Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 7 REPORT

EMIRE KORKMAZ
141044043

Course Assistant: Fatma Nur Esirci

NOT : Kitabın kaynak kodlarından yararlanılmıştır.
NOT 2 : Kitaptaki Dijkstra'nın Algoritması ve Prim'in Algortiması, soruları yaparken hata verdi. Hatanın sebebini çok uzun süre araştırmama rağmen bulamadım.

1 Q1

Bu bölümde bizden yönsüz, ağırlıklı ve çembersel olmayan bir çizge oluşturup bu çizgenin çembersel ve yönlü olup olmadığını kontrol etmemiz ve iki düğüm arasındaki en kısa mesafeyi bulmamız isteniyor.

1.1 Problem Solution Approach

10 düğümlü 20 kenarlı çizge oluşturmamız isteniyor. Bu şekilde rastegele değerlere sahip ağırlıklı bir çizge oluşturuldu. Bu çizgeyi görselleştirmek için plot_graph fonksiyonu yazıldı. Yönlü olup olmadığı ve çembersel olup olmadığı da is_undirected ve is_acyclic fonksiyonları yardımıyla test edilmiştir. En kısa mesafeyi de Dijkstra'nın Algrotitması yardımıyla bulmamız isteniyor. Dijkstra'nın Algoritması kullanılarak en kısa mesafe hesaplanmıştır.

```
public static void shortest_path(ListGraph g, int v1, int v2)
```

Bu fonksiyon bir çizge objesi ve bir başlangıç ve bir bitiş düğümü alarak, çizge içerisinde Dijkstra'nın Algoritması'nı kullanarak en kısa yolu buluyor ve ekrana bulduğu yolu yani düğümleri bastıyor.

```
public static boolean is_acyclic_graph(ListGraph graph)
```

İnternetteki araştırmalarıma göre bir çizgenin çembersel olup olmadığını anlamak için DFS algoritması kullanılıyormuş. Eğer daha önceden ziyaret edilen bir düğüme tekrar gelinirse bu durum bize çizgenin çembersel olduğunu gösteriyormuş. Bu sebeple Bu fonksiyonda DFS algoritmasından yararlandım.

```
public static boolean is_undirected(Graph graph)
```

Çizge oluşturulurken alınan is_directed değişkeninin aldığı değere göre bir değer döndürüyor ve verilen çizgenin yönlü olup olmadığını bize gösteriyor.

```
public static void dijkstrasAlgortihm(Graph graph, int start,
int[] pred, double[] dist)
```

Bu fonksiyon kitaptan alınmıştır. Bir çizge objesi, bir başlangiç düğümü, önceki düğümlerin tutulduğu bir dizi ve düğümler arasındaki mesafenin tutulduğu bir diziyi parametre olarak alıp işlemlerini yapıyor. Sonuç olarak da **pred** ve **dist** dizilerinde gerekli değerler bulunmuş oluyor.

```
public static void plot_graph(ListGraph listGraph)
```

Çizgenin ağırlıklı olup olmadığına göre çizgeyi çiziyor. Eğer ağırlıklıysa düğümlerin arasındaki kenarlara bu ağırlığı da ekleyerek ekrana basıyor. Diğer durumda ağırlık olmadan ekrana basıyor.

1.2 Test Cases

Show that this func results ->

plot_graph
is_undirected
is_acyclic_graph fonksiyonlari icin

```
0 --<1.1>-->1
0 --<7.5>-->17
1 --<9.5>-->2
1 --<3.4>-->3
2 --<8.9>-->7
3 --<3.3>-->7
3 --<7.1>-->14
4 --<0.1>-->7
5 --<0.8>-->0
5 --<1.9>-->14
5 --<9.0>-->18
6 --<2.1>-->10
9 --<5.7>-->8
11 --<2.3>-->12
12 --<2.2>-->14
13 --<4.1>-->19
15 --<3.0>-->9
15 --<8.3>-->18
16 --<6.3>-->11
19 --<7.7>-->14

Is this graph undirected : false
Is this graph acyclic : false
```

2 Q2

Bu bölümde bizden yönsüz, ağırlıksız ve çembersel olmayan bir çizge oluşturup bu çizgenin çembersel ve yönlü olup olmadığını kontrol etmemiz ve bu çizgenin herhangi 2 düğümünün bağlı olup olmadığını kontrol etmemiz isteniyor.

2.1 Problem Solution Approach

15 düğümlü 30 kenarlı çizge oluşturmamız isteniyor. Bu şekilde rastegele değerlere sahip ağırlıksız bir çizge oluşturuldu. Bu çizgeyi görselleştirmek için plot_graph fonksiyonu yazıldı. Yönlü olup olmadğı ve çembersel olup olmadığı da is_undirected ve is_acyclic fonksiyonları yardımıyla test edilmiştir. İki düğümün bağlı olup olmadığını kontrol etmek için is_connected fonksiyonu yazılmıştır.

```
public static boolean is_connected(ListGraph g, int v1, int v2)
```

Bu fonksiyon shortest_path fonlsiyonunu kullanarak iki düğümün bağlantılı olup olmadığını kotrol ediyor. Dijkstra'nın Algoritması'nda başlangıç değeri olarak mesafelere sonsuz değeri verilir. Bağlantılı olduğu görülen düğümlere yeni mesafe yazılır. Bağlantılı olmayan düğümlerde ise bu mesafe değişmeyeceği için sonsuzdur. Bunu kullanarak iki düğümün bağlantılı olup olmadığını kontrol ettim.

```
public static void shortest_path(ListGraph g, int v1, int v2)
```

Bu fonksiyon bir çizge objesi ve bir başlangıç ve bir bitiş düğümü alarak, çizge içerisinde Dijkstra'nın Algoritması'nı kullanarak en kısa yolu buluyor ve bu yolun uzunluğunu döndürüyor.

```
public static boolean is_acyclic_graph(ListGraph graph)
```

İnternetteki araştırmalarıma göre bir çizgenin çembersel olup olmadığını anlamak için DFS algoritması kullanılıyormuş. Eğer daha önceden ziyaret edilen bir düğüme tekrar gelinirse bu durum bize çizgenin çembersel olduğunu gösteriyormuş. Bu sebeple Bu fonksiyonda DFS algoritmasından yararlandım.

```
public static boolean is_undirected(Graph graph)
```

Çizge oluşturulurken alınan is_directed değişkeninin aldığı değere göre bir değer döndürüyor ve verilen çizgenin yönlü olup olmadığını bize gösteriyor.

```
public static void dijkstrasAlgortihm(Graph graph, int start,
int[] pred, double[] dist)
```

Bu fonksiyon kitaptan alınmıştır. Bir çizge objesi, bir başlangiç düğümü, önceki düğümlerin tutulduğu bir dizi ve düğümler arasındaki mesafenin tutulduğu bir diziyi parametre olarak alıp işlemlerini yapıyor. Sonuç olarak da **pred** ve **dist** dizilerinde gerekli değerler bulunmuş oluyor.

```
public static void plot_graph(ListGraph listGraph)
```

Çizgenin ağırlıklı olup olmadığına göre çizgeyi çiziyor. Eğer ağırlıklıysa düğümlerin arasındaki kenarlara bu ağırlığı da ekleyerek ekrana basıyor. Diğer durumda ağırlık olmadan ekrana basıyor.

2.2 Test Cases

Show that this func results ->

- plot_graph
- is_undirected
- is_acyclic_graph
- is_connected function (use least 3 different label pair)

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

Is this graph acyclic : false
Is this graph undirected : true
Given vertices are not valid.
```

3 Q3

Bu bölümde bizden yönsüz ve ağırlıksız bir çizge oluşturup bu çizgenin çembersel ve yönlü olup olmadığını kontrol etmemiz, DFS ve BFS uygulayıp daha sonra kapsayan ağaçları ekrana bastırmamız isteniyor.

3.1 Problem Solution Approach

10 düğümlü 20 kenarlı çizge oluşturmamız isteniyor. Bu şekilde rastegele değerlere sahip ağırlıksız bir çizge oluşturuldu. Bu çizgeyi görselleştirmek için plot_graph fonksiyonu yazıldı. Yönlü olup olmadığı ve çembersel olup olmadığı da is_undirected ve is_acyclic fonksiyonları yardımıyla test edilmiştir. DepthFirstSearch, breadthFirstSearch, primsAlgorithm fonksiyonları yazılmıştır.

```
public static int[] breadthFirstSearch(Graph graph, int start)
```

Bu fonksiyon kitaptan yardım alınarak yazılmıştır. Bir düğümden her bir düğüme olan en kısa yolları bulur.

```
public static ArrayList < Edge > primsAlgorithm(Graph graph,int
start)
```

Bu fonksiyonu kitaptan yardım alarak yazıldı. Kapsayan ağaç bir çizgede bulunan bütün düğümlerin sadece tek bir kenarla bağlantılı olduğu bir alt kümedir. Bu algoritma Dijkstra'nın Algoritmasına çok benzer bir algoritmadır. Kapsayan ağacı döndürür.

```
public DepthFirstSearch(Graph graph)
```

DepthFirstSearch Sınıf'ından bir obje oluştururken kullanılıp bir çizge objesi alıp içerisinde depthFirstSearch fonskiyonunu çağırır.

```
public void depthFirstSearch(int current)
```

Bu fonskiyon, başlangıç düğümünün bir kenarından başlayıp o kenar üzerinden gidilebilecek en uzak (derin) düğüme kadar sürdürür.

3.2 Test Cases

plot_graph
is_undirected
is_acyclic fonksiyonlari icin

```
3 -->1
3 -->14
4 -->7
5 -->0
5 -->14
5 -->18
6 -->10
7 -->2
7 -->3
7 -->4
8 -->9
9 -->8
9 -->15
10 -->6
11 -->1
11 -->12
11 -->16
12 -->11
12 -->14
13 -->19
14 -->5
14 -->12
14 -->3
14 -->19
15 -->9
15 -->18
16 -->11
17 -->0
18 -->5
18 -->15
19 -->13
19 -->14

Is this graph undirected : true
Is this graph acyclic : false
```

4 Q4

Daha net çıkması için bütün olarak kağıdı çekmek yerine farklı yerleri çekildi. Ödev tek bir sayfaya yazılmıştır.

4) Çizge üzerinde dolaşma çizgenin düğümleri ve kenarları üzerinde istenen bir işi yapacak veya bir problemi çözecek biçimde hareket etmektir.

Çizge üzerinde dolaşma yapan birçok yaklaşım vardır; en önemli iki türevi, kısaca, DFS (Depth First Search, derinlemesine arama) ve BFS (Breadth First Search, genişlemesine arama) olarak adlandırılmıştır. ve çizge üzerinde geliştirilen algoritmaların birçoğu bu yaklaşım yöntemlerine dayanmaktadır denilebilir.

DFS yöntemi

DFS, çizge üzerinde başlangıç düğümünün bir kenarından başlayıp o kenar üzerinden gidilebilecek en uzak (derin) düğüme kadar sürdürülür. İşlem adımları şu şekildedir; önce bir başlangıç node'u seçilir ve ziyaret edilir. Seçilen node'un bir komşusu seçilir ve ziyaret edilir. 2. adım ziyaret edilecek bir komşu kalmayıncaya kadar tekrar edilir. Komşu kalmadığında tekrar geri dönülür ve önceki ziyaret edilmiş node'lar için adım 2 ve 3 tekrar edilir.

BFS yöntemi

BFS yönteminin, DFS yönteminden farkı, dolaşmaya, başlangıç düğümünün bir kenarı ayrıtı üzerinden en uzağa gidilmesiyle değil de, başlangıç düğümünden gidilebilecek tüm komşu düğümlere gidilmesiyle başlanır. BFS yöntemi, diğer çizge üzerinde dolaşma yapan algoritmalara esin kaynağı olmuştur. Ağırlıksız çizgelerde en kısa yol algoritması olarak kullanılabilir. İşlem adımları şu şekildedir; seçilen düğümün tüm komşuları ziyaret edilir. Her komşu kuyruk yapısı içerisine alınır. Komşu kalmadığında kuyruk içerisindeki ilk düğüm alınır ve ilk adıma gidilir.

BFS Kullanım alanları → GPS Navigasyonlarda, Peer to Peer ağlarında, sosyal medya platformlarında

DFS → Labirent çözümünde, topolojik sıralamada, internet taramalarında kullanılır.

* BFS, DFS'den daha yavaştır. * BFS, DFS'den daha çok hafıza kullanır.

BFS

DFS

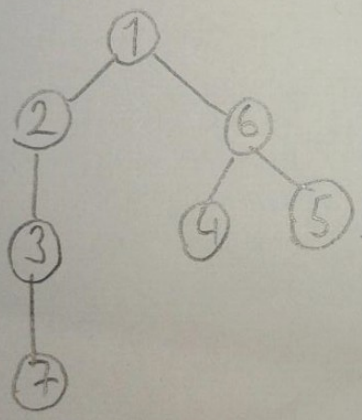
alını ve ilk adına gidilir.

BFS Kullanım alanları ⟹ GPS Navigasyonlarda, Peer to Peer ağlarında, sosyal medya platformlarında

DFS → Labirent çözümünde, topolojik sıralamada, internet taramalarında kullanılır.

* BFS, DFS'den daha yavaştır. * BFS, DFS'den daha çok hafıza kullanır

BFS



DFS