Middle East Technical University | Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring '2012-2013

Programming Assignment 3

Due date: 17 April 2013, Wednesday, 23:55

1 Introduction

The year is 2014, and finally Omer and I hacked the databases containing financial relationships all over the world. We don't know how but a government top secret facility found this hack and sent under cover agents to gather our big secret. Unfortunately, we are more than acute for these situations and hid everything not comprimising anything. Afterwards, they offered us jobs of being under cover agents working for them. They wanted us to analyse the graph in so many ways which we do not have the time to implement, and you know how governments are strict when the topic is about these. Hence, we needed help to alleviate our load on this situation, which we thought who else but only brilliant CENG students can help us.

We now ask you to implement two of the analysis of the social network which in this particular case financial network we gathered, in Haskell.

2 Specifications

The Network: The network we hacked is an acyclic directed weighted graph, where the nodes represent the business entities, the directions represent the "business demand" and the weight represent the profit. The details about what you need to do is given below.

1. **Graph Data Type (Network.hs) :** Use the following data type declaration

```
data Node a = Node
{ label ::  a,
adjacent ::  [(a,Int)] } deriving Show

data Network a = Graph [Node a] deriving Show
```

   - **Example :** We have 2 nodes labeled 0 and 1. The tuples in the inner list represent the edges. The neighbor node and the weight of that edge respectively.

```
Graph [ ( Node 0 [ ( 1 , 2 ) ] ) , ( Node 1 [] ) ]
```

2. **Elementary Operations: (Operations.hs)** In this task you are asked to implement some basic operations on given graphs.

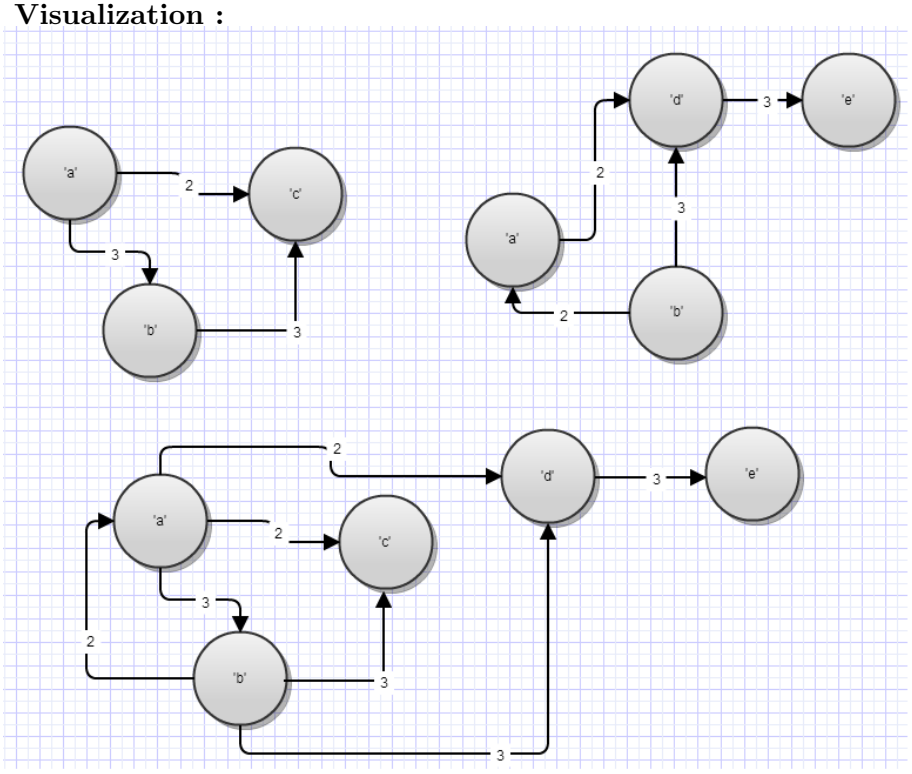
   (a) **Merge Function:** In this subtask you need to merge two given graphs according to the common nodes, if no common nodes exist then you should return only the first graph. Also for the sake of simplicity, there won't be any common edges between two graphs no matter what.

```
merge ::  ( Network a ) -> ( Network a ) -> ( Network a )
```

   - **Example :** (see figure below for the networks)
     `merge ( Graph [ ( Node 'a' [ ( 'b' , 3 ) , ( 'c' ,2 ) ] ) , ( Node 'b' [ ( 'c' , 3 ) ] ) , ( Node 'c' [] ) ] ) ( Graph [ ( Node 'a' [ ( 'd' , 2 ) ] ) , ( Node 'b' [ ( 'd' , 3 ) , ( 'a' , 2 ) ] ) , ( Node 'd' [ ( 'e' , 3 ) ] ) , ( Node 'e' [] ) ] )`

   - **Output :**
     `Graph [ ( Node 'a' [ ( 'b' , 3 ) , ( 'c' , 2 ) , ( 'd' , 2 ) ] ) , ( Node 'b' [ ( 'c' , 3 ) , ( 'd' , 3 ) , ( 'a' , 2 ) ] ) , ( Node 'c' [] ) ] , ( Node 'd' [ ( 'e' , 3 ) ] ) , ( Node 'e' [] ) ]`

   - **Visualization :**



(b) **Equality Operator :** In this subtask you need to compare the given graphs and return True if the graphs are equal, return False otherwise. To state that two graphs are equal, every label and every edge should be matched. For the following operator, define a new class called NetworkOp, and give instance of this operator for Network type. You should check for **Graph Equality** , meaning nodes and edges may be given in different order. Note that, the edge (0,1) in the first graph is not the same edge (1,0) in the second graph.

- **Example :** (see figure above for the networks)

```
Graph [ ( Node 'a' [ ( 'b' , 3 ) , ( 'c' ,2 ) ] ) , ( Node 'b' [ ( 'c' , 3
) ] ) , ( Node 'c' [] ) ] == Graph [ ( Node 'a' [ ( 'd' , 2 ) ] ) , ( Node
'b' [ ( 'd' , 3 ) , ( 'a' , 2 ) ] ) , ( Node 'd' [ ( 'e' , 3 ) ] ) , ( Node
'e' [] ) ]
False
```

- **Example : (see figure above for the networks)**

```
Graph [ ( Node 'a' [ ( 'b' , 3 ) , ( 'c' ,2 ) ] ) , ( Node 'b' [ ( 'c' , 3
) ] ) , ( Node 'c' [] ) ] == Graph [ ( Node 'a' [ ( 'b' , 3 ) , ( 'c' ,2 )
] ) , ( Node 'b' [ ( 'c' , 3 ) ] ) , ( Node 'c' [] ) ]
True
```

3. **Finding Frauds (AnalyzeNetwork.hs) : (25% Bonus Part)** In this task it is asked to find the frauds by giving patterns defined with subgraphs in this network. The function has the following type declaration

```
fraud ::  (Network a) -> (Network b) -> b ->[a]
```

It should take a main graph, a subgraph to be searched and a pivot vertex in the subgraph where it should return a list of vertices matching the pattern, i.e. subgraph, according to he pivot. The subgraph is a directed graph which will have edge weights only 0, stating that the matching will be regardless of the weights. When you are traversing the network vertex by vertex , searching for subgraphs, the pivot should match the current vertex, and according to that vertex you should search for the subgraph which will highly ease your burden.
Important restriction: the subgraph node count will **not exceed 6**.

- **Example : (see figure above for the networks)** `fraud`

```
( Graph [ ( Node 'a' [ ( 'b' , 3 ) , ( 'c' ,2 ) ] ) , ( Node 'b' [ ( 'c' , 3 )
] ) , ( Node 'c' [] ) ] ) , ( Graph [ ( Node 0 [ (1,0) ] ) , ( Node 1 [] ) ] )
0
```

- **Output :**

```
[ 'a' ]
```

- **Example : (see figure above for the networks)**

```
fraud ( Graph [ ( Node 'a' [ ( 'b' , 3 ) , ( 'c' , 2 ) , ( 'd' , 2 ) ] ) , ( Node
'b' [ ( 'c' , 3 ) , ( 'd' , 3 ) , ( 'a' , 2 ) ] ) , ( Node 'c' [] ) ] , ( Node
'd' [ ( 'e' , 3 ) ] ) , ( Node 'e' [] ) ] ) ( Graph [ ( Node 'a' [ ('b',0) ] )
, ( Node 'b' [] ) ] ) 'a'
```

- **Output :**

```
[ 'a' , 'b' , 'd' ]
```

3 Regulations

1. **Programming Language: Haskell**
2. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations.
3. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.
4. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications.
5. **Dependencies:**

    - Network.hs Network(type), Node(type)
    - Operations ¡¿ merge(function), (==)(operator)
    - AnalyseNetwork ¡¿ fraud(function)
    - The dependencies should be the following:
    - AnalyseNetwork ¡¿ import All
    - Operations ¡¿ import Network

4 Submission

Submission will be done via COW. Create a tar.gz file named hw3.tar.gz that contains all your source code files (Network.hs , Operations.hs , AnalyzeNetwork.hs and your own modules if you wish to write more).
Note: The submitted archive should not contain any directories! The following command sequence is expected to run your program on a Linux system:

```
$ tar -xf hw3.tar.gz
$ hugs AnalyzeNetwork.hs
```