

Gebze Technical University
Computer Engineering

CSE 222
2017 Spring

HOMEWORK 9 REPORT

STUDENT NAME : Muhammet Bedirhan Çağlar
STUDENT NUMBER : 141044073

Course Assistant: Ahmet Soyyiğit

~Problem solutions approach

=> **addRandomEdgesToGraph fonksiyonu için** girilen edgelimet max olacak şekilde 0 dan edgelimete kadar bir random sayısı üretilir ve sonra edgeler için üretilen sayısı kadar döngü döner ve edge var mı conditionı kontrol edilerek yoksa insert gerçekleşir.

=>**breadthFirstSearch fonksiyonu için** girilen başlangıç vortex i ile tüm vortexlere ulaşılma path i bir int array ile return edilir.[Kitaptaki static olan fonksiyonu çevirerek yaptım]

=>**isBipartiteUndirectedGraph fonksiyonu** graph objesinin Bipartite olup olmadığını kontrol ederek boolean deger return eder.Bir Hashmap yardımı ile key ve valueları Beyaz, Kırmızı şeklinde boyayarak çakışıp çakışmadıkları kontrol edilir.

=>**getConnectedComponentUndirectedGraph fonksiyonu için** graphlar,vartex degerleri ve edgeler için arraylist tutarak depth_first_search yardımı ile önce edgeleri doldurarak max vartex sayısına göre graph arrayliste yeni graph objesi oluşturuyoruz. depth_first_search fonksiyonun temel olarak visited arrayini doldurarak,o fonksiyondan faydalanarak tüm farklı graphleri bularak arraylistten arraye aktararak return ediyoruz.

~ Tests ~

1-) addRandomEdgesToGraph fonksiyonu için :

=> edgeLimete 3 ve 10 sayıları vererek üretilen ve eklenen edgeler test edilmiştir.

2-) breadthFirstSearch fonksiyonu için :

=> 3 farklı graph txt den okunarak test edilmiştir.

3-) isBipartiteUndirectedGraph fonksiyonu için :

=> 2 farklı graphta caselerin true ve false verdiği durumlar test edilmiştir.

4-) getConnectedComponentUndirectedGraph fonksiyonu için :

=> 3 farklı txt den 1 , 2 ve 3 graph olacak şekilde edgeler girilerek test edilmiştir.

5-) writeGraphToFile fonksiyonu için :

=> yukarda ki test caselerin txt diye yazılımı test edilmiştir.

Test1~

```
Output - hw9 (run) X AbstractGraphExtended.java X Main.java X graph_1.txt X AbstractGraph.java X graph_1_m1.txt X ListGraph.java X

run:
-----TEST (addRandomEdgesToGraph) Function-----
-----Random üretilen edgeler-----
(1) 0 4
Grapha eklenen edge sayi---> 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
Output - hw9 (run) X AbstractGraphExtended.java X Main.java X graph_1.txt X AbstractGraph.java X graph_1_m1.txt X ListGraph.java X

run:
-----TEST (addRandomEdgesToGraph) Function-----
-----Random üretilen edgeler-----
(1) 1 7
(2) 1 2
(3) 7 2
(4) 8 1
(5) 6 1
(6) 3 1
(7) 4 8
(8) 3 5
Grapha eklenen edge sayi---> 7
BUILD SUCCESSFUL (total time: 0 seconds)
```

Test2~

```
Output - hw9 (run) X AbstractGraphExtended.java X Main.java X graph_1.txt X AbstractGraph.java X graph_1_m1.txt X ListGraph.java X
run:
-----TEST (breadthFirstSearch) Function-----
0->-1
1->0
2->1
3->0
4->1
5->1
6->3
7->4
8->6
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
Output - hw9 (run) X AbstractGraphExtended.java X Main.java X graph_1.txt X AbstractGraph.java X graph_1_m1.txt X ListGraph.java X
run:
-----TEST (breadthFirstSearch) Function-----
0->-1
1->0
2->1
3->0
4->1
5->1
6->3
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
Output - hw9 (run) X AbstractGraphExtended.java X Main.java X graph_1.txt X AbstractGraph.java X graph_1_m1.txt X ListGraph.java X
run:
-----TEST (breadthFirstSearch) Function-----
0->-1
1->0
2->1
3->0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Test3~

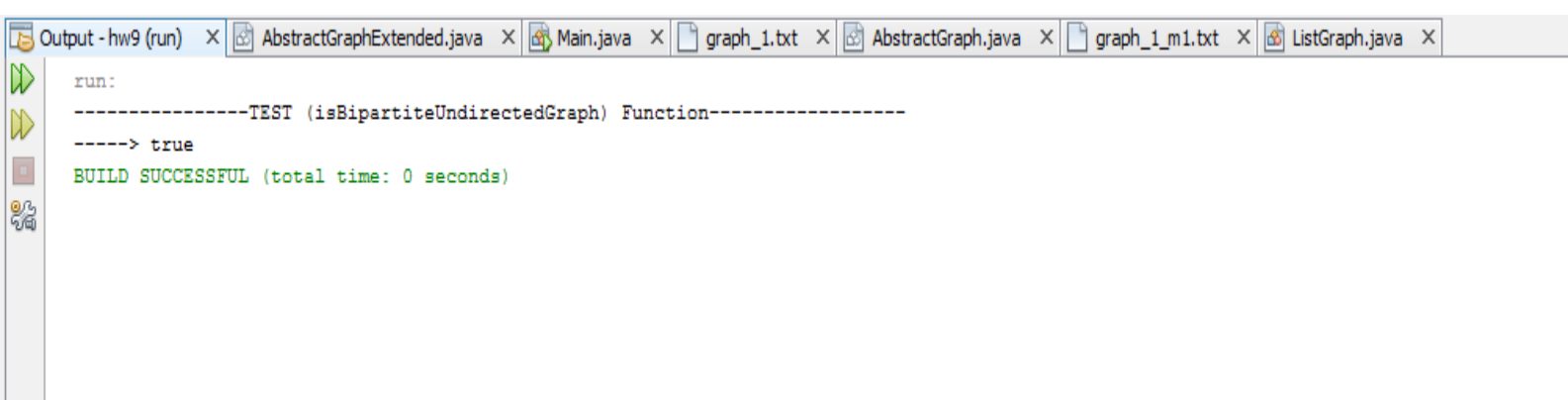
=> txde deki edgeler :

4

0 1

0 2

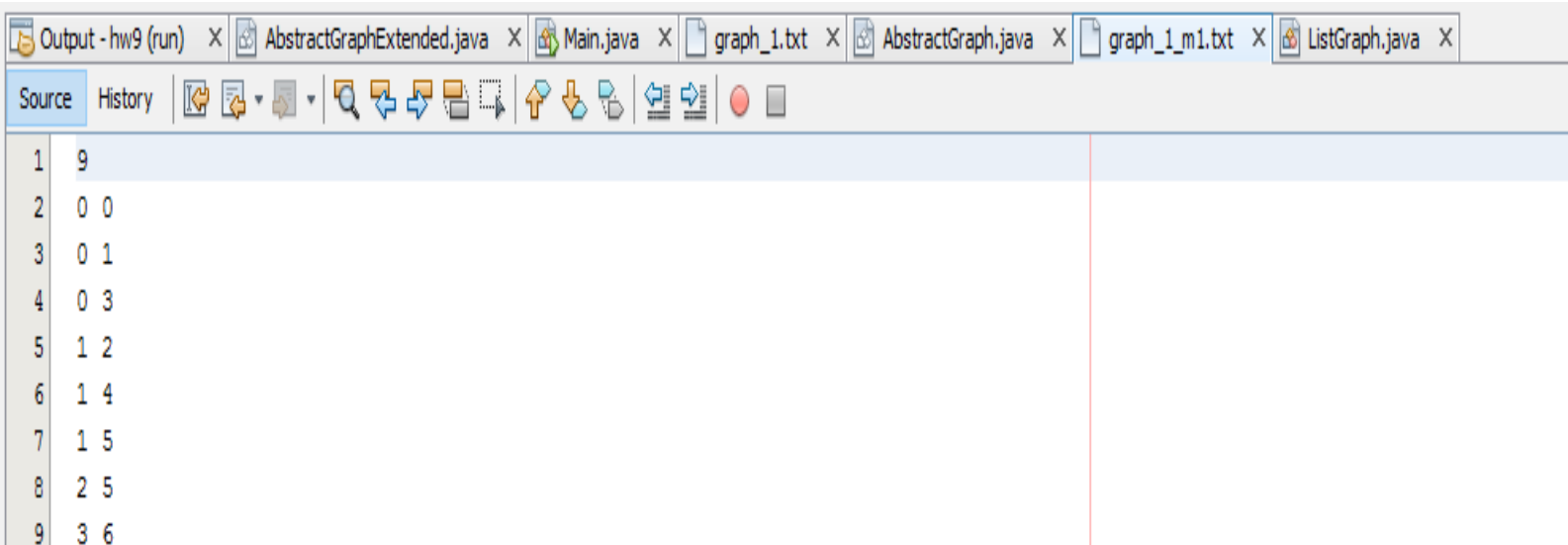
0 3



Output - hw9 (run) X AbstractGraphExtended.java X Main.java X graph_1.txt X AbstractGraph.java X graph_1_m1.txt X ListGraph.java X

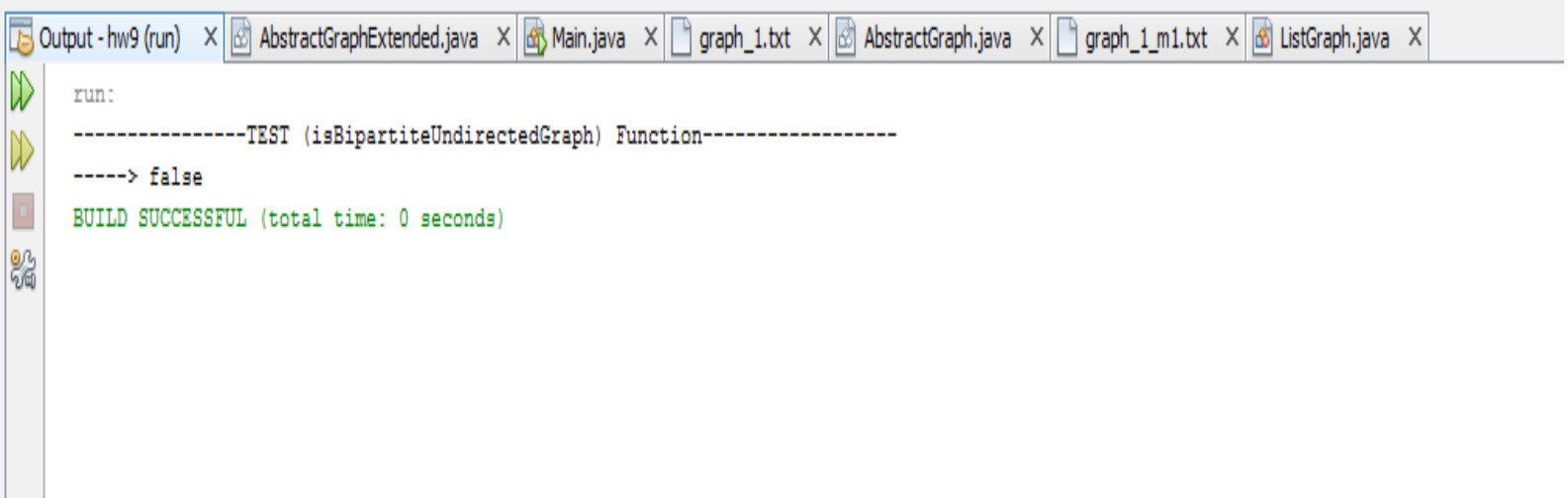
```
run:
-----TEST (isBipartiteUndirectedGraph) Function-----
-----> true
BUILD SUCCESSFUL (total time: 0 seconds)
```

=>txtdeki edgeler :



Source History | [Icons]

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



Output - hw9 (run) X AbstractGraphExtended.java X Main.java X graph_1.txt X AbstractGraph.java X graph_1_m1.txt X ListGraph.java X

```
run:
-----TEST (isBipartiteUndirectedGraph) Function-----
-----> false
BUILD SUCCESSFUL (total time: 0 seconds)
```

Test4~

NOT=> Txde ki edgeler sırasıyla aşağıdaki gibidir:

1-)

9
0 1
0 3
1 2
1 4
1 5
2 5
3 6
4 6
4 7
5 7
6 8
7 8

2-)

13
0 1
0 3
1 2
1 4
1 5
2 5
3 6
4 6
4 7
5 7
6 8
7 8
9 10
9 11
11 12

3-)

9
0 1
0 3
1 2
1 4
1 5
2 5
3 6
4 6
4 7
5 7
6 8
7 8
9 10
9 11
11 12
14 15
14 16
17 16


```
Output - hw9 (run) X AbstractGraphExtended.java X Main.java X graph_1.txt X AbstractGraph.java X graph_1_m1.txt X ListGraph.java X
run:
-----TEST (getConnectedComponentUndirectedGraph) Function-----
-----Graph (1)-----
0 1
0 3
1 0
1 2
1 4
1 5
2 1
2 5
3 0
3 6
4 1
4 6
4 7
5 1
5 2
5 7
6 3
6 4
6 8
7 4
7 5
7 8
8 6
8 7
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
Output - hw9 (run) X AbstractGraphExtended.java X Main.java X graph_1.txt X AbstractGraph.java X graph_1_m1.txt X ListGraph.java X
run:
-----TEST (getConnectedComponentUndirectedGraph) Function-----
-----Graph (1)-----
0 1
0 3
1 0
1 2
1 4
1 5
2 1
2 5
3 0
3 6
4 1
4 6
4 7
5 1
5 2
5 7
6 3
6 4
6 8
7 4
7 5
7 8
8 6
8 7
-----Graph (2)-----
0 1
0 2
1 0
2 0
2 3
3 2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Output - hw9 (run) X AbstractGraphExtended.java X Main.java X graph_1.txt X AbstractGraph.java X graph_1_m1.txt X ListGraph.java X

1 2
1 4
1 5
2 1
2 5
3 0
3 6
4 1
4 6
4 7
5 1
5 2
5 7
6 3
6 4
6 8
7 4
7 5
7 8
8 6
8 7

-----Graph (2)-----

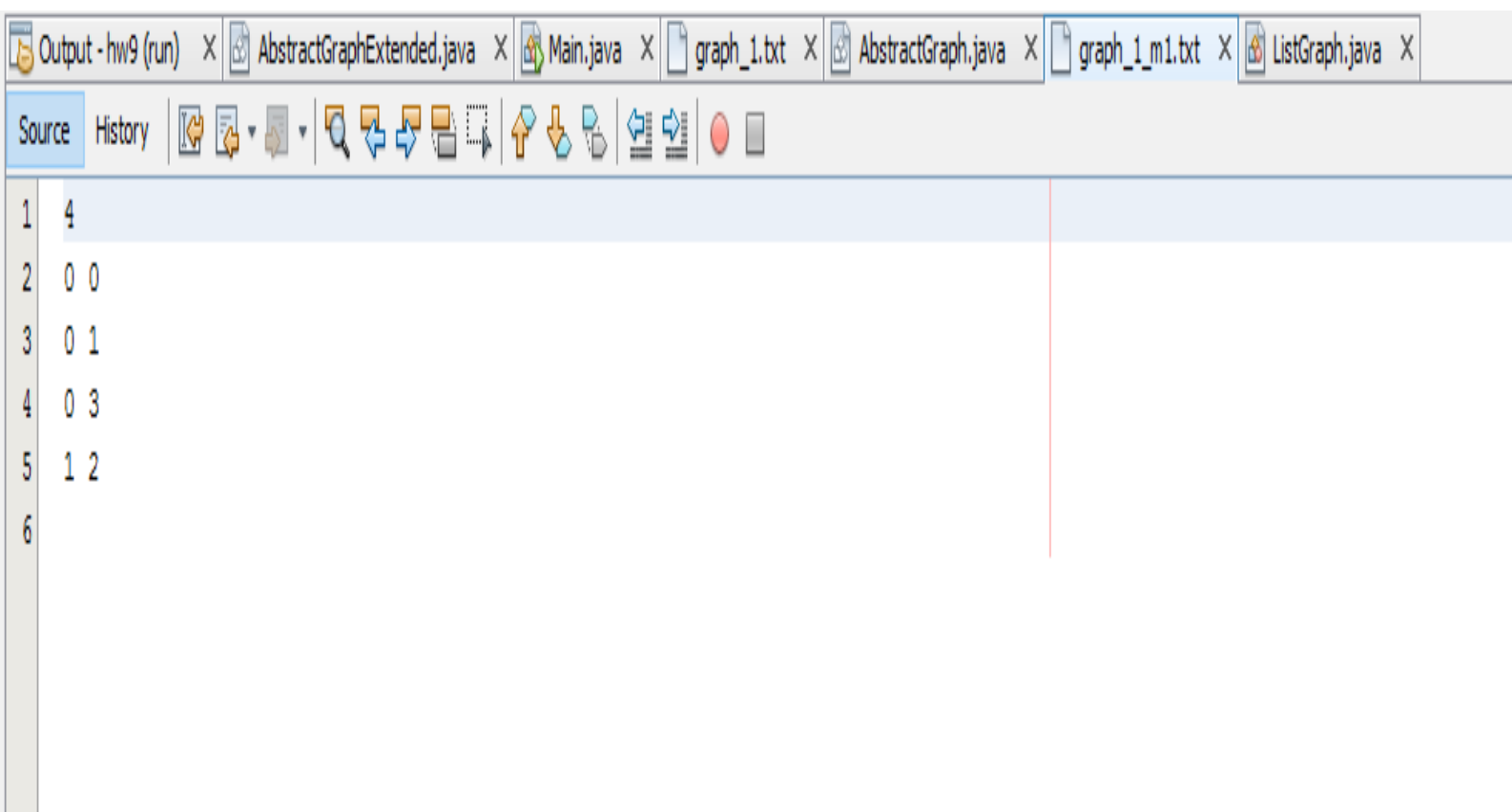
0 1
0 2
1 0
2 0
2 3
3 2

-----Graph (3)-----

0 1
0 2
1 0
2 0
2 3
3 2

BUILD SUCCESSFUL (total time: 0 seconds)

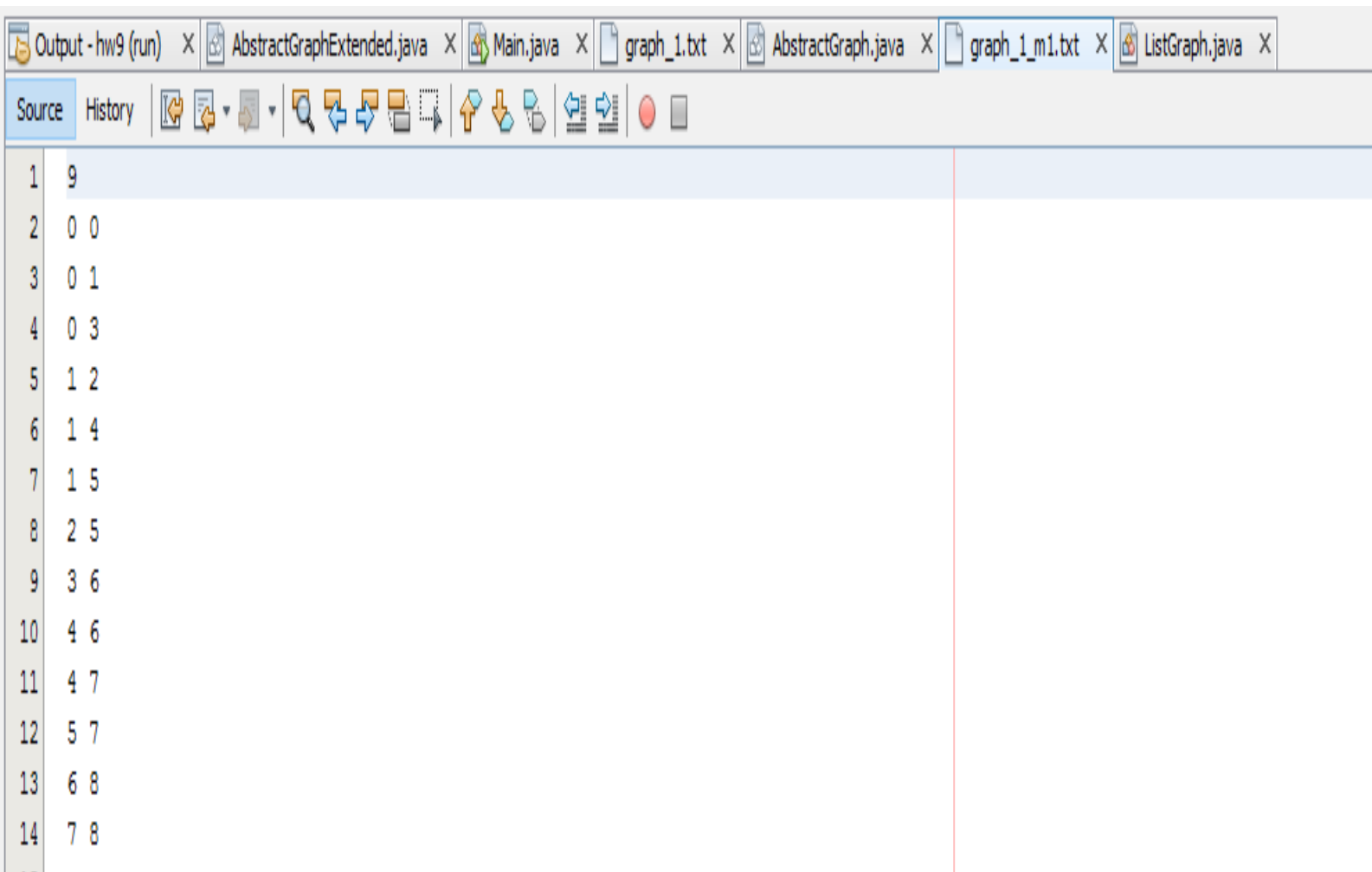
Test5



The screenshot shows an IDE window with the following tabs: Output - hw9 (run), AbstractGraphExtended.java, Main.java, graph_1.txt, AbstractGraph.java, graph_1_m1.txt, and ListGraph.java. The 'Source' tab is active, displaying the following code:

```
1 4
2 0 0
3 0 1
4 0 3
5 1 2
6
```

The code represents a graph with 6 nodes and 5 edges. The edges are: (1, 4), (2, 0), (3, 0), (4, 0), and (5, 1). The graph is visualized as a set of nodes and edges, with a red vertical line indicating the current position of the cursor.



The screenshot shows the same IDE window as above, but with the 'graph_1_m1.txt' tab selected. The code represents a graph with 14 nodes and 13 edges. The edges are: (1, 9), (2, 0), (3, 0), (4, 0), (5, 1), (6, 1), (7, 1), (8, 2), (9, 3), (10, 4), (11, 4), (12, 5), (13, 6), and (14, 7). The graph is visualized as a set of nodes and edges, with a red vertical line indicating the current position of the cursor.