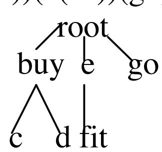
# Computer Science 308-250B Homework #5
# Due Tuesday April 13, 2004, 13:30

•**1)** Write a **Java** class that implements a generic Heapsort of an array of Objects (of any type). Your method must receive two parameters: the array and an extra object containing the method to compare elements of the array. You must test your method with objects of type String. You will also use this method for the rest of the homework.

•**2)** This problem deals with the Tree Isomorphism Problem. You will receive two trees from a file and you must determine whether they are isomorphic, meaning that they are the same tree upto relabelling of the vertices. Briefly this is what you program should do:

A) read two trees from a file
format: root(buy(c)(d))(e(fit))(go)
resulting tree:




B) Re-root each tree at its center.
C) Assign an integer label to each vertex
D) The trees are isomorphic iff their sets of labels are equal.
E) If the trees are isomorphic then print out a 1-1 mapping between the vertices.

**Algorithm** Isomorphic($T_1$, $T_2$)

**Input**: Rooted trees $T_1$ and $T_2$.
**Output**: If $T_1$ is isomorphic to $T_2$ then a 1-1 mapping between their vertices proving that they are isomorphic; otherwise, an empty mapping.

$C_1 \leftarrow$ FindCenter($T_1$)
ReRoot($T_1$, first vertex in $C_1$)
$C_2 \leftarrow$ FindCenter($T_2$)
ReRoot($T_2$, first vertex in $C_2$)

M $\leftarrow$ RootedIsomorphic($T_1$, $T_2$)
**IF** M is empty and size($C_2$) > 1 **THEN**
  ReRoot($T_2$, second vertex in $C_2$)
  RootedIsomorphic($T_1$, $T_2$)

**return** M

**Algorithm** FindCenter(T)

**Input**: Rooted tree T.
**Output**: Returns the set of vertices at the center of T; that is,
the set of vertices furthest from a leaf. The set has size either one or two.

CalculateDistancesFromLeaves(T)
max ← max{ distance(v) | v in T}
**return** { v | v in T and distance(v) = max }

**Algorithm** CalculateDistancesFromLeaves(T)

**Input**: Rooted tree T.
**Output**: For each vertex v in T, calculates v.distance which is
the length of the shorest path between w and a leaf in T.

Use a postorder traversal of T to initially set
v.distance ← min{ w.distance + 1 | w is a child of v }

Use a preorder traversal of T to finally set
v.distance ← min{ v.distance, v.parent().distance + 1 }

**Algorithm** ReRoot(T, v)

**Input**: T is a rooted tree and v a vertex in T.
**Output**: T rooted at v.

T.setRoot(v)
p ← T.parent(v)
**WHILE** p ≠ null **DO**
remove v from p's list of children
add p to v's list of children
g ← T.parent(p)
make p's parent v
v ← p
p ← g

**Algorithm** RootedIsomorphic($T_1$, $T_2$)

**Input**: $T_1$ and $T_2$ are rooted trees.
**Output**: If $T_1$ is rooted-isomorphic to $T_2$ then produces a 1-1 mapping between their vertices, proving that they are rooted-isomorphic; otherwise, an empty mapping.

```
h ← T1.height()
IF h ≠ T2.height() THEN
  return empty mapping
L1 ← an array of h+1 empty sequences
L2 ← an array of h+1 empty sequences
L1[h] ← Initialize(T1, h)
L2[h] ← Initialize(T2, h)
FOR i ← h-1, h-2, ..., 0 DO
 L1[i] ← LabelLevel(T1, L1[i+1], i)
 L2[i] ← LabelLevel(T2, L2[i+1], i)
  IF L1[i].size() ≠ L2[i].size() THEN
    return empty mapping
  FOR i ← 0, 1, ..., L1[i].size() DO
    v ← vertex i in L1[i]
    w ← vertex i in L2[i]
    IF orderedlabels(v) ≠ orderedlabels(w) THEN
      return empty mapping
M ← empty sequence
GenerateMapping(T1.root(),T2.root(), M)
return M
```

**Algorithm** Initialize(T, h)

**Input**: Rooted tree t of height h.
**Output**: Initializes the label, orderedchildren sequence and orderedlabels sequence of each vertex in T. Returns L, a sequence containing all vertices at depth h in T.

```
L ← an empty sequence
FOR each vertex v in T DO
  v.label ← 0
  v.orderedchildren ← empty sequence
  v.orderedlabels ← empty sequence
  IF T.depth(v) = h THEN
    L.insert(v)
return L
```

**<u>Algorithm</u>** <u>LabelLevel(T, P, d)</u>

**Input**: Rooted tree T, integer d between 0 and T.height(), and sequence P containing all vertices at depth d+1 in T arranged in ascending order of their integer labels.
**Output**: Calculates integer label for each vertex v at depth d in T so that if v and w are vertices in T at depth d then v.label = w.label if and only if the subtrees rooted at v and w are isomorphic.
Returns sequence L containing all vertices at depth d in T arranged in ascending order of their integer labels.

```
L ← empty sequence
FOR i ← 0, 1, ..., P.size()-1 DO
  v ← vertex i in P
  IF T.parent(v).orderedlabels.isEmpty() THEN
    L.insert(T.parent(v))
  T.parent(v).orderedlabels.insertAtBack(v.label)
  T.parent(v).orderedchildren.insertAtBack(v)
Sort L in descending order of the orderedlabels of each vertex
FOR each vertex v in L DO
  v.label ← k where v.orderedlabels is the kth largest in L
```

**<u>Algorithm</u>** <u>GenerateMapping(v, w, M)</u>

**Input**: Vertices v and w, roots of isomorphic subtrees, and sequence M.
**Output**: Adds 1-1 mapping between the vertices in the subtree rooted at v and the vertices in the subtree rooted at w proving that the subtrees are isomorphic.

```
M.insert(pair v,w)
FOR i ← 0, 1, ..., v.orderedchildren.size()-1 DO
  x ← vertex i in v.orderedchildren
  y ← vertex i in w.orderedchildren
        GenerateMapping(x, y, M)
```

**<u>PLEASE CONSULT</u>**

http://crypto.cs.mcgill.ca/~crepeau/CS250/HW5+.pdf

for more details about this algorithm.