler.

Queue size 0 olana kadar devam eder.

Komşuları ile 2. Vertex i karşılaştırır eşleşme varsa true döndürür yoksa orayı queuedan siler ve bir sonraki ile devam eder. Her gezdiği yeri işaretler bulamazsa false return eder.

Çalışma zamanı için lineer ve $O(n)$ olduğunu söyleyebiliriz.

Is_shortest path metodu için dijkstra metodu tekrar yazılmıştır(kitap source koda bulunan metodu bende çalışmadığı için)

Bu metot şöyle çalışır:

Başlangıç indexini queue ya ekler.cost ve pred ilklendirilir. En küçük cost olan bulunur daha sonra komşuları arasında en düşük cost olan bulunur ve cost toplanıp ordan devam edilir. Kullanılan vertex silinir bir sonraki vertexten devam edilir queue.size 0 olana kadar. Çalışma zamanı için lineerden biraz daha yavaş olabilir diyebiliriz. Daha sonra shortest_path predecessorsleri ekler

başlangıç indexinden başlayarak gidilmek istenen yere kadar ve vektöre yazılır. Eğer bağlantı yoksa boş vektör return edilir.

İs_undirected metodu bütün vertexleri gezer lineer zamanda çalışır. Bir vertexten başlar komşusuna bakar komşusu ile kendisi arasındada edge varsa ve costlar eşitse devam eder bunun dışında herhangi bir durumda false return eder çünkü bağlantı iki yönlü değildir ya da costlar eşit değildir. Lineer zamanda çalışır.

İs_acycle metodu breadthfirstsearch metodu kullanılarak üzerine ekleme yapılarak oluşturulmuştur. BreadthFirstSearch kitaptan source koddan alınmıştır. Komşulara bakarken directed olup olmadığına bakar directed değilse bu şu demektir başladığı noktaya döndüğünde parentı değişecektir. Bunu kontrol eder. Eğer directed ise üç sonraki komşusu kendisi mi diye bakar doğruysa cycledır false return eder. Çalışma zamanı $O(n^2)$ dir.

Plot_graph→

Bütün vertexleri dolaşır kendisini ve komşularını ekrana yazdırır. Lineer Zamanda Teta(n)

Spanning_method→

Bu metod part3 için yazılmıştır. Kitapta bulunan 585 teki pseduo kode dikkate alınarak yazılmıştır. Her metotta olduğu gibi gezilenler array liste kaydedilir. Ve vertexleri Queue yardımı ile gezilir. Vertex in komşusu gezilmedi ise kendisi yeni bir grapha eklenir. Daha sonra komşularına geçer onları queue ya ekler. eğer komşusu var ise queuedaki diğer eleman ile devam eder. Bu sayede gereksiz bağlantılar çıkarılır ve eski graph işlevini gören bir graph oluşturulur. Çünkü eğer komşusu varken eklenseydi fazladan bir yol eklenmiş olacaktır. Çalışma zamanı lineerdir.

3.2 Test Cases

```
graph
├── AbstractGraph
├── BreadthFirstSearch
├── DepthFirstSearch
├── Edge
├── Graph
├── ListGraph
├── Main
├── MatrixGraph
└── hw7_151044007.iml
External Libraries
Run: Main (1) Main (2)
"C:\Program Files\Java\jdk1.8.0_161\bin\java" ...
Directed acyclic graph have random weight(v=15) w=constant----->
0-->1-->2-->4
1-->0-->2-->3-->3
2-->1-->0-->3
3-->2-->1-->4-->1
4-->0-->3
if cycle-graph returns false---->>false
After spanning method--->0-->1-->2-->4
1-->0-->3
2-->0
3-->1
4-->0
BreadthFirstSearch(this method taken by book sourcecode) parent label--> after spanning
-1 0 0 1 0
1 -1 0 1 0
2 0 -1 1 0
1 3 0 -1 0
Process finished with exit code 0
```

4 Q4

BFS kökten dolaşmaya başlar ve seviye seviye dalaşır(laverOrderSearch) kök düğümüne yakın olanları daha çabuk bulabilir.

İmplement edilirken seviye seviye olduğu için queue kullanılır. Tek aşamada çalışır. Queue dan eleman alınır kullanılır ve bir sonraki ile devam edilir.

DFS ye göre daha çok memory kullanır ve bunada bağlı olarak daha yavaştır.

En kısa yolu bulmak için, Spanning ağacında, Connectivity de kullanılır.

DFS kökten bağlar ve height i baz alarak daha aşağı seviyelere bakar.

İmplement edilirken seviye seviye olduğu için stack kullanılır.

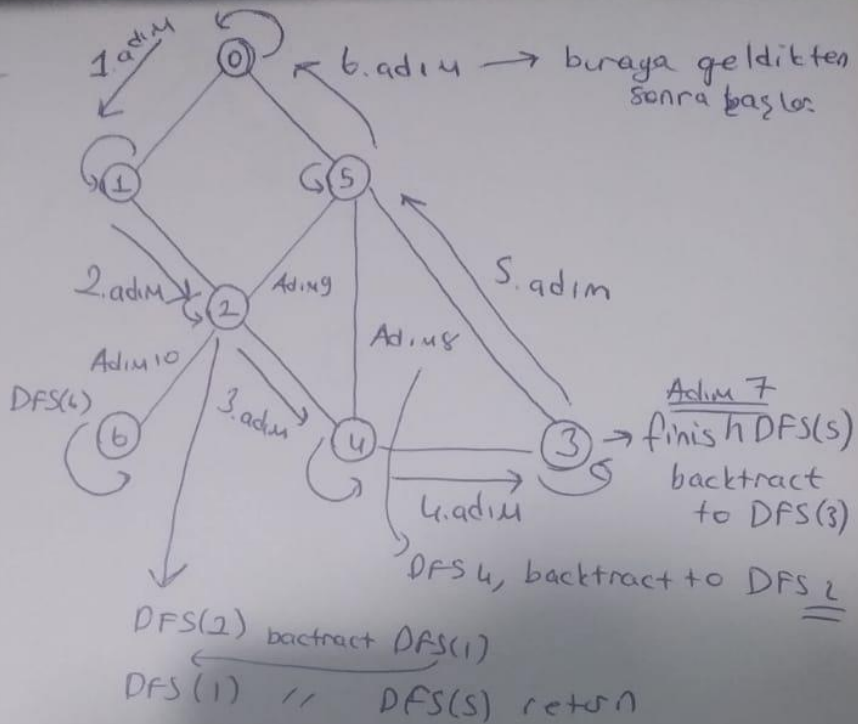
Bu algoritma iki aşamada çalışır - ilk aşamada, ziyaret edilen köşeler stack e eklenir daha sonra ziyaret edilecek bir tepe noktası yok olduğunda stackten alarak aramaya başlar.

BFS ye göre daha az memory kullanır ve bunada bağlı olarak daha hızlıdır.

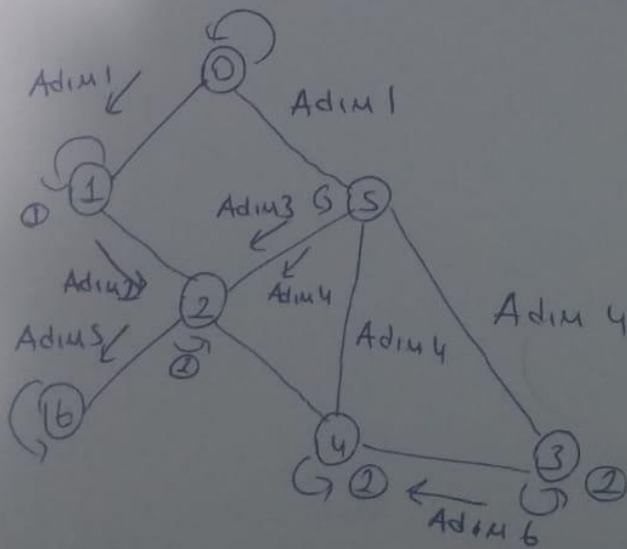
Çevrim tespiti için kullanışlıdır, Bağlantı testinde, Grafikte a ve b arasında bir yol bulmak.

Spanning tree aksine forest treeler için daha kullanışlıdır .Daha kompleks olduğu için.

DFS =



BFS =



0 a bakar sonra komşularına gider sonra 1'in komşularına bakar. 5'in komşularına bakar. Level 2 ye bakar. daha sonra enson 6 ya (level 3 e bakar)