Joint Advanced Student School

Explanation for ‘Tree isomorphism’ talk

by Alexander Smal (avsmal@gmail.com)

Saint-Petersburg, Russia
2008

Abstract

In this talk we considered a problem of tree isomorphism. We made several attempts to find a complete invariant for rooted trees isomorphism and obtain efficient algorithm from it. In conclusion, we discussed main ideas of algorithm from "The Design and Analysis of Computer Algorithms" by Aho, Hopcroft and Ullman [1].

The idea of this talk is based on [2].

1 Motivation

In gene splicing, protein analysis, and molecular biology the chemical structures are often trees with millions of vertices. So, the problem of checking whether two structures are equal corresponds to the problem of checking whether two trees are isomorphic. Thus, in mentioned applications difference between $O(n)$, $O(n \log n)$, and $O(n^2)$ isomorphism algorithms is not just theoretical importance.

2 The idea

2.1 Graph isomorphism

Let's start with a definition of graph isomorphism.

Definition 1. ***Isomorphism of graphs*** *$G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ is a bijection between the vertex sets $\varphi : V_1 \to V_2$ such that*

$$\forall u, v \in V_1 \quad (u, v) \in E_1 \Leftrightarrow (\varphi(u), \varphi(v)) \in E_2.$$

There are several common facts about graph isomorphism.

- No algorithm, other than brute force, is known for testing whether two arbitrary graphs are isomorphic.
- It is still an open question(!) whether graph isomorphism is $\mathcal{NP}$ complete.
- Polynomial time isomorphism algorithms for various graph subclasses such as trees are known.

2.2 Rooted trees

We need a quick way to determine whether two ordinary trees are isomorphic. Consider a trick: let's convert trees to rooted trees and try to determine whether they are isomorphic as rooted trees. The idea of this trick is that it should be easier to determine rooted trees isomorphism than ordinary trees isomorphism because rooted trees give us a little bit more information.

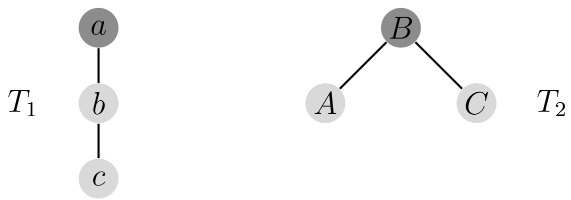
Definition 2. ***Rooted tree*** (V, E, r) *is a tree* (V, E) *with selected root* $r \in V$.

Definition 3. ***Isomorphism of rooted trees*** $T_1(V_1, E_1, r_1)$ *and* $T_2(V_2, E_2, r_2)$ *is a bijection between the vertex sets* $\varphi : V_1 \to V_2$ *such that*

$$\forall u, v \in V_1 \quad (u, v) \in E_1 \Leftrightarrow (\varphi(u), \varphi(v)) \in E_2 \text{ and } \boxed{\varphi(r_1) = r_2}.$$

The only difference between graphs and rooted trees isomorphism is that trees isomorphism preserves root. Here you can see an example of two rooted trees, that are isomorphic as *graphs* (and as ordinary trees) but **not** as *rooted trees*.

Example.



Lemma 1. *If there is* $O(n)$ *algorithm for rooted trees isomorphism, then there is* $O(n)$ *algorithm for ordinary trees isomorphism.*

Proof.

1. Let $\mathcal{A}$ be $O(n)$ algorithm for rooted trees.
2. Let T_1 and T_2 be ordinary trees.
3. Let's find centers of this trees. There are three cases:
   (a) each tree has only one center (c_1 and c_2 respectively)
   return $\mathcal{A}(T_1, c_1, T_2, c_2)$
   (b) each tree has exactly two centers (c_1, c_1' and c_2, c_2' respectively)
   return $\mathcal{A}(T_1, c_1, T_2, c_2)$ **or** $\mathcal{A}(T_1, c_1', T_2, c_2)$
   (c) trees has different number of centers
   return False

□

To understand this lemma we should define center of tree and propose a way to find it.

2.3 Diameter and center

Definition 4. *The* ***diameter*** *of tree is the length of the longest path.*

Definition 5. *A* ***center*** *is a vertex* v *such that the longest path from* v *to a leaf is minimal over all vertices in the tree (a half of diameter).*

Tree centers can be found using simple algorithm.

Algorithm. (Centers of tree)

1: Choose a random root r.
2: Find a vertex v_1 — the farthest from r.
3: Find a vertex v_2 — the farthest from v_1.
4: Diameter is a length of path from v_1 to v_2.
5: Centers are median elements of path from v_1 to v_2.

This is $O(n)$ algorithm. So, we can use it in lemma 1.

2.4 The idea

Definition 6. ***Isomorphism invariant*** *is a function* $f(T)$ *such that* $f(T_1) = f(T_2)$ *for all pairs of isomorphic trees* T_1 *and* T_2*.*

Definition 7. ***Complete isomorphism invariant*** *is a function* $f(T)$ *such that two trees* T_1 *and* T_2 *are isomorphic if and only if* $f(T_1) = f(T_2)$*.*

Idea. If we find a complete isomorphism invariant we can obtain algorithm from it.

WARNING! Starting from this point *tree* always means *rooted tree*!

3 Complete invariant candidates

Consider several candidates to be a complete isomorphism invariant.

3.1 Candidate 1

Observation 1. *The level number of a vertex is a tree isomorphism invariant.*

Using this observation...

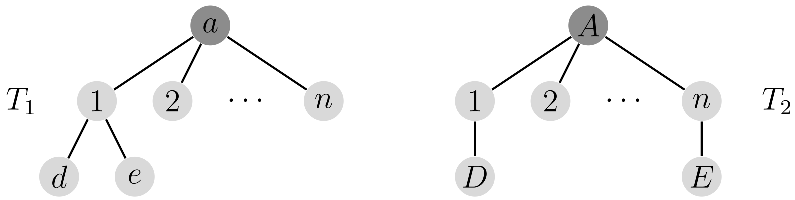
Conjecture 1. *Two trees are isomorphic if and only if they have the same number of levels and the same number of vertices on each level.*

Ok. It seems to be a good try. But...

Observation 2. *The number of the leaves is a tree isomorphism invariant.*

Using observation 2 we can construct a contrary instance.

Contrary instance. Here you can see two trees that have the same number of vertices on each level, but different number of leaves (observation 2 is violated). It means that conjecture 1 is wrong.



Each tree has the same number of vertices on each level:
1 vertex on level 0, n vertices on level 1 and 2 vertices on level 2,
but T_1 and T_2 have different number of leaves ($n + 1$ and n respectively).

3.2 Candidate 2

What's wrong with candidate 1? We didn't take into account the *degree spectrum* of a tree.

Definition 8. ***Degree spectrum*** *of a tree is the sequence of non-negative integers* $\{d_j\}$, *where* d_j *is the number of vertices that have* j *children.*

Let's fix it.

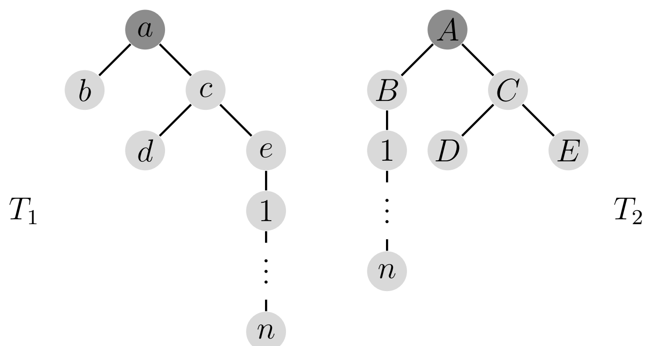
Conjecture 2. *Two trees are isomorphic if and only if they have the same degree spectrum.*

It seems to be better than first candidate, but...

Observation 3. *Since a tree isomorphism preserves longest paths from the root, the number of levels in a tree is a tree isomorphism invariant.*

Using observation 3 we can construct a contrary instance.

Contrary instance. This two trees have the same degree spectrum but different number of levels (observation 3 is violated).



Each tree has degree spectrum $(3, n, 2, 0, \ldots)$: there are 3 vertices with no children,
n vertices with only one child and 2 vertices with 2 children,
but T_1 has $n + 2$ levels whereas T_2 has only $n + 1$ levels.

3.3 Candidate 3

What's wrong with candidate 2? We used some integral property for it. Consider an analogy: algorithm for determining whether two integer arrays are equal which just compares their sums. Obviously, it is a wrong algorithm.

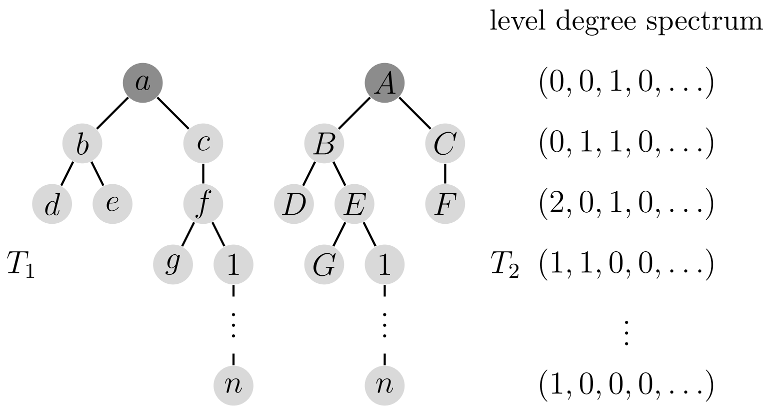
Conjecture 3. *Two trees are isomorphic if and only if they have the same degree spectrum at each level.*

This conjecture is very tricky. If two trees have the same degree spectrum at each level, then they must automatically have the same number of levels, the same number of vertices at each level, and the same global degree spectrum!

Observation 4. *The number of leaf descendants of a vertex and the level number of a vertex are both tree isomorphism invariants.*

And again using observation 4 we can construct a contrary instance.

Contrary instance. You can see two trees below that have the same degree spectrum at each level but on level 2 in T_1 there are two vertices, b and c, with 2 and $n+2$ descendants respectively and in T_2 there are two vertices, B and C, with $n+3$ and 1 descendants respectively. So, observation 4 is violated.



4 AHU algorithm

We have failed tree times. Let's look for some existing algorithm and understand it.

4.1 Algorithm by Aho, Hopcroft and Ullman

This algorithm is from [1]. There are two main properties of this algorithm.

- Determine tree isomorphism in time $O(|V|)$.
- Uses *complete history of degree spectrum of the vertex descendants* as a complete invariant.

The idea of AHU algorithm. The AHU algorithm associates with each vertex a tuple that describes the complete history of its descendants.

Hard question. Why our previous invariants are not complete?

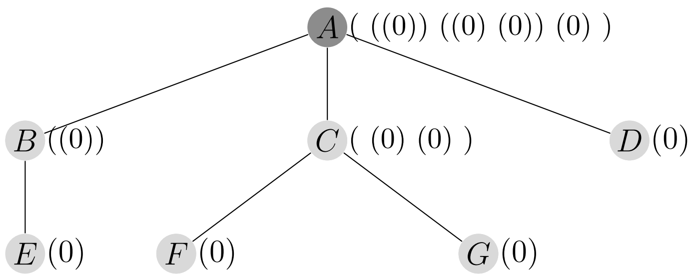
Answer. We discussed weakness of first two candidates. The third invariant was better than second but it also uses some integral properties.

Our plan. Let's discuss AHU algorithm. We start from $O(|V|^2)$ version and then we discuss how to make it faster ($O(|V|)$).

4.2 Understanding AHU algorithms

Knuth tuples. Let's assign parenthetical tuples to all tree vertices.

Knuth tuples example. Knuth tuples are very intuitive easy. So, formal definition is unnecessary. Let's consider an example.



You should have noticed that all leaves have (0) label
and each non-leafs label consists of children tuples enclosed in parentheses.

There is an algorithm ASSIGN-KNUTH-TUPLES that visits every vertex once or twice.

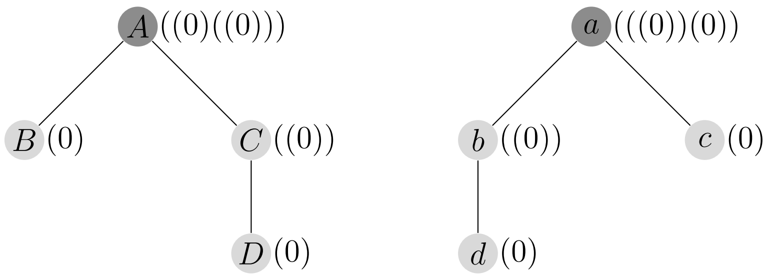
Algorithm. ASSIGN-KNUTH-TUPLES(v)

if v is a leaf **then**
  Give v the tuple name (0)
else
  for all child w of v **do**
    ASSIGN-KNUTH-TUPLES(w)
  end for
end if
Concatenate the names of all children of v to *temp*
Give v the tuple name *temp*

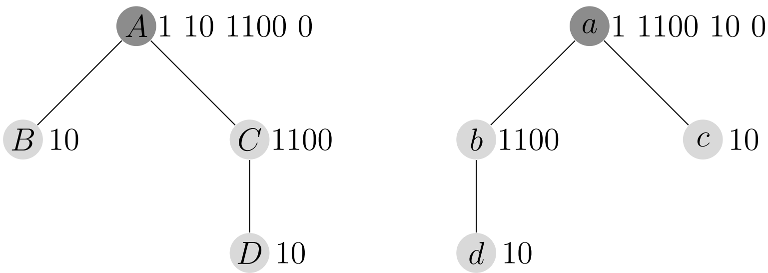
Observation 5. *There is no order on parenthetical tuples.*

Why do we need an order? Let's consider an example.

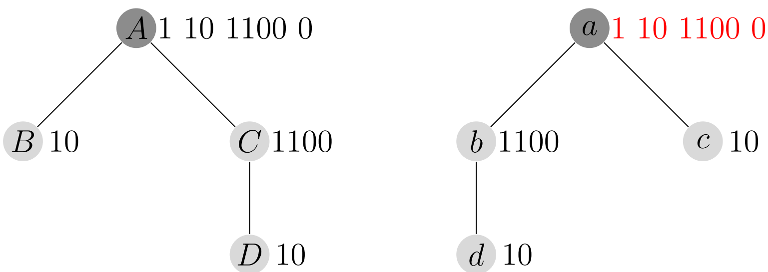
Example. Here you can see two isomorphic trees with assigned Knuth tuples.



We know that trees are isomorphic but roots have different assigned tuples. Let's convert parenthetical tuples to *canonical names*. We should drop all "0"-s (zeros are not necessary) and replace "(" and ")" with "1" and "0" respectively.



Canonical names are just numbers. So, we can sort them. Let's sort canonical names of children for each non-leaf node.



Roots has the same assigned canonical names.

There is an algorithm ASSIGN-CANONICAL-NAMES that visits every vertex once or twice (it is a modification of ASSIGN-KNUTH-TUPLES).

Algorithm. ASSIGN-CANONICAL-NAMES(v)

```
if v is a leaf then
    Give v the tuple name "10"
else
    for all child w of v do
        ASSIGN-CANONICAL-NAMES(v)
    end for
```

end if
Sort the names of the children of v
Concatenate the names of all children of v to *temp*
Give v the name **1***temp***0**

Conjecture 4. *Two trees are isomorphic if and only if they have the same canonical name assigned to root.*

We should discuss some important questions.

Invariant? Is canonical name of a root a *tree isomorphism invariant*?

Answer. Yes. Obviously, two isomorphic trees have the same canonical name assigned to root.

Complete invariant? Is canonical name of a root a *complete tree isomorphism invariant*?

Answer. Yes. We can show it easily by reconstructing tree from root canonical name. So, there is a bijection between tree and roots canonical names.

Algorithm. AHU-Tree-Isomorphism(T_1, T_2)

$r_1 \leftarrow \text{root}(T_1)$
$r_2 \leftarrow \text{root}(T_2)$
Assign-Canonical-Names(r_1)
Assign-Canonical-Names(r_2)
if $\text{name}(r_1) = \text{name}(r_2)$ **then**
  return True
else
  return False
end if

4.3 AHU algorithm improvement

Observation 6. *Consider a tree of n vertices in one long strand. Time needed to compute the root name of this tree is proportional to $1 + 2 + \cdots + n$, which is $\Omega(n^2)$.*

This observation shows that AHU-Tree-Isomorphism is $O(|V|^2)$. But there is a way we can improve it to be $O(|V|)$.

Observation 7. *For all levels i, the canonical name of level i is a tree isomorphism invariant.*

Observation 8. *Two trees T_1 and T_2 are isomorphic if and only if for all levels i canonical level names of T_1 and T_2 are identical.*

Using this two observations…

The idea 1. Assign canonical names for level, sort by level, and check by level that the canonical level names agree.

The idea 2. Assign canonical names for level and if canonical level names agree than replace canonical names with integers.

These ideas show us how improve AHU-Tree-Isomorphism to make it $O(|V|)$. Using unique integers on each level instead of strings we can get rid of effect we discovered in observation 6. On slides you can find animated demo of AHU-Tree-Isomorphism algorithm work.

But… There is a hole in our discussion — we have forgotten about sorting in Assign-Canonical-Names. So, in this case AHU-Tree-Isomorphism is just $O(n \log n)$… But there is a way to sort canonical names in linear time — you can read about it in [1].

5 Resume

- We have made three unsuccessful attempts to construct complete tree isomorphism invariant.
- We discussed $O(|V|^2)$ version of AHU algorithm.
- We discussed ways of AHU algorithm improvement to make it work in $O(|V|)$ time.

References

[1] A. Aho, J. Hopcroft, and J. Ullman *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Co., Reading, MA, 1974, pp. 84-85.

[2] D.M. Campbell, D. Radford, *Tree isomorphism algorithms: Speed vs. clarity.* Math. Mag. 64, No. 4, 1991, pp. 252-261.