Tree Isomorphism Algorithms with time complexity $\mathcal{O}(n)$

Arist Kojevnikov

Based on Aho, Hopcroft and Ullman book

```
Tree_Isomorphism(T1, T2 : trees)
Begin
  Assign all vertices of T1 and T2 to level numbers lists
  and let hi be the largest level number in Ti;

  If h1 <> h2 then
    write('trees are not isomorphic'); Halt;
  else
    set h to h1; {h1 = h2}

  { process from bottom to top level }
  for i := h downto 0
  begin
    { assign vertices their string name }
    For all vertices v of level i do
      If v is a leaf then
        assign v the string 10
      Else
        assign v the tuple 1 i1 i2 ... ik 0, where i1 i2 ... ik
        are the strings associated with the children
        of v in non-decreasing order;
      { assign vertices to temporary sorting lists }
      For all vertices v of level i do
        If v belongs to Tj then
          add v's string to Tj(i);
      Sort T1(i) and T2(i) lexicographically;
      If T1(i) <> T2(i) then
        write('trees are not isomorphic t level',i); Halt;

      { assign condensed canonical names }
      For all vertices v of level i do
        If v is the k-th element in Tj(i) then
          assign v the binary string for the integer k
  end;
  write('the trees are isomorphic');
end
```

AHU algorithm

Input: trees T_1 and T_2.

1. Assign to all leaves of T_1 and T_2 the integer 0.

2. Inductively, assume that all vertices of T_1 and T_2 at level $i-1$ have been assigned integers. Assume L_1 is a list of the vertices of T_1 at level $i-1$ sorted by non-decreasing value of the assigned integers. Assume L_2 is the corresponding list for T_2.

3. Assign to the non-leaves of T_1 at level i a tuple of integers by scanning the list L_1 from left to right and performing the following actions: For each vertex on list L_1 take the integer assigned to v to be the next component of the tuple associated with the father of v. On completion of this step, each non-leaf w of T_1 at level i will have a tuple $(i_1, i_2, \ldots, i_k)$ associated with it, where $i_1, i_2, \ldots, i_k$ are integers, in non-decreasing order, associated with the sons of w. Let S_1 be the sequence of tuples created for the vertices of T_1 on level i.

4. Repeat step 3 for T_2 and let S_2 be the sequence of tuples created for the vertices of T_2 on level i.

5. Sort S_1 and S_2 lexicographically. Let S_1' and S_2' respectively, be the sorted sequences of tuples.

6. If S_1' and S_2' are not identical then halt; the trees are not isomorphic. Otherwise, assign the integer

1 to those vertices of T_1 on level i represented by the first distinct tuple on S_1', assign the integer 2 to the vertices represented by the second distinct tuple, and so on. As these integers are assigned to the vertices of T_1 on level i, make a list L_1 of the vertices so assigned. Append to the front of L_1 all leaves of T_1 on level i. Let L_2 be the corresponding list of vertices of T_2. These two list can now be used for the assignment of tuples to vertices of level $i+1$ by returning to step 3.

7. If the roots of T_1 and T_2 are assigned the same integer, T_1 and T_2 are isomorphic.

AHU algorithm for lexicographical sorting

Input: Sequence of strings $A_1, A_2, \ldots, A_n$ from alphabet $0 \ldots m-1$. We set $l_i = |A_i| = |(a_{i1}, \ldots, a_{il_i})|$ and $l_{max} = \max_i l_i$

Output: Permutation $B_1, B_2, \ldots, B_n$ of $A_1, A_2, \ldots, A_n$ where $B_1 \leq B_2 \leq \ldots \leq B_n$

1. Create set of pairs (l, a_{il}) for all $1 \leq i \leq n, 1 \leq l \leq l_i$. Sort them.
   Form lists $NONEMPTY[l]$ with sorted numbers j: $a_{il} = j$ for some i.

2. Form lists $LENGTH[l]$ with sorted strings of length l (it is easy to do with pointers)

3. Sort strings by literals in the fixed position from $l := l_{max}$ down to 1:
   Add to QUERY all strings from $LENGTH[l]$
   Sort QUERY by literals at position l

Complexity

Theorem: Algorithm uses the time

$$\mathcal{O}(l^* + m),$$

where $l^* = \sum_{i=1}^{n} l_i$

At the first step: $\mathcal{O}(l^*)$ for creation and $\mathcal{O}(l^* + m)$ for sorting

At the second step: $\mathcal{O}(l^*)$

At the third step: $\mathcal{O}(l^*)$