

Work Integrated
Learning Programmes



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Structure Algorithm and Design (SSZG519)

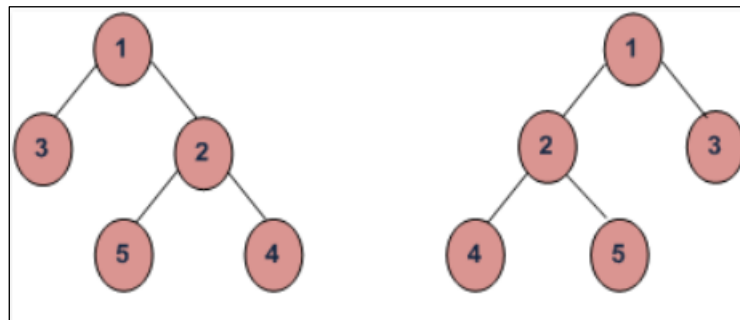
Assignment

Submitted By:

Nilesh D. Ghodekar (2018ht12544)

Q 1: Give an algorithm for converting a tree to its mirror. Mirror of a tree is another tree with left and right children of all non-leaf nodes interchanged.

Ans:



Algorithm using Recursive Method:

- (1) Call Mirror for left-subtree i.e., Mirror(left-subtree)
- (2) Call Mirror for right-subtree i.e., Mirror(right-subtree)
- (3) Swap left and right subtrees.

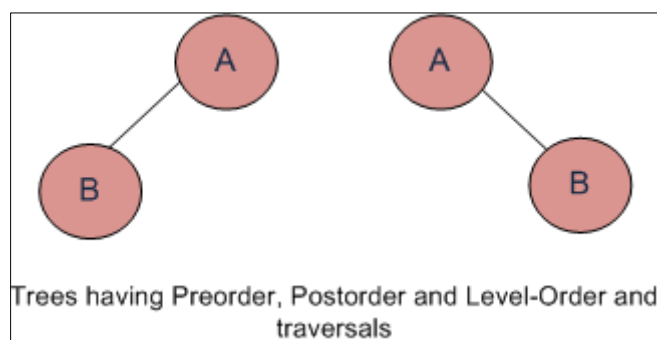
temp = left-subtree

left-subtree = right-subtree

right-subtree = temp

Q 2: If we are given two traversal sequences, can we construct the binary tree uniquely?

Ans: It depends on what traversals are given. If one of the traversal methods is Inorder then the tree can be constructed, otherwise not.



With following combination can uniquely identify a tree.

Inorder and Preorder.

Inorder and Postorder.

Inorder and Level-order.

And with below combination we cannot identify a tree.

Postorder and Preorder.

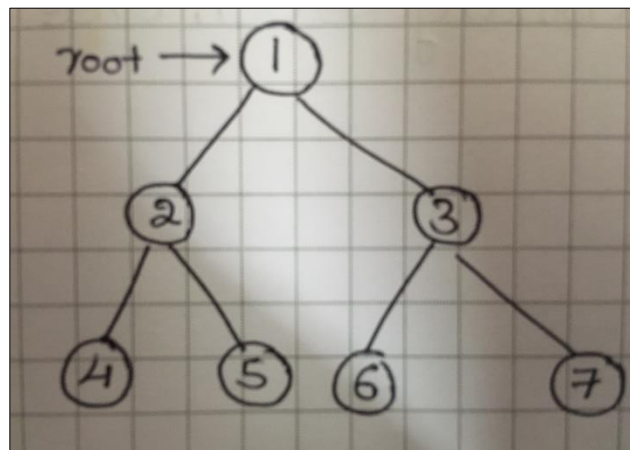
Preorder and Level-order.
Postorder and Level-order.

For example, Preorder, Level-order and Postorder traversals are same for the trees given in above diagram.

Preorder Traversal = AB
Postorder Traversal = BA
Level-Order Traversal = AB

So, even if three of them (Pre, Post and Level) are given, the tree cannot be constructed.

Q 3: Zig Zag Tree Traversal: Give an algorithm to traverse a binary tree in Zig Zag order.
For example, the output for the tree below should be: 1 3 2 4 5 6 7.



Ans:

1. Take 2 Stack, one for current level and one for next level.
2. For filling the next Level stack, we have to see whether it need to be filled Left to Right or Right to left as we are reading the Level Order spirally.
3. When we are reading current level stack, all the children's of current level stack will go to next level stack.
4. When current level stack has no value and is Empty at that time point current level stack Reference to hold next level stack values that is current level stack will now point to next level stack and next level stack should be reinitialize for its next level.

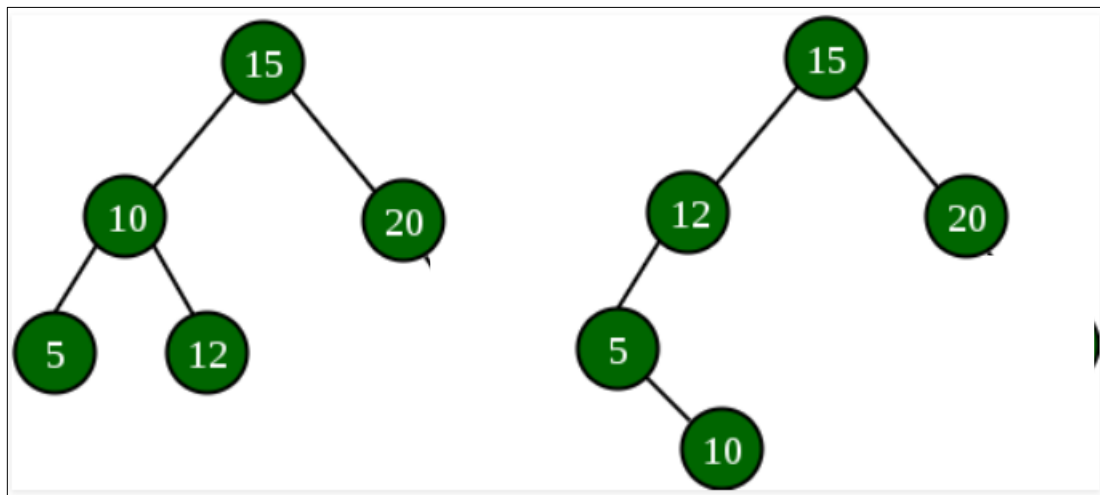
Also, this is the time when we need to alter our reading style for children that are it need to be read in left - right fashion.

Q 4: Given two BSTs, check whether the elements of them are the same or not. For example: two BSTs with data 10 5 20 15 30 and 10 20 15 30 5 should return true and the dataset with 10 5 20 15 30 and 10 5 30 20 5 should return false. Note: BSTs data can be in any order.

Ans: Given two Binary Search Trees consisting of unique positive elements, we have to check whether the two BSTs contains same set or elements or not.

Note: The structure of the two given BSTs can be different.

For example,



The above two BSTs contains same set of elements {5, 10, 12, 15, 20}

Method 1: The most simple method will be to traverse first tree and store its element in a list or array. Now, traverse 2nd tree and simultaneously check if the current element is present in the list or not. If yes, then mark the element in the list as negative and check for further elements otherwise if no, then immediately terminate the traversal and print No. If all the elements of 2nd tree is present in the list and are marked negative then finally traverse the list to check if there are any non-negative elements left. If Yes then it means that the first tree had some extra element otherwise the both tree consists same set of elements.

Time Complexity: $O(n * n)$, where n is the number of nodes in the BST.

Auxiliary Space: $O(n)$.

Method 2: This method is an optimization of above approach. If we observe carefully, we will see that in the above approach, search for element in the list takes linear time. We can optimize this operation to be done in constant time using a hashmap instead of list. We insert elements of both trees in different hash sets. Finally we compare if both hash sets contain same elements or not.

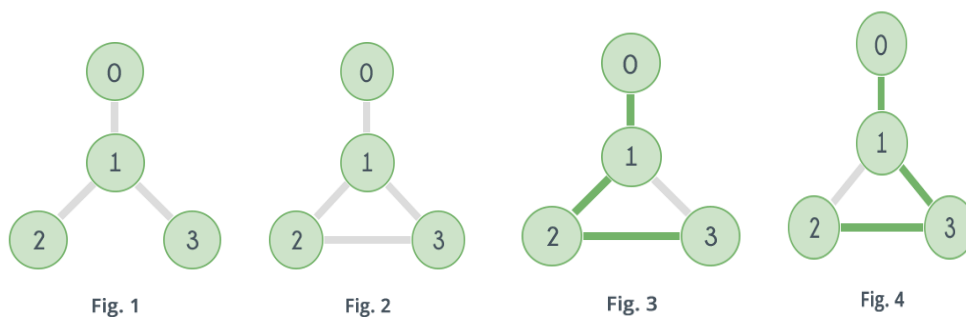
Time Complexity: $O(n)$.

Auxiliary Space: $O(n)$.

Q 6: The Hamilton cycle problem: Is it possible to traverse each of the vertices of a graph exactly once, starting and ending at the same vertex?

Ans:

Hamiltonian Path is a path in a directed or undirected graph that visits each vertex exactly once. The problem to check whether a graph (directed or undirected) contains a Hamiltonian Path is NP-complete, so is the problem of finding all the Hamiltonian Paths in a graph. Following images explain the idea behind Hamiltonian Path more clearly.



Graph shown in Fig.1 does not contain any Hamiltonian Path. Graph shown in Fig. 2 contains two Hamiltonian Paths which are highlighted in Fig. 3 and Fig. 4

Following are some ways of checking whether a graph contains a Hamiltonian Path or not.

1. A Hamiltonian Path in a graph having N vertices is nothing but a permutation of the vertices of the graph $[v_1, v_2, v_3, \dots, v_{N-1}, v_N]$, such that there is an edge between v_i and v_{i+1} where $1 \leq i \leq N-1$. So it can be checked for all permutations of the vertices whether any of them represents a Hamiltonian Path or not. For example, for the graph given in Fig. 2 there are 4 vertices, which means total 24 possible permutations, out of which only following represents a Hamiltonian Path.

0-1-2-3

3-2-1-0

0-1-3-2

2-3-1-0

Following is the pseudo code of the above algorithm:

```
function check_all_permutations(adj[], n)
  for i = 0 to n
    p[i]=i
    while next permutation is possible
      valid = true
      for i = 0 to n-1
```

```

        if adj[p[i]][p[i+1]] == false
            valid = false
            break
    if valid == true
        return true
    p = get_next_permutation(p)
return false

```

Time complexity of the above method can be easily derived. For a graph having N vertices it visits all the permutations of the vertices, i.e. $N!$ iterations and in each of those iterations it traverses the permutation to see if adjacent vertices are connected or not i.e N iterations, so the complexity is $O(N * N!)$.

Q 7: Given an array F with size n. Assume the array content $F[i]$ indicates the length of the i^{th} file and we want to merge all these files into one single file. Check whether the following algorithm gives the best solution for this problem or not?

Algorithm: Merge the files contiguously. That means select the first two files and merge them. Then select the output of the previous merge and merge with the third file, and keep going.

Note: Given two files A and B with sizes m and n, the complexity of merging is $O(m+n)$.

Ans:

The implementation of the above algorithm is attached in below GITHUB link

<https://github.com/nileshGh/BitsDataAlgorithm/blob/master/MergeArrays.java>



```

1  class MergeArrays {
2  {
3      /** function to merge a elements at the end of array arr[] */
4      void mergeSort(int arr[], int n)
5      {
6          if (n <= 1) return;
7          int i = 0;
8          while (i < n-1)
9          {
10             if (arr[i] < arr[i+1])
11                 swap(arr[i], arr[i+1]);
12             i++;
13         }
14     }
15
16     /** Merge array arr[] of size n into array arr[]
17     of size m */
18     void mergeSort(int arr[], int i1, int n1, int n2)
19     {
20         int i = i1;
21         int j = i1 + n1;
22         int k = j;
23
24         /** Current index of 1st part of arr[] */
25         int l = i;
26         /** Current index of 2nd part of arr[] */
27         int m = j;
28         /** Current index of output arr[] */
29         int k = i;
30
31         while (l < m)
32         {
33             /** Take an element from arr[] if
34             it is value of the current element is smaller and we have
35             not reached end of it.
36             If it is then reached end of it.
37             If (l < m) then arr[l] is smaller than arr[m]
38             {
39                 arr[k] = arr[l];
40                 l++;
41             }
42             else // otherwise take element from m
43             {
44                 arr[k] = arr[m];
45                 m++;
46             }
47             k++;
48         }
49     }
50
51     /** Utility that prints out an array in a line */
52     void printArray(int arr[], int n)
53     {
54         int i;
55         for (i = 0; i < n; i++)
56             System.out.print(arr[i] + " ");
57         System.out.println();
58     }
59
60     public static void main(String[] args)
61     {
62         MergeArrays merge = new MergeArrays();
63
64         /** Initial array */
65         int arr[] = {5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100};
66         int n = arr.length;
67         int m = arr.length + 1;
68
69         /** Merge the n elements at the end of arr[] */
70         merge.mergeSort(arr, n);
71         /** Merge the m elements at the end of arr[] */
72         merge.mergeSort(arr, m);
73         /** Print the merged array */
74         merge.printArray(arr, m);
75     }
76 }

```

Q 8: Number of railway-platforms: at a railway station, we have a time – table with the trains arrivals and departures. We need to find the minimum number of platforms so that all the trains can be accommodated as per their schedule.

Ans :

The implementation of above can be found at GITHUB link

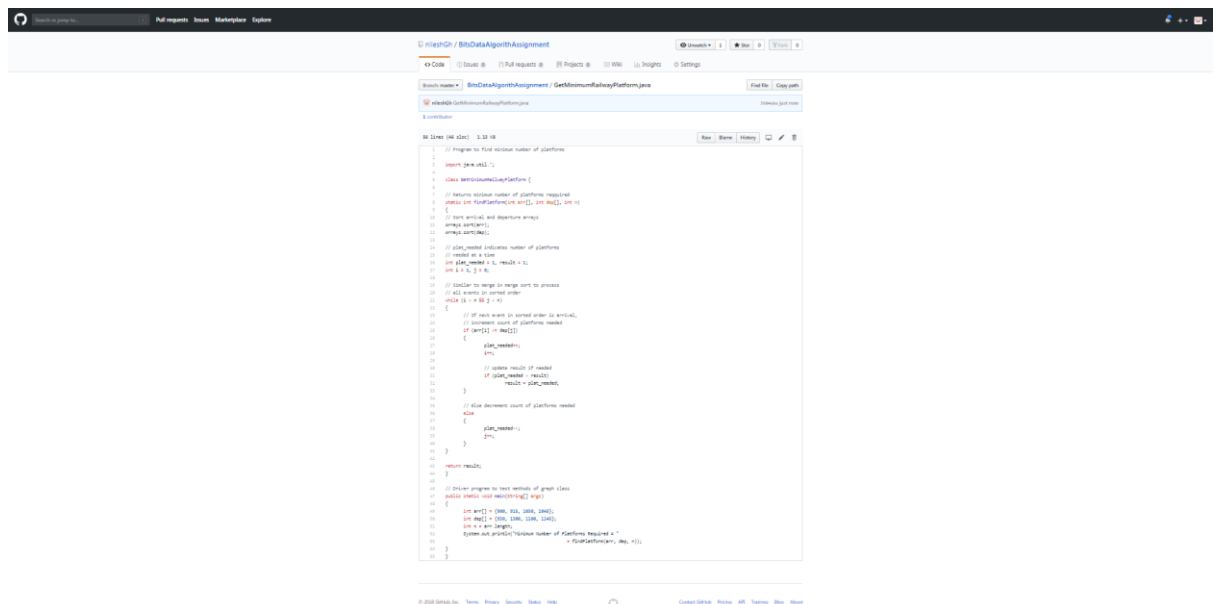
<https://github.com/nileshGh/BitsDataAlgorithmAssignment/blob/master/GetMinimumRailwayPlatform.java>

Time(sec) : 0.06

Memory(MB) : 55.0859

Output:

Minimum Number of Platforms Required = 3



```
1 // Program to find minimum number of platforms
2
3 import java.util.*;
4
5 class GetMinimumRailwayPlatform {
6
7     // Returns minimum number of platforms required
8     // arr1[] contains arrival times of trains, arr2[] contains departure times
9     // arr1 and arr2 are sorted in ascending order
10    int minPlatforms(int arr1[], int arr2[]) {
11        // Sort arrival and departure arrays
12        Arrays.sort(arr1);
13        Arrays.sort(arr2);
14
15        // plat_needed indicates number of platforms
16        // needed at a time
17        int plat_needed = 1, result = 1;
18        int i = 1, j = 0;
19
20        // Iterate through the arrays
21        while (i < arr1.length) {
22            // If next train starts before the previous train
23            // has departed, then we need an additional platform
24            if (arr1[i] < arr2[j]) {
25                plat_needed++;
26                result = Math.max(result, plat_needed);
27            }
28            // Else, the previous train has departed, so we can
29            // use the same platform for the next train
30            else {
31                j++;
32            }
33            i++;
34        }
35
36        return result;
37    }
38
39    // Driver program to test the above method
40    public static void main(String[] args) {
41        GetMinimumRailwayPlatform obj = new GetMinimumRailwayPlatform();
42        int arr1[] = { 900, 940, 950, 1100, 1200 };
43        int arr2[] = { 1000, 1200, 1300, 1400, 1500 };
44        int n = arr1.length;
45        System.out.println("Minimum number of Platforms required = "
46            + obj.minPlatforms(arr1, arr2));
47    }
48 }
```