Cartesian Tree Sorting

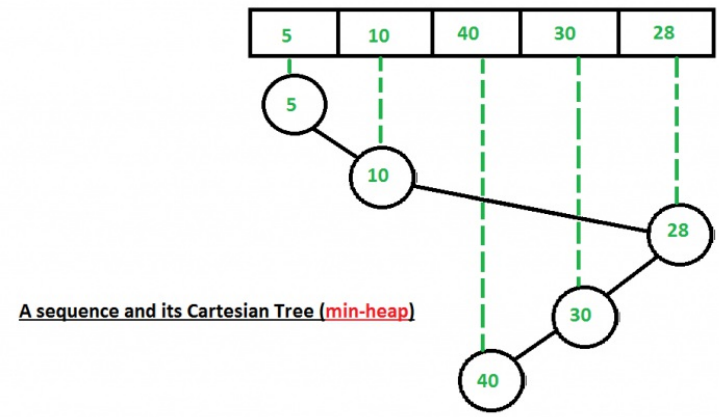
Prerequisites : Cartesian Tree

Cartesian Sort is an Adaptive Sorting as it sorts the data faster if data is partially sorted. In fact, there are very few sorting algorithms that make use of this fact.

For example consider the array {5, 10, 40, 30, 28}. The input data is partially sorted too as only one swap between "40" and "28" results in a completely sorted order. See how Cartesian Tree Sort will take advantage of this fact below.

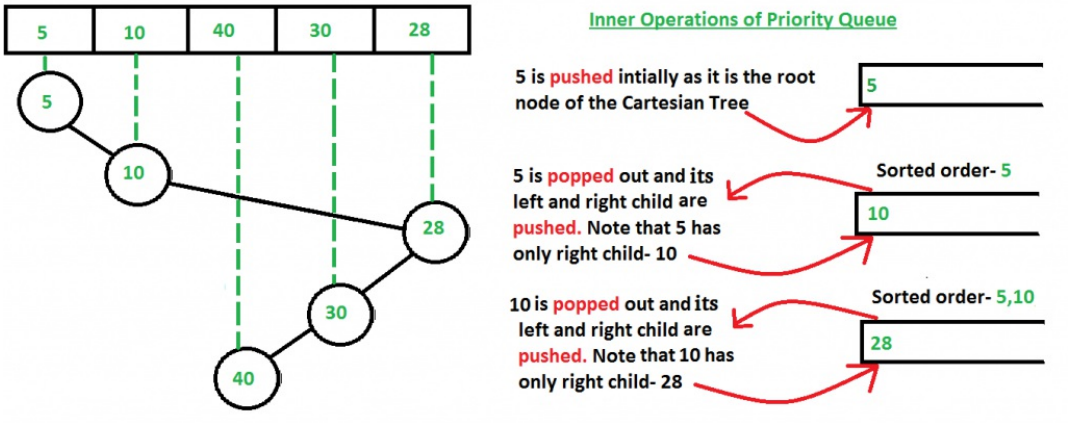
Below are steps used for sorting.

Step 1 : Build a (min-heap) Cartesian Tree from the given input sequence.



Step 2 : Starting from the root of the built Cartesian Tree, we push the nodes in a priority queue.
Then we pop the node at the top of the priority queue and push the children of the popped node in the priority queue in a pre-order manner.

1. Pop the node at the top of the priority queue and add it to a list.
2. Push left child of the popped node first (if present).
3. Push right child of the popped node next (if present).



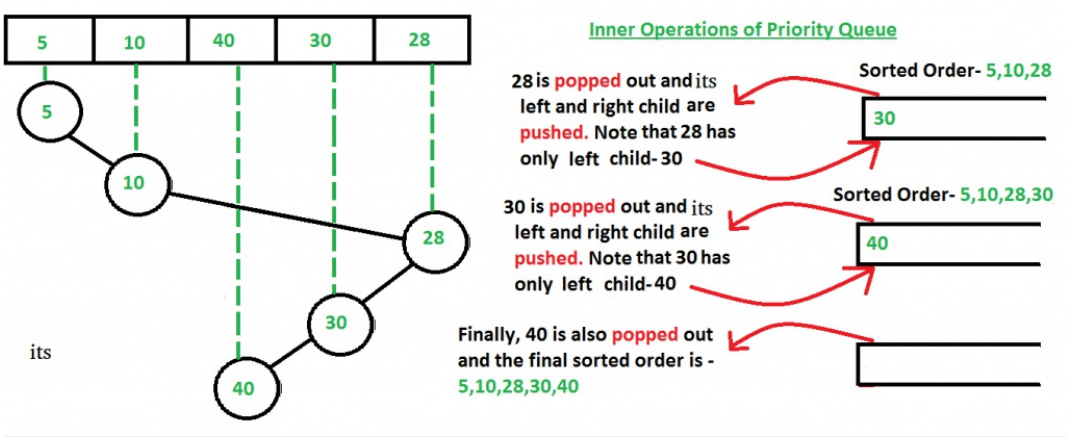
How to build (min-heap) Cartesian Tree?
Building min-heap is similar to building a (max-heap) Cartesian Tree (discussed in previous post), except the fact that now we scan upward from the node's parent up to the root of the tree until a node is found whose value is smaller (and not larger as in the case of a max-heap Cartesian Tree) than the current one and then accordingly reconfigure links to build the min-heap Cartesian tree.

Why not to use only priority queue?
One might wonder that using priority queue would anyway result in a sorted data if we simply insert the numbers of the input array one by one in the priority queue (i.e- without constructing the Cartesian tree).

But the time taken differs a lot.

Suppose we take the input array – {5, 10, 40, 30, 28}

If we simply insert the input array numbers one by one (without using a Cartesian tree), then we may have to waste a lot of operations in adjusting the queue order everytime we insert the numbers (just like a typical heap performs those operations when a new number is inserted, as



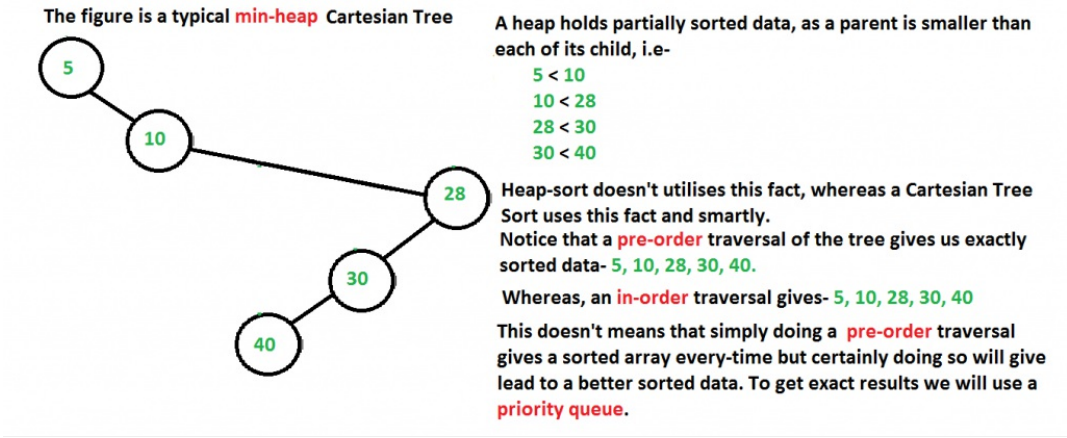
priority queue is nothing but a heap).

Whereas, here we can see that using a Cartesian tree took only 5 operations (see the above two figures in which we are continuously pushing and popping the nodes of Cartesian tree), which is linear as there are 5 numbers in the input array also. So we see that the best case of Cartesian Tree sort is O(n), a thing where heap-sort will take much more number of operations, because it doesn't make advantage of the fact that the input data is partially sorted.

Why pre-order traversal?

The answer to this is that since Cartesian Tree is basically a heap- data structure and hence follows all the properties of a heap. Thus the root node is always smaller than both of its children. Hence, we use a pre-order fashion popping-and-pushing as in this, the root node is always pushed earlier than its children inside the priority queue and since the root node is always less than both its child, so we don't have to do extra operations inside the priority queue.

Refer to the below figure for better understanding-




```
// A C++ program to implement Cartesian Tree sort
// Note that in this program we will build a min-heap
// Cartesian Tree and not max-heap.
#include<bits/stdc++.h>
using namespace std;

/* A binary tree node has data, pointer to left child
  and a pointer to right child */
struct Node
{
   int data;
   Node *left, *right;
};

// Creating a shortcut for int, Node* pair type
typedef pair<int, Node*> iNPair;

// This function sorts by pushing and popping the
// Cartesian Tree nodes in a pre-order like fashion
void pQBasedTraversal(Node* root)
{
   // We will use a priority queue to sort the
   // partially-sorted data efficiently.
   // Unlike Heap, Cartesian tree makes use of
   // the fact that the data is partially sorted
   priority_queue <iNPair, vector<iNPair>, greater<iNPair>> pQueue;
  pQueue.push (make_pair (root->data,root));
```

```
    // Resembles a pre-order trave
rse as first data
    // is printed then the left and t
hen right child.
    while (! pQueue.empty())
    {
     iNPair popped_pair = pQue
ue.top();
   printf("%d ",popped_pair.firs
t);

     pQueue.pop();

  if (popped_pair.second->left !
= NULL)
       pQueue.push (make_pair(
popped_pair.second->left->data
,
                          popped_pai
r.second->left));

       if (popped_pair.second->ri
ght != NULL)
        pQueue.push (make_pair(
popped_pair.second->right->da
ta,
                             popped_
pair.second->right));
    }

    return;
}


Node *buildCartesianTreeUtil(i
nt root, int arr[],
         int parent[], int leftchild[
], int rightchild[])
{
    if (root == -1)
       return NULL;

    Node *temp = new Node;

    temp->data = arr[root];
    temp->left = buildCartesianT
reeUtil(leftchild[root],
               arr, parent, leftchild,
rightchild);

    temp->right = buildCartesian
TreeUtil(rightchild[root],
               arr, parent, leftchild,
rightchild);

    return temp ;
}

// A function to create the Cartes
ian Tree in O(N) time
Node *buildCartesianTree(int ar
r[], int n)
{
    // Arrays to hold the index of
parent, left-child,
    // right-child of each number i
n the input array
    int parent[n],leftchild[n],right
child[n];

    // Initialize all array values as
-1
    memset(parent, -1, sizeof(par
```

```
ent));
   memset(leftchild, -1, sizeof(le
ftchild));
   memset(rightchild, -1, sizeof(
rightchild));

   // 'root' and 'last' stores the in
dex of the root and the
   // last processed of the Cartes
ian Tree.
   // Initially we take root of the
Cartesian Tree as the
   // first element of the input ar
ray. This can change
   // according to the algorithm
   int root = 0, last;

   // Starting from the second ele
ment of the input array
   // to the last on scan across th
e elements, adding them
   // one at a time.
   for (int i=1; i<=n-1; i++)
   {
      last = i-1;
      rightchild[i] = -1;

      // Scan upward from the no
de's parent up to
      // the root of the tree until a
 node is found
      // whose value is smaller th
an the current one
      // This is the same as Step
2 mentioned in the
      // algorithm
      while (arr[last] >= arr[i] &
& last != root)
         last = parent[last];

      // arr[i] is the smallest elem
ent yet; make it
      // new root
      if (arr[last] >= arr[i])
      {
         parent[root] = i;
         leftchild[i] = root;
         root = i;
      }

      // Just insert it
      else if (rightchild[last] == -
1)
      {
         rightchild[last] = i;
         parent[i] = last;
         leftchild[i] = -1;
      }

      // Reconfigure links
      else
      {
         parent[rightchild[last]] =
 i;
         leftchild[i] = rightchild[la
st];
         rightchild[last]= i;
         parent[i] = last;
      }

   }
```

```
    // Since the root of the Cartesi
an Tree has no
    // parent, so we assign it -1
    parent[root] = -1;

    return (buildCartesianTreeUti
l(root, arr, parent,
                        leftchild,
rightchild));
}

// Sorts an input array
int printSortedArr(int arr[], int
n)
{
    // Build a cartesian tree
    Node *root = buildCartesian
Tree(arr, n);

    printf("The sorted array is-\n
");

    // Do pr-order traversal and in
sert
    // in priority queue
    pQBasedTraversal(root);
}

/* Driver program to test above
functions */
int main()
{
    /*  Given input array- {5,10,4
0,30,28},
        it's corresponding unique C
artesian Tree
        is-

        5
         \
          10
           \
            28
           /
          30
         /
        40
    */

    int arr[] = {5, 10, 40, 30, 28};
    int n = sizeof(arr)/sizeof(arr[0
]);

    printSortedArr(arr, n);

    return(0);
}
```

Output :

```
The sorted array is-
5 10 28 30 40
```

Time Complexity : O(n) best-case behaviour (when the input data is partially sorted), **O(n log n)** worst-case behavior (when the input data is not partially sorted)

Auxiliary Space : We use a priority queue and a Cartesian tree data structure. Now, at any moment of time the size of the priority queue doesn't exceeds the size of the input array, as we are constantly pushing and popping the nodes. Hence we are using O(n) auxiliary space.

References :

https://en.wikipedia.org/wiki/Adaptive_sort

http://11011110.livejournal.com/283412.htmlhttp://gradbot.blogspot.in/2010/06/cartesian-tree-sort.htmlhttp://www.keithschwarz.com/interesting/code/?dir=cartesian-tree-sorthttps://en.wikipedia.org/wiki/Cartesian_tree#Application_in_sorting

This article is contributed by **Rachit Belwariar**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

2 Comments Category: Advanced Data Structure Sorting

Related Posts:

- Smallest Subarray with given GCD
- GCDs of given index ranges in an array
- Disjoint Set Data Structures (Java Implementation)
- Tarjan's off-line lowest common ancestors algorithm
- Cartesian Tree
- Sparse Set
- Persistent data structures
- Centroid Decomposition of Tree

Previous post in category Next post in category

Quy • a month ago

Why when i try on IDE abow with array {45, 51, 34, 87, 44, 88, 1, 5, 4, 3, 8, 2};
I got a result:

The sorted array is-
1 2 3 8 4 5 34 44 88 87 45 51

Rachit Belwariar ➚ Quy • a month ago

The algorithm is completely fine. The bug is in the way we are using priority queue, as we have made a priority queue of pointers to the node of Cartesian Tree, instead of data . So it is sorting it according to the address of the nodes of Cartesian Tree.
This will be fixed soon. Thanks for pointing out.

Popular Posts

- Top 10 Algorithms and Data Structures for Competitive Programming

- Top 10 algorithms in Interview Questions
- How to begin with Competitive Programming?
- Reflection in Java
- Memory Layout of C Programs
- Push Relabel Algorithm
- Heavy Light Decomposition
- Sorted Linked List to Balanced BST
- Generics in Java
- Aho-Corasick Algorithm for Pattern Searching
- Binary Search , QuickSort , MergeSort , HeapSort

- Common Interview Puzzles
- Interview Experiences
- Advanced Data Structures
- Design Patterns
- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Pattern Searching
- Divide & Conquer
- Geometric Algorithms
- Searching
- Sorting
- Hashing
- Analysis of Algorithms
- Mathematical Algorithms
- Randomized Algorithms
- Recursion

Tags

Adobe Advance Data Structures Advanced Data Structures Amazon array Backtracking Bit Magic C++ CN c puzzle D-E-Shaw DBMS design Divide and Conquer Dynamic Programming Flipkart GATE GATE-CS-C-Language GATE-CS-DS-&-Algo GATE-CS-Older GFacts Goldman Sachs Google Graph Greedy Algorithm Hashing Interview Experience Java MAQ Software MathematicalAlgo Matrix Microsoft Morgan Stanley Operating systems Oracle Pattern Searching puzzle Python Recursion Samsung SAP Labs SnapDeal stack Stack-Queue STL

Subscribe and Never Miss an Article

Email Address Subscribe

Like us on Facebook

Follow us on Twitter

Recent Comments

Shambhavi Shinde without using dummy node c++...

Rearrange a given linked list in-place. · 27 minutes ago

sulabh kumar http://imgur.com/pNcwl1L

Hope u understand.

Articulation Points (or Cut Vertices) in a Graph · 36 minutes ago

Bala Murali Krishna I don't think time complexity is O(n). Try with...

Connect nodes at same level using constant extra space · 1 hour ago

texas yes it is do-able, but it uses a lot of extra...

Diameter of a Binary Tree · 1 hour ago

disqus_rrOqG3928B http://ideone.com/nTuu6s

Remove duplicates from a sorted linked list · 1 hour ago

Junaid Alam Simplest code.....

Merge two sorted linked lists · 1 hour ago

All Categories

Select Category

- **GeeksQuiz**
- **GeeksforGeeksIDE**
- **GeeksforGeeks Practice**
- **Data Structures**
- **Algorithms**
- **C Programming**
- **C++ Programming**
- **Java Programming**
- **Books**
- **Interview Experiences**
- **GATE CS**

- GATE CS Forum
- Android App