Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 7 REPORT

SERKAN SORMAN
151044057

Course Assistant: Fatma Nur Esirci

1 Q1

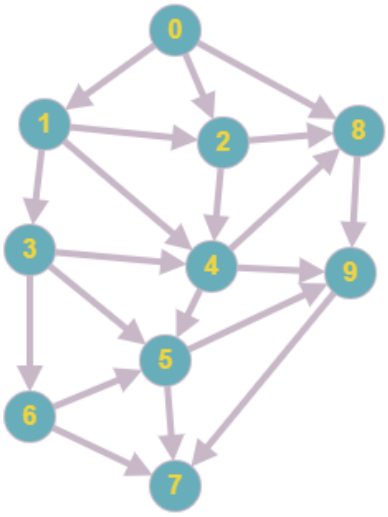
This part about Question1 in HW7

1.1 Problem Solution Approach

Directed graph oluşturulurken adjacency list yapısı kullanıldı.10 vertex ve 20 edge eklendi. Eklenen edgelerin weightleri random olarak oluşturuldu. Edgeler eklenirken graphın acyclic olma koşulu korundu. Plot_graph metodu içinde tüm graph adjacency list gösterimi şeklinde ekrana basıldı weight değeri ok işaretinin arasında belirtildi. 2 -5.0-> 6 (2den 6 ya 5 weight var). İs_undirected metodu için var olan edgelerin zıt yönde ve aynı weighte bir edgesi olması koşulu arandı. is_acyclic_graph metodunda DFS yapılarak tüm graphta en az 1 tane cycle arandı. Bu metod directed grapha özel olarak yazıldı. Çünkü bir çevrim olması sağlansa bile aynı yönde bir çevrim olmayabileceğinden bu cycle olarak değerlendirilemez.

shortest_path metodunda verilen iki nokta arasındaki en kısa yol bulunurken dijkstra algoritması kullanıldı. İçinde vector olarak path ve double weight tutan Path class yapısı yazıldı. Başlangıç nodundan bitiş noduna olan tüm yollar bulundu weightleri hesaplandı ve weightleri ile beraber bir Path referansı oluşturuldu. Tüm bu Pathler bir array liste atılarak weightlerine göre sıralandı ve en küçük weighte sahip Pathin nodeları vector olarak return edilip toplam weightleri ekrana basıldı.

1.2 Test Cases



Şekildeki directed graph main testinde random weightler ile oluşturuldu ve ardından list temsili hali ekrana plot_graph ile bastırıldı. Ardından is_undirected metodu ile graphın directed olduğu onaylandı. is_acyclic_graph metodu ile graphın acyclic olduğu test edildi. Daha sonra 3 farklı start-end ile shortest_path metodu test edildi ve en kısa pathleri total weightleri ile beraber ekrana basıldı.

```
0 -6.0-> 1 -9.0-> 2 -11.0-> 8
1 -11.0-> 2 -9.0-> 3 -11.0-> 4
2 -9.0-> 4 -4.0-> 8
3 -9.0-> 4 -7.0-> 5 -10.0-> 6
4 -5.0-> 5 -6.0-> 8 -7.0-> 9
5 -9.0-> 7 -4.0-> 9
6 -11.0-> 5 -7.0-> 7
7
8 -8.0-> 9
9 -7.0-> 7

Directed Graph
Acyclic Graph

Total weight: 18.0
Shortest path between 1 - 9: [1, 4, 9]

Total weight: 26.0
Shortest path between 0 - 7: [0, 8, 9, 7]

Total weight: 16.0
Shortest path between 1 - 5: [1, 3, 5]
```

2 Q2

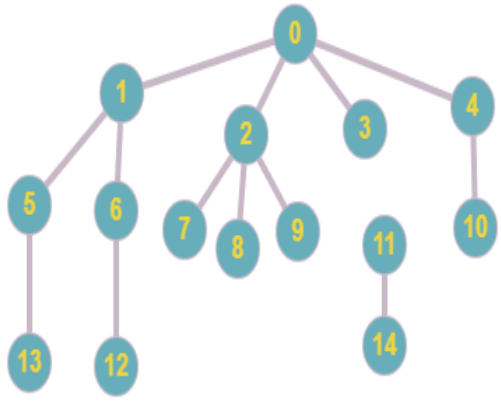
This part about Question2 in HW7

2.1 Problem Solution Approach

Undirected graph oluşturulurken adjacency list yapısı kullanıldı.15 vertex ve 13 edge eklendi. Eklenen edgelerden bir tanesi diğerlerinden unconnectted olarak eklendi. Edgeler eklenirken graphın acyclic olma koşulu korundu. Directed graph için yazılan plot graph metodu kullanıldı. Tek fark olarak grap unweighted olduğu için weightler ekrana basılmadı. İs_undirected metodu directed graphınki kullanıldı. is_acyclic_graph metodunda DFS yapılarak tüm graphta en az 1 tane cycle arandı. Bu metod undirected grapha özel olarak yazıldı. Cycle koşulu en az 3 vertexden oluşan bir cycle bulunma durumu ile arandı.

İs_connected metodunda BFS kullanılarak belirtilen iki node arasında bir yol olup olmadığı check edildi. Aranan nodeların graphta bulunmaması durumu için exception fırlatıldı.

2.2 Test Cases



Şekildeki graph main testinde oluşturuldu ve ardından list temsili hali ekrana plot_graph ile bastırıldı. Ardından is_undirected metodu ile graphın undirected olduğu onaylandı. is_acyclic_graph metodu ile graphın acyclic olduğu test edildi. Daha sonra 0-12, 4-7, 5-10 pathlerinin olup olmadığı is_connected metodu test edildi ve doğrulandı.Ardından 2-11 nodeları arasında path olup olmadığı test edildi ve olmadığı doğrulandı. Son olarak ise 10-20 nodeları için test edildi fakat 20 nodu olmadığı için exception try catch bloğu içinde yakalandı.

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

Undirected Graph
Acyclic Graph

Node 0 and 12 are connected
Node 4 and 7 are connected
Node 5 and 10 are connected
Node 2 and 11 are not connected
Node can not found
```

3 Q3

This part about Question3 in HW7

3.1 Problem Solution Approach

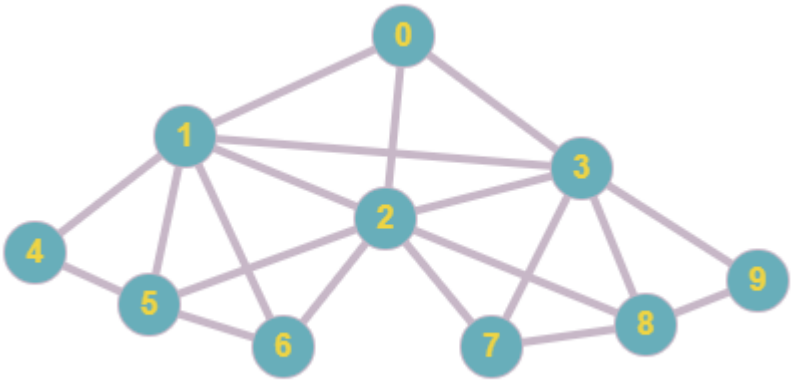
Undirected graph oluşturulurken adjacency list yapısı kullanıldı.10 vertex ve 20 edge eklendi. Edgeler eklenirken graphda birçok cycle oluşturuldu. plot graph, İs_undirected ve is_acyclic_graph metodları part2deki ile aynı şekilde kullanıldı.

Grapha ait spanning treelerin elde edilmesi için ilk olarak BreathFirstSearch yapılarak, gezilen nodelar queuya atıldı ve bu queue üzerinden kontrol işlemleri yapılarak mümkün olabilecek en az sayıda edge bulundurulan ve bir nodedan diğer tüm nodelara erişim sağlanabilecek spanning tree elde edilmesi için traverse yapıldı. Bu traverse sırasında gerekli edgeler graphtan silindi ardından spanning tree ekrana basıldı. Silinme işlemi için edgeleri silen remove metodu yazıldı.

Aynı işlemler DFS ile spanning tree elde edilmesinde de yapıldı. Fakat bu işlem sırasında queu yerine stack kullanıldı ve aynı kontrol işlemleri bu stack üzerinden yapılarak gerekli edgeler graphtan silindi ardından spanning tree ekrana basıldı

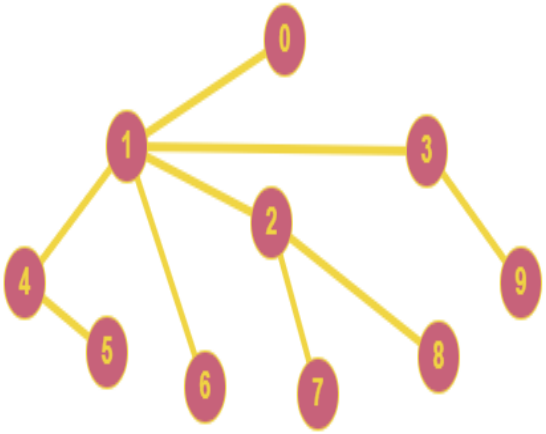
Ardından diğer spanning treelerin ortaya çıkarılması için sırayla graph üzerindeki tüm noktalar başlangıç noktası alınarak spanning treeler elde edildi ve ekrana basıldı.

3.2 Test Cases



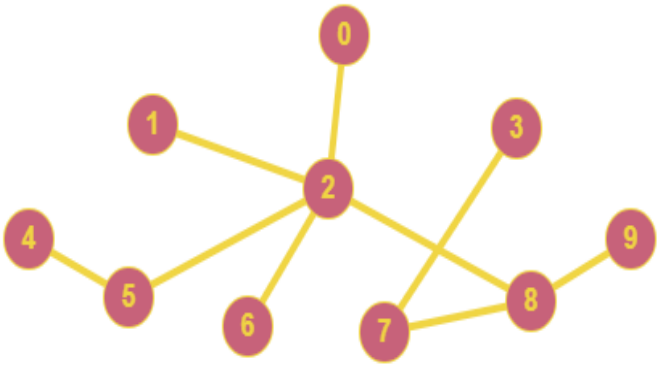
Şekildeki graph main testinde oluşturuldu ve ardından list temsili hali ekrana plot_graph ile bastırıldı. Ardından is_undirected metodu ile graphın undirected olduğu onaylandı. is_acyclic_graph metodu ile graphın cyclic olduğu test edildi

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

Undirected Graph
Cyclic Graph

    ############ BFS SPANNING TREE  ############
0 -> 1
1 -> 0 -> 2 -> 3 -> 4 -> 6
2 -> 1 -> 7 -> 8
3 -> 1 -> 9
4 -> 1 -> 5
5 -> 4
6 -> 1
7 -> 2
8 -> 2
9 -> 3

Acyclic Graph
```



.
Daha sonra BFS kullanılan getSpanningTreeBFS metodu ile üstteki spanning tree elde edilip ekrana basıldı. is_acyclic_graph metodu ile bu spanning tree test edildi ve acyclic olduğu görülüp doğrulandı.

```
    ############ DFS SPANNING TREE  ############
0 -> 2
1 -> 2
2 -> 0 -> 1 -> 5 -> 6 -> 8
3 -> 7
4 -> 5
5 -> 2 -> 4
6 -> 2
7 -> 3 -> 8
8 -> 2 -> 7 -> 9
9 -> 8

Acyclic Graph
```



Aynı işlem DFS kullanılan getSpanningTreeDFS ile yapılarak başka bir spanning tree elde edilip ekrana basıldı. is_acyclic_graph metodu ile bu spanning tree test edildi ve acyclic olduğu görülüp doğrulandı.

Ardından diğer spanning treeler elde edilerek ekrana basıldı.

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

Acyclic Graph

    ############ SPANNING TREE 2 ############
0 -> 1
1 -> 0 -> 2 -> 3 -> 4 -> 5 -> 6
2 -> 1 -> 7 -> 8
3 -> 1 -> 9
4 -> 1
5 -> 1
6 -> 1
7 -> 2
8 -> 2
9 -> 3

Acyclic Graph
```

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

Acyclic Graph

    ############ SPANNING TREE 4 ############
0 -> 3
1 -> 3 -> 4 -> 5 -> 6
2 -> 3
3 -> 0 -> 1 -> 2 -> 7 -> 8 -> 9
4 -> 1
5 -> 1
6 -> 1
7 -> 3
8 -> 3
9 -> 3

Acyclic Graph
```

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

Acyclic Graph

    ############ SPANNING TREE 6 ############
0 -> 1
1 -> 0 -> 3 -> 5
2 -> 5 -> 7 -> 8
3 -> 1 -> 9
4 -> 5
5 -> 1 -> 2 -> 4 -> 6
6 -> 5
7 -> 2
8 -> 2
9 -> 3

Acyclic Graph
```

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

Acyclic Graph

    ############ SPANNING TREE 8 ############
0 -> 2
1 -> 2 -> 4
2 -> 0 -> 1 -> 5 -> 6 -> 7
3 -> 7 -> 9
4 -> 1
5 -> 2
6 -> 2
7 -> 2 -> 3 -> 8
8 -> 7
9 -> 3

Acyclic Graph
```

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

Acyclic Graph

    ############ SPANNING TREE 10 ############
0 -> 3
1 -> 3 -> 4 -> 5 -> 6
2 -> 3
3 -> 0 -> 1 -> 2 -> 7 -> 9
4 -> 1
5 -> 1
6 -> 1
7 -> 3
8 -> 9
9 -> 3 -> 8

Acyclic Graph
```

4 Q4

Bfs ve dfs arama algoritmaları graphlar üzerinde gezinmek için kullanılan algoritmalardır. Bfs graph üzerinde başlangıç düğümünden itibaren level order bir şekilde gezinirken, Dfs başlangıç nodundan itibaren ilk önce alt seviyesinde bulunan komşularının üzerinde gezinme işlemi yapar. Dfs ve bfs nin yaygın olarak kullanım alanları, avantaj ve dezavantajları ise şu şekildedir.

- Graph üzerinde bir nodedan diğer tüm nodelara olan shortest path bulunurken BFS kullanılır.
- Graph üzerinde cycle tespit etme işlemi için DFS kullanılır.
- Spanning treelerin elde edilmesi için BFS veya DFS kullanılabilir.
- BFS queue veri yapısı kullanılıp FIFO temeli benimsenmiştir.
- DFS ile stack veri yapısı kullanılıp LIFO temeli benimsenmiştir.
- BFS, DFS ye göre daha yavaştır ve daha fazla hafızaya ihtiyaç duyar.
- BFS bir alt level nodeda bir başka node aranması durumunda gereksiz olarak aynı level nodelara bakılmasına sebep olabilirken. Aynı levelde bir node aranması durumu olduğunda DFS ile gereksiz yere alt level tüm nodelara bakılması sorunu oluşabiir.
- Bittorent gibi Peer to peer networkler için aynı leveldeki komşu nodelar bulunurken BFS kullanılır.
- BFS Sosyal ağlarda belirli uzaklıktaki kişileri bulurken belirlenen levele kadar arama yapılmasına olanak sağlar.
- Tek bir çözümü olan bir puzzleın çözülmesi probleminde DFS kullanılabilir.
- BFS GPS navigasyon sisteminde komşu olan konumların bulunmasında kullanılır.