Advances in Computer, Signals and Systems (2023)
Clausius Scientific Press, Canada

DOI: 10.23977/acss.2023.070312
ISSN 2371-8838 Vol. 7 Num. 3

A Study of Isomorphic Trees in Programming Competitions

Zhiyong Feng, Yike Shi, Junping Shi*

The School of Computer Science and Engineering, Jishou University, Jishou, Hunan, 416000, China

**Corresponding author*

Keywords: ACM programming, tree isomorphism determination algorithm, tree isomorphism, algorithm optimization

Abstract: Tree isomorphism refers to the question whether two trees are exactly the same when the structure and node labels are the same. Tree isomorphism algorithm can be used to judge whether two trees are similar and classify similar trees. Tree isomorphism algorithms are usually implemented using depth-first search and hash tables, and there are also methods using graph theory and linear algebra algorithms. Tree isomorphism algorithm is widely used in many fields, such as bioinformatics, computer science and mathematics. It can help researchers better understand and analyze tree-structured data and provide a basis for subsequent research and development. At the same time in college students programming competition for the problem of tree isomorphism is more widely studied, usually use Algorithm of Aho, Hopcroft, and Ullman(AHU) to solve the isomorphism tree, but simple AHU algorithm time complexity is, cannot well solve the problem in the competition, consider to improve the AHU algorithm. This paper starts from the definition of homogenous tree, tells the basic concept of homogenous tree, then introduces the principle and implementation of naive AHU algorithm and analyzes its time complexity, and then improves the naive AHU algorithm, which greatly improves the running efficiency of AHU algorithm, and finally introduces the application of improved AHU in the form of program design competition.

1. Introduction

Be advised that papers in a technically unsuitable form will be returned for retyping. After returned the manuscript must be appropriately modified. College programming competitions are very high in gold among college competitions, with the ICPC [1-3] being known as the Olympics of college programming competitions, and the data structures and algorithms [4-6] that are the focus of the programming competitions are the focus of research, with graph isomorphism and tree isomorphism problems being the focus of it. In November 2015, University of Chicago mathematician and computer scientist László Babai announced that he has proved that the graph isomorphism problem can be solved in quasi-polynomial time. The isomorphic tree problem is a classical problem in computer science that involves comparing two trees and determining whether they are isomorphic. The isomorphic tree problem has a wide range of applications in computer

science and graph theory. Different advances have been made both at home and abroad for isomorphism problems, such as: maximal out-of-plane graph isomorphism [7-8], tree graph isomorphism [9,10], most of the domestic and international literature proposes the use of intelligent algorithms to solve isomorphism problems or related problem discussions, literature [11] uses genetic algorithms to solve isomorphism problems, converting them to problems with minimum values by mapping the population code to the solution space and performing genetic operations to find the optimal value of the problem, and also by neural network [12] algorithms. All of the above literature provides useful solution methods, but there is still much to be desired.

In the ACM-ICPC competition for isomorphic trees generally uses the tree isomorphism problem existence validity algorithm-Algorithm of Aho, Hopcroft, and Ullman (AHU) to determine whether two rooted trees are isomorphic. However, the time complexity of the plain AHU algorithm is too high and the memory overhead is too high, so the plain AHU algorithm is not used in actual competitions. So we consider to improve the AHU algorithm to reduce the complexity of the algorithm and to solve the problems in the competition better. In practical applications such as network protocol design, the improved AHU algorithm can quickly compare the structure of two packets and use it to determine whether they are identical.

2. The of Study of Isomorphic Trees

2.1. Definition of Isomorphic Tree

Trees can be classified as rooted trees and unrooted trees. A rooted tree has a definite root node, while an unrooted tree has an indefinite root, and any node can be used as the root of the tree. So the problem of tree isomorphism is also divided into rooted tree isomorphism and unrooted tree isomorphism. The definition of tree isomorphism can be simply understood as given two trees T1 and T2, two trees are said to be isomorphic if T1 can become T2 by swapping left and right children several times. For example, the two trees given in Figure 1 are isomorphic, because when we swap the left and right children of nodes A, B, and G of one of the trees, we get another tree.

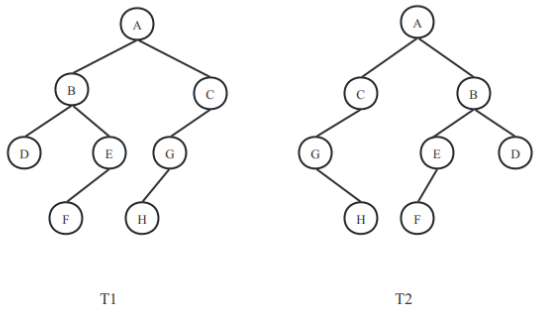

Figure 1: Example of an isomorphic tree.

2.2. Rooted Tree Isomorphism

For two rooted trees $T_1(V_1, E_1, r_1)$ and $T_2(V_2, E_2, r_2)$, if there is a double shot $\varphi : V_1 \to V_2$, such that $\forall\, u, v \in V_1(u, v) \in E_1 \leftrightarrow (\varphi(u), \varphi(v)) \in E_2$ and $\varphi(r_1) = r_2$ holds, then the rooted tree $T_1(V_1, E_1, r_1)$ and $T_2(V_2, E_2, r_2)$ isomorphism are said to be present.

2.3. Rootless Tree Isomorphism

For two unrooted trees $T_1(V_1, E_1)$ and $T_2(V_2, E_2)$, a rooted tree $T_1(V_1, E_1)$ and $T_2(V_2, E_2)$ is said to be

isomorphic if there exists a bijection $\varphi : V_1 \to V_2$ such that $\forall u, v \in V_1(u, v) \in E_1 \leftrightarrow (\varphi(u), \varphi(v)) \in E_2$holds. Simply put, two trees are said to be isomorphic if the tree T_1 and T_2 can be made identical by renumbering all nodes of the tree T_1.

2.4. The Question Conversion

In fact, the unrooted tree isomorphism is convertible to the rooted tree isomorphism problem. For two unrooted trees two unrooted trees and, the specific steps are as follows.

Step 1, find all the centers of gravity of the two unrooted trees respectively

Step 2, if the centers of gravity are different, then the two unrooted trees must not be isomorphic.

Step 3, if the number of centers of gravity are both 1, marked as n_1 and n_2 respectively, then if there is a root tree $T_1(V_1, E_1, n_1)$ and $T_2(V_2, E_2, n_2)$ isomorphism, then there is no root tree $T_1(V_1, E_1)$ and $T_2(V_2, E_2)$ isomorphism, and vice versa.

Step 4, If the number of centers of gravity are both 2 and label their centers of gravity as n_1,n_1' and n_2,n_2' respectively, then if there is a root tree $T_1(V_1, E_1, n_1)$ and $T_2(V_2, E_2, n_2)$ isomorphism or $T_1(V_1, E_1, n_1')$ and $T_2(V_2, E_2, n_2')$ isomorphism, then there is no root tree $T_1(V_1, E_1)$ and $T_2(V_2, E_2)$ isomorphism. Conversely, it is not isomorphic.

According to the above determination steps, we can know that once we solve the rooted tree isomorphism problem, then we can convert the unrooted tree isomorphism problem to the rooted tree isomorphism problem according to the above steps, and then we can solve the unrooted tree isomorphism problem. Therefore, the complexity of the algorithm for solving the rooted tree isomorphism problem is the same as that of the unrooted tree isomorphism problem.

If the centers of gravity are different, then the two unrooted trees must not be isomorphic.

3. The AHU Algorithm

3.1. Definition of AHU

The plain AHU algorithm is based on bracket sequences, where a rooted tree has a unique legal bracket sequence and a tree's bracket sequence is stitched together by the bracket sequences of its subtrees. The tree corresponding to the new bracket sequence is isomorphic to the tree corresponding to the original bracket sequence if that edge subtree bracket sequence is spliced in that order.

3.2. Principle of AHU

Tree isomorphisms have transferability. That is, if T_1 and T_2 isomorphism, T_2 and T_3isomorphism, then T_1 and T_3 isomorphism. Then the recursive algorithm for finding the tree bracket order is introduced, splicing the subtree in backtracking and awarding the sequence with small dictionary order in splicing first. And the final result is noted as NAME. The algorithm steps are shown in Figure 2.

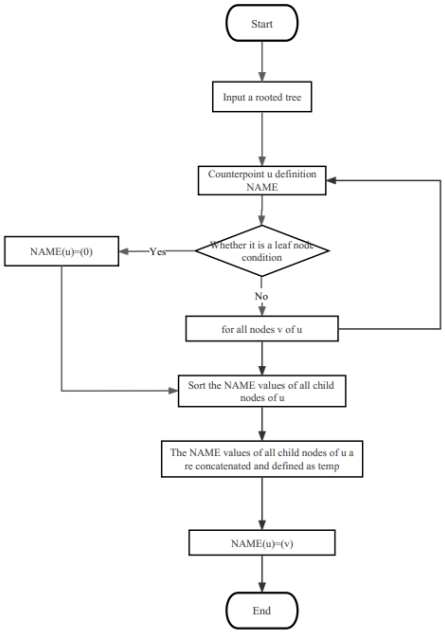

Figure 2: Basic flowchart of AHU algorithm.

The NAME of the subtree rooted at node r is taken as the NAME of node r, noted as NAME(r), then for a rooted tree $T_1(V_1, E_1, r_1)$ and $T_2(V_2, E_2, r_2)$, if NAME(r_1)=NAME(r_2), then T_1 and T_2isomorphic. For a rooted tree with n nodes, assuming that the rooted tree is chained, then the maximum length of the nodes can be n. Then the complexity of the NAME algorithm is a constant multiple of that of $(1 + 2 + \cdots + n)$, the time complexity of the plain AHU algorithm is $O(n^2)$.

3.3. Improvements of AHU

The drawback of the plain AHU algorithm is that the length of the tree's NAME may be too long, so the tree is hierarchically divided, and the shortest distance from the node at the i-th level to the root is i. The NAME located at the ith level can be obtained by splicing only the NAME of the node located at the i-th level. So the original NAME of the node can be replaced by its rank within the layer, and then the original spliced node NAME is replaced by an array of joined elements.

First note that the total length of the NAME obtained by the splicing of the i-th layer is the sum of the degrees of the ith layer, that is, the total number of points of the layer, which is expressed below. THE NEXT STEP OF THE ALGORITHM WILL TREAT THESE NAMES AS STRINGS (ARRAYS) AND SORT THEM, AND THEN TRANSPOSE THEM TO THEIR RANK WITHIN THE LAYER (I.E., REMAP THEM TO A TREE). While sorting algorithms can use cardinality sorting, cardinality sorting [12] can complete the sorting in a time frame, where L represents the size of the character set.

4. Practical Application

4.1. Description of the Title

Take the title SPOJ-TREEISO as an example.

Given two undirected trees T1 and T2 with equal number of vertices N ($1 \leqslant N \leqslant 100{,}000$) numbered 1 to N, find out if they are isomorphic.Two trees T1 and T2 are isomorphic if there is a bijection f between the vertex sets of T1 and T2 such that any two vertices u and v of T1 are adjacent in T1 if and only if f(u) and f(v) are adjacent in T2.

Input:

The first line of input contains the number of test cases nTest (1<= nTest <= 400). Each test case contains:

The first line contains the number of nodes N.

Each of next N-1 lines contain two integers A, B, denoting that there is an edge in T1 between nodes A and B ($1 \leq A, B \leq N$).

Each of next N-1 lines contain two integers A, B, denoting that there is an edge in T2 between nodes A and B ($1 \leq A, B \leq N$).

The sum of N over all test cases will not exceed 100,000.

Output:

For each test case print YES if T1 and T2 are isomorphic and NO otherwise.

4.2. Problem Analysis

First of all, the description of the problem shows that this is a tree isomorphism problem, which requires to determine whether two unrooted trees are isomorphic or not. The AHU algorithm is considered, but the complexity of the plain AHU algorithm is not enough to pass the problem, but the complexity of the optimized AHU algorithm above is good enough to pass the problem.

5. Conclusion

Programming competitions are constantly evolving, and more and more algorithms are being studied, and the requirements for the efficiency of the algorithms are getting higher and higher, especially the programming competitions nowadays pay special attention to the solution of tree-like problems, so it is essential and necessary to study the problem of isomorphic trees in ACM competitions, and it is also necessary for ACM to focus on the efficiency of the algorithms to reduce their time complexity. In this thesis, the AHU algorithm is improved to reduce the time complexity to $O(n)$, which is important for the practical application of the algorithm.

References

[1] Yonghui Wu and Jiande Wang. Algorithm Design Practice for Collegiate Programming Contests and Education. CRC Press, 2018

[2] Ferrada Héctor. A sorting algorithm based on ordered block insertions. Journal of Computational Science, 2022, 64

[3] Rick H. de Boer and Cassio P. de Campos. A retrospective overview of International Collegiate programming contest data. Data in Brief, 2019, 25: 104382.

[4] Dovier A, Formisano A, Gupta G, et al. Parallel Logic Programming: A Sequel[J]. arXiv e-prints, 2021.

[5] Andre Droschinsky and Nils Kriege and Petra Mutzel. Faster Algorithms for the Maximum Common Subtree Isomorphism Problem. CoRR, 2016, abs/1602.07210

[6] V. Arvind et al. The isomorphism problem for k -trees is complete for logspace. Information and Computation, 2012, 217: 1-11.

[7] Mitchell S, Beyer T, Jones W. Linear Algorithms for Isomorphism of Maximal Outerplanar Graphs. Journal of the Acm, 1979, 26(4):603-610.

[8] Cole R, Crochemore M, Galil Z, et al. Optimally fast parallel algorithms for preprocessing and pattern matching in one and two dimensions[C]// IEEE Foundations of Computer Science. IEEE Computer Society, 1993.

[9] Buss S R. Alogtime Algorithms for Tree Isomorphism, Comparison, and Canonization. 1997.

[10] Wang Y K, Fan K C, Liu C W, et al. [IEEE 1995 IEEE International Conference on Evolutionary Computation - Perth, WA, Australia (29 Nov.-1 Dec. 1995)] Proceedings of 1995 IEEE International Conference on Evolutionary Computation - Adaptive optimization for solving a class of subgraph isomorp[J]. 1995, 1:44.

[11] Hopfield J. Neural computation of decisions in optimization problems. Biological Cybernetics, 1985, 52.

[12] Guo Zhengchu and Hu Ting and Shi Lei. Distributed spectral pairwise ranking algorithms. Inverse Problems, 2023, 39(2)