Data Structure

http://smartlead.hallym.ac.kr

Instructor: Jin Kim
010-6267-8189(033-248-2318)
jinkim@hallym.ac.kr
Office Hours:

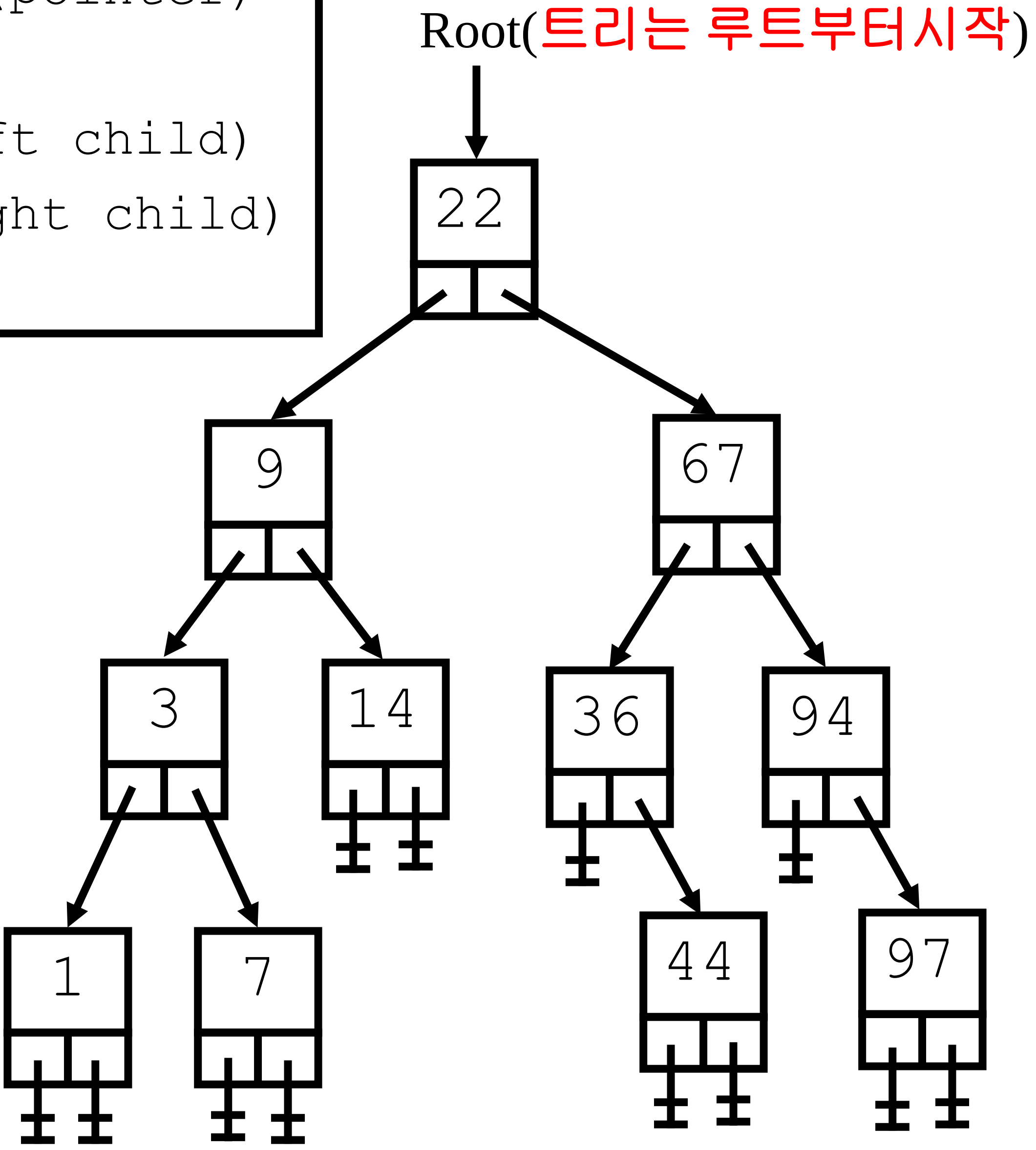
Non Linear Data Structure

- Data structure we will consider this semister:
  - Tree
  - Binary Search Tree
  - Graph
  - Weighted Graph
  - Sorting
  - Balanced Search Tree

Threaded binary tree
쓰레드 이진 트리

Thread ? 실, 흐름의 단위,
프로세스보다도 작은 실행 흐름의
최소 단위
Threaded ? 실로 꿰어진
Threaded binary tree? 쓰레드 이진트리

```
Proc PostOrderPrint(pointer)
 pointer NOT NIL?
L PostOrderPrint(left child)
R PostOrderPrint(right child)
P print(data)
```



Threaded Binary Trees
스레드이진트리

Given a binary tree with *n* nodes,

⇨ the total number of links in the tree is 2*n*.

Each node (except the root) has exactly one incoming arc

⇨ only *n* - 1 links point to nodes

⇨ remaining *n* + 1 links are null. (널 수가 너무 많다, 유용하게사용하자)

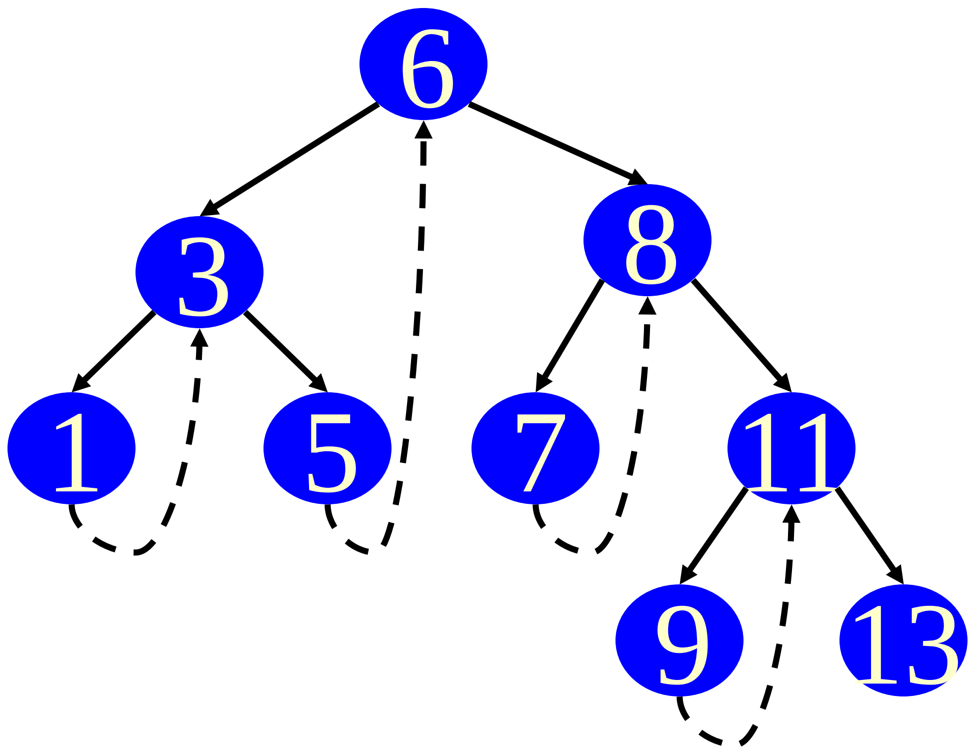
One can use these null links to simplify some traversal processes.

A **threaded binary search tree** is a BST with unused links employed to point to other tree nodes.

⇨ *Traversals* (as well as other operations, such as backtracking) made *more efficient.*

A BST can be threaded with respect to inorder, preorder or postorder successors. 중위,후위,전위순회에 사용될 수 있다

Threaded Tree Example

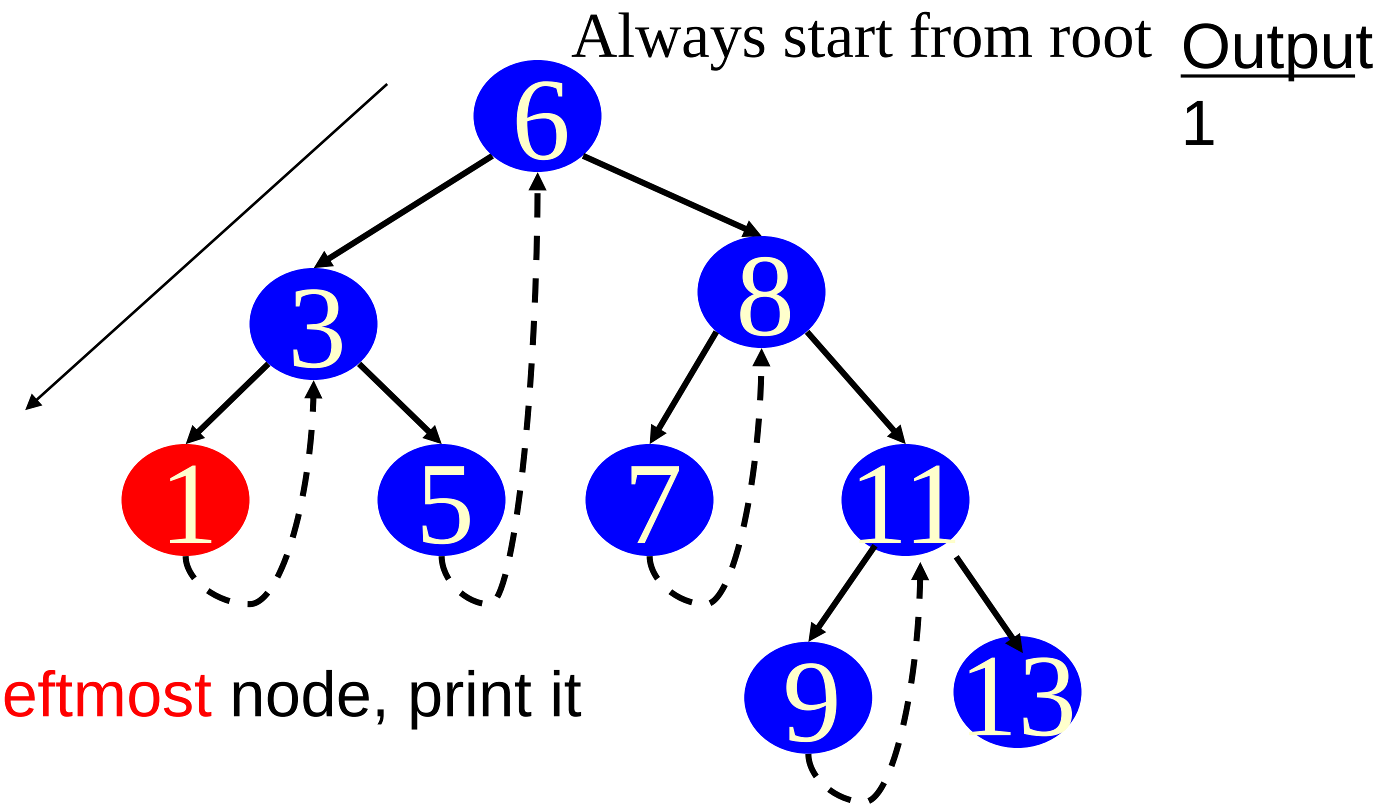


Threaded Tree Traversal
(스레드트리중위순회 알고리즘)

- We start at the leftmost node in the tree, print it, and follow its right thread(루트에서 시작, 왼쪽으로 끝까지 내려가서 데이터출력, 이후 오른쪽을 본다)
- If we follow a thread(다음방문노드에 대한 주소) to the right, we output the node and continue to its right(오른쪽이 스레드이면 그 스레드를 따라가서 출력)
- If we follow a link(오른쪽자식에대한주소) to the right, we go to the leftmost node, print it, and continue(오른쪽이 링크이면, 오른쪽자식을 방문 다시 왼쪽으로 한없이가서출력)

Threaded Tree Traversal

Inorder threaded tree중위순회용 쓰레드이진트리



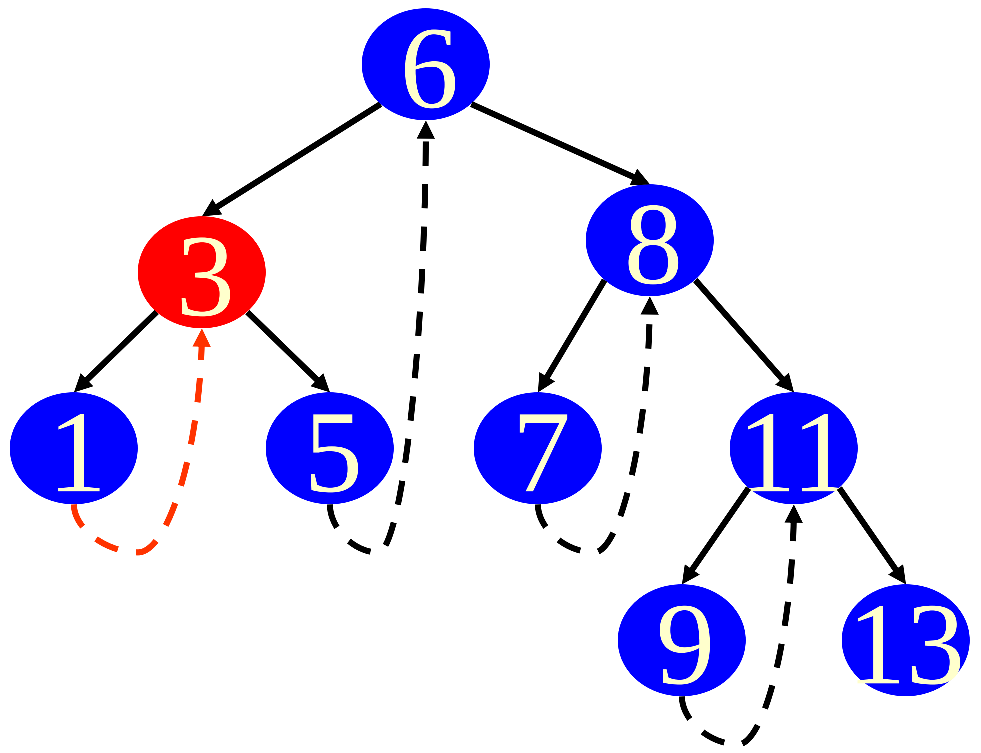
Start at leftmost node, print it

⟵ 링크. 왼쪽자식에 대한 끈

⤴ 쓰레드. 다음 방문할 노드에 대한 끈

Threaded Tree Traversal

Inorder threaded tree중위순회용 쓰레드이진트리

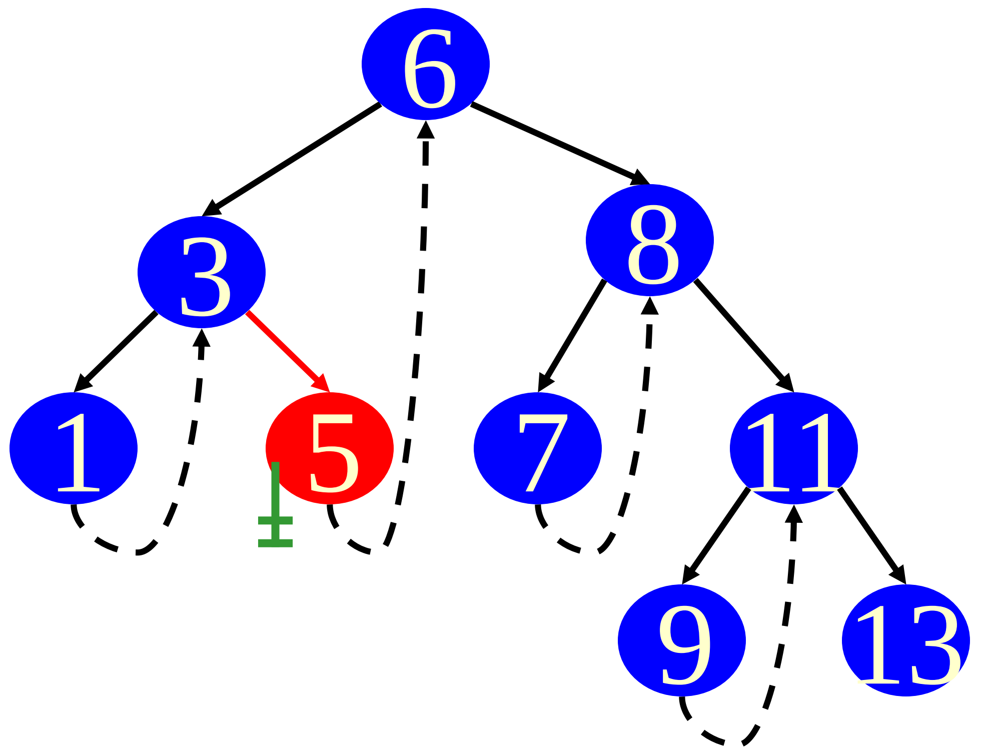


Output
1
3

Follow thread to right, print node

Threaded Tree Traversal

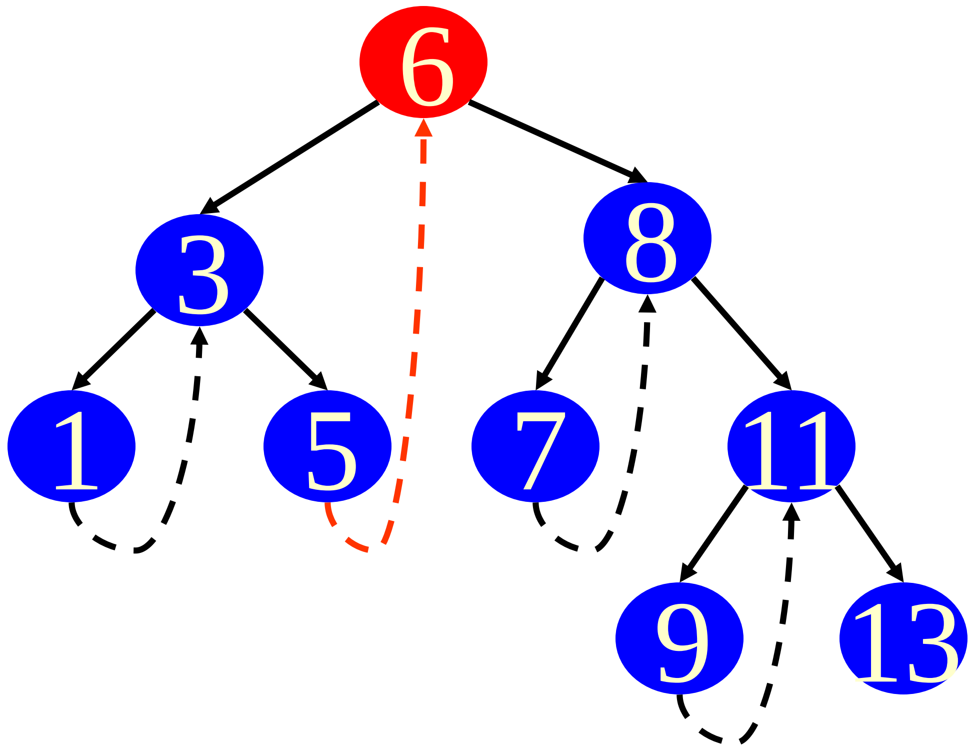
Inorder threaded tree중위순회용 쓰레드이진트리



Output
1
3
5

Follow link to right, go to leftmost node and print

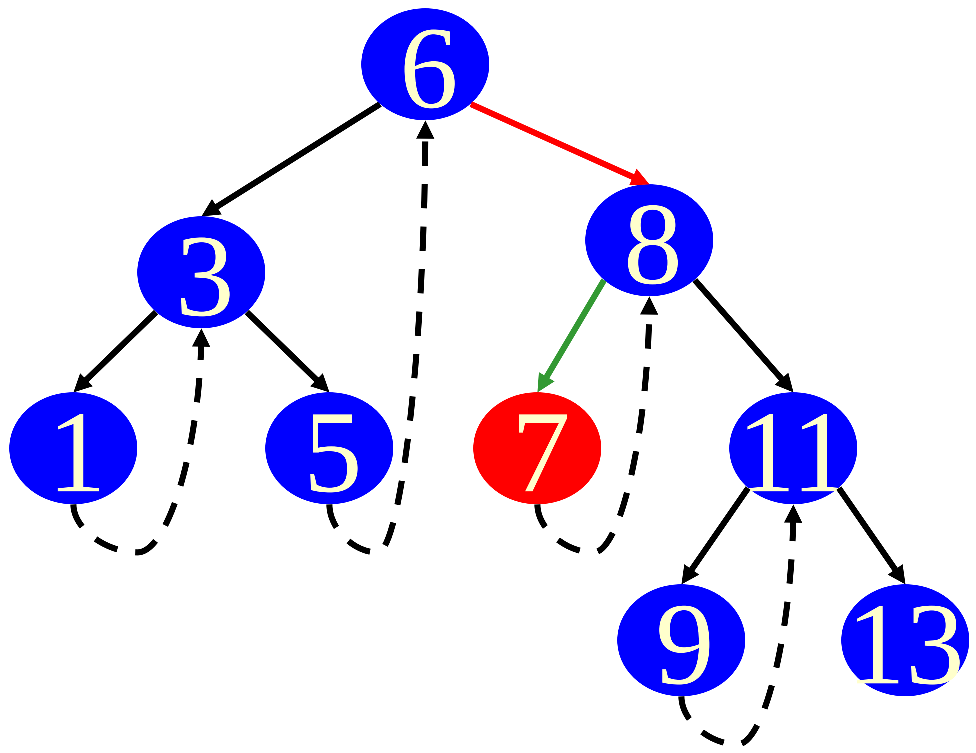
Threaded Tree Traversal



Output
1
3
5
6

Follow thread to right, print node

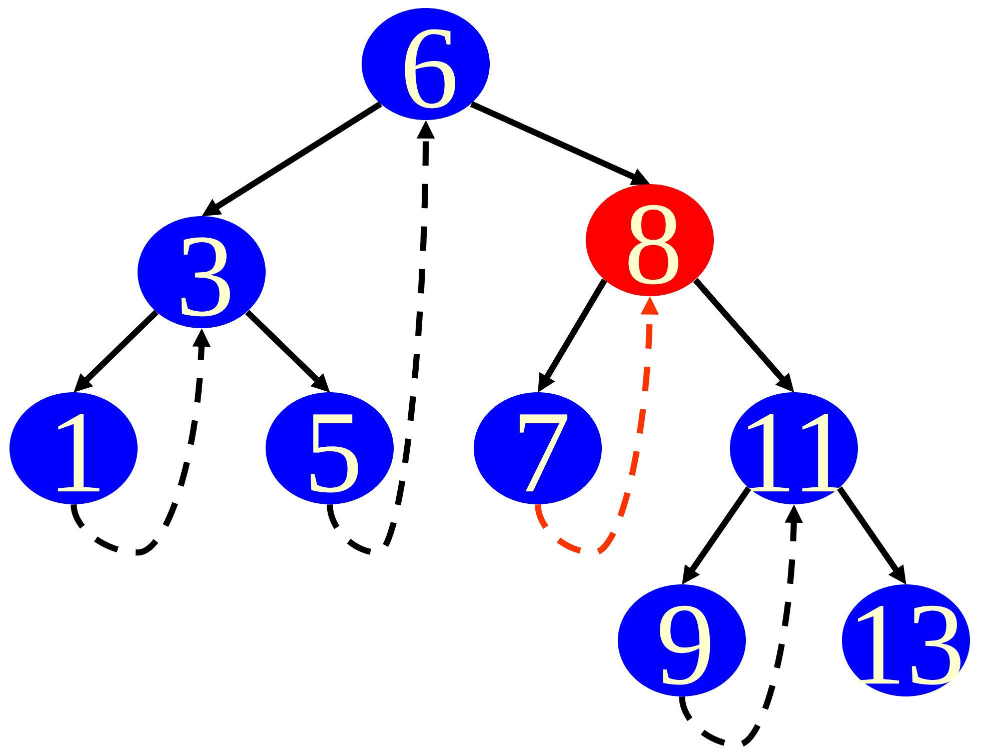
Threaded Tree Traversal



Output
1
3
5
6
7

Follow link to right, go to leftmost node and print

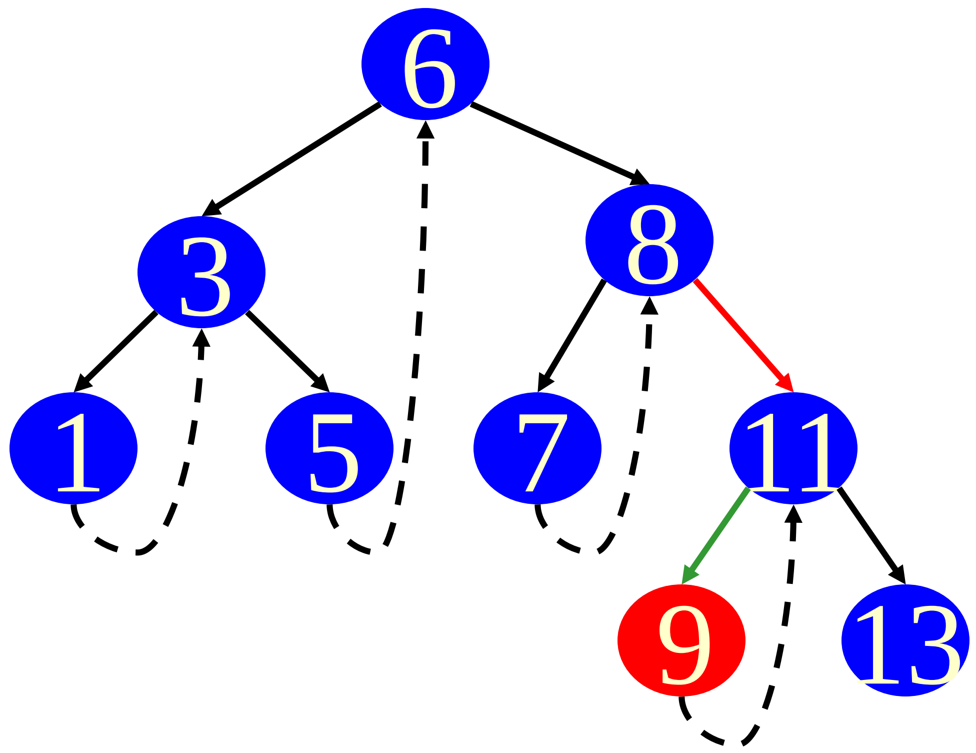
Threaded Tree Traversal



Output
1
3
5
6
7
8

Follow thread to right, print node

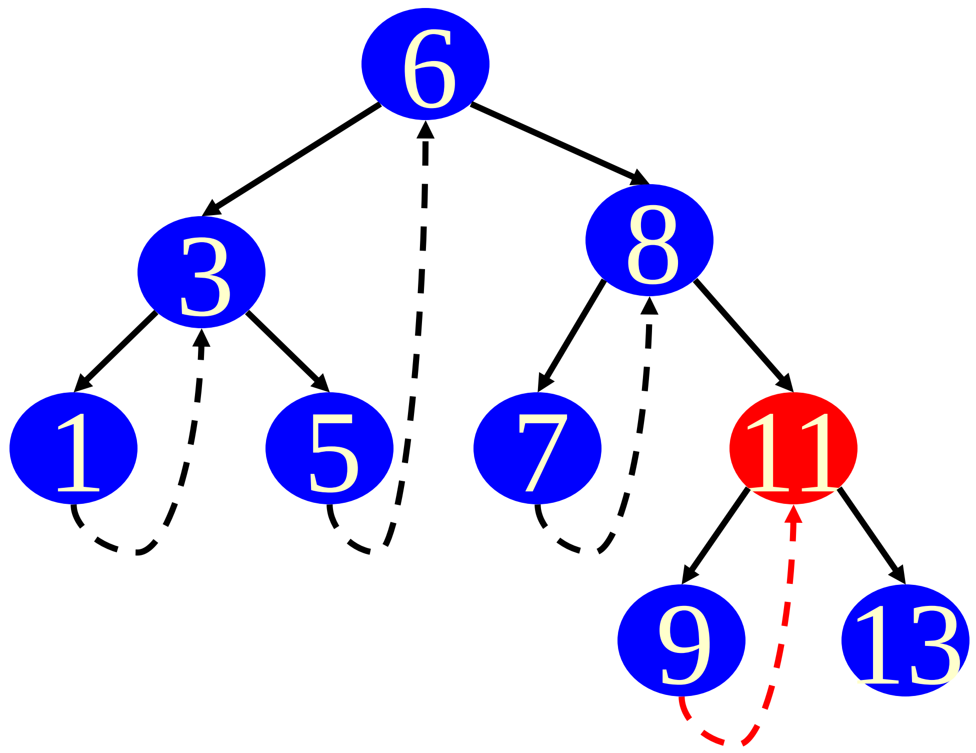
Threaded Tree Traversal



Output
1
3
5
6
7
8
9

Follow link to right, go to leftmost node and print

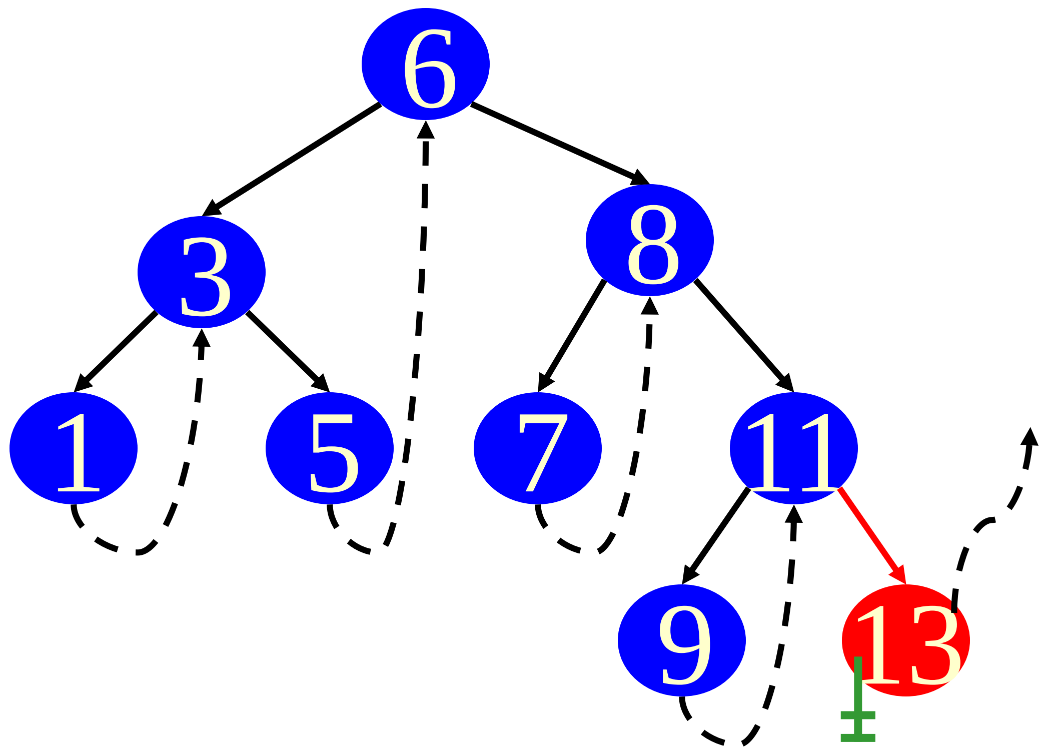
Threaded Tree Traversal



Output
1
3
5
6
7
8
9
11

Follow thread to right, print node

Threaded Tree Traversal



Output
1
3
5
6
7
8
9
11
13

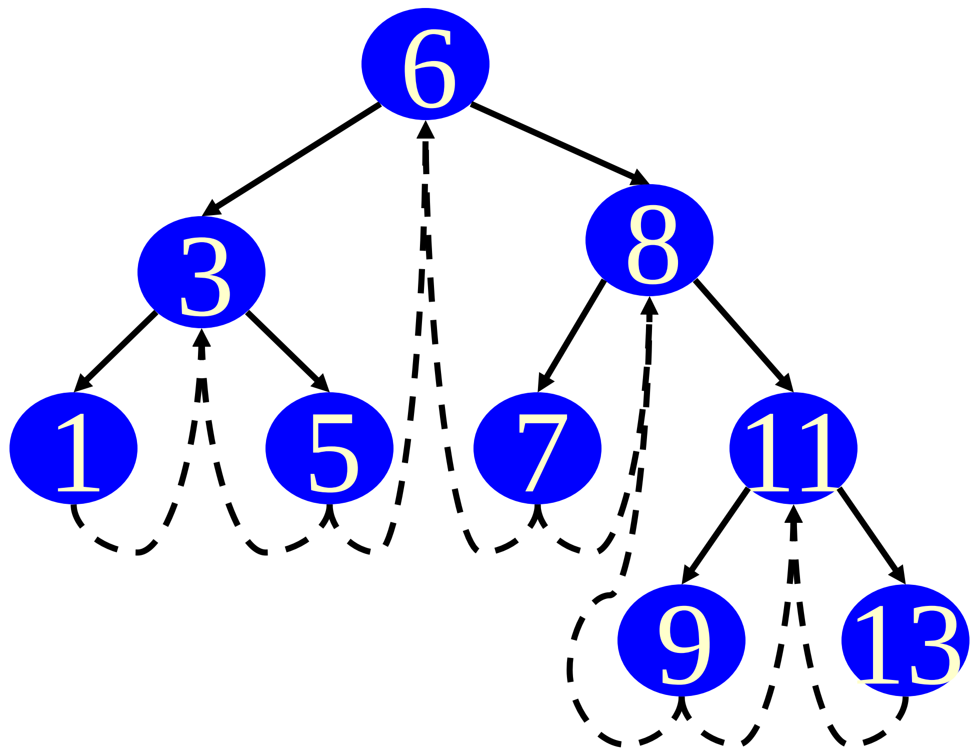
Follow link to right, go to leftmost node and print

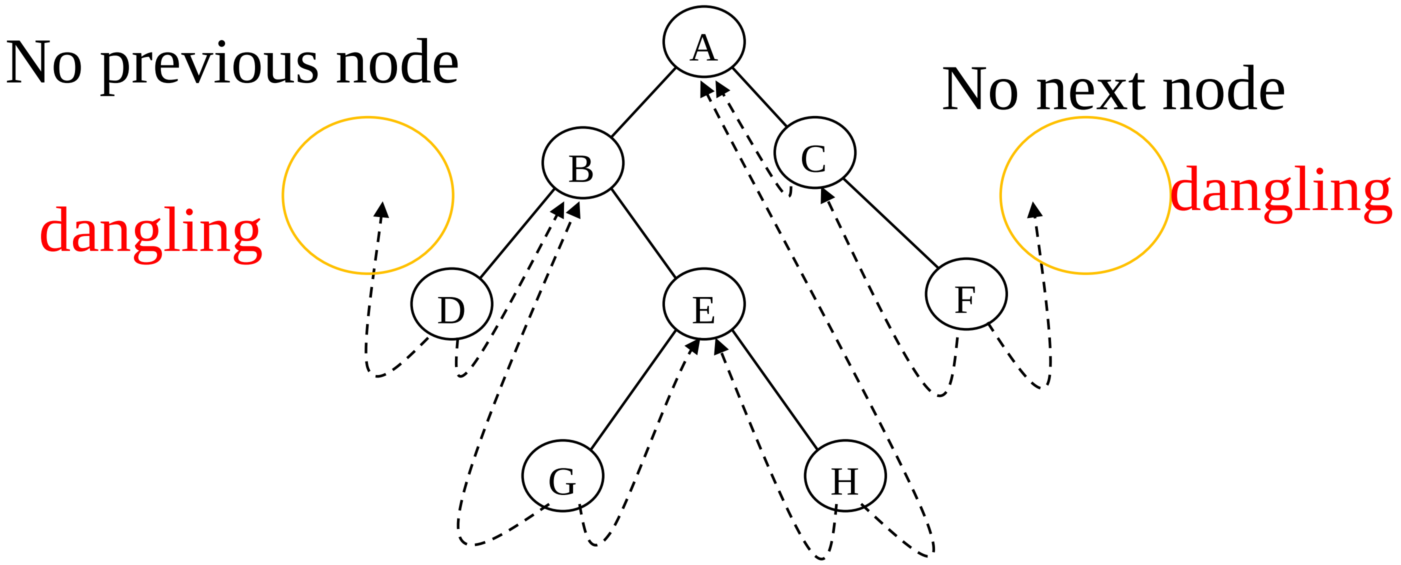
Threaded Tree Modification 쓰레드 트리 수정

- We’re still wasting pointers, since half of our leafs’ pointers are still null
- We can add threads to the previous node in an inorder traversal as well, which we can use to traverse the tree backwards or even to do postorder traversals(중위순회뿐아니라 후위순회용 쓰레드이진트리도 만들수있다)

Threaded Tree Modification

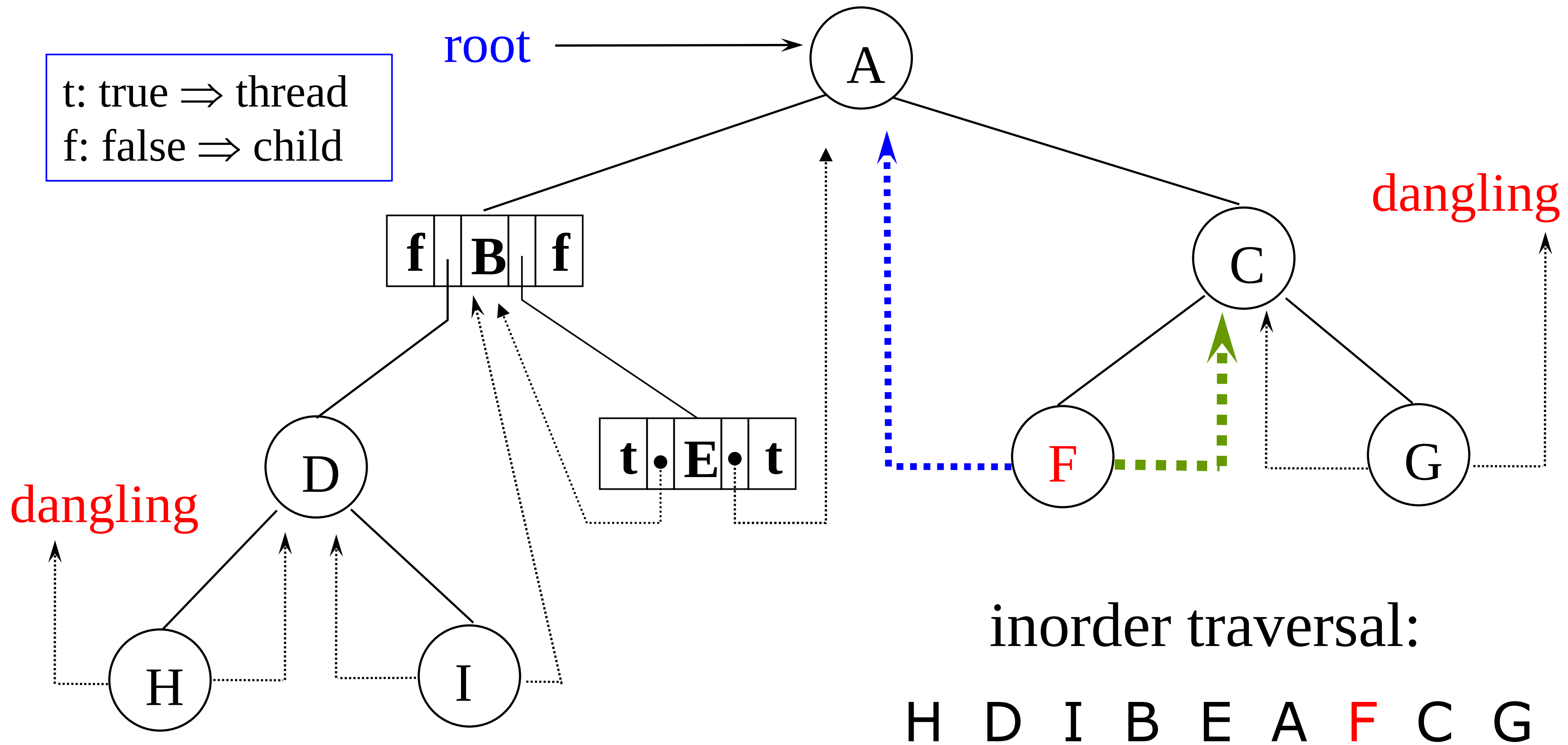
왼쪽 null에 이전에 방문했던 노드의 주소저장




No previous node
dangling
A
B
C
D
E
F
G
H
No next node
dangling

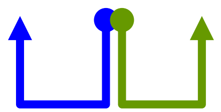
Threaded Binary Trees (3/10)

- A Threaded Binary Tree



inorder traversal:

H D I B E A F C G



Dangling(대롱대롱매달린, 허공에 매달린)

Threaded Binary Trees (5/10)

- If we don’t want the left pointer of H and the right pointer of G to be dangling pointers, we may create root node(헤더노드) and assign them pointing to the root node

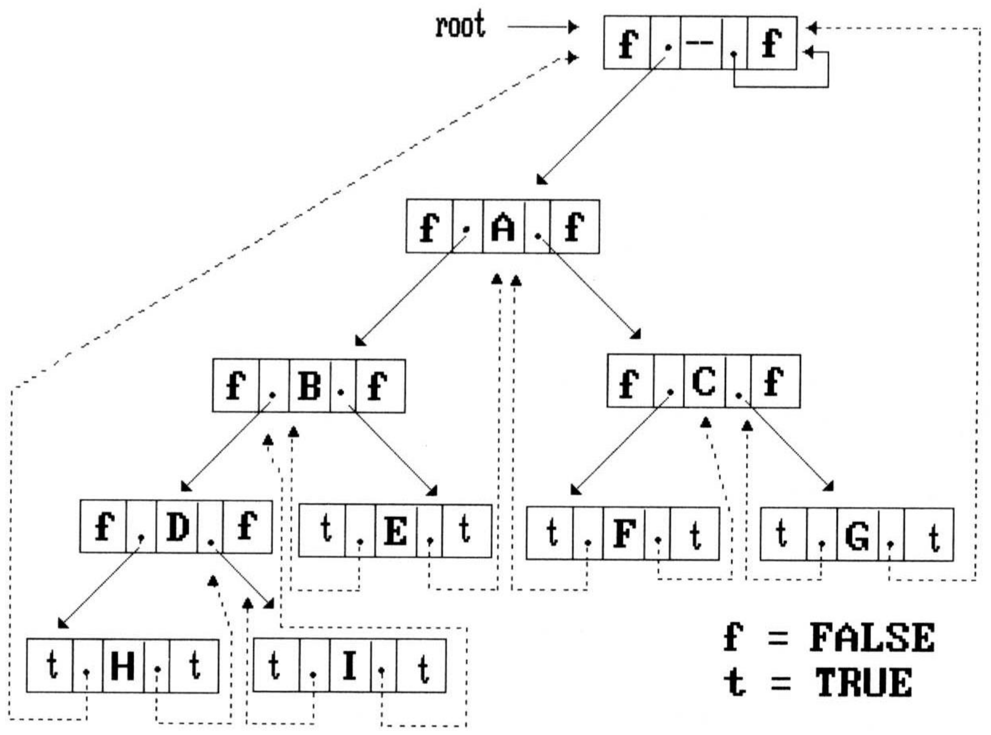
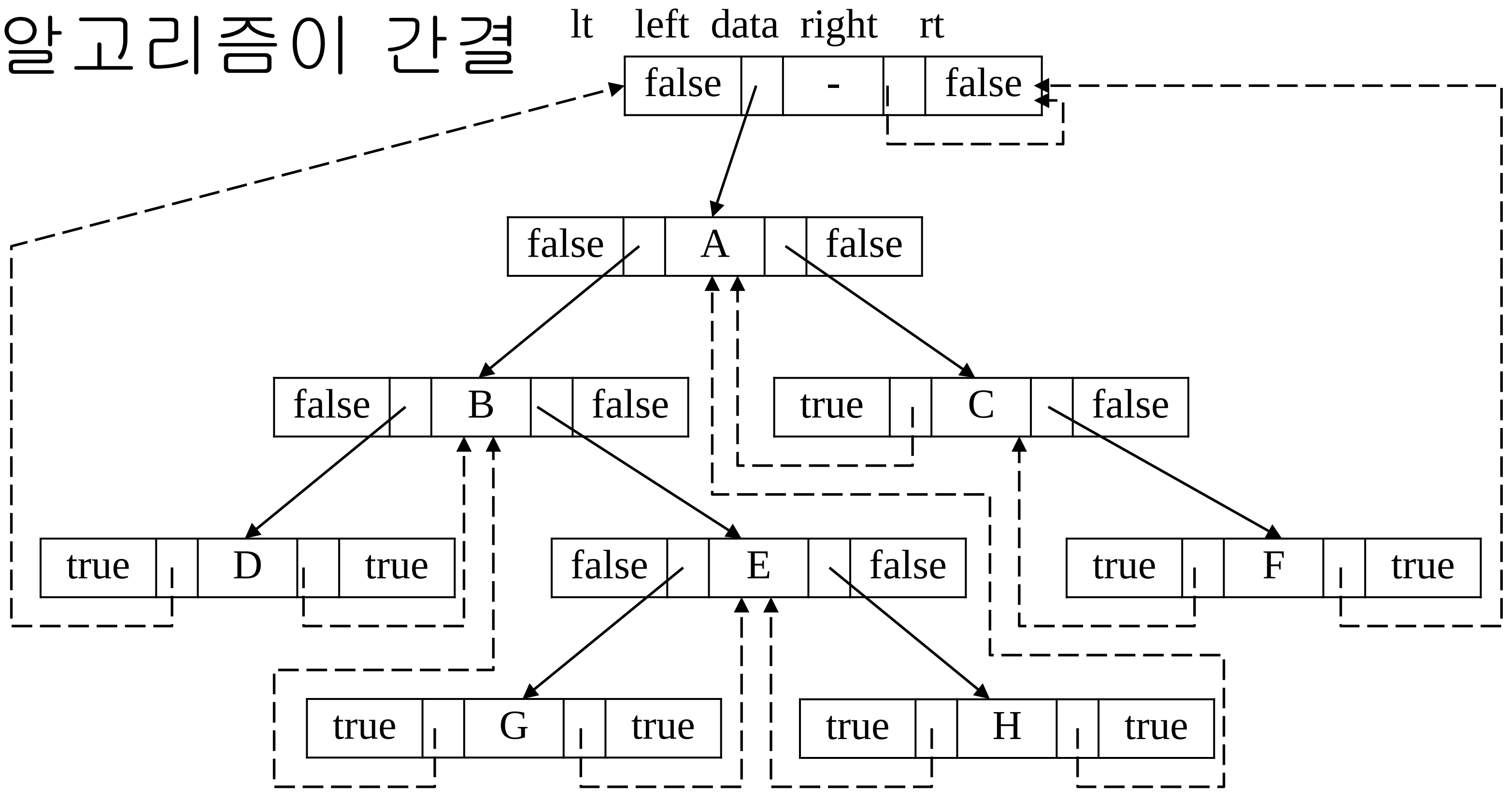


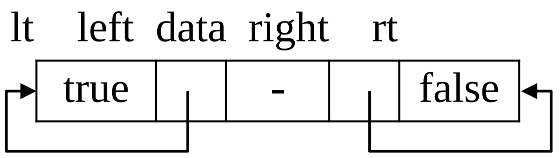
Figure 5.23: Memory representation of a threaded tree

Threaded binary tree with header node

- TBT with header node(헤더노드가짐)
- 알고리즘이 간결



- Null TBT



Threaded Binary Tree

- **Advantages of threaded binary tree장점:**
- Threaded binary trees have numerous advantages over non-threaded binary trees listed as below:
  - The traversal operation is more faster than(빠르다) that of its unthreaded version, because with threaded binary tree non-recursive(재귀가아니기때문에) implementation is possible which can run faster and does not require the botheration of stack management.

Threaded Binary Tree

- **Advantages of threaded binary tree장점:**
  - The second advantage is more understated with a threaded binary tree, we can efficiently determine the predecessor and successor(선행자, 후속자를 효율적으로 결정가능) nodes starting from any node. In case of unthreaded binary tree, however, this task is more time consuming and difficult. For this case a stack is required to provide upward pointing information in the tree whereas in a threaded binary tree, without having to include the overhead of using a stack mechanism the same can be carried out with the threads.

Threaded Binary Tree

- **Advantages of threaded binary tree장점:**
  - Any node can be accessible from any other node. (원형연결리스트의 성격을 가졌으므로, 어느노드에서 출발해도 모든 노드를 방문가능) Threads are usually more to upward whereas links are downward. Thus in a threaded tree, one can move in their direction and nodes are in fact circularly linked. This is not possible in unthreaded counter part because there we can move only in downward direction starting from root.

Threaded Binary Tree

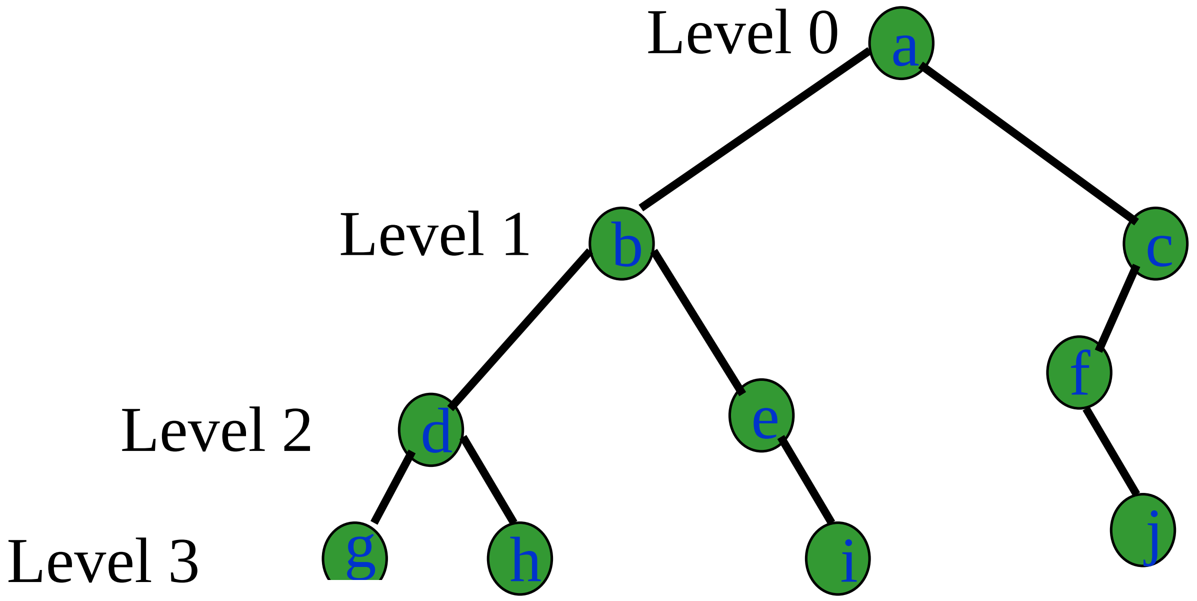
- **Disadvantages of threaded binary tree단점:**
  - Insertion and deletion from a threaded tree are very time consuming(삽입삭제가 복잡) operation compare to non-threaded binary tree.
  - This tree requires additional bit(자식, 스레드구별을 위한 추가적인 비트필요) to identify the threaded link.

Questions?

Sequential Representation of Binary Trees(이진트리순차표현)

- There are several ways to represent a binary tree. Although it may appear that we must use pointers, this is not required.
- One way to represent binary trees is using arrays. For this to be possible we need to number the elements in such a way that operations to find the other nodes can be done in a systematic way
- The best way to number the elements (nodes) is using a level-by-level numbering. Start from the topmost number the node with 1, move down to the next level and number the nodes with 2 and 3 starting from the left, repeat the process until there are no more levels
- Sequential representation is more cost-effective when the tree is complete

Level Order binary tree traversal 레벨오더순으로 노드 방문



- Traverse by level order
- Use queue
- From top to bottom 위에서 아래로
- Left to right 왼쪽에서 오른쪽으로 방문

a b c d e f g h i j

Level order tree traversal algorithm 레벨오더트리방문 알고리즘

Queue 큐 사용

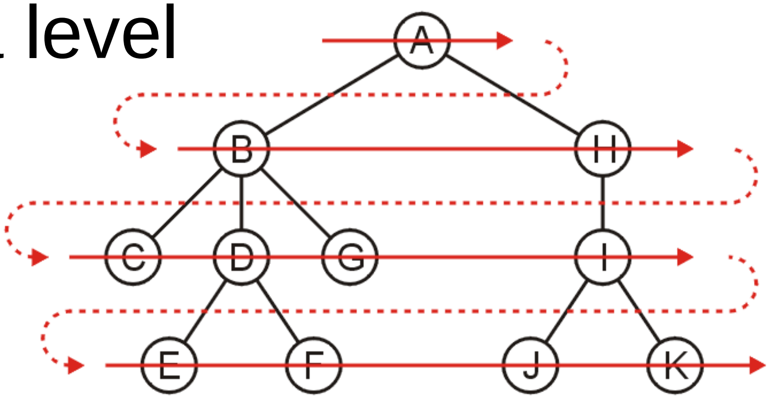
```
levelorder(T)
   initialize queue; // 큐를 초기화
   enqueue(queue,T);
   while (not (isEmpty(queue))) do {
       p←dequeue(queue);
       if (p≠null) then {
           visit p.data;
           enqueue(queue, p.left);
           enqueue(queue, p.right);
       }
   }
 end levelorder()
```

Application(응용)

We would rather search the more shallow directories first then plunge deep into searching one sub-directory and all of its contents

One such search is called a *breadth-first traversal(level order traversal) 너비우선탐색에 사용*

- Search all the directories at one level before descending a level



Applications(너비우선탐색)

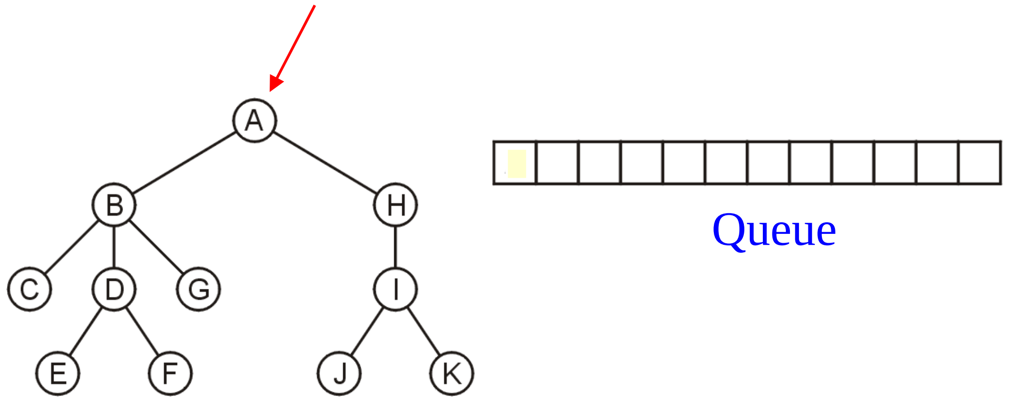
The easiest implementation 구현방법is:

- Place the root directory into a queue루트노드를 큐에 입력
- While the queue is not empty큐가 비어있지않을 동안:
  - Pop the directory at the front of the queue큐에서원소를팝
  - Push all of its sub-directories into the queue 팝한원소의 모든자식들을 큐에삽입

The order in which the directories come out of the queue will be in breadth-first order

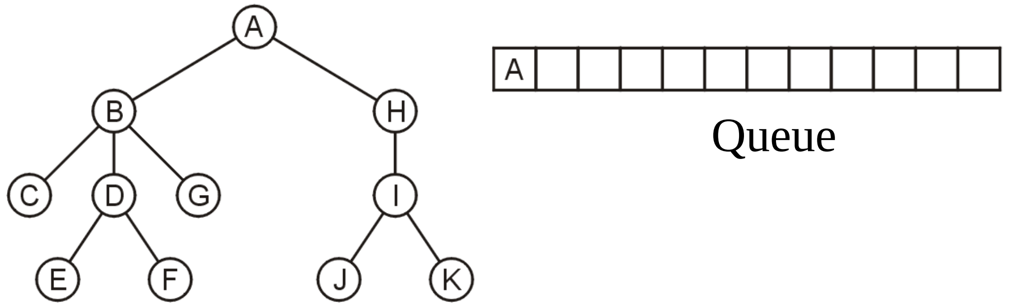
Application

We always start from root 항상 루트부터 입력



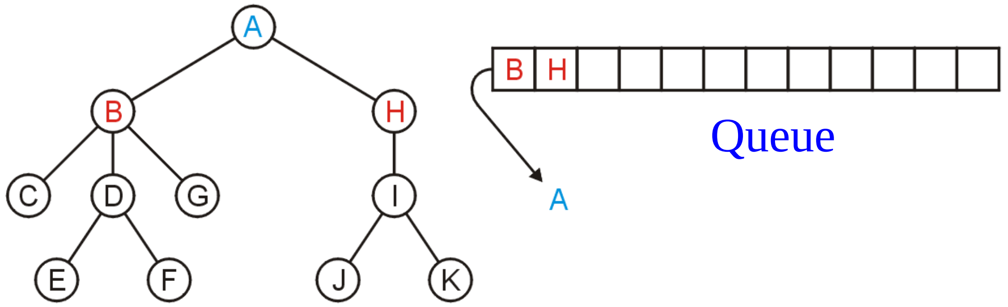
Application

Push the root directory A



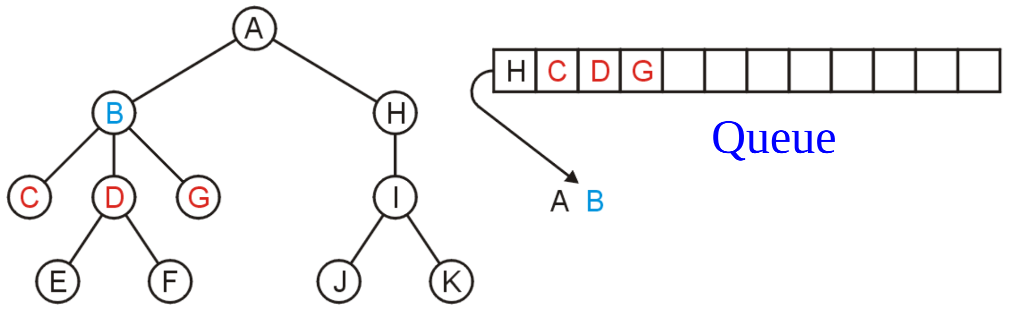
Application

Pop A and push its two sub-directories: B and H



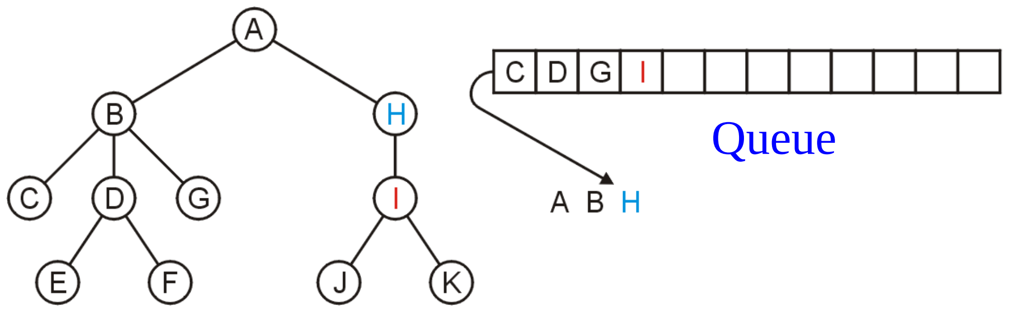
Application

Pop B and push C, D, and G



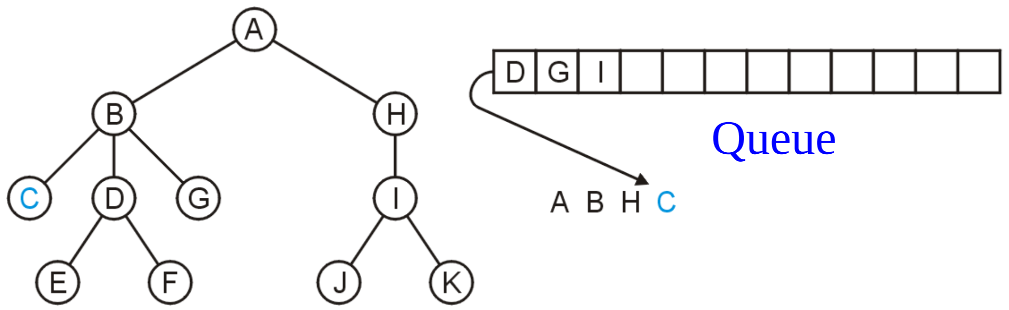
Application

Pop H and push its one sub-directory I



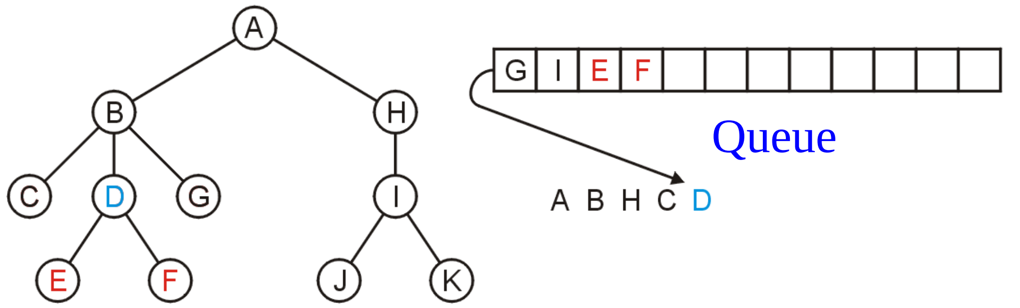
Application

Pop C: no sub-directories



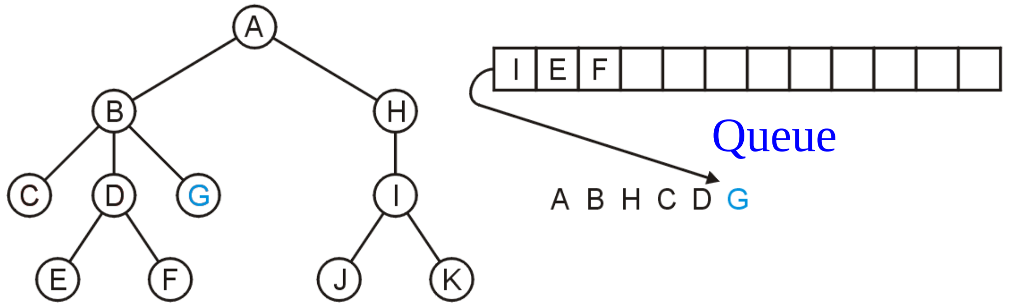
Application

Pop D and push E and F



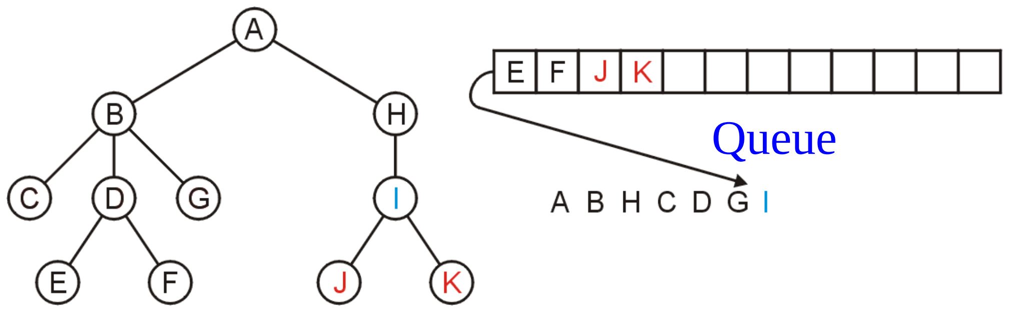
Application

Pop G



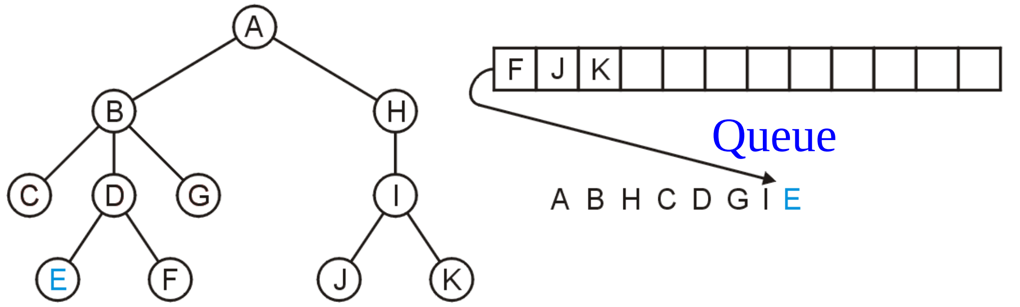
Application

Pop I and push J and K



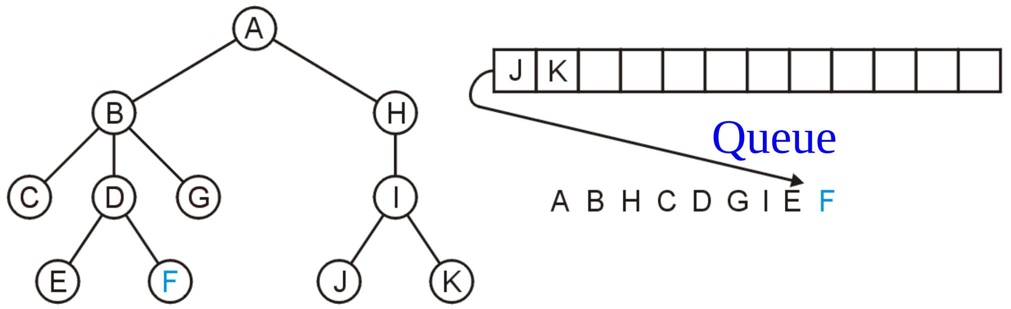
Application

Pop E



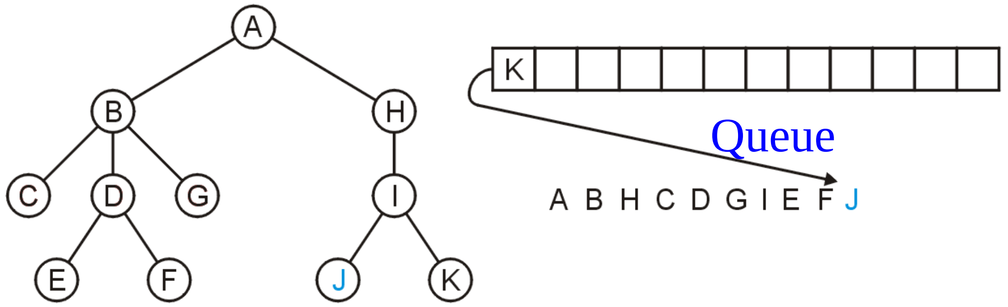
Application

Pop F



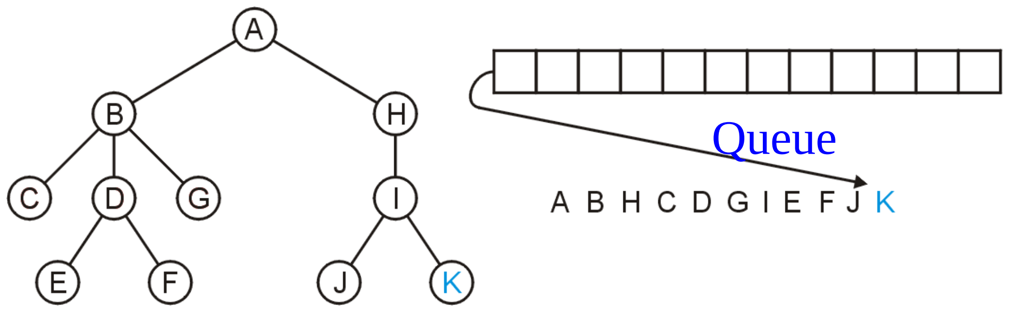
Application

Pop J



Application

Pop K and the queue is empty

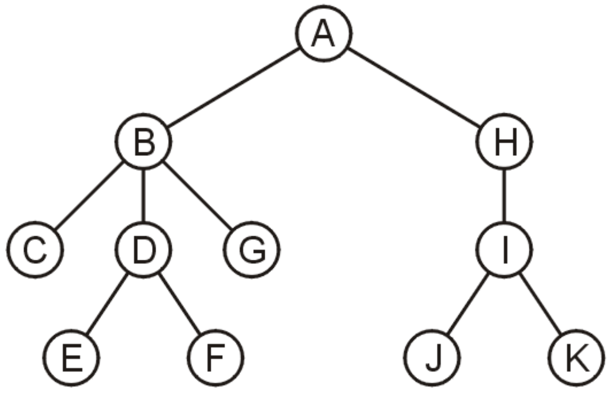


Application

The resulting order 레벨순서별로 출력결과

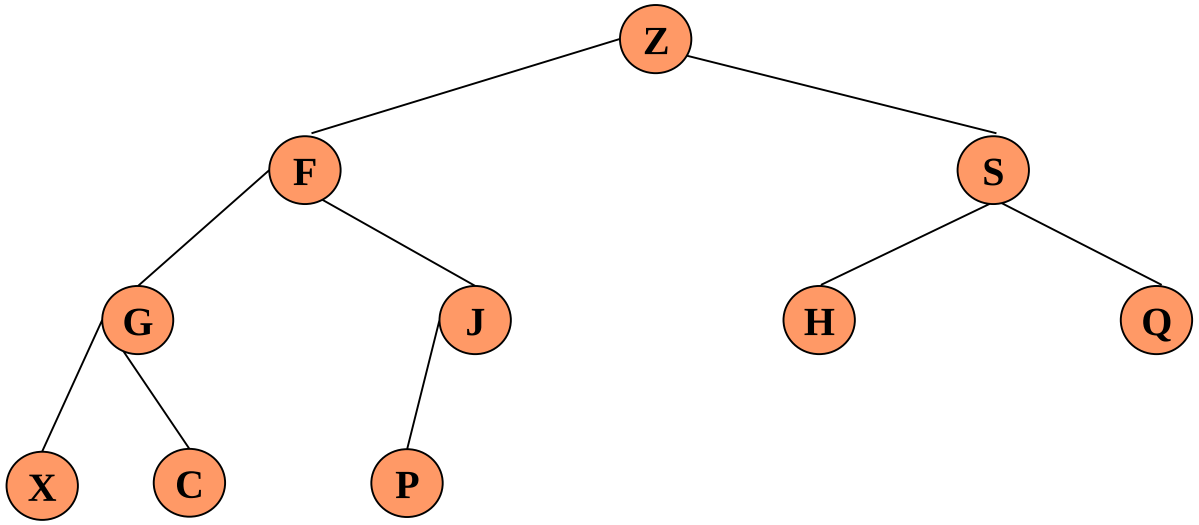
A B H C D G I E F J K

is in breadth-first order(level order):



Level order of a binary tree

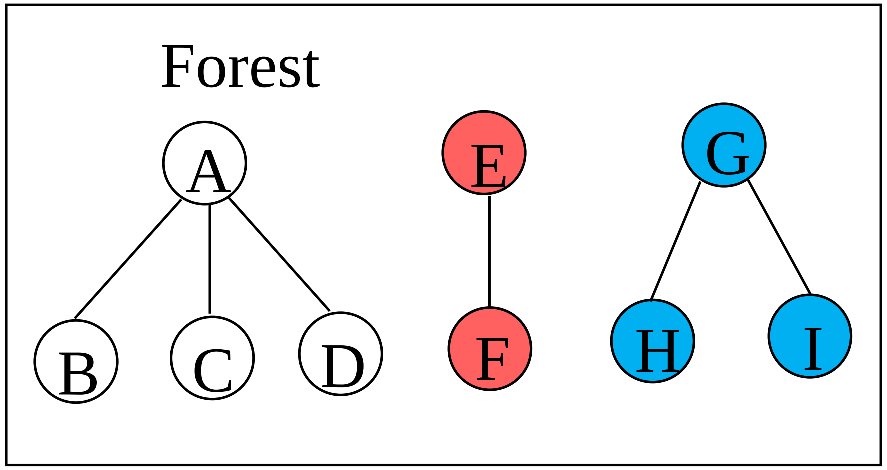
- Given a complete binary tree as below if we perform the level-by-level numbering we get…



Forests(숲) (1/4)

- Definition:
  - A *forest* is a set of $n \geq 0$ disjoint trees(나무들의 집합)
- Transforming a forest into a binary tree
  - Definition: If $T_1,\ldots,T_n$ is a forest of trees, then the binary tree corresponding to this forest, denoted by $B(T_1,\ldots,T_n)$:
  - is empty, if $n = 0$
  - has root equal to root(T_1); has left subtree equal to B($T_{11},T_{12},\ldots,T_{1m}$); and has right subtree equal to B($T_2,T_3,\ldots,T_n$)
    - where $T_{11},T_{12},\ldots,T_{1m}$ are the subtrees of root (T_1)

Forest=tree1+tree2+tree3



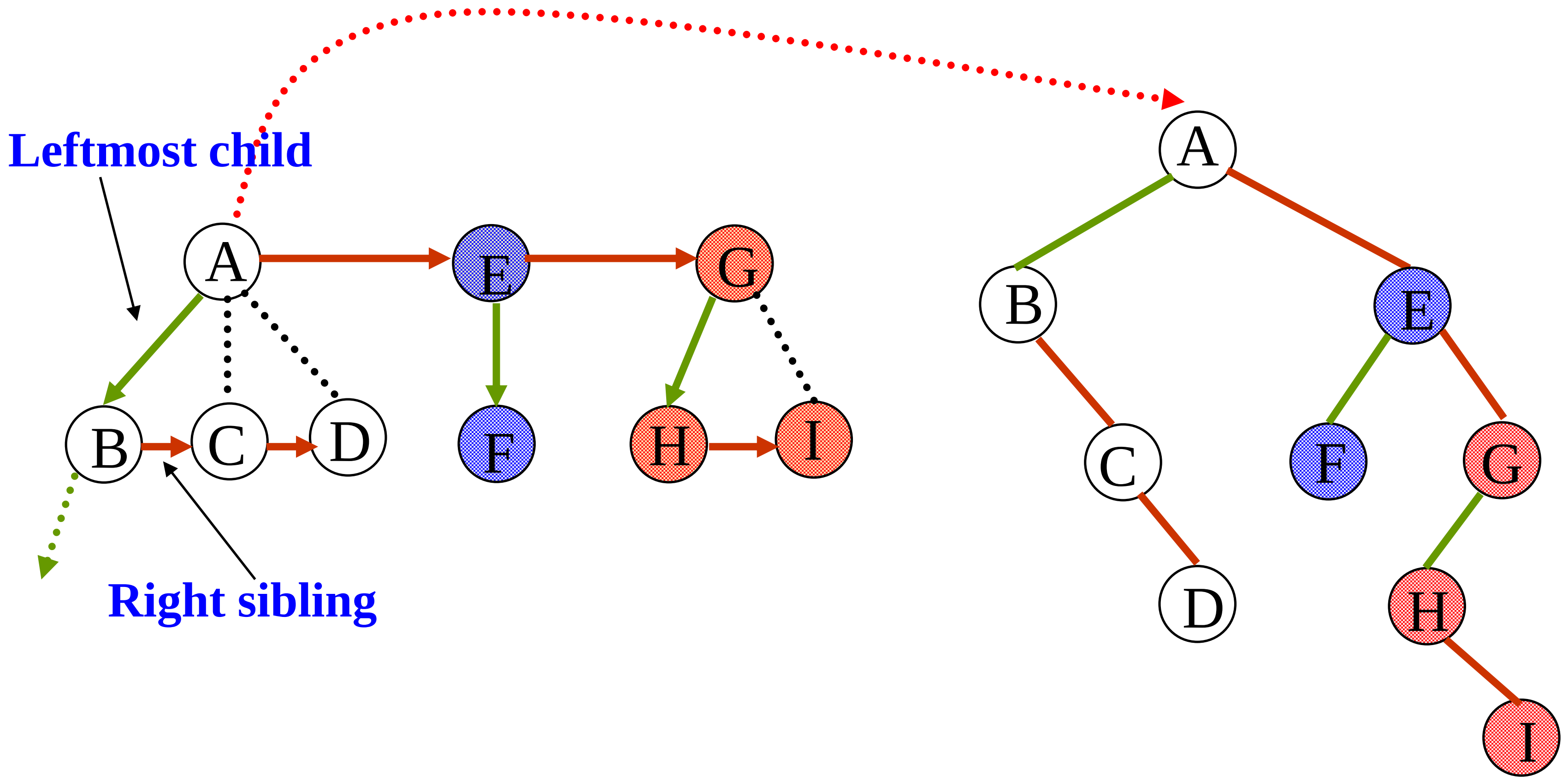
Tree1 tree2 tree3

Transform a forest into a binary tree

- T1, T2, …, T*n*: a forest of trees
  B(T1, T2, …, Tn): a binary tree corresponding to this forest
- algorithm
  (1) empty, if n = 0
  (2) has root equal to root(T1)
      has left subtree equal to B(T11,T12,…,T1*m*)
      has right subtree equal to B(T2,T3,…,Tn)

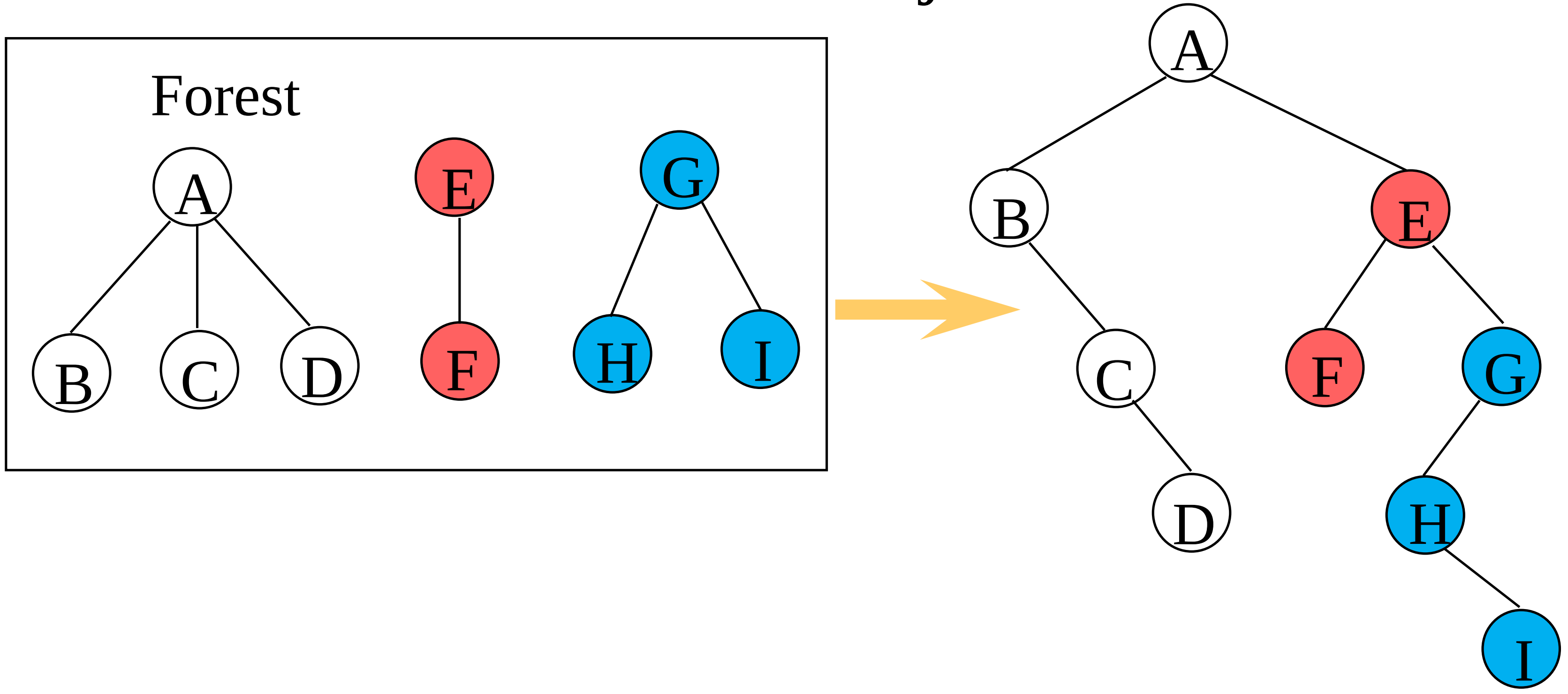
Forests (2/4)

- Rotate the tree clockwise by 45 degrees



Forest->binary tree

- A forest is a set of n >= 0 disjoint trees



감사합니다.