The goal of this assignment is to implement a *floor cut tree* that represents a cut floorplan. A *cut floorplan* divides a rectangle with horizontal and vertical *cuts*. A cut floorplan can be represented by a binary tree, called a *floor cut tree*, whose internal nodes represent the cuts, and whose external nodes represent the basic rectangles into which the floorplan is decomposed by the cuts. (See the figure below.)

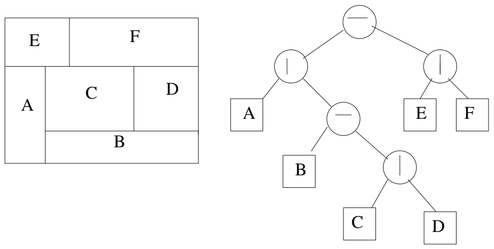

Figure 1: Cut Floorplan and corresponding Floor Cut Tree

The *compaction problem* for a cut floorplan is defined as follows. Assume that each basic rectangle of a cut floorplan is assigned a minimum width w and minimum height h. The compaction problem is to find the smallest possible height and width for each rectangle of the cut floorplan that is compatible with the minimum dimensions of the basic rectangles. Namely this problem requires assignment of values $h(v)$ and $w(v)$ o each node of the floor cut tree such that

$$w(v) = \begin{cases} w & \text{if v is an external node whose basic rectangle has minimum weight w.} \\ max(w(w), w(z)) & \text{if v is an internal node associated with a horizontal cut with left} \\ & \text{child w and right child z.} \\ w(w) + w(z) & \text{if v is an internal node associated with a vertical cut with left} \\ & \text{child w and right child z.} \end{cases} \tag{1}$$

$$h(v) = \begin{cases} h & \text{if v is an external node whose basic rectangle has minimum height h.} \\ h(w) + h(z) & \text{if v is an internal node associated with a horizontal cut with left} \\ & \text{child w and right child z.} \\ max(h(w), h(z)) & \text{if v is an internal node associated with a vertical cut with left} \\ & \text{child w and right child z.} \end{cases} \tag{2}$$

You will design a tree data structure for cut floorplans that supports the operations:

- Create a floorplan consisting of a single basic rectangle.
- Decompose a basic rectangle by means of a horizontal or a vertical cut.
- Assign minimum weight and height to a basic rectangle.
- Compact a floorplan.
- Display the tree representing a floorplan.

The details for your implementation should be as follows.

(1) Write a `FloorCutTree` class. The tree MUST be implemented as a linked structure. Each node has an associated minimum height and minimum width. For external nodes (the basic rectangles) these values are explicitly assigned. For internal nodes, these values are computed by *compaction.* In addition, each node in the tree should have a **unique** node_label, which is a String, containing only letters. Once assigned, the label of a node cannot be changed. Your FloorCutTree should have **at least** the following `public` methods. Note that, this list is not complete, you will need other standard binary tree methods. You may extend your LinkedBinaryTree implementation from before, or you may write a FloorCutTree from scratch.

- the constructor: Creates an empty tree.
- `create_root(String v)`: Creates the root node corresponding to the initial rectangle and labels it with `v`.
- `cut_horizontal(String v, String lv, String rv)`: Decomposes the node with label `v` (which corresponds to a rectangle) into two by means of a horizontal cut. Label the bottom (left) child `lv` and the top (right) child as `rv`. It should throw an IllegalArgumentException if the node with label `v` is an internal node. No need to check for duplicate labels, however should throw an IllegalArgumentException if the given label is not a valid label, i.e., not a String containing only letters.
- `cut_vertical(String v, String lv, String rv)`: Decomposes the node with label `v` into two by means of a vertical cut. Label the left child `lv` and the right child as `rv`. It should throw an IllegalArgumentException if the node with label `v` is an internal node. No need to check for duplicate labels, however should throw an IllegalArgumentException if the given label is not a valid label, i.e., not a String containing only letters.
- `assign_width(String v, int width)`: Assigns a minimum width to the node whose label is `v`. If the node with label `v` corresponds to an internal node, it should throw an IllegalArgumentException.
- `assign_height(String v, int height)`: Assigns a minimum height to the node whose label is `v`. If the node with label `v` corresponds to an internal node, it should throw an IllegalArgumentException.

- `compact()`: Should *compact* the tree as described above. That is, the width and height values should be computed and assigned to all internal nodes as appropriate.
- `display()`: Display the tree as following,
  - Nodes should be printed according to their order in an *inorder* traversal, one node per line.
  - The entry for a node should be preceded by *depth* number of dots, where *depth* denotes the depth of the node in the tree.
  - For an internal node that corresponds to a horizontal cut, with width `w` and height `h`, the entry should be printed as `(-:(w,h))`. (If the width and/or height has not been assigned to a node yet, display 0 as placeholder, for example, `(-:(0,0))`.)
  - For an internal node that corresponds to a vertical cut, with width `w` and height `h`, the entry should be printed as`(|:(w,h))`.
  - For an external node with width `w`, height `h`, and label `L`, the entry should be printed as `(L:(w,h))`.

  For example, for the tree in the above figure, let width=10 and height=12 for the external node labeled with 'C', and let width=5 and height=11 for the external node labeled with 'D'. After compaction, if we display the tree, the part of the output corresponding to these two nodes and their parent should be:

```
. . . . (C:(10,12))
. . . (|:(15,12))
. . . . (D:(5,11))
```

(2) Write a driver program, which creates an instance of a `FloorCutTree` class, and gradually constructs the tree as follows. You will read from a file "test-hw3.txt". This file consists of a sequence of directives, each indicating an operation to be performed on the tree. There will be one directive per line. Each directive consists of the name of the directive followed by 0 or more arguments, separated by tab. Here are the possible directives and their meanings.

```
create-root L           Create the root of the tree and label it as L
cut-h       L  LC  LR   Divide the node labeled L into two by a horizontal cut,
                             label bottom(left) child as LC, label top(right) child as LR
cut-v       L  LC  LR   Divide the node labeled L into two by a vertical cut,
                           label the left child as LC, label the right child as LR
assign-w    L x         Set the width of the node labeled as L to x
assign-h    L x         Set the height of the node labeled as L to x
compact                 Perform the compaction algorithm on the tree
display                 Display the tree as described above
quit                    This is the last directive
```