Algorithm Design
M. T. Goodrich and R. Tamassia
John Wiley & Sons
Solution of Exercise C-6.9

1. No. Not always unique. It's possible that the remaining tree has two nodes. Consider again, a tree that consists of a unique path having an even number of nodes. And, of course, we don't like to remove the leaves of a two-node tree (there will be nothing left!).
2. First, observe that the center of a tree T does not change, if we remove all leaves of the tree (or, if we add some children to every leaf of the tree). Moreover, a leaf can not be the center of a tree of at least 3 nodes. The idea here is, thus, to keep pruning the leaves until one or two nodes are left:

   (a) Remove all leaves of T. Let the remaining tree be T_1.
   (b) Remove all leaves of T_1. Let the remaining tree be T_2.
   (c) Repeat the "remove" operation as follows: Remove all leaves of T_i. Let remaining tree be T_{i+1}.
   (d) Once the remaining tree has only one node or two nodes, stop. Suppose now the remaining tree is T_k.
   (e) If T_k has only one node, that is the center of T. The *eccentricity* of the center node is k.
   (f) If T_k has two nodes, either can be the center of T. The *eccentricity* of the center node is $k+1$.

   This high level algorithm description gives us a linear time algorithm. (Worst case, when the tree is just a path.)
   Now, we can either really perform removals in a copy of our tree, or simulate the removal by marking the leaves. Any "remove" operation can be performed by traversing the tree (starting from an arbitrary node) and deleting/marking the leaves. However, a "remove" operation may require $O(n)$ time, yielding a total quadratic time complexity. Another way to see that: consider a degenerate tree which is actually a path (or list). At each step only two leaves are removed, and these two leaves are found in time proportional to the current size of the path. This worst case example shows that the required time can be of the order of $\sum_{i=1}^{n} i = O(n^2)$.

We give the pseudo-code for the algorithm. The algorithm only returns the center of G and not the eccentricity (this can be easily done by storing the number of leaf "removal" operations).

```
Algorithm Center(G):
  G' ← a copy of G
  while G'.numVertices()> 3 do
    RemoveLeaves(G',G'.aVertex(),NULL)
  return G'.aVertex()
```

```
Algorithm RemoveLeaves(G,v,u):
{u is the parent of v in the tree traversal}
  c ← 0
  for all e ∈ G.incidentEdges(v) do
    c ← c + 1
    w ← G.opposite(v,e)
    if w ≠ u then
      RemoveLeaves(G,w,v)
  if c = 1 then
    removeVertex(v)
```