Trees

① Algorithm of Recursive Traversal for

i) Preorder

PREORDER (ROOT, LEFT, RIGHT, INFO) :- Given a linked binary tree whose root node pointed to by a pointer ROOT. A local variable PTR is a pointer that points to current being processed.

1) PTR ← ROOT
2) if (PTR ≠ NULL) then
    a) Print INFO [PTR]
    b) Call PREORDER (LEFT [PTR])
    c) Call PREORDER (RIGHT [PTR])
3) EXIT

ii) INORDER

INORDER (ROOT, LEFT, RIGHT, INFO)

1) PTR ← ROOT
2) if (PTR ≠ NULL) then
    a) call INORDER (LEFT [PTR])
    b) Print INFO [PTR]
    c) call INORDER (RIGHT [PTR])
3) EXIT

iii) POSTORDER

POSTORDER (ROOT, LEFT, RIGHT, INFO)

1) PTR ← ROOT
2) if (PTR ≠ NULL) then
    a) Call POSTORDER (LEFT [PTR])
    b) Call POSTORDER (RIGHT [PTR])

c) Print INFO[PTR]

3) EXIT

② Algorithm of non-recursive Traversal for

i) Preorder

PREORDER (INFO, LEFT, RIGHT, ROOT) :- A Binary Tree T is in memory. This algorithm does a preorder Traversal of T. An array stack is used to temporarily hold the addresses of nodes.

1) Set TOP = 1 , STACK[1] = NULL and PTR = ROOT
2) Repeat step 3 to 5 while (PTR ≠ NULL)
3) Apply PROCESS to INFO[PTR]
4) if (RIGHT[PTR] ≠ NULL) then
      Set TOP = TOP + 1 and STACK[TOP] = RIGHT[PTR]
   [end if]
5) if (LEFT[PTR] ≠ NULL) then
      set PTR = LEFT[PTR]
   else
      set PTR = STACK[TOP] and TOP = TOP - 1
   [end if]
6) EXIT

ii) Inorder

INORDER (INFO, LEFT, RIGHT, ROOT) :- This algorithm does an inorder traversal of binary tree T.

1) Set TOP = 1 , STACK[1] = NULL and PTR = ROOT
2) Repeat while (PTR ≠ NULL)
   a) Set TOP = TOP + 1 and STACK[TOP] = PTR
   b) Set PTR = LEFT[PTR]
   [end loop]

3) Set PTR = STACK[TOP] and TOP = TOP-1
4) Repeat step 5 to 7 while (PTR ≠ NULL)
5) Apply PROCESS to INFO[PTR]
6) if (RIGHT[PTR] ≠ NULL) then
    a) Set PTR = RIGHT[PTR]
    b) Go to step 2
  [end if]
7) set PTR = STACK[TOP] and TOP = TOP-1
  [end loop step 4]
8) EXIT

iii) Postorder

POSTORDER (LEFT, RIGHT, INFO, ROOT) :- This algorithm does a postorder traversal of binary tree T.

1) Set TOP = 1 , STACK[1] = NULL and PTR = ROOT
2) Repeat step 3 to 5 while (PTR ≠ NULL)
3) Set TOP = TOP+1 and STACK[TOP] = PTR
4) if (RIGHT[PTR] ≠ NULL) then
    set TOP = TOP+1 and STACK[TOP] = -RIGHT[PTR]
  [end if]
5) Set PTR = LEFT[PTR]
  [end loop step 2]
6) Set PTR = STACK[TOP] and TOP = TOP - 1
7) Repeat while (PTR > 0)
    a) Apply PROCESS to INFO[PTR]
    b) set PTR = STACK[TOP] and TOP = TOP-1
  [end loop]
8) if (PTR < 0) then
    a) Set PTR = -PTR
    b) Go to step 2
9) EXIT

(3) Algorithm to search a node in the Binary Tree

FIND (INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR) :- A binary search tree is in memory and an ITEM of information is given. This procedure finds the location LOC of ITEM in T and also the location PAR of the parent of ITEM. There are 3 cases:

(i) LOC = NULL and PAR = NULL will indicate empty tree
(ii) LOC ≠ NULL and PAR = NULL will indicate ITEM is the root of T
(iii) LOC = NULL and PAR ≠ NULL will indicate ITEM is not in T and can be added to T as a child of node N with location PAR

```
1)  if (ROOT = NULL) then
        set LOC = NULL and PAR = NULL and Return
    [end if]
2)  if (ITEM = INFO[ROOT]) then
        Set LOC = ROOT and PAR = NULL and Return
    [end if]
3)  if (ITEM < INFO[ROOT]) then
        set PTR = LEFT[ROOT] and SAVE = ROOT
    else
        set PTR = RIGHT[ROOT] and SAVE = ROOT
    [end if]
4)  Repeat step 5 and 6 while (PTR ≠ NULL)
5)  if (ITEM = INFO[PTR]) then
    set LOC = PTR and PAR = SAVE and Return
6)  if (ITEM < INFO[PTR]) then
        set SAVE = PTR and PTR = LEFT[PTR]
    else
        set SAVE = PTR and PTR = RIGHT[PTR]
    [end if]
```

[end loop step4]

7) Set LOC = NULL and PAR = SAVE

8) EXIT

(4) Algorithm to insert a node in Binary Tree

INSBST (INFO, LEFT, RIGHT, ROOT, AVAIL, ITEM, LOC) :- A binary search tree T is in memory and an ITEM of information is given. This algorithm finds the location LOC of ITEM in T and/or adds ITEM as a new node in T at location LOC.

1. Call FIND (INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR)
2. if (LOC ≠ NULL) then
            Exit
   [end if]
3. a) if (AVAIL = NULL) then
            Print "Overflow"
            Return
      [end if]
   b) set NEW = AVAIL , AVAIL = LEFT[AVAIL] ; INFO[NEW]=ITEM
   c) set LOC = NEW , LEFT[NEW] = NULL and RIGHT[NEW] = NULL
4. if (PAR = NULL) then
            Set ROOT = NEW
   else if (ITEM < INFO [PAR]) then
            set LEFT [PAR] = NEW
   else
            set RIGHT [PAR] = NEW
   [end if]
5. EXIT

(5) Algorithm to delete a node in Binary Tree

(i) CASEA(INFO, LEFT, RIGHT, ROOT, LOC, PAR) :- This procedure deletes the node N at location LOC, where N does not have two children. The pointer PAR gives the location of parent of N or else PAR=NULL indicates that N is the root node. The pointer CHILD gives the location of the only child of N, or else CHILD = NULL indicates N has no children.

```
1.  if (LEFT[LOC] = NULL and RIGHT[LOC] = NULL) then
            Set CHILD = NULL
    else if (LEFT[LOC] ≠ NULL) then
            Set CHILD = LEFT[LOC]
    else
            Set CHILD = RIGHT[LOC]
    [end if]
2.  if (PAR ≠ NULL) then
            if (LOC = LEFT[PAR]) then
                    Set LEFT[PAR] = CHILD
            else
                    Set RIGHT[PAR] = CHILD
            [end if]
    else
            Set ROOT = CHILD
      [end if step 2]
3.  Return
```

(ii) CASEB (INFO, LEFT, RIGHT, ROOT, LOC, PAR) :- This procedure will delete the node N at location LOC, where N has two children. The pointer PAR gives location of parent of N or else PAR = NULL indicates N is root node. The pointer SUC gives the location of inorder successor of N and PARSUC gives the location of the parent of the inorder successor.

1) a) Set PTR = RIGHT [LOC] and SAVE = LOC

   b) Repeat while (LEFT[PTR] ≠ NULL)

      Set SAVE = PTR and PTR = LEFT[PTR]

      [end loop]

   c) Set SUC = PTR and PARSUC = SAVE

2) Call CASEA (INFO, LEFT, RIGHT, ROOT, SUC, PARSUC)

3) a) if (PAR ≠ NULL) then

      if (LOC = LEFT [PAR]) then

         set LEFT [PAR] = SUC

      else

         set RIGHT [PAR] = SUC

      [end if]

   else

      Set ROOT = SUC

   b) Set LEFT[SUC] = LEFT[LOC]

      Set RIGHT [SUC] = RIGHT [LOC]

4) Return

(iii) DEL (INFO, LEFT, RIGHT, ROOT, AVAIL, ITEM) :- A binary search tree T is in memory and an ITEM of Information is given. This algorithm deletes ITEM from the Tree.

1) Call FIND (INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR)

2) if (LOC = NULL) then

      Print " Item not in tree"

      EXIT

   [end if]

3) if (RIGHT [LOC] ≠ NULL and LEFT [LOC] ≠ NULL) then

      call CASEB (INFO, LEFT, RIGHT, ROOT, LOC, PAR)

else
    Call CASEA (INFO, LEFT, RIGHT, ROOT, LOC, PAR)
[end if]
4) Set LEFT[LOC] = AVAIL and AVAIL = LOC
5) EXIT

17/10/22

Output :-

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Find
7. Quit

Enter choice 1
Enter number to insert 10
Enter choice 1
Enter number to insert 8
Enter choice 1
Enter number to insert 20
Enter choice 3
8 10 20
Enter choice 2
Enter number to delete 8
Enter choice 3
10 20

f) Graphs

① Algorithm of BFS (Breadth-First Search)

1) Initialize all nodes to the ready state (STATUS = 1).
2) Put the starting node A in Queue and change its status to the waiting state (STATUS = 2).
3) Repeat step 4 and 5 until Queue is empty:
4) Remove the front node N of Queue. Process N and change the status of N to the processed state (STATUS = 3).
5) Add to the rear of Queue all the neighbours of N that are in the ready state (STATUS = 1), and change their status to the waiting state (STATUS = 2).
   [end loop step 3]
6) Exit

28/10/22

② Algorithm of DFS (Depth-First Search)

1) Initialize all nodes to the ready state (STATUS = 1).
2) Push the starting node A onto stack and change its status to the waiting state. (STATUS = 2)
3) Repeat step 4 and 5 until stack is empty.
4) Pop the top Node N of stack. Process N and changes its status to the processed state (STATUS = 3).
5) Push onto stack all the neighbours of N that are still in ready state (STATUS = 1), and change their status to the waiting state.
6) Exit