Project Category- Advanced Data Structure

Project Name/Purpose- To construct a **Cartesian Tree** from a given input array/inorder array in linear [**O(N)**] time.

Programming Paradigm/Algorithm Used- Property of heaps .

Data Structure Used- A C styled-structure (“**struct”**) to hold the node of the cartesian tree.

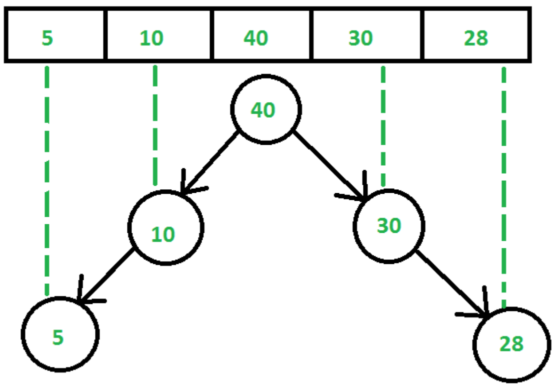
What is a Cartesian Tree?

A Cartesian tree is a tree data structure created from a set of data that obeys the following structural invariants:

1. *The tree obeys in the min (or max) heap property - each node is less (or greater) than its children.*
2. *An inorder traversal of the nodes yields the values in the same order in which they appear in the initial sequence.*

Suppose we have an input array- {5,10,40,30,28}.

Then the **max-heap** Cartesian Tree would be-

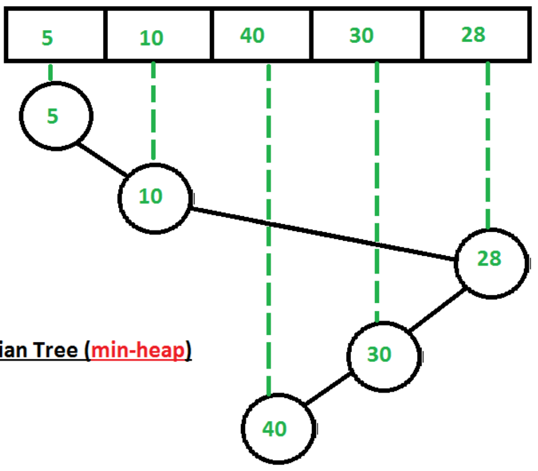


A sequence and its corresponding Cartesian tree

Note that this is a max-heap Cartesian Tree.

Similarly a min-heap Cartesian Tree is also possible.

A **min-heap Cartesian Tree** of the above input array will be-



A sequence and its Cartesian Tree (min-heap)

Note-

1) Cartesian Tree is not a height-balanced tree.
2) Cartesian tree of a sequence of distinct numbers is always unique.

Proof -
We will prove the above two statements

1) Cartesian Tree is not a height-balanced tree.

This doesn't require any proof. A Cartesian- tree acts like a min/max heap and heaps are not height balanced. Hence Cartesian trees are also height-unbalanced.

2) Cartesian tree of a sequence of distinct numbers is always unique.

We will prove this using induction.
As a base case, if the input sequence is empty, then the empty tree is the unique Cartesian tree over that sequence. For the i nductive case, assume that for all trees containing n' < n elements, there is a unique Cartesian tree for each sequence of n' n odes. Now take any sequence of n elements. Because a Cartesian tree is a min-heap, the smallest element of the sequence must be the root of the Cartesian tree. Because an inorder traversal of the elem ents must yield the input sequence, we know that all nodes to the left of the min element must be in its left subtree and simi larly for the nodes to the right. Since the left and right subtree are both Cartesian trees with at most n-1 elements in them (since the min element is at the root), by the induction hypothesis there is a unique Cartesian tree that cou ld be the left or right subtree. Since all our decisions were forced, we end up with a unique tree, completing the induction.

Naive Method-

See the below link for the naive method

http://www.geeksforgeeks.org/construct-binary-tree-from-inorder-traversal/

However, the above program constructs the “special binary tree” (**which is nothing but a Cartesian tree**) in **O(N^2)** time which is very inefficient.

An O(N) Algorithm -

It's possible to build a Cartesian tree from a sequence of data in linear time.

Beginning with the empty tree, scan across the sequence from the left to the right adding new nodes as follows:

Step 1.
Position the node as the right child of the rightmost node.

Step 2.
Scan upward from the node's parent up to the root of the tree until a node is found whose value is **g reater** than the current value.

Step 3.
If such a node is found, set its right child to be the new node, and set the new node's left child to be the previous right child.

Step 4.
If no such node is found, set the new child to be the root, and set the new node's left child to be the previous tree.

Time Complexity-

If you see the code at first sight, then it will create an impression of having O(N^2) time complexity as there are two loops in the *buildCartesianTree()* function. But actually, it takes linear time only.

The inner **while loop** represent the process in which we scan the tree upwards in the search of finding a suitable place to insert the new element.

A keen observation will prove that in total, the **whole while loop**(i.e- over all values from i=1 to i<n) takes **O(N)** time and not every time.
For more reference go to-
http://wcipeg.com/wiki/Cartesian_tree#Analysis_of_running_time

Hence, the overall time complexity is **O(N)**.

Space Complexity-

We declare a structure for every node as well as three extra arrays- leftchild[], rightchild[], parent[] to hold the indices of left-child, right-child, parent of each value in the input array. Hence the overall **O(4*N)** = **O(N)** extra space.

Application of Cartesian Tree-

- Cartesian Tree Sorting
- A range minimum query on a sequence is equivalent to a lowest common ancestor query on the sequence's Cartesian tree. Hence, RMQ may be reduced to LCA using the sequence's Cartesian tree.
- The treap, a balanced binary search tree structure, is a Cartesian tree of (key,priority) pairs; it is heap-ordered according to the priority values, and an inorder traversal gives the keys in sorted order.
- The suffix tree of a string may be constructed from the suffix array and the longest common prefix array. The first step is to compute the Cartesian tree of the longest common prefix array.

References-

http://wcipeg.com/wiki/Cartesian_tree
https://en.wikipedia.org/wiki/Cartesian_tree