TREE ISOMORPHISM PROBLEM

A HASHING APPROACH

Li Kwing Hei

Jon Sze Pok

February 2018

TREE ISOMORPHISM PROBLEM
A HASHING APPROACH

TABLE OF CONTENTS

1 Abstract

This research investigates the efficiency of determining instances of tree isomorphism from a large number of rooted, labelled trees by initial hashing of the trees. The time taken to detect isomorphic trees with the proposed algorithm in different situations, such as variation is the number of trees and order of trees, are compared with that of the AHU algorithm. Applications and further modifications are discussed.

2 Relevant Concepts

2.1 Tree

A tree is a set of straight line segments connected at their ends, forming nodes, containing no closed loops.[1] A tree graph possesses the following properties:

1. It has N nodes and N-1 graph edges.
2. It is bipartite.
3. Any 2 nodes are connected by a simple unique path.

2.2 Rooted Tree

A rooted tree is a tree graph with one node designated the root. On the contrary, a free tree is a tree graph without a root. In this research, rooted trees will be the main focus.

2.3 Labelled Tree:

A labelled tree is a tree graph with its nodes labelled,[2] such that nodes with different labels are distinct. Cayley's formula, devised in 1889 by Arthur Cayley, determines that the number of possible trees with N distinctly labelled nodes is N^{N-2}. However in this research, the nodes in a tree are not necessarily distinctly labelled.

2.4 Isomorphic Trees:

Two trees are isomorphic if and only if one of them can be obtained from other by a series of flips, i.e. by swapping the order of children connected to a node.[3] In a rooted tree, an alternative way of defining isomorphism is that the resulting tree preserves root and edge incidence of the initial tree, such that the notion direct paths from any node to another is invariant.

2.5 Tree Isomorphism Problem

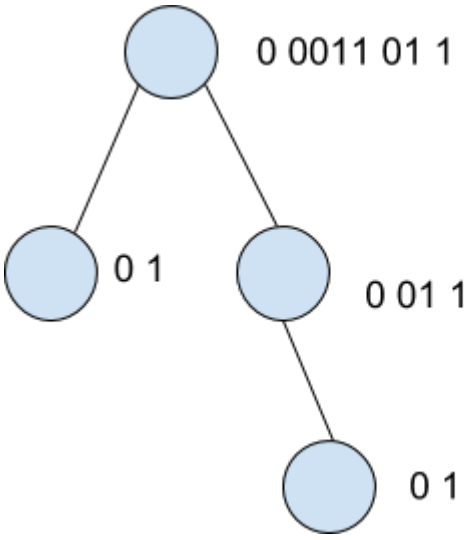
The tree isomorphism problem is a well-known problem in computational group theory that studies the algorithmic determination of whether two trees are isomorphic. The type of tree is not specified, thus it includes sub-problems about different types of trees, such as labelled trees versus unlabelled trees.

3 Similar Solutions

3.1 AHU algorithm

The AHU algorithm is an $O(N^2)$ algorithm able to exhibit $O(N)$ with optimisation that determines whether 2 trees are isomorphic.[4] For each node of a tree, this algorithm computes the canonical name of the node. Two trees are isomorphic if the canonical name assigned to their respective roots are the same. The canonical names are computed by the following function:

```
Algorithm. Findcanonical(v)
    if v has no children then
       give v canonical name "01"
     else
          for all child w of v do
               Findcanonical(w)
          end for
    end if
    Sort canonical names of the children of v
    Concatenate the names of all children of v to temp
    Give v the name "0"+temp+"1"
```

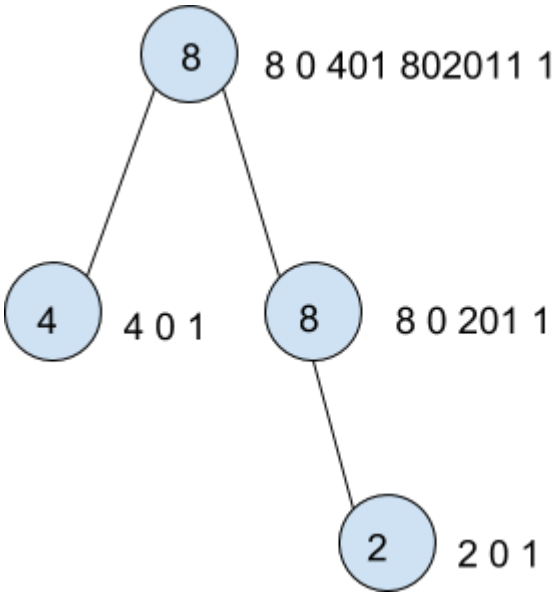


Example 3.1: AHU algorithm for unlabelled trees

3.2 Limitations

However, there are limitations to the algorithm:

1. AHU algorithm is used for non-labelled trees. If it is used for labelled trees, it must be modified. Later in the study, when comparisons of labelled trees are tested with the AHU algorithm, the label of each node will be inserted in front of its canonical name.
2. If there are a large number of trees, the AHU algorithm is quite slow. When a lot of trees are considered, the AHU algorithm first computes the canonical name of all trees and compares each pair. Comparing canonical names of the trees take a significant amount of time.



Example 3.2: AHU algorithm for labelled trees

4 Proposed Solution

4.1 Refined problem

Consider a set of M rooted, labelled trees, with an average of N nodes each. Our aim is to find all pairs of isomorphic trees within the set.

4.2 Solution with AHU algorithm

The AHU algorithm not only requires modification to work on labelled trees, but the time taken to solve the problem is quite long.

The pre-computation time for the canonical names is $O(N^2M)$ and since each comparison takes $O(N)$, with the length of the canonical names being around $2N$; having to make around M^2 comparisons, the time complexity for the comparisons is $O(NM^2)$. It follows that the full algorithm is of time complexity class $O(N^2M+NM^2)$, which is not efficient.

4.3 Hashing algorithm

In order to obtain a more efficient solution to the problem, we propose the assignment of a hash function for the trees such that each tree is assigned a number. It is similar to the AHU algorithm with two main differences:

1. It is applicable to labelled trees
2. Each canonical name is hashed into an integer

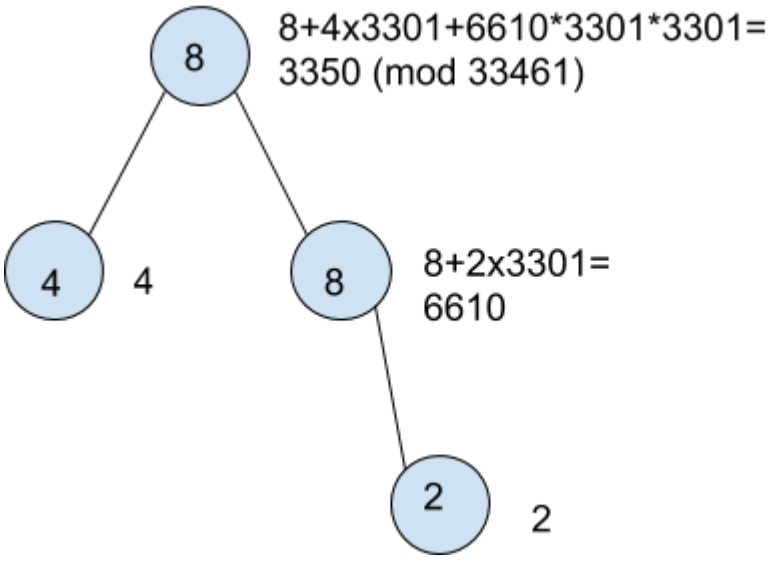
Definitions:
Let $H[i]$ denote the hash for the i-th node
Let K_i denote the label for the i-th node
Let C_{ij} denote the j-th child of the i-th node
Let P and p denote two distinct prime numbers

The hash of a node is given by the hash function H as follows:

$$H[i] = (K_i + H[C_{i1}] \cdot p^1 + H[C_{i2}] \cdot p^2 + H[C_{i3}] \cdot p^3 + ...) \bmod P$$

The hash can be computed by the following function:

```
Algorithm. Findhash(v)
          H[v] := K[v]
     if v has one or more children then
          for all child w of v do
               Findhash(w)
          end for
                    Sort hash of the children of v
                    temp:=1
                       for each child i of v
                                 temp:= temp * p % P
                                 H[v] := H[v] + temp * H[i]
                                 H[v] := H[v] % P
                       end for
     end if
     H[v] := H[v] % P
```



Example 4.3: Hashing algorithm for labelled trees (p=3301; P=33461)

5 Theoretical Evaluation

5.1 Time Complexity

The process in pre-computing hashes in the hashing algorithm and pre-computing canonical names in the AHU algorithm is very similar. Therefore, it can be deduced that the time complexity for precomputing hashes of M trees with an average of N nodes each is also of the complexity class $O(N^2M)$. However, unlike comparisons of canonical names, each comparison of hashes is $O(1)$ as bounded by the range of the modulo function, and thus the time complexity for comparing M trees is $O(M^2)$. It follows that the hashing algorithm is of time complexity class $O(N^2M+M^2)$, which is always more efficient than the AHU algorithm, which is of time complexity class $O(N^2M+NM^2)$.

5.2 Soundness

The hash of a node is given by the hash function H as follows:

$H[i] = (K_i + H[C_{i1}] \cdot p^1 + H[C_{i2}] \cdot p^2 + H[C_{i3}] \cdot p^3 + ...) \bmod P,$

and the hashes of the roots on 2 trees within the set of M trees are compared to determine isomorphism.

First consider the case of an alternate hash function H* where

$H^*[i] = K_i + H[C_{i1}] \cdot p^1 + H[C_{i2}] \cdot p^2 + H[C_{i3}] \cdot p^3 + ...$

And deduce hash for a node i with j children in the 2nd level bottom-up from a node itself having no children.

If $K_i < p \ \forall$ i,

H*[i] in base $p = C_{ij} || C_{i(j-1)} || C_{i(j-2)} || \dots || C_{i2} || C_{i1} || K_i$

where || denotes concatenation.

If $\exists$ i | $K_i \geq p$,

$$H^*[i] \text{ in base } p = \left\{\left\lfloor \frac{K_i}{p^j} \right\rfloor + \sum_{k=0}^{j-1} \left\lfloor \frac{C_{i(j-k)}}{p^k} \right\rfloor\right\} || \left\{\coprod_{m=1}^{j-1} \left(\left\lfloor \frac{K_i}{p^j} \right\rfloor mod\ p + \sum_{k=m}^{j-1} \left\lfloor \frac{C_{i(j-k)}}{p^k} \right\rfloor mod\ p\right)\right\} || \{K_i\ mod\ p\}$$

where $\coprod$ denotes a large-scale operator for sequential concatenation.

It follows from the fundamental theorem of arithmetic and the rules of modular arithmetic that the hash H* must be unique for a specific tree if

$\nexists$ i,j | $K_i \neq K_j$, $K_i \equiv K_j \pmod p$; AND

$\nexists$ i | $K_i = p^n$, $n \in Z^+$

Replacing the concatenated form with an algebraic form and modulo P gives an expression for H[i]; and similarly, the hash H must unique for a specific tree if the two above conditions hold true, as well as the four below:

$\nexists$ i,j | $K_i \neq K_j$, $K_i \equiv K_j$ (mod P); AND

$\nexists$ i,j | $H[K_i] \neq H[K_j]$, $H[K_i] \equiv H[K_j]$ (mod P); AND

$\nexists$ i | $K_i = P^n$, n ϵ Z^+; AND

$\nexists$ i | $H[K_i] = P^n$, n ϵ Z^+

The probability of all six conditions simultaneously holding true for an arbitrarily generated tree is greater than or equal to the probability of a tree guaranteeing a unique hash H for the root distinct from all other possible trees bar the final modulo P. Assume the labels K are generated randomly within the range from 1 to L, where L is a sufficiently large number, with an equal probability of each integer within the range being generated.

The probability of these three conditions holding true tends to 1 as L tends to infinity, and thus effect is negligible for large L,

$\nexists$ i | $K_i = p^n$, n ϵ Z^+;

$\nexists$ i | $K_i = P^n$, n ϵ Z^+;

$\nexists$ i | $H[K_i] = P^n$, n ϵ Z^+

such that the probability of hash uniqueness bar modulo P can be determined by the probability of the other three conditions holding true.

$\nexists$ i,j | $K_i \neq K_j$, $K_i \equiv K_j$ (mod p);

$\nexists$ i,j | $K_i \neq K_j$, $K_i \equiv K_j$ (mod P);

$\nexists$ i,j | $H[K_i] \neq H[K_j]$, $H[K_i] \equiv H[K_j]$ (mod P)

Since the labels K are evenly distributed across the integer range, the probability of each of the above conditions being violated for each pair of nodes is 1 divided by the modulus p or P.

In a tree with N nodes, an expression for the probability of hash uniqueness bar modulo P is obtained, given by

$$\text{Probability} \geq \left(\frac{(P-1)^2(p-1)}{P^2 p}\right)^{\frac{N(N-1)}{2}}$$

In a selection of M trees the probability of all trees preserving this characteristic is obtained by applying M as an exponent. Accounting as well for the final step of modulo P in the final step of the hashing algorithm, this gives a final probability of the algorithm holding as

$$\text{Probability} \geq \left(\frac{(P-1)^2(p-1)}{P^2 p}\right)^{\frac{MN(N-1)}{2}} \times \prod_{i=0}^{M}\left[1-\left(\frac{i}{P}\right)\right]$$

As can be observed, choosing larger primes p and P for the hashing process increase the probability of accurate output.

6 Experimental Evaluation

6.1 Comparison between hashing algorithm and AHU algorithm

Experiments were conducted to determine whether the hashing algorithm is more efficient than the AHU algorithm. The test data of each experiment is a certain number of rooted, indexed trees with the goal of finding all pairs of isomorphic trees. The time taken to complete each task is recorded and repeated. The respective times taken for the two algorithms are then compared. Note that the test data used for each trial is different. The number of trees M and average number of nodes N are listed.

6.2 Test Data

Note: all programs and test data used are uploaded to the following website: https://github.com/hei411/Treeresearch

Experiment 1:

Number of trees used: 2

Average number of nodes: 5

	1st trial	2nd trial	3rd trial	Mean time taken	Difference of mean time taken
Hashing algorithm	0.04221 s	0.03967 s	0.03841 s	0.0401 s	0.008 s
AHU algorithm	0.04833 s	0.04987 s	0.04612 s	0.0481 s	

Experiment 2:

Number of trees used: 2

Average number of nodes: 100

	1st trial	2nd trial	3rd trial	Mean time taken	Difference of mean time taken
Hashing algorithm	0.04092 s	0.04339 s	0.04325 s	0.04252 s	0.024 s
AHU algorithm	0.0668 s	0.06785 s	0.06578	0.06681 s	

Experiment 3:

Number of trees used: 2

Average number of nodes: 1000

	1st trial	2nd trial	3rd trial	Mean time taken	Difference of mean time taken
Hashing algorithm	0.05031 s	0.04734 s	0.04921 s	0.04895 s	0.186 s
AHU algorithm	0.2356 s	0.2336 s	0.2345 s	0.2346 s	

Experiment 4:

Number of trees used: 100

Average number of nodes: 5

	1st trial	2nd trial	3rd trial	Mean time taken	Difference of mean time taken
Hashing algorithm	0.04583 s	0.04946 s	0.04914 s	0.04814 s	0.047 s
AHU algorithm	0.09586 s	0.09509 s	0.09383 s	0.0949 s	

Experiment 5:

Number of trees used: 100

Average number of nodes: 100

	1st trial	2nd trial	3rd trial	Mean time taken	Difference of mean time taken
Hashing algorithm	0.05962 s	0.05599 s	0.05904 s	0.05821 s	0.119 s
AHU algorithm	0.1768 s	0.176 s	0.1787 s	0.1772 s	

Experiment 6:

Number of trees used: 100

Average number of nodes: 1000

	1st trial	2nd trial	3rd trial	Mean time taken	Difference of mean time taken
Hashing algorithm	0.1834 s	0.173 s	0.1767 s	0.1777 s	0.225 s
AHU algorithm	0.4001 s	0.4051 s	0.4036 s	0.4029	

Experiment 7:

Number of trees used: 1000

Average number of nodes: 5

	1st trial	2nd trial	3rd trial	Mean time taken	Difference of mean time taken
Hashing algorithm	0.05671 s	0.05458 s	0.05676 s	0.05602 s	0.109 s
AHU algorithm	0.1668 s	0.1638 s	0.1637 s	0.1648 s	

Experiment 8:

Number of trees used: 1000

Average number of nodes: 100

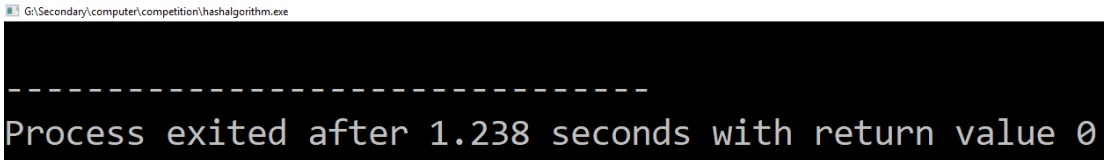
	1st trial	2nd trial	3rd trial	Mean time taken	Difference of mean time taken
Hashing algorithm	0.1722 s	0.1822 s	0.1865 s	0.1803 s	0.164 s
AHU algorithm	0.3255 s	0.3733 s	0.3336 s	0.3441 s	

Experiment 9:

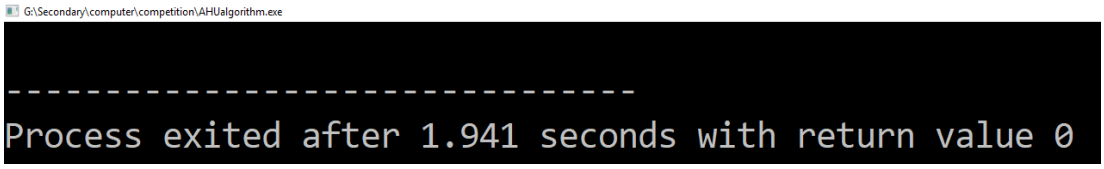
Number of trees used: 1000

Average number of nodes: 1000

	1st trial	2nd trial	3rd trial	Mean time taken	Difference of mean time taken
Hashing algorithm	1.238 s	1.237 s	1.231 s	1.234 s	0.717 s
AHU algorithm	1.941 s	1.96 s	1.951 s	1.951 s	



Example 6.2a: Hashing algorithm, experiment 9, 1st trial result



Example 6.2b: AHU algorithm, experiment 9, 1st trial result

Differences between the mean times taken between the two algorithms are calculated:

	2 trees	100 trees	1000 trees
5 nodes	0.008 s	0.047 s	0.109 s
100 nodes	0.024 s	0.119 s	0.164 s
1000 nodes	0.186 s	0.225 s	0.717 s

Note that positive values denote less time taken for the hashing algorithm compared to the AHU algorithm.

6.3 Interpretation and Analysis

It can be determined from the test data that the hashing algorithm works faster than the AHU algorithm in all scenarios varying M and N. On top of that, two trends can be observed:

1. The difference between the mean times taken between the hashing algorithm and the AHU algorithm increases with the number of trees M. Since comparing integers is more efficient than comparing strings (canonical names), therefore as the number of comparisons increase with the number of trees, the difference in efficiency of the algorithms increases, asymptotically tending to infinity.
2. The difference between the mean times taken between the hashing algorithm and the AHU algorithm increases with the average number of nodes N. Since the length of the canonical names produced from the AHU algorithm increases with the number of nodes, therefore the time taken for comparing two canonical names increases. On the contrary, the hash (in the form of an integer) produced from the hashing algorithm remains at a constant range under the modulo operation, hence the time taken for comparing two hashes does not increase with the average number of nodes. Moreover, although both algorithms involves sorting processes when computing the canonical names and hashes, sorting integers is more efficient than sorting strings. Therefore as the average number of nodes increase, the difference in efficiency of the algorithms increases, asymptotically tending to infinity.

These observations agree with the results predicted by the theoretical evaluation.

7 Application

When a webpage is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.[5] The model can be constructed as a tree of objects. Considering every website as a tree, the hashing algorithm can be used to check whether there are websites which are structurally identical. This is extremely useful especially when determining whether a website is exhibits plagiarism of another webpage. This can help identify plagiarism cases more easily and protect the rights of web developers.

8 Modifications

8.1 Unrooted trees

The hashing algorithm can be modified to compare unrooted trees. This can be done by converting the each unrooted tree into a rooted one with its centre as its root. If there are two centres, the unrooted tree will be hashed twice with each centre being the root. This concept is similar to that discussed here.[6]

8.2 Unlabelled trees

The hashing algorithm can be modified to compare unlabelled trees. This can be done by relabelling all nodes with the integer 1.

8.3 Improvement on soundness

As explained in section 5.2 there are trees which are not isomorphic but produce the same hash. This is a disadvantage of using hashing algorithms, as it is sometimes inaccurate. One can decrease the chance of this happening by repeating the hashing process with different pairs of primes, taking the intersection of the results from these trials. Another trick is to use larger primes, as mentioned previously. These measures will increase the accuracy of the algorithm, and the benefit can be calculated using the expressions in the aforementioned section.

8.4 Similar trees

The proposed algorithm can only detect whether two trees are isomorphic. If the label of a single node of the tree is changed, the hash of that tree will be changed as well. Therefore, the hashing algorithm cannot determine the structural similarity between two trees. The possibility of modification to include this type of evaluation for an array of trees is left as a topic for further discussion.

9 Conclusion and Further Discussion

Though it can be observed that the hashing algorithm proposed in this paper is indeed more efficient in determining tree isomorphism compared to the AHU algorithm, especially for a large number of trees or nodes, there are still problems yet to be addressed. For example, whether the increase in speed of the hashing algorithm is adequate compensation for its occasional inaccuracy is a trade-off to be further investigated. Theory in how the hashing algorithm may be improved to suit daily-life scenarios or whether there are better methods to determine tree isomorphism in specific cases is still to be developed. Experiments keeping track on the number of times the hashing algorithm incorrectly determines two non-isomorphic trees as isomorphic can also be done to validate the theoretical deductions in this paper. And as mentioned above, an interesting study might involve finding an algorithm to determine the structural similarity between two trees effectively and accurately, something this algorithm is unable to achieve.

References

[1] http://mathworld.wolfram.com/Tree.html

[2] http://mathworld.wolfram.com/LabeledTree.html

[3] https://www.geeksforgeeks.org/tree-isomorphism-problem/

[4] https://logic.pdmi.ras.ru/~smal/files/smal_jass08.pdf

[5] https://www.w3schools.com/js/js_htmldom.asp

[6] https://logic.pdmi.ras.ru/~smal/files/smal_jass08.pdf

Photo