First, let's make some statements:

M1 = list of "{"-nodes; M2 = list of "}"-nodes, m1 = "{"-nodes, m2 = "}"-nodes

root-node = The root of the tree.

-> **Property 0: The root-node has no parent, it cannot be folded and it inherits every node that has no other parent.**

Excess = e.g: { } } } – The excess for "{"-node is 2, because we can add 2 more children ("{"-nodes) to this node and it would still have a pair. This excess tells me how many children I could add to this node before it will have no pair left. Based on the excess definition we can come up with some properties:

-> **Property 1: Excess can be used not only by a node, but by its parent as well = inheritance of excess**. (This is true, because if a node has an excess of N_excess, then its parent will inherit an excess of N_excess – 1 from this node).

->**Property 2**: **An m1 node with excess cannot have a brother, unless his parent is the root**. (This is true, because if there is an excess for a node, then its parent will have at least a pair (if not an excess as well) inherited from this child).

Shortage = e.g: { { } – The shortage for the first "{" is 1, because we must add at least 1 new "}" se it would have a pair; the shortage for the second "{" is 0, because it already has a pair. The shortage is inherited as well as the excess, because if a node has a shortage of 2, for example, then its parent will have a shortage of 3. This part can actually be generalized and so we can define the next properties:

-> **Property 3: If a node (N1) has n children, then only one of them (the last one) can have a shortage and N1's shortage will be its child's shortage + 1 = inheritance of shortage**. (This is true, because if there is a node that has a shortage, then all the nodes below this node will be its children (or children of children, but not brothers). Or, in other words:

-> **Property 4**: **A node with shortage cannot have any brothers**. (This is true, because any node that will be below me it will be my child).

The main idea is that I have two types of nodes: m1 and m2 (as I said before). Then we have a function called Update_self that will be called every time we insert or delete an m1-node or an m2-node, because the tree structure will change. All m1-nodes have a list of m1-nodes (M1) and a list of m2-nodes (M2). M2[1], the first element in M2, is the actual pair. We distinguish three different cases depending on the size of M2 list:

1) If M2 size > 1, than there is excess (see definition and properties above) of m2-nodes and these nodes will be kept in order to use them later, if needed (when more m1 child – nodes will be inserted).

E.g: N1 is a node with M2 = [a,b,c,d]. N1 uses only one node (a) to form a pair, so it will share the rest of the list, [b,c,d], with its parent (let's call it N2). Now, both N1 and N2 share an excess,

so we can add a new child to any of them and there will still be a pair for everyone. Furthermore, if N2's parent was root, then the root would have been the parent for the remaining two nodes as well ('c' and 'd') to help me search through the nodes.

2) if M2 size = 1, then this node has a pair, but if an m1-child will be added to this node, then this child would keep its parent's pair and the parent will have a shortage.

3) if M2 size < 1, then there is shortage (see definition and properties above).

There are 4 different actions :
a) Insert an m1-node
b) Insert an m2-node
c) Delete an m1-node
d) Delete an m2-node

Now let's detail them a little:
a) When a "{" (m1-node) is inserted :
-> The node searches for its parent
-> Its parent will pass on some of his properties like this:
-> M1 and M2 lists of the parent will be divided (not shared) between this current node and its parent (based on child-node position, m1 nodes and m2 nodes position).
-> If the parent has a shortage, then M2 won't be divided anymore (because it's empty) and the child-node will just inherit its parent's shortage and attach an empty list as M2 list
-> The child-node will call Update_children() function
-> The child-node will call Update_Parent(excess (child-node M2))

b) When a "{" is inserted there are there are two possibilities:
-> Insert the "}" after a "{" and then there will be a "{" with a pair (and without shortage)
-> Insert the "}" after another "}" and then the first "{" above will have an excess that he will share with its parent.
-> In both cases it's enough to call Update_self() after inserting the new "}" into the M2 list.

c) When a "{" is deleted:
-> All the properties from this node will pass on to its parent (children, M2 list)
-> The parent will merge these properties with its own
-> It will call Update_self() to restore the tree.

d) When a "}" is deleted there are two posbilities (same as b):
-> Delete a "}" that is after a "}"
-> Delete a "}" after a "{"

-> In both cases it's enough to call Update_self() after removing the deleted "}" from the M2 list.

Update_parent (new_M2_Excess, new_shortage) :

-> This node will update:

-> Its M2 list (it will merge its M2 list with new_M2_Excess received from its modified child, based on Property 1)

-> If new_shortage > 0, then shortage = new_shortage + 1 (Based on Property 3)

-> If this is the root-node, then the algorithm stops (because the root-node is the last node in hierarchy and it doesn't have any brothers anyway – Property 0)

-> Then, it calls Update_self()

Update_self() : This is s a very important function, that restores the tree after has been modified. Brief explanation: What can happen after the tree structure changes (inserting or deleting a node) is that a brother of this node can become its child if this node loses its pair or one of its children may become its brother if this node doesn't have a shortage anymore. So here's a solution to solve this:

-> If size (M2) > 0, then this node might have lost some children (and these nodes will became its brothers). This happens if M2[0] (this node's pair) is above a child. That child will become its brother.

-> Because this node will lose a child and its M2 list is the merger of all its children excesses, this list has to be restored. So, a new M2 list will be calculated using its remaining children's excesses.

-> If size (new M2) <= 0, then this node has a shortage. If there is a shortage, then this node cannot have any brothers (Property 4), so all its brothers will be appended as its children. Then, its M2 list will be calculated as the merger of all its children's excesses.

-> Update_parent (excess(this node M2 list),shortage of this node) will then be called for its parent.