CENG 242

Homework #1

(Due: March 18th 2011, Friday 23:55)

In this homework, you will be given a list of simple paths (i.e. paths with no repeated vertices) where each vertex is labeled and has an associated value. Your aim is to construct a graph such that an edge between two vertices exists if these vertices are adjacent in one of the paths. The values of the vertices should be equal to sum of the values of vertices having the same label in the list of paths.

For the sake of simplicity, it will be assumed that the resulting graph will always be a binary tree where a vertex 'u' is an ancestor of another vertex 'v' only if 'u' precedes 'v' in every path in which 'v' is present.

Assume that Path and Tree types are defined as:

```
data Path = Nil | Node Char Integer Path
data Tree = Empty | Leaf Char Integer
                  | Branch Char Integer Tree Tree
```

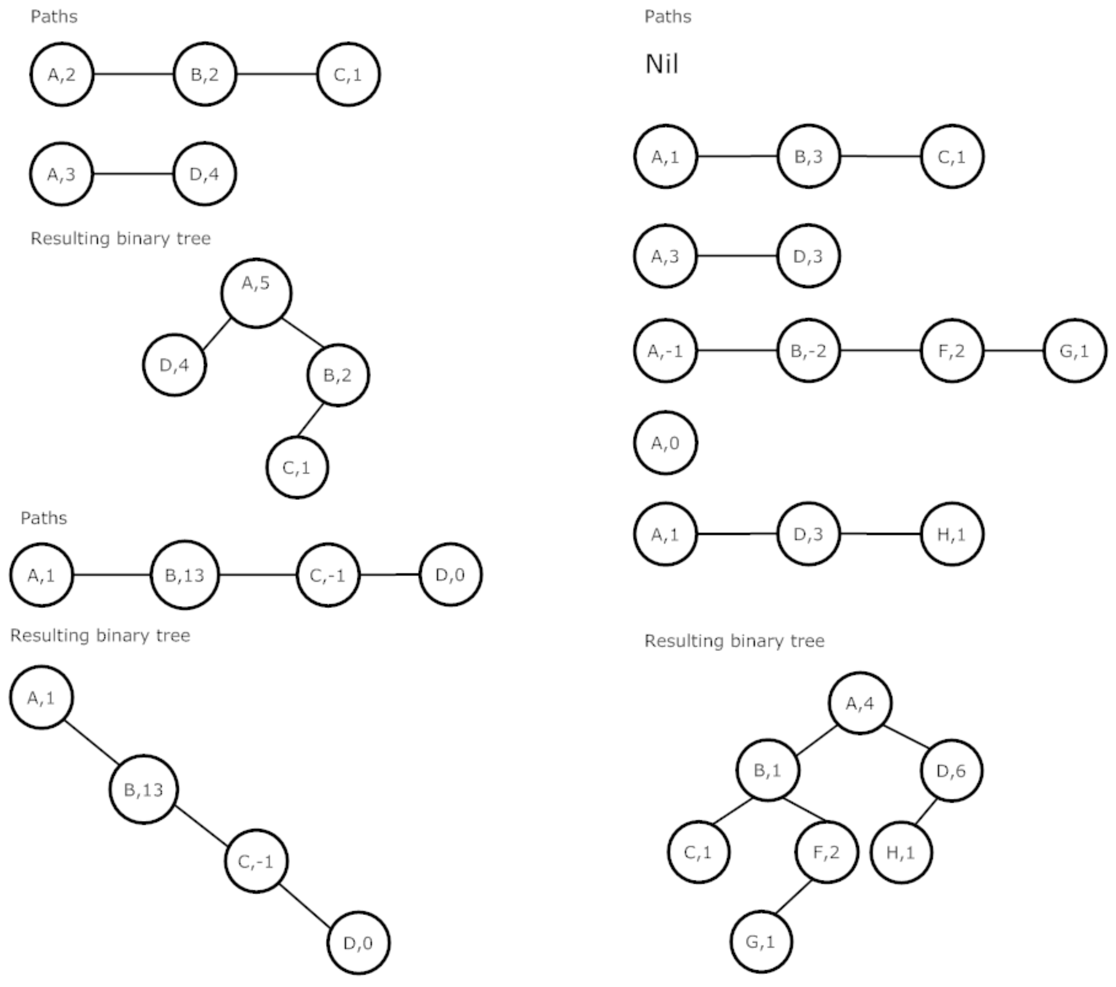


Figure 1

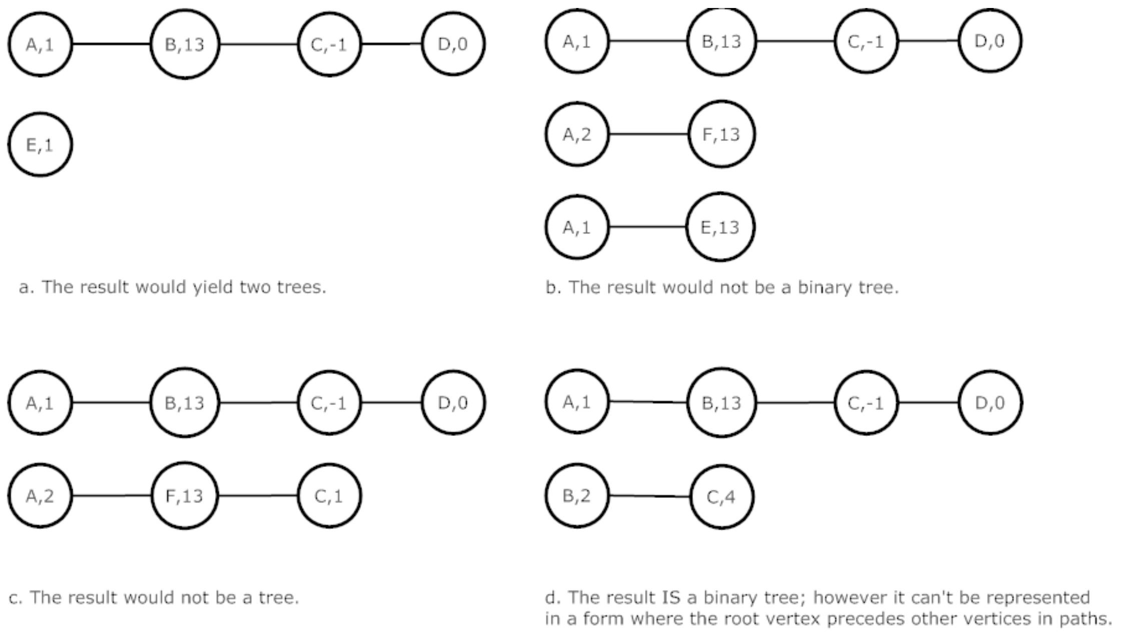
The character in a path node denotes the label of the vertex, whereas the integer denotes the value of the vertex. Same idea applies to the tree definition.

You will write a function **hw1** that will return the resulting tree, in the form:

```
hw1 :: [Path] -> Tree
```

Notes

- The assumption on the resulting graph (i.e. that it will be a binary tree) has important consequences. As an example, every non-empty path in the list will have the same root vertex. You should make use of these implications in your implementation. Figure.2 shows sample inputs that will NOT be tested.
- You should note that the tree structure is redundant. For the output, your function can produce any possible tree representation for a tree.



A,1
B,13
C,-1
D,0
E,1
a. The result would yield two trees.
A,1
B,13
C,-1
D,0
A,2
F,13
A,1
E,13
b. The result would not be a binary tree.
A,1
B,13
C,-1
D,0
A,2
F,13
C,1
c. The result would not be a tree.
A,1
B,13
C,-1
D,0
B,2
C,4
d. The result IS a binary tree; however it can't be represented in a form where the root vertex precedes other vertices in paths.

Figure 2

Examples

```
hw1 [(Node 'A' 2 (Node 'B' 2 (Node 'C' 1 Nil))), (Node
'A' 3 (Node 'D' 4 Nil))] -check Figure.1

>>> (Branch 'A' 5 (Branch 'D' 4 Empty Empty) (Branch 'B'
2 (Leaf 'C' 1) Empty)) or

>>> (Branch 'A' 5 (Branch 'B' 2 (Branch 'C' 1 Empty
Empty) Empty) (Branch 'D' 4 Empty Empty)) or any other
equivalent representation

hw1 [(Node 'A' 1 (Node 'B' 13 (Node 'C' (-1) (Node 'D' 0
Nil))))] -check Figure.1

>>>(Branch 'A' 1 (Branch 'B' 13 (Branch 'C' -1 (Leaf 'D'
0) Empty) Empty) Empty) or any other equivalent
representation.
```

```
hw1 [Nil, (Node 'A' 1 (Node 'B' 3 (Node 'C' 1 Nil))),
(Node 'A' 3 (Node 'D' 3 Nil)), (Node 'A' (-1) (Node 'B'
(-2) (Node 'F' 2 (Node 'G' 1 Nil)))), Node 'A' 0 Nil,
(Node 'A' 1 (Node 'D' 3 (Node 'H' 1 Nil)))] -check
Figure.1

>>> (Branch 'A' 4 (Branch 'B' 1 (Leaf 'C' 1) (Branch 'F'
2 (Leaf 'G' 1) Empty)) (Branch 'D' 6 (Leaf 'H' 1) Empty))
or any other equivalent representation
```

Specifications:

- All work should be done individually.
- Your codes should be written in Haskell and have the name “hw1.hs”. You should create a module named Hw1 and implement hw1 function with the specified type.
- For the inputs that won’t be tested, the behavior of your program is unspecified (you are free in your implementation for such cases, you do not have to report error in any way).
- In evaluation, black box method will be used. So be careful about the name of functions, data structures etc.
- You will submit your code through Cow system.
- You should test your codes in inek machines with hugs before submitting.