Algorithm:

Step 1: Start.

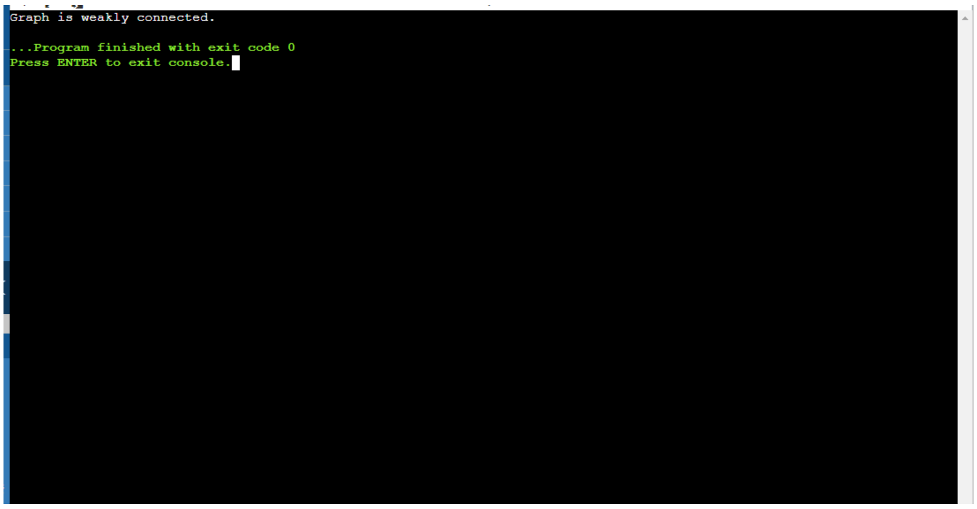
Step 2: Call DFS(G) to compute finishing time f(U) for each vertex U.

Step 3: Compute G^T.

Step 4: Call DFS(G^T).

Step 5: Produce the output of DFS forest as separated strongly connected components.

Output



```
Graph is weakly connected.

...Program finished with exit code 0
Press ENTER to exit console.
```

Algorithm:

Tree Insert

Step 1: Find the node where the new node has to be inserted and store the visited node in the temp variable.

Step 2: Now, the new node data will be compared with temp.

Step 3: If the data is larger than temp data then insert the new node at the right of the temp.

Step 4: If the data is smaller then insert the new node at the left of the temp.

Tree Delete

case 1: Deleting the leaf node

To delete the leaf node we need to traverse to the required leaf node then delete that node and update its parent.

case 2: Delete a node with one child

Remove that given node and replace it with its children.

Search a Node

Step 1: Compare the current node data with the key if:

If the key is found, then return the node.

If the key is lesser than the node data, move the current to the left node and again repeat step 1.

If the key is greater then move to the right and repeat step 1.

Step 2: If the node is not found then return NULL.

Traversal

a) Inorder

Step 1: Traverse to the left to a given node.

Step 2: Print the node data.

Step 3: Traverse to the right to a given node.

b) Postorder

Step 1: Traverse to the left to a given node.

Step 2: Traverse to the right to a given node.

Step 3: Print the data of the node.

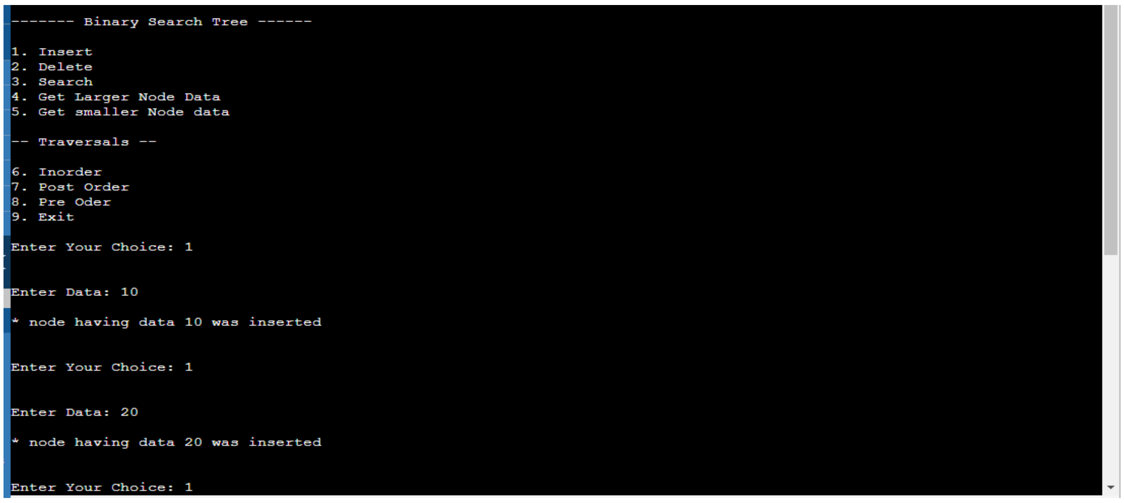
c) Preorder

Step 1: Print the node data.

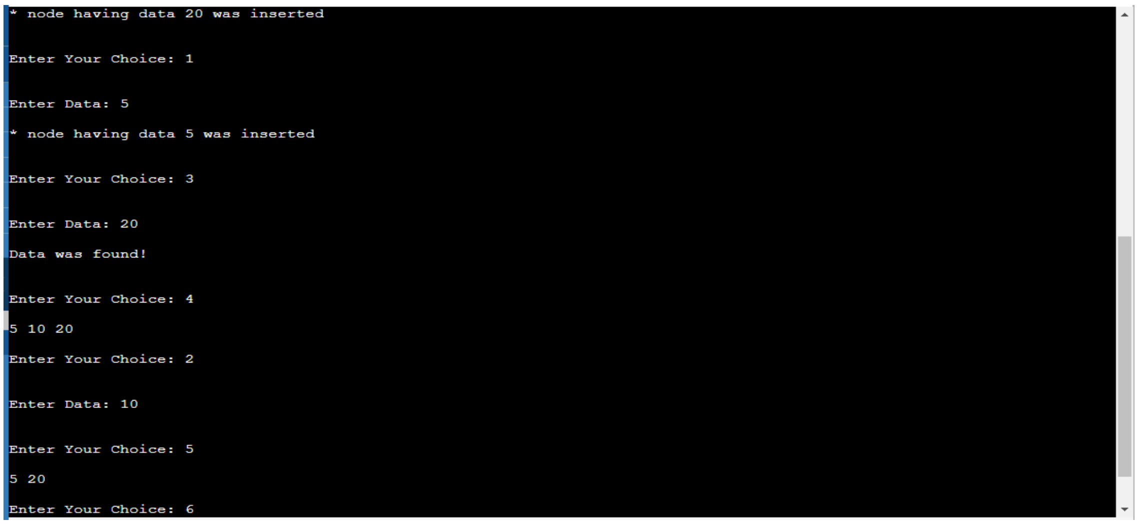
Step 2: Traverse to the left to a given node.

Step 3: Traverse to the right to a given node.

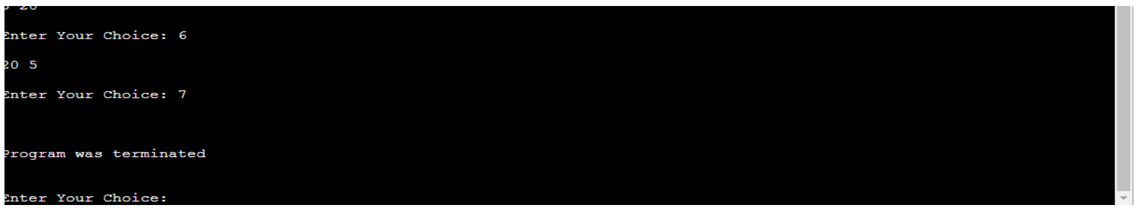
Output



```
------- Binary Search Tree ------

1. Insert
2. Delete
3. Search
4. Get Larger Node Data
5. Get smaller Node data

-- Traversals --

6. Inorder
7. Post Order
8. Pre Oder
9. Exit

Enter Your Choice: 1


Enter Data: 10

* node having data 10 was inserted


Enter Your Choice: 1


Enter Data: 20

* node having data 20 was inserted


Enter Your Choice: 1
```



```
* node having data 20 was inserted


Enter Your Choice: 1


Enter Data: 5

* node having data 5 was inserted


Enter Your Choice: 3


Enter Data: 20

Data was found!


Enter Your Choice: 4

5 10 20

Enter Your Choice: 2


Enter Data: 10


Enter Your Choice: 5

5 20

Enter Your Choice: 6
```



```
Enter Your Choice: 6

20 5

Enter Your Choice: 7


Program was terminated

Enter Your Choice:
```

Algorithm:

Step 1: Start.

Step 2: Initialize universal set and two non-empty set.

Step 3: Convert into bitstring.

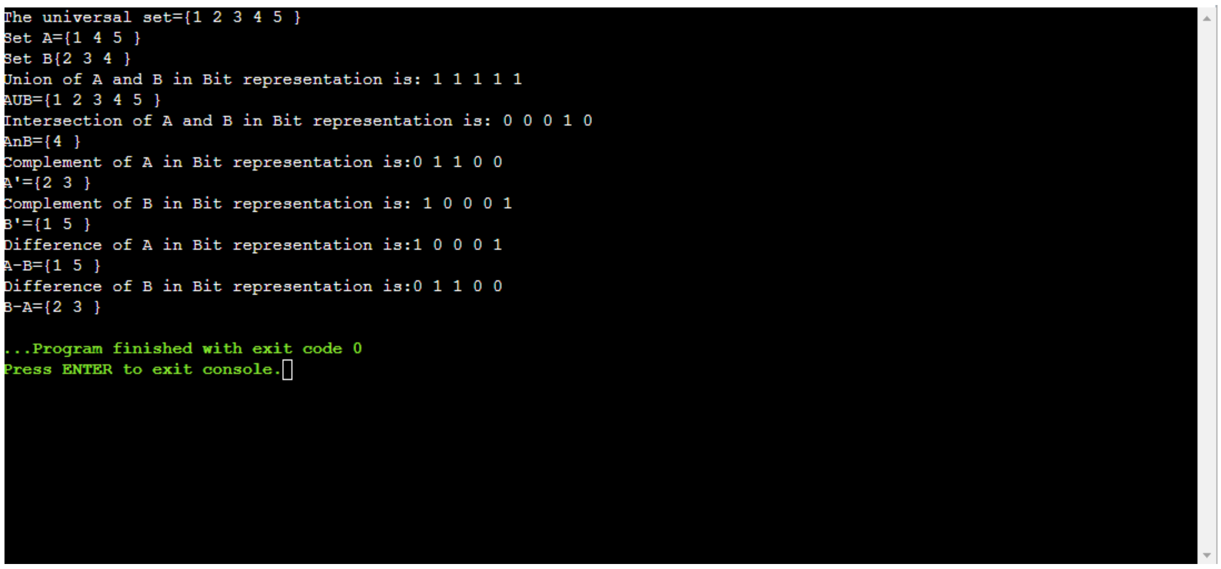
Step 4: Using bit string find the union of two non-empty set and print.

Step 5: Using bit string find the common element in two sets and print the intersection.

Step 6: Print difference of two sets A-B and B-A.

Step 7: Stop.

Output



```
The universal set={1 2 3 4 5 }
Set A={1 4 5 }
Set B{2 3 4 }
Union of A and B in Bit representation is: 1 1 1 1 1
AUB={1 2 3 4 5 }
Intersection of A and B in Bit representation is: 0 0 0 1 0
AnB={4 }
Complement of A in Bit representation is:0 1 1 0 0
A'={2 3 }
Complement of B in Bit representation is: 1 0 0 0 1
B'={1 5 }
Difference of A in Bit representation is:1 0 0 0 1
A-B={1 5 }
Difference of B in Bit representation is:0 1 1 0 0
B-A={2 3 }

...Program finished with exit code 0
Press ENTER to exit console.
```