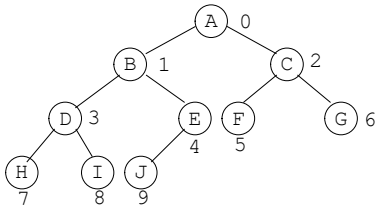
Threaded Binary Trees

5 -1

1

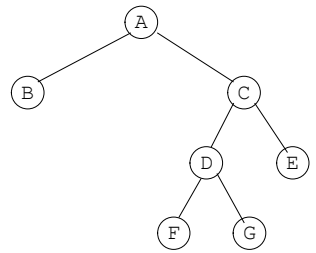
Implicit array representation for Binary Tree



0	1	2	3	4	5	6	7	8	9
A	B	C	D	E	F	G	H	I	J

5 -2

2



0	1	2	3	4	5	6	7	8	9	10	11	12
A	B	C			D	E					F	G

5 -3

3

Linked Representation of the Tree

5 -4

4

Drawback with BT

- Too many null pointers representation of binary trees
  - n: number of nodes
  - number of non-null links: n-1
  - total links: 2n
  - null links: 2n-(n-1)=n+1
- Solution????

  Replace these null pointers with some useful pointers known as "threads".

5 -5

5

Drawbacks with Normal BT continued ..

- Traversing operation is the most frequently used operation on tree.
- Temporary data structure(stack) is required to implement non recursive traversal algorithm.

5 -6

6

Need for TBT

> linked representation , there are more 0/NULL - links than actual pointers.

– There are
  - ➢n+ 1 0/NULL-links and 2n total links
  - ➢Wastage of memory

Thread : a pointer to other nodes in the tree for replacing the 0/NULL-link.

By doing this we can reutilize the Null Pointers. Will result in :

1. No Wastage of memory for null pointers.
2. Non Recursive traversal without stack.

5 -7

7

Traversal Without A Stack

Two methods:

1. Use of parent field to each node.
2. **Use of two bits per node to represents binary trees as threaded binary trees**

5 -8

8

Threaded Binary Trees (*Continued*)

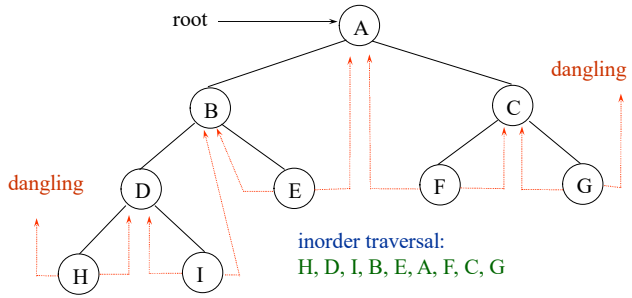
To construct threads we use following rules (assume that ptr represents a node):

- If `ptr->left_child` is null,
  replace it with a pointer to the node that would be visited *before* `ptr` in an *inorder traversal*
  (i.e Inorder predecessor)
- If `ptr->right_child` is null,
  replace it with a pointer to the node that would be visited *after* `ptr` in an *inorder traversal*
  (i.e Inorder successor)

5 -9

9

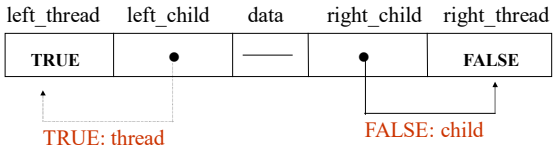
A Threaded Binary Tree



5 -10

10

Data Structures for Threaded BT




```
typedef struct threaded_tree *threaded_pointer;
typedef struct threaded_tree
{
   threaded_pointer left_child;
   threaded_pointer right_child;
   char data;
   int left_thread;
   int right_thread;
};
```

5 -11

11

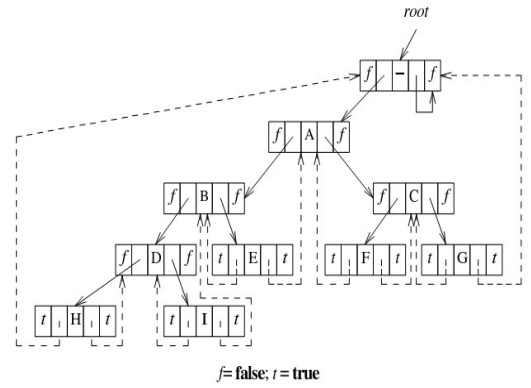


Figure 5.23: Memory representation of threaded tree

12

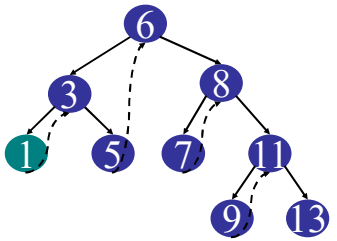
Advantage of TBT in Traversals

- By using threads, we can perform an inorder, preorder and postorder traversal without making use of a stack
- For any node, ptr, in a threaded binary tree , if ptr->rightThread= TRUE, the inorder successor of ptr is ptr's parent. Otherwise, we obtain the inorder successor of ptr by following a path of left-child links from the right-child of ptr until we reach a node with left Thread= TRUE

5 -13

13

Threaded Tree Traversal



Output

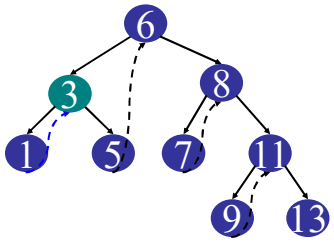
1

Start at leftmost node, print it

5 -14

14

Threaded Tree Traversal



Output

1

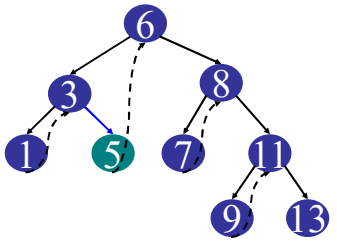
3

Follow thread to right, print node

5 -15

15

Threaded Tree Traversal



Output

1

3

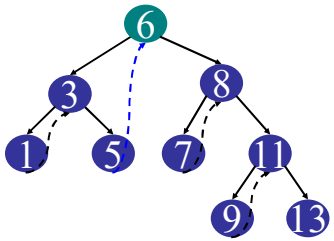
5

Follow link to right, go to leftmost node and print

5 -16

16

Threaded Tree Traversal



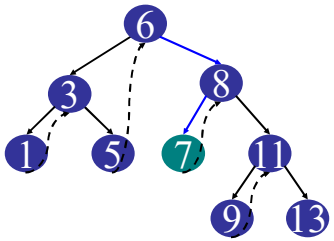
Output
1
3
5
6

Follow thread to right, print node

5 -17

17

Threaded Tree Traversal (Inorder)



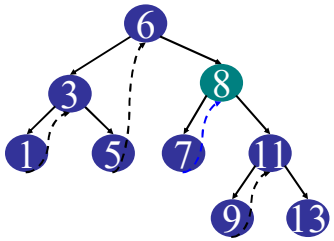
Output
1
3
5
6
7

Follow link to right, go to leftmost node and print

5 -18

18

Threaded Tree Traversal



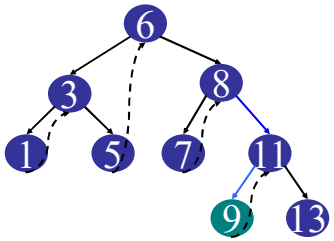
Output
1
3
5
6
7
8

Follow thread to right, print node

5 -19

19

Threaded Tree Traversal



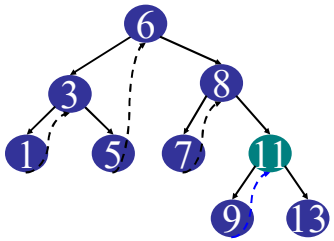
Output
1
3
5
6
7
8
9

Follow link to right, go to leftmost node and print

5 -20

20

Threaded Tree Traversal



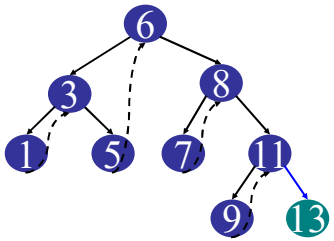
Output
1
3
5
6
7
8
9
11

Follow thread to right, print node

5 -21

21

Threaded Tree Traversal



Output
1
3
5
6
7
8
9
11
13

Follow link to right, go to leftmost node and print

5 -22

22

TBT root creation

FUNCTION CREATE_ROOT(ROOT):

1. IF ROOT=NULL
   THEN TEMP←GETNODE(X)
   LTHREAD(TEMP)←RTHREAD(TEMP)← 1
   LPTR(TEMP)←RPTR(TEMP)←HEAD
   LPTR(HEAD)←TEMP
   ROOT←TEMP
2. RETURN(ROOT)

5 -23

23

TBT Insertion

FUNCTION INSERT_NODE (HEAD , X):
1.IF LPTR(HEAD)=HEAD
THEN WRITE('CREATE ROOT FIRST')
LPTR(HEAD)←CREATE_ROOT(HEAD)
RETURN HEAD

2. PARENT←LPTR(HEAD)
FLAG←TRUE

3. REPEAT THRU STEP 4 WHILE FLAG=TRUE

4. WRITE('ROOT IS' , DATA(PARENT))
WRITE('1.INSERT AT LEFT , 2.INSERT AT RIGHT')
WRITE('ENTER CHOICE')
INPUT(CH)

5 -24

24

TBT Insertion contd..

```
SELECT CH
  CASE 1:  IF (LTHREAD (PARENT) = 1)
             THEN NEV←GETNODE(X)
                LPTR(NEV)←LPTR(PARENT)
                RPTR(NEV)←PARENT
                LTHREAD(NEV)←RTHREAD(NEV)← 1
                LPTR(PARENT)←NEV
                LTHREAD(PARENT)← 0
                FLAG=FALSE
           ELSE
                PARENT←LPTR(PARENT)

  CASE 2:  IF (RTHREAD (PARENT) = 1)
             THEN NEV←GETNODE(X)
                RPTR(NEV)←RPTR(PARENT)
                LPTR(NEV)←PARENT
                LTHREAD(NEV)←RTHREAD(NEV)← 1
                RPTR(PARENT)←NEV
                RTHREAD(PARENT)← 0
                FLAG=FALSE
           ELSE
                PARENT←RPTR(PARENT)

5. RETURN(PARENT)
```


25

TBT-Preorder Traversal

```
PROCEDURE PRETHREAD(HEAD):

1.CURRENT←LPTR(HEAD)

2.IF(CURRENT= HEAD)
  THEN WRITE('EMPTY TREE')
     RETURN

3. REPEAT THRU STEP 5 WHILE CURRENT != HEAD

4. WRITE(DATA(CURRENT))

5. IF(LTHREAD(CURRENT)=0)
     CURRRENT←LPTR(CURRENT)
  ELSE
    REPEAT WHILE(RTHREAD(CURRENT)=1)
        CURRENT←RPTR(CURRENT)
    CURRENT←RPTR(CURRENT)

6. RETURN
```


26

TBT-Inorder Traversal

```
PROCEDURE INTHREAD(HEAD):

1.CURRENT←LPTR(HEAD)

2.IF(CURRENT= HEAD)
  THEN WRITE('EMPTY TREE')
     RETURN

3. REPEAT WHILE  LTHREAD(CURRENT)=0
          CURRENT←LPTR(CURRENT)

4. REPEAT THRU STEP 6 WHILE CURRENT != HEAD

5. WRITE(DATA(CURRENT)

6. IF  RTHREAD(CURRENT) =1
       CURRRENT←RPTR(CURRENT)
  ELSE
     CURRENT←RPTR(CURRENT)
     REPEAT WHILE(LTHREAD(CURRENT)=0)
        CURRENT←LPTR(CURRENT)
7. RETURN
```


27

TBT-Postorder Traversal (Initialize)

```
PROCEDURE INITIALIZE(HEAD):

1.CURRENT←LPTR(HEAD)

2.IF(CURRENT= HEAD)
  THEN WRITE('EMPTY TREE')
     RETURN

3. REPEAT THRU STEP 4 WHILE CURRENT != HEAD

5. IF(LTHREAD(CURRENT)=0)
     CURRRENT←LPTR(CURRENT)
     FLAG(CURRENT)=0

  ELSE
    REPEAT WHILE(LTHREAD(CURRENT)=1)
        CURRENT←RPTR(CURRENT)
        FLAG(CURRENT)=0

   CURRENT←RPTR(CURRENT)
   FLAG(CURRENT)=0

6. RETURN
```


28

TBT-Postorder Traversal

```
PROCEDURE POSTTHREAD(HEAD):

1.TEMP←left(HEAD)
  FLAG(TEMP)←0

2. REPEAT THRU STEP 5 WHILE TEMP != HEAD

3. if Lbit(TEMP)=0 AND FLAG(left(TEMP))!=2)
  then TEMP←left(TEMP)
          FLAG(TEMP) ← 0
  return

4. if Rbit(TEMP)=0 AND FLAG(right(TEMP))!=2)
  then TEMP←right(TEMP)
          FLAG(TEMP) ← 1
  return
```

```
5. WRITE (DATA(TEMP))
   if(FLAG(TEMP) = 0)
   then FLAG(TEMP)=2
            WHILE(Rbit(TEMP)=0)
                TEMP=right(TEMP)
            TEMP=right(TEMP)
   else
            FLAG(TEMP)=2
            WHILE(Lbit(TEMP)=0)
                TEMP=left(TEMP)
            TEMP=left(TEMP)

6. return
```

29

TBT-Postorder Traversal

```
PROCEDURE POSTTHREAD(HEAD):

1.TEMP←LPTR(HEAD)
  FLAG(TEMP)←0

2. REPEAT THRU STEP 5 WHILE TEMP != HEAD

3. IF LTHREAD(TEMP)=0 AND FLAG(LPTR(TEMP))!=2)
  THEN TEMP←LPTR(TEMP)
          FLAG(TEMP) ← 0
  RETURN

4. IF RTHREAD(TEMP)=0 AND FLAG(RPTR(TEMP))!=2)
  THEN TEMP←RPTR(TEMP)
          FLAG(TEMP) ← 1
  RETURN
```

5 -30

30

TBT-Postorder contd..

```
5. WRITE (DATA(TEMP))
   IF(FLAG(TEMP) = 0)
   THEN FLAG(TEMP)=2
            WHILE(RTHREAD(TEMP)=0)
                TEMP=RPTR(TEMP)
            TEMP=RPTR(TEMP)
   ELSE
            FLAG(TEMP)=2
            WHILE(LTHREAD(TEMP)=0)
                TEMP=LPTR(TEMP)
            TEMP=LPTR(TEMP)

6. RETURN
```

5 -31

31

TBT EXP creation

```
TBT_ExptTee()
{
    while(str1(i)
!='\0'){
    if(isalpha(str1[i]))
        {parent = new node
        parent->data = str1[i]
        parent->lthd = true
        parent->rthd = true
        parent->rlink = NULL
        parent->llink = NULL
    push( stack_data,parent)
    }
```

```
else // when the token is
   oerator
    {   parent = new node
        parent->data = str1[i]
        parent->lthd = false
        parent->rthd = false
        parent->rlink = pop()
        parent->llink = pop()
        t1= parent ->lrlink
        t2 =parent->rlink
        if(t2->llink==NULL)
            t2->llink = parent
        if(t1->rlink==NULL)
        t1->rlink = parent
```

32

TBT Expt Tree Creation

```
temp1 = t1 temp2 = t2
  while(!(temp1->rthd))
        temp1 = temp1->rlink
 temp1->rlink = parent
 while(!(temp2->lthd))
     temp2 = temp2->llink
temp2->llink = parent;
     push(parent);
     }
     i++ //  take next token
    }
```

```
root = stack_data.pop_data()
    head->llink=root
    temp1 = root->llink
    temp2 = root->rlink
  while(!(temp1->lthd))
        temp1 = temp1->llink
  temp1->llink = head
  while(!(temp2->rthd))
        temp2 = temp2->rlink
  temp2->rlink = head
}
```