Let v be a *non-root* node in a rooted tree T. The operation $cut(v)$ *cuts the link* between v and its parent. This results in two trees, respectively, the subtree T_v rooted at v and the tree $T = B_r \setminus V(T_v)$.

Recall that we had already introduced this operation in the Homework on binomial trees and heaps. There, we observed that the *cut* operation is rather costly on a binomial heap. This was owing to the fact that cutting out a subtree from a binomial tree results in a tree that is no longer binomial and many more cuts may be needed to re-enforce the binomial heap property.

A *Fibonacci heap* is very similar to the lazy variant of binomial heaps, except that we *relax* the requirement that every tree in the heap is a binomial tree. Instead, by implementing *cut* in a controlled manner, we ensure that the trees in a Fibonacci heap retain the useful properties of binomial trees, while supporting efficient cuts.

To this end, $cut(v)$ is modified to work as follows.

1. Let u be the parent of v.
2. If u is a root node, we simply cut v from u. That is, we delete the edge (u, v), add v to the heap's root list and update the min pointer, if required.
3. If v is the first node cut from u, we will cut v from u, as before.
4. Otherwise, if v is the second child to be from u, we shall cut v from u and recursively cut u from its parent.

The rank of a node is defined as the number of children it has. By extension, we redefine the rank of a tree as the rank of its root. This redefinition is necessitated as the other characterizations of rank in a binomial heap do not hold for Fibonacci heaps.

As in the case of binomial heaps, we will only link a tree into another tree if both trees have the same rank.

1. Let T_k be a tree of rank k in a Fibonacci heap. Give a tight upper bound on the number of nodes that T_k can have.

2. Suppose that we obtained the tree T_k (with rank k) by cutting out the largest subtree from each node in the binomial tree B_{k+1}.

   (a) Give a recurrence relation for the number of nodes in T_k.

   (b) Using this recurrence relation, show that the number of nodes in T_k is F_k, the k^{th} number in a Fibonacci series with $F_0 = 1$ and $F_1 = 2$.

   (c) Suppose that the children of all the nodes in T_k are sorted in the ascending order of their ranks. Then, for any node v such that $rank(v) \geq 3$ and some i such that $2 \leq i \leq rank(v) - 1$, show that $rank(v_i) = i - 2$, where v_i is the i^{th} child of v.

3. Write algorithms to implement $decreaseKey$ and $deleteMin$ in a Fibonacci heap, using only cut as a sub-task.

4. What is the actual cost of the modified version of cut? Design a crediting scheme such that its amortized cost is only $\mathcal{O}(1)$.