Heaven's Light is Our Guide
Computer Science & Engineering
Rajshahi University of Engineering & Technology

Lab Manual

Module- 08

Course Title: Sessional based on CSE 1201
Course No. : CSE 1202

Experiment No. 8

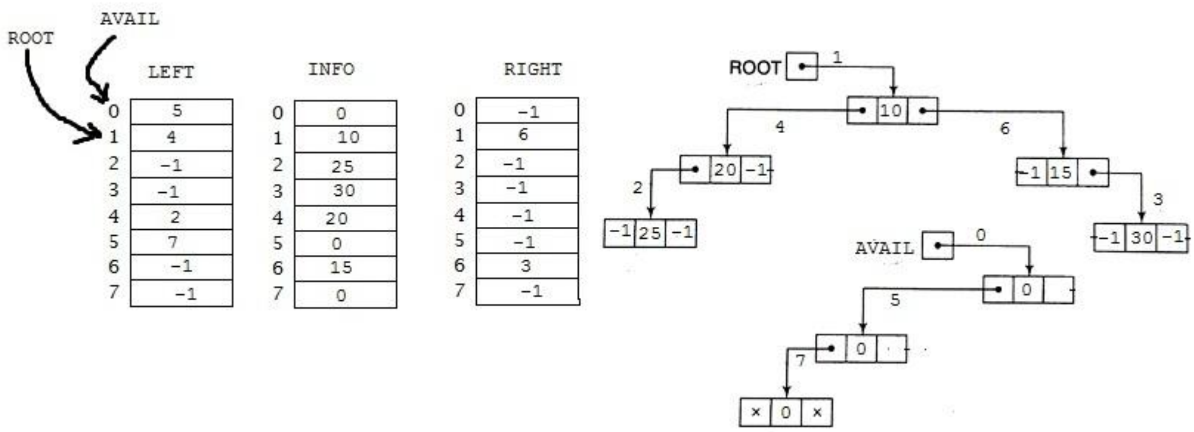
Name of the Experiment: Tree

Duration: 1 cycle

Background Study: Chapter 7 (Theory and Problems of Data Structures Written by Seymour Lipschutz)

8.1 Representation of Binary Tree
8.1.1 Linked Representation of binary tree using Parallel Array:
- ✓ Three Parallel Arrays, INFO,LEFT and RIGHT and a pointer variable ROOT as Follows
- ✓ N^{th} node of Tree will correspond to a location K such that:
  - INFO[K] contains the data at the N^{th} node
  - LEFT[K] contains the location of the left child of N^{th} node
  - RIGHT[K] contains the location of the right child of N^{th} node.
- ✓ ROOT will contain the location of the root R of a Tree.
- ✓ If any sub tree is empty, then the corresponding pointer will contain the null value.
- ✓ IF the tree itself is empty, then ROOT will contain the null values.

Remark:
1. We will use the LEFT array contain the pointer for the AVAIL list.
2. Any invalid address may be chosen for the null pointer denoted by NULL. In actual practice, 0 or negative number is used for NULL.



8.1.2 Sequential Representation of Binary Tree:
- ✓ Use a single linear array to represent a TREE.

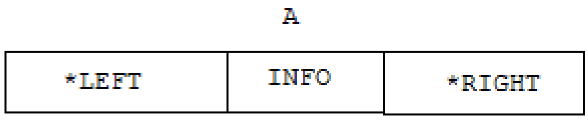
We can represent the above TREE by using a linear array, i.e.
ROOT=0

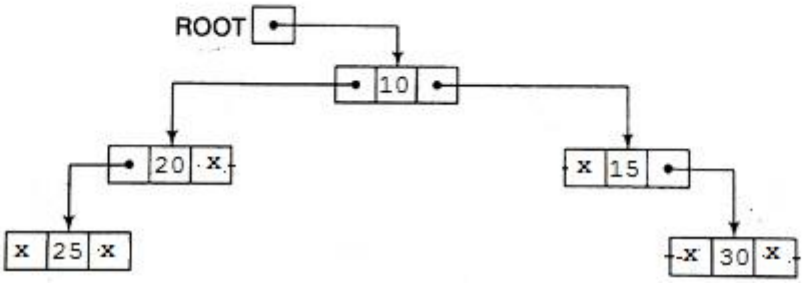
	TREE	
0	10	ROOT
1	20	2xROOT +1= 1, LEFT CHILD(ROOT)
2	15	2xROOT +2=2, RIGHT CHILD(ROOT)
3	25	2x1 +1=3, LEFT CHILD(1)
4		
5		
6	30	2x2+2=6, RIGHT CHILD(2)

8.1.3 Linked Representation of Binary Tree using pointer and structure:
```
struct node{
    int INFO;
    struct node *LEFT;
```

```
    struct node *RIGHT;
};

struct node A;
```



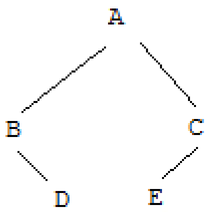


Problem I: Write Down a program to take a binary tree as input and represent it by different technique.

8.2 Traversing Binary Tree
8.2.1 Preorder Traverse:

1 Process the Root R
2 Traverse the left sub tree of R in Preorder
3 Traverse the right sub tree of R in Preorder

Example:



A (Left Sub tree) (Right Sub tree)
A (B (Left Sub tree) (Right Sub tree)) (C (Left Sub tree) (Right Sub tree))
A (B (Right Sub tree)) (C (Left Sub tree))
A (B (D (Left Sub tree) (Right Sub tree))) (C (E (Left Sub tree) (Right Sub tree)))
A (B D) (C E)
ABDCE

Problem II: Preorder traverse of a tree
Algorithm 8.1: PREORDER(INFO, LEFT,RIGHT,ROOT)
A binary tree T is in memory. The algorithm does a preorder traversal of T, applying an operation PROCESS to each of its nodes. An array STACK is used to temporarily hold the addresses of nodes.

1. Set TOP:=1, STACK [1]:=NULL and PTR:=ROOT.
2. Repeat Steps 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 of If statement]
5. If LEFT[PTR] ≠ NULL, then:
   Set PTR:=LEFT[PTR]
   Else:

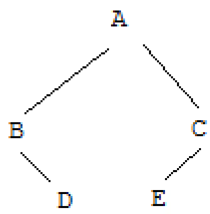
Set PTR:=STACK[TOP] and TOP:=TOP-1.
[End of If statement]
[End of step 2]
6. Exit.

Flow Chart: Draw a flow chart.

8.2.2 In order Traverse:

1 Traverse the left sub tree of R in Preorder
2 Process the Root R
3 Traverse the right sub tree of R in Preorder

Example:



(Left Sub tree) A (Right Sub tree)
((Left Sub tree) B (Right Sub tree)) A ((Left Sub tree) C (Right Sub tree))
(B (Right Sub tree)) A ((Left Sub tree) C)
(B ((Left Sub tree) D (Right Sub tree))) A (((Left Sub tree) E (Right Sub tree)) C)
(B D) A (E C)
BDAEC

Problem III: Inorder traverse of a tree
Algorithm 8.2: INORDER(INFO, LEFT,RIGHT,ROOT)
A binary tree T is in memory. The algorithm does a inorder traversal of T, applying an operation PROCESS to each of its nodes. An array STACK is used to temporarily hold the addresses of nodes.

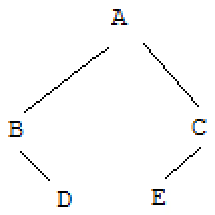
1. Set TOP:=1, STACK [1]:=NULL and PTR:=ROOT.
2. Repeat while PTR $\neq$ NULL
   (a) Set TOP:=TOP+1 and STACK[TOP]:=PTR
   (b) Set PTR:=LEFT[PTR]
   [End the Loop]
3. Set PTR:=STACK[TOP] amd TOP:=TOP-1
4. Repeat Steps 5 to 7 while PTR $\neq$ NULL
5. Apply PROCESS to INFO[PTR].
6. If RIGHT[PTR] $\neq$ NULL, then:
   (a) Set TOP:=TOP+1 and STACK[TOP]:= RIGHT[PTR]
   (b) Go to step 3.
   [End of If statement]
7. Set PTR:=STACK[TOP] amd TOP:=TOP-1
   [End of step 4]
8. Exit.

Flow Chart: Draw a flow chart.

8.2.3 Post order Traverse:

1 Traverse the left sub tree of R in Preorder
2 Traverse the right sub tree of R in Preorder
3 Process the Root R

Example:



(Left Sub tree) (Right Sub tree) A
((Left Sub tree) (Right Sub tree) B) ((Left Sub tree) (Right Sub tree) C) A
((Right Sub tree) B) ((Left Sub tree) C) A
(((Left Sub tree) (Right Sub tree) D) B) (((Left Sub tree) (Right Sub tree) E) C)
(D B) (E C) A
DBECA

Problem IV: Postorder traverse of a tree
Algorithm 8.3: POSTORDER(INFO, LEFT,RIGHT,ROOT)
A binary tree T is in memory. The algorithm does a postorder traversal of T, applying an operation PROCESS to each of its nodes. An array STACK is used to temporarily hold the addresses of nodes.

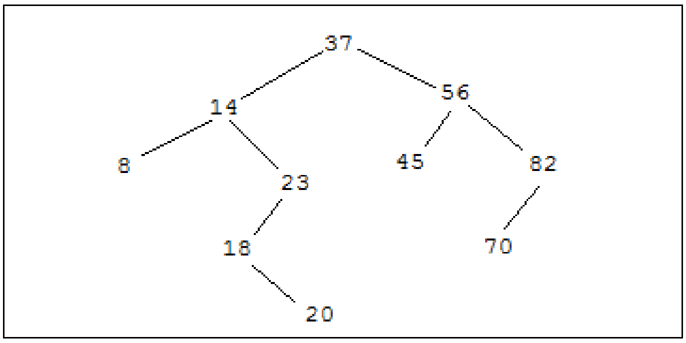
1. Set TOP:=1, STACK [1]:=NULL and PTR:=ROOT.
2. Repeat Steps 3 to 5 while PTR ≠ NULL
3. Set TOP:=TOP+1 and STACK[TOP]:= PTR
4. If RIGHT[PTR] ≠ NULL, then:
   Set TOP:=TOP+1 and STACK[TOP]:= -RIGHT[PTR]
   [End of If statement]
5. Set PTR:=LEFT[PTR].
   [End of step 2]
6. Set PTR:=STACK[TOP] amd TOP:=TOP-1
7. Repeat while PTR>0:
   (a) Apply PROCESS to INFO[PTR].
   (b) Set PTR:=STACK[TOP] amd TOP:=TOP-1
   [End of loop]
8. If PTR<0, then:
   (a) Set PTR:=-PTR.
   (b) Go to step 2
   [End of If statement]
9. Exit.

Flow Chart: Draw a flow chart.

8.3 Binary Search Tree

ROOT = 1

	LEFT	INFO	RIGHT
1	2	38	7
2	3	14	4
3	0	8	0
4	5	23	0
5	0	18	6
6	0	20	0
7	8	56	9
8	0	45	0
9	10	82	0
10	0	70	0



Problem V: Searching in binary search tree
Algorithm 8.4: FIND(INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR)
A binary tree T is in memory and an ITEM of information is given. This algorithm finds the location LOC of ITEM in T and also the location PAR of the parent of ITEM. There are four special cases:

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

1. If ROOT = NULL then: Set LOC=NULL, PAR=NULL and Return.
2. If ITEM=INFO[ROOT] then: Set LOC=ROOT, PAR=NULL and Return.
3. SAVE:=ROOT
4. If ITEM<INFO[ROOT] then:
   Set PTR:=LEFT[ROOT].
   Else:
   Set PTR:=RIGHT[ROOT].
   [End of IF statement]
5. Repeat Steps 6 and 7 while PTR≠NULL
6. If ITEM=INFO[PTR] then: Set LOC=PTR, PAR=SAVE and Return.
7. If ITEM<INFO[ROOT] then:
   Set SAVE=PTR, PTR=LEFT[PTR].
   Else:
   Set SAVE=PTR, PTR=RIGHT[PTR].
   [End of If statement]
   [End of Step 5 loop]
8. Set LOC:=NULL and PAR:=SAVE.
9. Exit

Flow Chart: Draw a flow chart.

Problem VI: Inserting in binary search tree
Algorithm 8.5: INSBST(INFO, LEFT, RIGHT, ROOT, AVAIL, ITEM, LOC)
A binary tree T is in memory and an ITEM of information is given. This algorithm finds the location LOC of ITEM in T 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
3. (a) If AVAIL=NULL, then: Write: OVERFLOW and Exit.
   (b) Set NEW:=AVAIL, AVAIL:=LEFT[AVAIL] and 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 LEFT[PAR]:=NEW.
   [End of If statement]
5. Exit.

Flow Chart: Draw a flow chart.

Problem VII: Deleting in binary search tree
Algorithm 8.6(a): 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 the parent of N, or else PAR = NULL indicates that N is the root node. The pointer CHILD = NULL indicates N has no children.

1. If LEFT[LOC] = NULL and RIGHT[NULL] = NULL, then:
   Set CHILD := NULL.
   Else if LEFT[LOC] ≠ NULL, then:
   Set CHILD := LEFT[LOC].
   Else:
   Set CHILD := RIGHT[LOC].
   [End of If statement]
2. If PAR ≠ NULL, then:
   If LOC = LEFT[PAR], then:
   Set LEFT[PAR]:=CHILD.
   Else:
   Set RIGHT[PAR]:=CHILD.
   [End of If statement]
   Else:
   Set ROOT:=CHILD.
   [End of If statement]
3. Return.

Algorithm 8.6(b): CASEB(INFO, LEFT, RIGHT, ROOT, LOC, PAR)
This procedure deletes the node N at location LOC, where N have **two children.** The pointer PAR gives the location of the parent of N, or else PAR = NULL indicates that N is the root node. The pointer SUC gives the location of the 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 of 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 of If statement]
   Else:
   Set ROOT:=CHILD.
   [End of If statement]
   (b) Set LEFT[SUC]:=LEFT[LOC] and RIGHT[SUC]:= RIGHT[LOC].
4. Return.

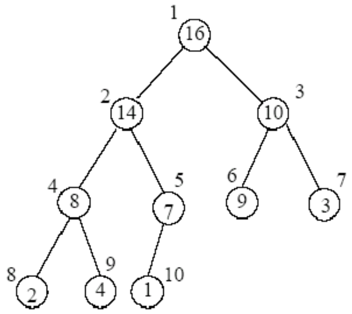
Algorithm 8.6(c): DEL(INFO, LEFT, RIGHT, ROOT, AVAIL, ITEM)
A binary search tree T is in memory and an ITEM of information is given. This algorithm deletes ITEM from the tree.

1. **CALL FIND(INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR)**
2. If LOC = NULL, then Write: ITEM is not in tree, and Exit.
3. If RIGHT[LOC]≠NULL and LEFT[LOC]≠NULL, then:
   CASEB(INFO, LEFT, RIGHT, ROOT, LOC, PAR)
   Else:
   CASEA(INFO, LEFT, RIGHT, ROOT, LOC, PAR)
   [End of If statement]
4. Set LEFT[LOC]:=AVAIL and AVAIL:=LOC.
5. Exit.

Flow Chart: Draw a flow chart.

8.4 Heap and Heapsort

Suppose H is a complete binary tree. Then H is called heap or maxheap, if each node N of H has the following property: "**the value at N is greater than or equal to the value at any of the descendants of N**".



1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

Fig. Heap and Sequential Representation

Problem VIII: Inserting into a heap(max heap)
Algorithm 8.7: INSHEAP(TREE, N, ITEM)
A heap H with N elements is stored in the array TREE, and an ITEM of information is given. The procedure inserts ITEM as a new element of H. PTR gives the location of ITEM as it rises in the tree, and PAR denotes the location the location of the parent of ITEM.

1. Set N:=N+1 and PTR:=N.
2. Repeat Steps 3 to 6 while PTR<1.
3. Set $\text{PAR} := \lfloor \text{PAR}/2 \rfloor$
4. If ITEM ≤ TREE[PAR], then:
   Set TREE[PTR]:=ITEM, and Return.
   [End of If statement]
5. Set TREE[PTR]:=TREE[PAR].
6. Set PTR:=PAR.
   [End of Step 2 loop]
7. Set TREE[1]:=ITEM.
8. Return.

Flow Chart: Draw a flow chart.

Problem IX: Deleting the root of a heap(max heap)
Algorithm 8.8: DELHEAP(TREE, N, ITEM)
A heap H with N elements is stored in the array TREE. This procedure assigns the root TREE[1] of H to the variable ITEM and then reheap the remaining elements. The variable LAST saves the value of the original last node of H. The pointer PTR, LEFT and RIGHT give the location of LAST and its left and right children as LAST sinks in the tree.

1. Set ITEM:=TREE[1].
2. Set LAST:=TREE[N] and N:=N-1.
3. Set PTR:=1, LEFT:=2 and RIGHT:=3.
4. Repeat Steps 5 to 7 while RIGHT≤N:
5. If LAST≥TREE[LEFT] and LAST≥TREE[RIGHT], then:
   Set TREE[PTR]:=LAST and Return.
   [End of If statement]
6. If TREE[RIGHT]≤ TREE[LEFT] then:
   Set TREE[PTR]:= TREE[LEF] and PTR:=LEFT.
   Else:
   Set TREE[PTR]:= TREE[RIGHT] and PTR:=RIGHT.
   [End of If statement]
7. Set LEFT:=2*PTR and RIGHT:=LEFT+1.

[End of Step 4 loop]
8. If LEFT = N and If LAST<TREE[LEFT], then: Set PTR:=LEFT.
9. Set TREE[PTR]:=LAST.
10. Return.

Flow Chart: Draw a flow chart.

Problem X: Application of Heap(Sorting)
Algorithm 8.9: HEAPSORT(A, N)
An array A with N elements is given. This algorithm sorts the elements of A.
1. Repeat fro J=1 to N-1:
   CALL INSHEAP(A, J, A[J+1])
   [End of loop]
2. Repeat while N>1
   (a) **Call DELHEAP(A, N, ITEM)**
   (b) Set A[N+1]:=ITEM.
   [End of loop]
3. Exit.

Complexity: The running time of the step 1 of heap sort is proportional to $n\log_2 n$
The running time of the step 2 of heap sort is proportional to $n\log_2 n$
So The running time of heapsort is proportional to $n\log_2 n$, that is, $f(n) = O(n\log_2 n)$
Flow Chart: Draw a flow chart.

Exercise:
- **[1] Computer Implementation of Huffman's Algorithm.**
- **[2] Application to coding**

MORE PROBLEMS

1. Programming Problems of Chapter 7 of "Data Structures" by Seymour Lipschutz.

LAB REPORT: You have to submit all assigned problems in next lab.