Presenting a Parallel Algorithm for Constructing Cartesian Trees and its Application in Generating Separate and Free Trees

Azad Shojaei [*1], Abdoljabar Asadi[2]

[1] Department of Computer, Saghez Branch, Islamic Azad University, Saghez, Iran
[2]Department of Computer, Sanandaj Branch, Islamic Azad University, Sanandaj,Iran
[*]Azad.Shojaei@gmail.com

Abstract: In the present study, an n-element array is introduced in order to present a parallel algorithm for generating Cartesian trees. Furthermore, required time, space, and operations of the algorithm are investigated. The all nearest smaller values problem is explained briefly because it influences on the problems of Cartesian trees. Two applications of Cartesian trees in generating suffix and separating trees are explored. Suffix trees and separating trees are among the most important data structures in processing fields.
[Azad Shojaei, Abdoljabar Asadi. **Presenting a Parallel Algorithm for Constructing Cartesian Trees and its Application in Generating Separate and Free Trees.** *J Am Sci* 2012;8(6):811-813]. (ISSN: 1545-1003). http://www.jofamericanscience.org. 103

Key words: Parallel algorithm, Cartesian tree, ANSV, Separating tree, Suffix tree

1. Introduction

Introduced by Vuillemin, Cartesian tree is a data structure on an n-position sequence like A where $A = \{a_1, a_2, \ldots, a_n\}$ which is a binary tree and it has two attributes: (1) Tree is generated in a heap-sort way; (2) Inorder tree traversal generates an ordered sequence. The root of the tree is the minimum value of (a_i) in the sequence. There is a Cartesian tree with nodes of $\{a_1, a_2, \ldots, a_{i-1}\}$ under the left tree and one with nodes of $\{a_{i+1}, a_{i+2}, \ldots, a_n\}$ under the right tree. If the positions of the sequence are different, the generated Cartesian tree will be a unique one, otherwise it cannot be unique. Generating a Cartesian tree, in a sequential form on a processor, is accomplished using an n-position sequence in the time of $O_{(n)}$ and a free tree in the time of $O_{(nlogn)}$ [1].

In this article, a parallel algorithm is introduced for generating Cartesian trees, which is in multi-algorithmic time, linear space, and linear operations [2]. This algorithm is based on the method of division and solution and presented in two versions. The difference between these two versions is related to the operation of "merging" which can be in two forms of sequential or parallel. The first version which applies "sequential merge" runs on CREW PRAM model in the time of O(min{dlogn, n}) where "d" is the depth of the tree. The second version is a more complicated and it runs on CREW PRAM model in the time of $O(\log^2 n)$. Space and operations are linear in both versions. The algorithm generating Cartesian trees in linear form is located on a core with a speed of about 3x but in parallel form it is located on 32 cores with a speed of 30x. By adding multi-threaded attribute and using 32 cores, speed can be increased to 45x. In the following paragraphs, the application of Cartesian trees in generating separate and suffix trees is briefly explained and their time order is specified. Moreover, there is a short review of the problem of all nearest value which helps solve the problems.

2. The Parallel Algorithm of Cartesian Trees

The parallel algorithm for generating Cartesian trees is based on the method of division and solution and it acts in a reversal way. The algorithm receives the n-position array of A as the input and divides it into two sub-arrays, then generates a Cartesian tree for each sub-array and finally merges the results in order to construct the final Cartesian tree for the input array of A. Concepts of "right spine" and "left spine" for a Cartesian tree are defined as follow:

Right spine (left spine) includes all of the nodes located on a path which begins from the root and continues to the most right (left) part of the tree node. It is vivid that the most left and the most right elements are the first and the last elements in the ordered sequence, respectively. Merging operation is carried out by combining the right spine in the left sub-tree with the left spine in the right sub-tree. The first version of the algorithm (Algorithm 1) is partially performed in a linear way; however, the second version is quite parallel. Algorithm 1 is performed in the time of where "d" is the depth of the optimal tree. In most cases; however, the algorithm takes less time. For instance, in the sequence of {1 , 2, 3 , … , n} the algorithm is carried out in the time of O(lign) although the depth of the resulted tree is

"n". This algorithm receives an array with n positions (nodes) and divides it into two sub-arrays in a reversal way. It generates a Cartesian tree for each sub-array and afterwards combines the right and left Cartesian trees together. The algorithm carries on breaking the array into two sub-arrays till there are less than two positions in each sub-array. The complete code of the algorithm in C language is presented in [2]. An important point about the algorithm is that heap and inorder attributes remain in all steps and phases of the algorithm. In its basic form, a Cartesian tree with a node holds both attributes. This algorithm need the space of O(n) and the operation of O(n). Another point is that the algorithm applies a sequential merging although it carries out the operation of reversal recall in a parallel way. In its worst form, this type of merging can have an equal time to the depth of the tree. To solve this problem, the second version was introduced which performs a binary search on every spine, divides the spine into two sub-trees base on a specific value, and replaces the sequential merge with a parallel merge. The second algorithm carries out the breaking operations on the spines. The second version of the algorithm is used to generate Cartesian trees with the operation order of O(n), space O(n), and time of $O(\log^2 n)$ on CREW PRAM model. In this algorithm, all of the parallel breakings and combinations are carried out in the time of O(logn). The algorithm does the parallel combining operation in the time of O(logn) and the reversal recall depth of the algorithm is of order O(logn); therefore, the whole time of the algorithm is of order $O(\log^2 n)$.

3.The All Nearest Smaller Values Problem (ANSV)

For each position in a sequence, this problem finds the nearest smaller position than it from right and left hand. This problem was introduced by Berkman as a subroutine to help solve a lot of problems in parallel computations [5]. Berkman's algorithm for the problem of ANSV has the operation order of O(n) and the time of O(loglogn) on CREW PRAM model. ANSV can be used to generate Cartesian trees. The root of the tree is the minimum value of the sequence and for every other node; the parent is the closest previous or following smaller value provided that these values are available. If both values are available, the bigger value is placed in the root. Therefore, by using ANSV problem it is likely to construct Cartesian trees in linear time base.

5. Separating trees

In the pattern matching permutations problem, a permutation T = (t1, t2, ..., tn) of 1, 2, ..., n is considered as the main text and another permutation P = (p1, p2, ..., pk) of 1, 2, ..., k, $k \leq n$ is taken as a pattern. The problem is to find a subsequence with length k like T' of the sequence of T, where $T = (t_{i1}, t_{i2}, ..., t_{ik})$ with $i_1 < i_2 < ... < i_k$, and all the elements of T' are ordered based on the permutation of P and also $t_{ir} < t_{is}$ iff $p_r < p_s$. This problem was proposed by Wilf. If T' is available, it can be said that T contains P or P matches in T. When P = (1, 2, ..., k) is a sequence of numbers, then the problems finds an increasing subsequence of length k in T [3]. According to the source definition, P is separable if it does not contain the sub-pattern (3 , 1 , 4 , 2) or its reverse that is (2 , 4 , 1 , 3). Separable permutation is defined as follow:

Consider a subsequence of consecutive elements. This subsequence is located in a special order in the sequence. Any consecutive subsequence of elements can be reversed in the main sequence. This is a separable permutation if it can be ordered using the ordering algorithm mentioned in [3].

A separating tree for a separable permutation P is a binary tree T with leaves $(p_1, p_2, \ldots, p_k)$ such that a contiguous range will be generated in the main sequence for each node of the tree if the leaves of the sub-tree with the root v are $p_i, p_{i+1}, \ldots, p_{i+j}$. Yugandhar and Saxena have presented parallel algorithm for generating separating trees. The firs algorithm runs in O(logn) time and uses $O(n^2)$ operations. This algorithm is not efficient compared to the sequential algorithm which uses O(n) operations. They have introduced another parallel algorithm which is more efficient. This algorithm runs in O(dlogn) time and uses O(nd) operations, where "d" is the depth of the optimal tree. This algorithm is efficient when "d" is not big.

The main difficulty in generating a separating tree is to find the node v which if considered as the root of a sub-tree, its leaves are a contiguous and ordered range in the permutation. To efficiently solve this problem, ANSV concepts which were referred above can be applied so that construction time and operations will be optimized. In this case, the algorithm needs $O(\log^2 n)$ time and $O(n\log^{1.5} n)$ operation on CREW PRAM model and O(log2n) time and O(nlognloglogn) operation on CRCW PRAM mode.To build a Cartesian tree, the smallest value should be put in the root, and ANSV concept is used in finding that value. And due to this attribute of Cartesian trees that inorder tree traversal displays an ordered sub-tree, these concepts can be used in generating separating trees to optimize the order of the algorithm.

6. Suffix Trees

For a string s of length n in a character set $\sum \leq \{1, 2, \ldots, n\}^1$ the suffix tree is a data structure that stores all the suffixes of s in a pat tree [2] and in O(|t|) time supports searching operation for any string $t \in \sum^*$ in s. In addition, suffix trees support many other string-related operations like longest common substring, maximal repeats, longest repeated substrings, and so on. To construct a suffix tree, first a suffix array should be generated and then it should be converted to a suffix tree. The suffix array is built using a parallel algorithm called "Skew" which is administered in a partially linear way for an input range of length n in an alphabetical sequence. Suffix and algorithm trees with linear time were first introduced by Wiener. Wiener's algorithm on an n-member sequence of characters works in linear time. This algorithm is naturally a sequential one.

The first linear algorithm for generating suffix trees was first proposed by Apostolico *et al.* This algorithm uses a parameter $0 < \epsilon \leq 1$ in $O(\frac{1}{\epsilon} logn)$ time, order operations $O(\frac{n}{\epsilon} logn)$, and $O(\log^{1+\epsilon})$ space on CREW PRAM model.

The later algorithm improved the order of the previous algorithm and was administered in operations, linear space, and time of $O(\log^4 n)$ on CREW PRAM model. The optimal linear algorithm for building suffix trees could reduce administering time to $O(\log^2 n)$ by using the concepts of ANSV and Cartesian trees. The explained attributes of ANSV and Cartesian trees are extremely effective in generating a suffix tree [2] and they can reduce the order of the algorithm to an optimized value.

7. Conclusion

In this brief article, Cartesian trees and a linear algorithm to construct them were introduced. This algorithm has two versions. In the first version, combination operation is carried out sequentially in O(min { dlogn , n}) time and with order operation and space of O(n). In the second version one, the algorithm is applied in a completely parallel way because it uses parallel combination operation and changes the administering time to $O(\log^2 n)$. Both versions are administered on CREW PRAM model. Then separating and suffix trees were introduced briefly. The time of this algorithm can be optimized using the attributes of Cartesian trees.

Corresponding Author:
Azad Shojaei
Department of computer, saghez Branch, Islamic Azad University, Saghez, Iran.
E-mail: Azad.Shojaei@gmail.com

References

[1] Brian, C. D. Raghuveer Mohan, 2010.Building Cartesian Trees from Free Trees. August 28.
[2] Guy E. Belloch and Julian Shun, 2011. A Simple Parallel Cartesian Tree Algorithm And its Application to Suffix Tree Construction.
[3] Yijie Han. Sanjeev Saxena and Xiaojun Shen. 2010. An efficient Parallel Algorithm for building The Separator Tree. Journal of Parallel Distribution and Computer 70, 625 – 629.
[4] Weiner, P, 1973. Linear pattern matching algorithm, 14th Annual IEEE Symposium on Switching and Automata Theory, pp. 1-11.
[5] Dommelen, L. van and Rundensteiner, E. A, 1989. Fast, adaptive summation of point forces in the two dimensional Poisson equation. Journal of Computer and Physics, 83(1):126-147.

5/5/2012