제 9 강의 . 트리의 탐색

1. 이진트리 탐색 알고리즘
2. 쓰레드(Threaded) 이진 트리
3. 이진트리를 다루는 알고리즘

1. 이진트리 탐색 알고리즘

트리의 **탐색(traversal)**은 트리의 각 노드를 방문하는 작업을 말한다.

(왜 방문할 까요?)

다음과 같은 방법들을 생각해 볼 수 있다.

방법 1) 레벨 순 : 레벨이 낮은 순으로 방문

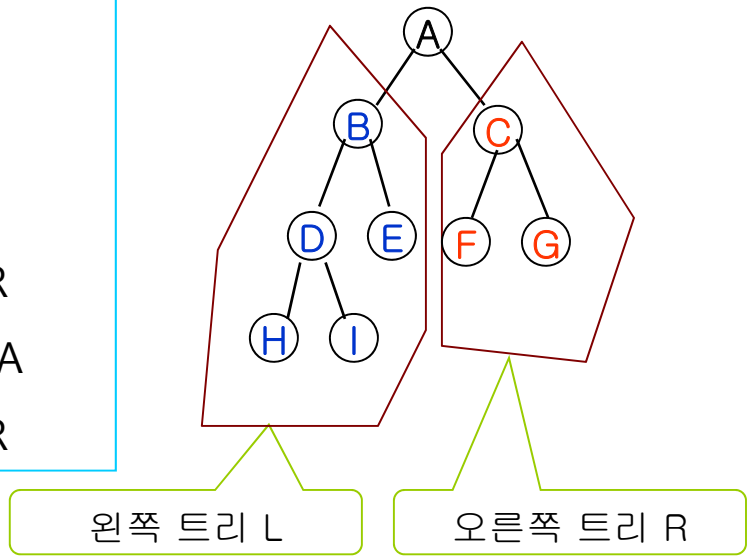
A B C D E F G H ...

3가지 다른 방법이 나올 수 있다.

방법 2) 왼쪽트리 L -> A -> 오른쪽트리 R

방법 3) 왼쪽트리 L -> 오른쪽 트리 R -> A

방법 4) A -> 왼쪽트리 L -> 오른쪽트리 R



트리를 탐색하는 방법 연습 : 아래 트리의 예에서 4가지 탐색 방법에 따라 방문 되는 노드를 살펴보면 다음과 같다.
[참고] http://www.student.seas.gwu.edu/~idsv/idsv.html

1) 중위탐색(inorder traversal) : LVR (Left Visit Right)

2) 전위탐색(preorder traversal) : VLR (Visit Left Right)

3) 후위탐색(postorder traversal) : LRV (Left Right Visit)

다음과 같은 4가지 방법들을 생각해 볼 수 있다.

방법 1) 레벨 순

A B C D E F G H I

방법 2) 중위탐색

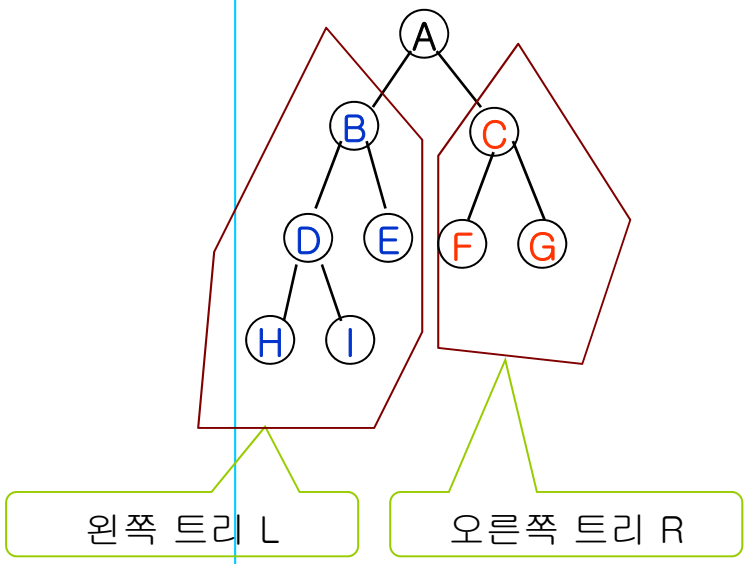
H D I B E A F C G

방법 3) 전위탐색

A B D H I E C F G

방법 4) 후위탐색

H I D E B F G C A



(1) 중위 탐색(inorder traversal)

중위 탐색은

왼쪽 트리 -> 루트노드 -> 오른쪽 트리 순으로 방문을 하게 된다.

주의할 점은 왼쪽 트리를 방문할 때 왼쪽 트리 자체가 트리이기 때문에 왼쪽 트리 안에서도 다시 중위 탐색을 한다

그러므로 왼쪽 트리로 간 다음
왼쪽 트리의 왼쪽 트리, 왼쪽 트리의 루트노드, 왼쪽 트리의 오른쪽 트리 순이 된다.
트리 탐색 알고리즘은 **순환 알고리즘(recursive algorithm)**이 간단하다.

```
void inorder(tree_ptr ptr) {
    if(ptr) {
        inorder(ptr->left_child);
        printf("%d",ptr->data);
        inorder(ptr->right_child);
    }
}
```

중위탐색(inorder traversal) 알고리즘

참고 : 순환 알고리즘(recursive algorithm)

순환 알고리즘이란 프로그램이 자기 자신을 호출하는 프로그램을 말한다.
순환 알고리즘에 대응되는 말은 반복 알고리즘이다.

(예) n! (n factorial 이라고 부름) 함수는 다음과 같이 정의된다.
6! = 6 * 5 * 4 * 3 * 2 * 1 이다.

(반복 알고리즘) 1부터 n 까지 곱한 값을 누적하면 된다.

```
int factorial(int n)
{       int f =1, i=1;
        for(i =1; i <= n; i++)
        {  f = f * i; }
        return f;
}
```

(순환 알고리즘) 프로그램을 순환 알고리즘으로 작성하면 다음과 같다.
6! = 6 * 5! (단 1! = 1)

```
int factorial(int n)
{
        if (n ==1) return 1
        else return ( n * factorial(n-1) );
}
```

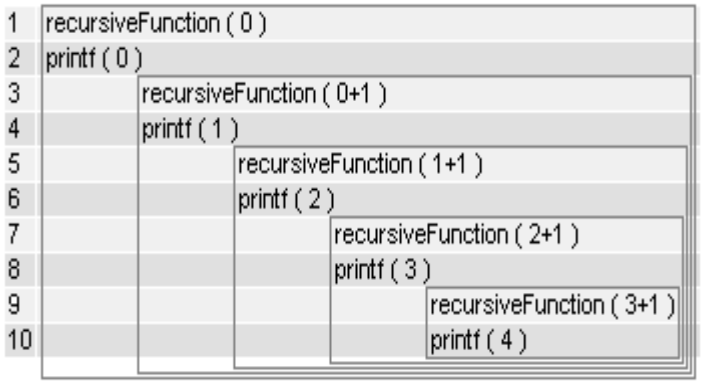
순환 알고리즘(recursive algorithm) 예

(예) Greatest common divisor

$$\gcd(x,y) = \begin{cases} x & \text{if } y = 0 \\ \gcd(y, \text{remainder}(x,y)) & \text{if } y > 0 \end{cases}$$

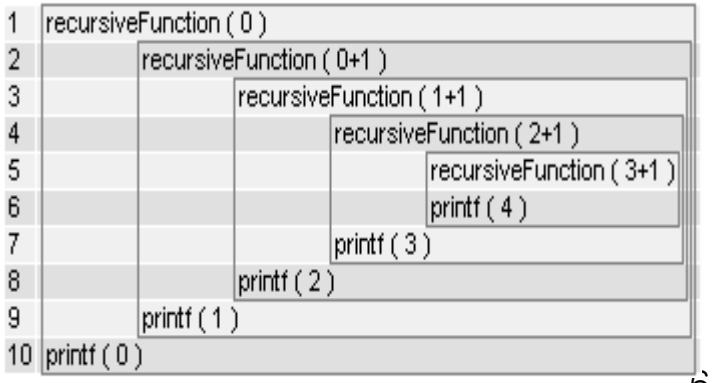
(예)

```
void recursiveFunction(int num)
{ printf("%d₩n", num);
   if (num < 4)
         recursiveFunction(num + 1);
}
```



(예)

```
void recursiveFunction(int num)
{ if (num < 4)
       recursiveFunction(num + 1);
  printf("%d₩n", num);
}
```



중위탐색 예 1)

중위 탐색을 하면

1. 왼쪽 트리(B D E H I), 루트(A), 오른쪽 트리(C F G)이기

   때문에 왼쪽 트리를 방문해야 한다.

2. 왼쪽 트리(B D E H I)가 트리를 구성하기 때문에 왼쪽 트리를 다시 중위 방문을 하면 다음과 같은 방법들을 생각해 볼 수 있습니다. 왼쪽 트리(H D I), 루트(B), 오른쪽 트리(E)

3. 다시 왼쪽 트리를 (H D I)를 중위 탐색을 하면
왼쪽 트리(H), 루트(D), 오른쪽 트리(I)가 된다.

4. 왼쪽 트리(H)를 중위탐색을 하면 왼쪽(없음), 루트노드(H), 오른쪽 트리(없음)이 되기 때문에 H가 결과가 되고 출력된다(출력 => H)

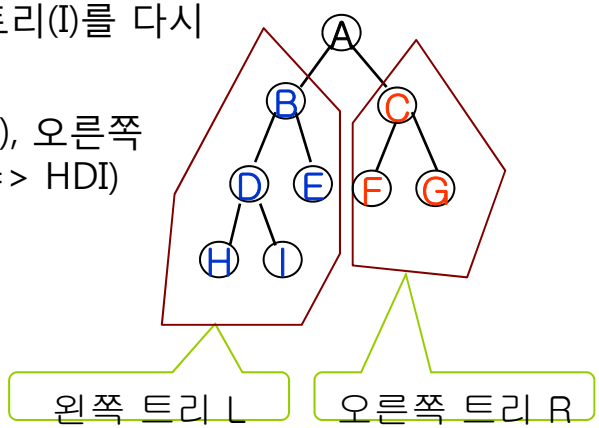
5. 3번으로 돌아가서 루트방문(출력 => HD)하고, 오른쪽 트리(I)를 다시 중위 탐색을 한다.

6. 오른쪽 트리(I)를 중위탐색을 하면 왼쪽(없음), 루트노드(I), 오른쪽 트리(없음)이 되기 때문에 I가 결과가 되고 출력된다(출력 => HDI)

7. 3번이 끝났으므로 2번에서 루트방문(출력 => HDIB)
오른쪽 트리(E)를 방문한다(출력 => HDIBE)

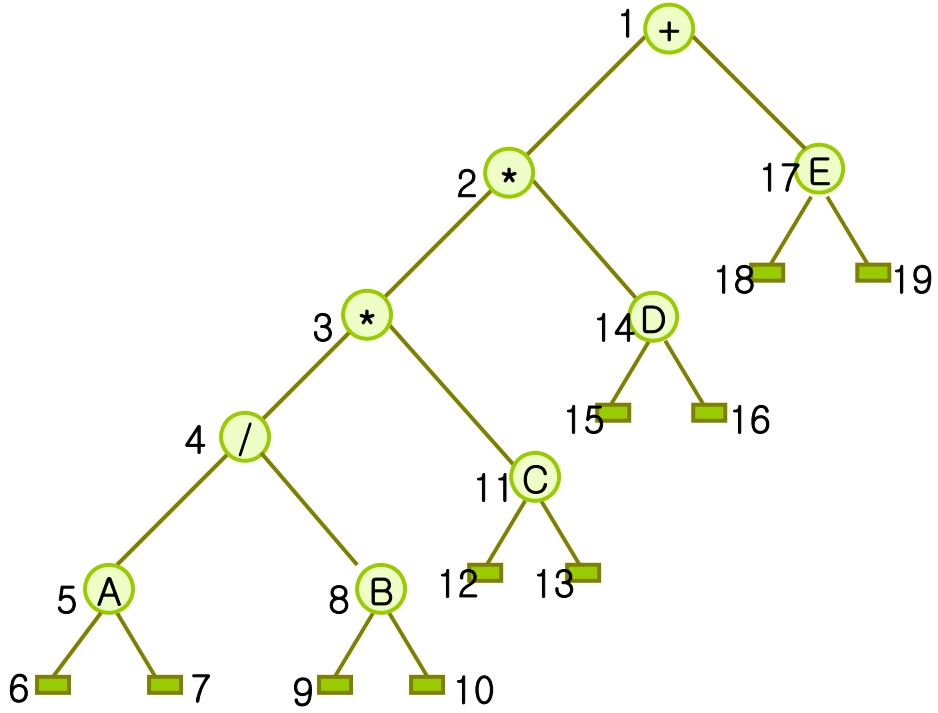
8. 2번이 끝나면 1번의 루트방문(출력 => HDIBEA)하고
1번의 오른쪽 트리를 방문한다.

9. 1번의 오른쪽을 마찬가지 중위 탐색을 하면 결과는
=> HDIBEAFCG 가 된다.



중위탐색 예 2)

결과 => **A / B * C * D + E** (중위표기식)



트리의 예 – 수식 트리(expression tree)

순서	호출노드번호	루트의 값	동작
1	1	+	없음
2	2	*	없음
3	3	*	없음
4	4	/	없음
5	5	A	없음
6	6	NULL	없음
7	5	A	printf
8	7	NULL	없음
9	4	/	printf
10	8	B	없음
11	9	NULL	없음
12	8	B	printf
13	10	NULL	없음
14	3	*	printf

순서	호출노드번호	루트의 값	동작
15	11	C	없음
16	12	NULL	없음
17	11	C	printf
18	13	NULL	없음
19	2	*	printf
20	14	D	없음
21	15	NULL	없음
22	14	D	printf
23	16	NULL	없음
24	1	+	printf
25	17	E	없음
26	18	NULL	없음
27	17	E	printf
28	19	NULL	없음

중위탐색 알고리즘과 중위 탐색 과정 => A / B * C * D + E

(2) 전위 탐색(preorder traversal)

전위 탐색은

루트노드 -> 왼쪽트리 -> 오른쪽트리 순으로 방문을 하게 된다. 중위탐색과 마찬가지로 왼쪽 트리를 방문할 때 왼쪽 트리 자체가 트리이기 때문에 왼쪽 트리에서도 다시 전위 탐색을 한다.
그러므로
왼쪽 트리의 루트노드, 왼쪽 트리의 왼쪽 트리, 왼쪽 트리의 오른쪽 트리순이 된다.
노드가 1개 있는 트리는 그냥 왼쪽, 오른쪽 트리가 없기 때문에 바로 방문을 한다.
이 알고리즘은 다음과 같다. 마찬가지로 순환 알고리즘이다.

```
void preorder(tree_ptr ptr) {
    if(ptr) {
        printf("%d", ptr->data);
        preorder(ptr->left_child);
        preorder(ptr->right_child);
    }
}
```

전위탐색(preorder traversal) 알고리즘

(3) 후위 탐색(postorder traversal)

후위 탐색은

왼쪽 트리 -> 오른쪽 트리 -> 루트노드 순으로 방문을 하게 된다. 중위탐색과 마찬가지로 왼쪽 트리를 방문할 때 왼쪽 트리 자체가 트리이기 때문에 왼쪽 트리에서도 다시 후위 탐색을 한다.
그러므로
왼쪽 트리의 왼쪽 트리, 왼쪽 트리의 오른쪽 트리, 왼쪽 트리의 루트노드 순이 된다.
노드가 1개 있는 트리는 그냥 왼쪽, 오른쪽 트리가 없기 때문에 바로 방문을 한다.
이 알고리즘은 다음과 같다

```
void postorder(tree_ptr ptr) {
    if(ptr) {
        postorder(ptr->left_child);
        postorder(ptr->right_child);
        printf("%d", ptr->data);
    }
}
```

후위탐색(postorder traversal) 알고리즘

트리 탐색을 꼭 순환 알고리즘으로 작성해야 하는 것은 아니다. 아래 예는 중위 탐색 알고리즘을 반복적인 알고리즘으로 작성한 것이다. 방문하면서 과거 노드를 LIFO 순으로 기억을 해야 하기 때문에 스택을 사용하였다.
순환 알고리즘과 비교하여 보자

반복적 중위탐색(inorder traversal) 알고리즘

```
/* iterative inorder tree traversal */
void iter_inorder(tree_ptr node) {
    int top = -1;
    tree_ptr stack[MAX_STACK_SIZE];
    while(1) {
        while(node) {
            push(&top, node);
            node = node->left_child;
        }
        node = pop(&top);
        if(!node) break;
        printf("%d", node->data);
        node = node->right_child;
    }
}
```

(4) 레벨 탐색(level order traversal)

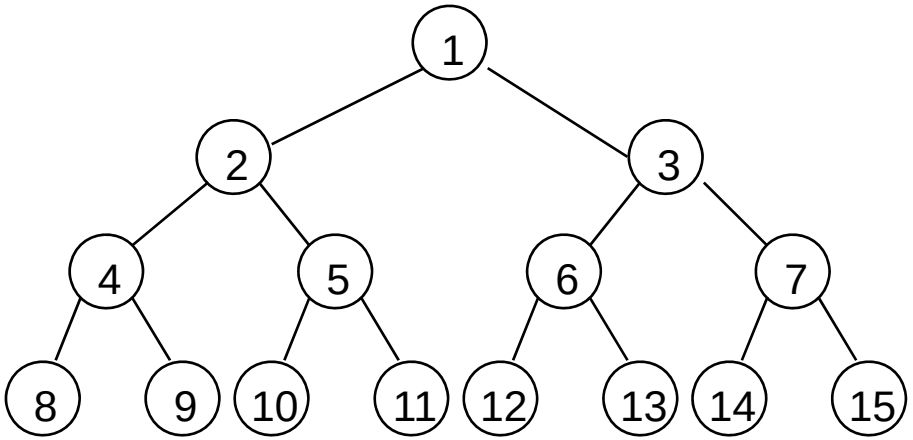
레벨 탐색은 레벨 1 노드들 -> 레벨 2 노드들 -> 레벨 3 노드들 -> ...

순서대로 탐색을 한다.

레벨 i를 탐색할 때 방문 노드의 자식 노드를 저장을 해둔 다음 레벨 i를 다 방문하면 다음 방문지는 저장해둔 노드를 꺼내어 방문한다. 이 때 저장해 둔 노드들은 먼저 저장된 노드를 먼저 꺼내어 방문을 하면 된다. 즉 큐(FIFO) 자료구조가 되는 것이다.

아래 트리의 경우 레벨 순으로 방문을 하면 다음과 같다.

=> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15



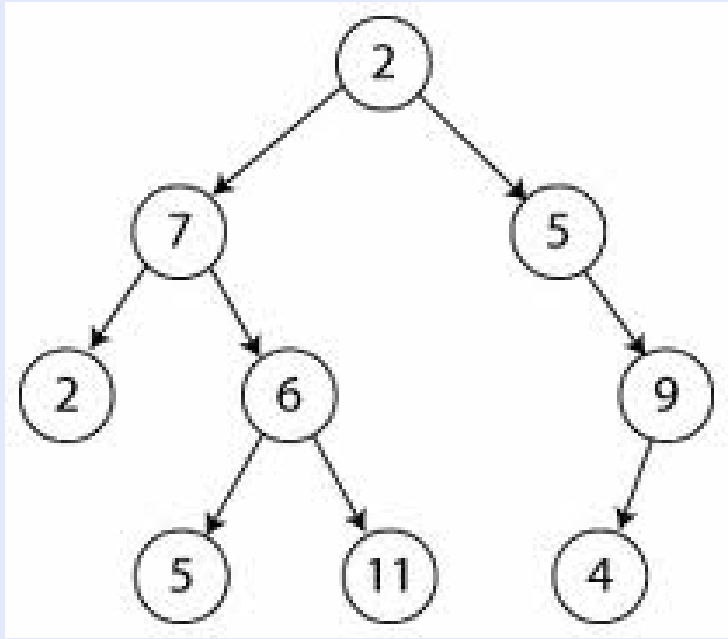
레벨탐색(level order traversal) 알고리즘

```
void level_order(tree_ptr ptr) {
    int front = rear = 0;
    tree_ptr queue[MAX_QUEUE_SIZE];
    if(!ptr) return;
    add(front,&rear,ptr);
    for(;;) {
        ptr = delete(&front, rear);
        if(ptr) {
            printf("%d", ptr->data);
            if(ptr->left_child)
                    add(front,&rear,ptr->left_child);
            if(ptr->right_child)
                    add(front, &rear,ptr->right_child);
        }
        else break;
    }
}
```

Q/A

다음 이진트리를 탐색한 결과는?

1. 중위탐색
2. 후위탐색
3. 전위탐색
4. 레벨탐색

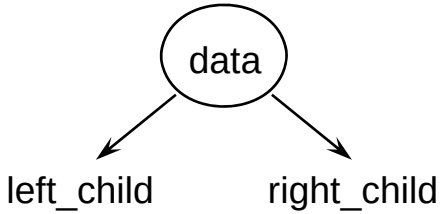


2. 쓰레드(Threaded) 이진트리

• 문제 제기 : 이진트리를 탐색할 때 일정한 순서에 따라 노드를 탐색하게 된다. 이때 노드의 순서에 따라 미리 다음 노드를 연결시켜 놓으면 훨씬 검색 속도가 빠르지 않을까?

• 방법 : 트리의 링크에서 자식이 없는 노드에 링크 필드 값을 NULL로 비워두지 말고 다음 방문할 곳이나 바로 앞 방문했던 노드를 가리키도록 한다.

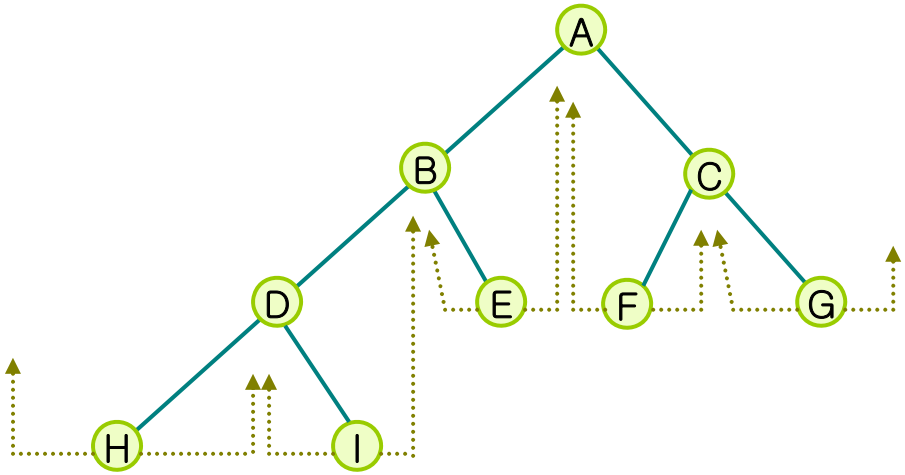
트리의 전체 노드 수는 N 개이고 링크의 수는 2N 개이며 이 중 N+1개의 링크는 비어있게 된다. 이 N+1개의 링크를 활용한다, 이 링크를 thread라고 부른다.



•쓰레드 만들기

비어있는 링크 값이 왼쪽인지 오른쪽인지에 따라 다음과 같이 구축한다.

1) 왼쪽 링크가 비어있으면(if ptr->left_child is null),
   => 노드의 링크 값을 중위 탐방 순서의 바로 전 노드 값을 저장한다.
2) 오른쪽 링크가 비어있으면(if ptr->right_child is null),
   => 노드의 링크 값을 중위 탐방 순서의 바로 다음 노드 값을 저장한다



쓰레드 트리의 예

노드의 링크에 저장된 값이 자식 노드에 대한 포인터인지, 쓰레드인지 구분을 어떻게 하는가?

(정답) 필드를 두어서 표시하여야 한다

필드 변수 이름 : *left_thread* and *right_thread*

변수 값 :

- *ptr->left_thread* = TRUE

  ptr->left_child 가 thread 값일 경우

- *ptr->left_thread* = FALSE

  ptr->left_child 가 왼쪽 자식일 경우

- *ptr->right_thread* = TRUE

  ptr->right_child 가 thread 값일 경우

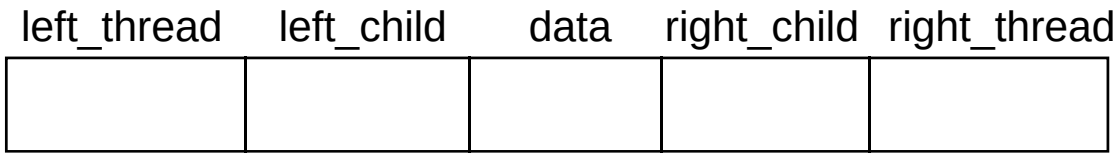
- *ptr->right_thread* = FALSE

  ptr->right_child 가 오른쪽 자식일 경우

•쓰레드 트리를 위한 노드의 선언과 구조

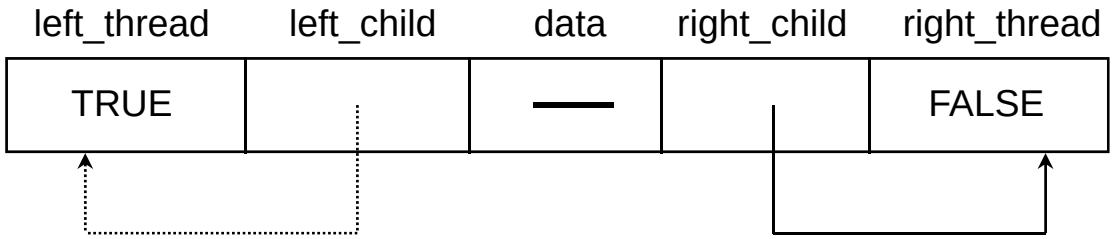
쓰레드 트리를 만들기 위해서는 일반 트리와 같으나 쓰레드인지 구분하는 필드를 두게된다.

```
struct tnode {
    short int left_thread;
    struct tnode *left_child;
    char data;
    struct tnode *right_child;
    short int right_thread;
};
typedef struct tnode threaded_tree;
typedef threaded_tree *threaded_ptr;
```

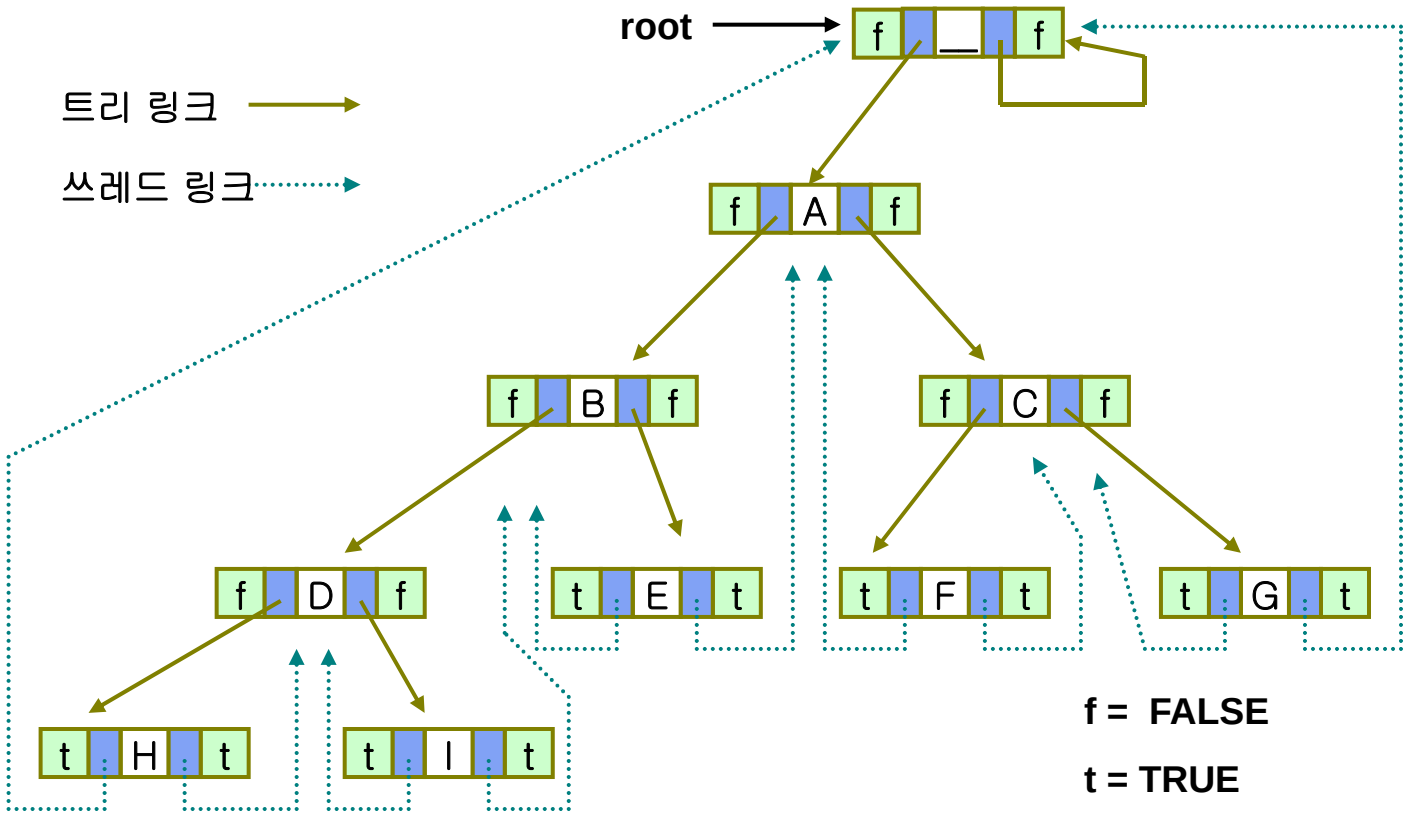


– 머리노드 : 쓰레드 트리에서는 왼쪽 끝과 오른쪽 끝의 두 노드는 가리킬 곳이 없으므로 전체 트리를 관리하는 머리노드(헤드노드)를 둔다. 트리가 비어있어도 머리노드는 남아있게 된다.

예) 머리노드의 예 : 일반 노드와 구조는 같다.



예) 헤드노드를 가진 쓰레드 이진 트리 :



•쓰레드 트리의 중위 탐색

이진 트리를 쓰레드 트리로 바꾼 후에는 중위탐색을 편하게 할 수 있다.
자식노드가 있으면 자식노드에 대한 링크 값을 바로 알 수 있고 자식노드가 없으면
쓰레드 링크에 다음 방문지 링크 값이 포함되어 있으므로 역시 바로 찾을 수 있다.

이 과정을 알고리즘으로 설명하면 다음과 같다. 먼저 임의의 노드의
다음 방문 링크를 찾는 insucc() 함수를 만들어 보자

```
알고리즘 insucc() :  (inorder traversal of a threaded binary tree)
    {  find the inorder successor of ptr
       - if ptr->right_thread=TRUE,
           then ptr->right_child
       - otherwise
           follow a path of left-child links from the right-child of ptr
           until we reach a node with left_thread=TRUE
    }
```

- **쓰레드 트리의 중위탐색에 대한 다음 노드 값 찾기 – insucc()**

노드의 포인터 값 tree를 인자로 주면 반환되는 값은 주위탐색의 다음 방문 노드이다.

```
threaded_ptr insucc(threaded_ptr tree) {
    threaded_ptr temp;
    temp = tree->right_child;
    if(!tree->right_thread)
        while(!temp->left_thread)
            temp = temp->left_child;
    return temp;
}
```

프로그램 tinorder() : 쓰레트 트리의 중위탐색 알고리즘

- **쓰레드 트리의 중위탐색 알고리즘**

쓰레드트리의 중위탐색 알고리즘은 루트 노드에서 시작하여서 다음 노드를 계속 찾아나가면 된다.

```
void tinorder(threaded_ptr tree) {
    threaded_ptr temp = tree;
    for(;;) {
        temp = insucc(temp);
        if(temp = tree) break;
        printf("%3c", temp->data);
    }
}
```

프로그램 tinorder() : 쓰레트 트리의 중위탐색 알고리즘

쓰레드 이진트리에 새로운 노드를 첨가한다면 어떻게 될까?

•알고리즘 insert_right() : 쓰레드 트리에서 노드의 오른쪽자식노드 삽입

{

1) parent->right_threaded 를 FALSE 로 변경

2) child->left_thread 과 child->right_thread 를 TRUE 로 변경

3) child->left_child 를 parent 로 변경

4) child->right_child 를 parent->right_child 로 변경

5) parent->right_child 를 child 로 변경

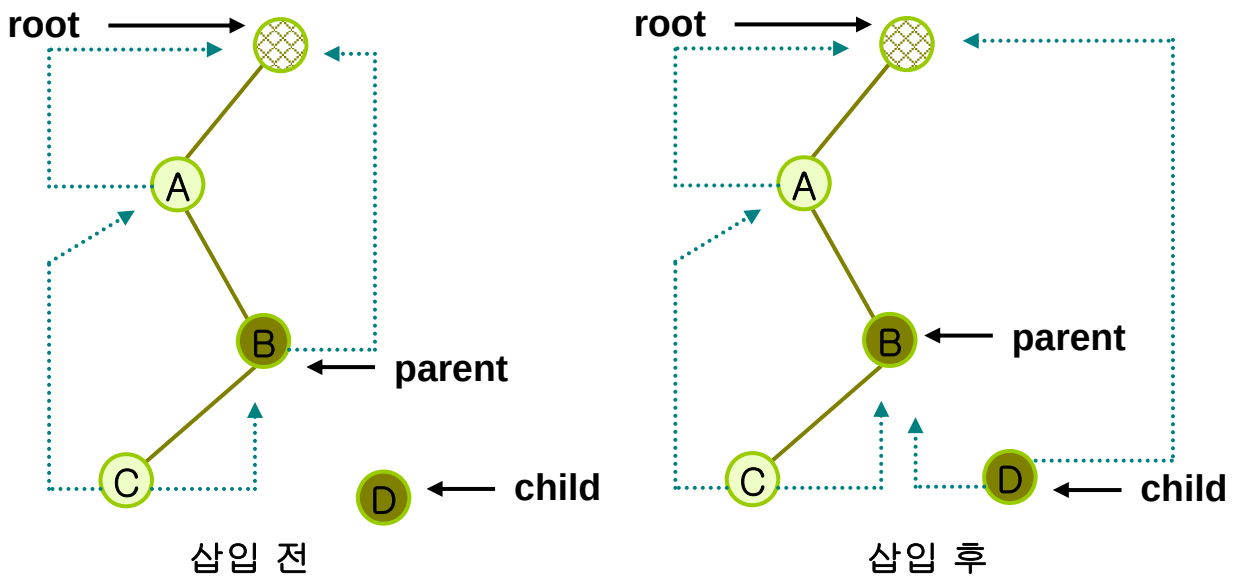
}

•프로그램 insert_right() : 쓰레드 트리에서 임의의 노드의 오른쪽에 새로운 노드를 삽입하는 프로그램

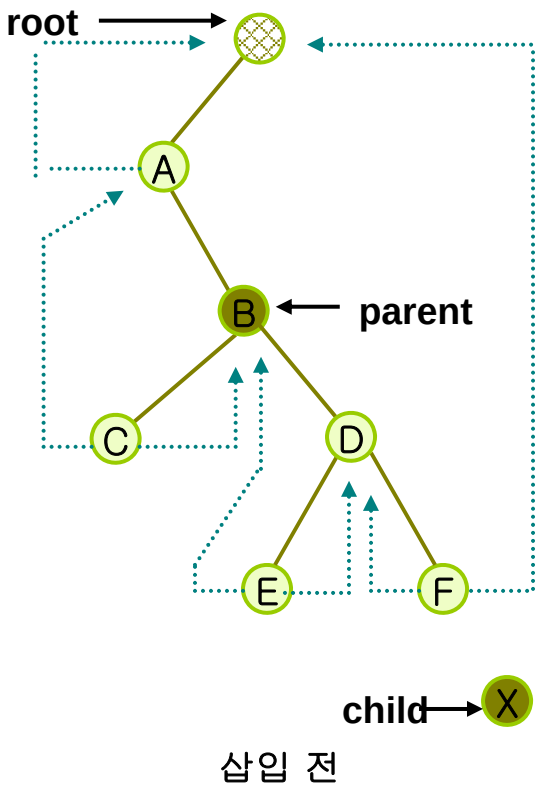
```
void insert_right(threaded_ptr parent, threaded_ptr
child)
{
    threaded_ptr temp;
    child->right_child=parent->right_child; (4)
    child->right_thread=parent->right_thread; (2)
    child->left_child=parent; (3)
    child->left_thread=TRUE; (2)
    parent->right_child=child; (5)
    parent->right_thread=FALSE; (1)
    if(!child->right_thread) { /* 오른쪽 자식이 있는 경우
    */
        temp=insucc(child);
        temp->left_child=child;
    }
}
```

프로그램 tinorder() : 쓰레트 트리의 중위탐색 알고리즘

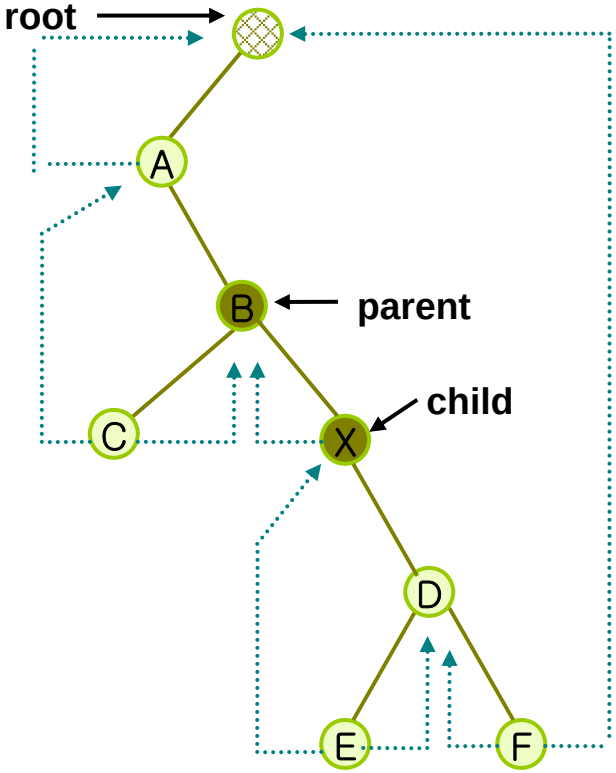
예 1) 쓰레드 트리에서 임의의 노드의 오른쪽에 새로운 노드를 삽입하는 그림 – 오른쪽 자식이 없는 경우



예 2) 쓰레드 트리에서 임의의 노드의 오른쪽에 새로운 노드를 삽입하는 그림
- 오른쪽 자식이 있는 경우



삽입 전



삽입 후

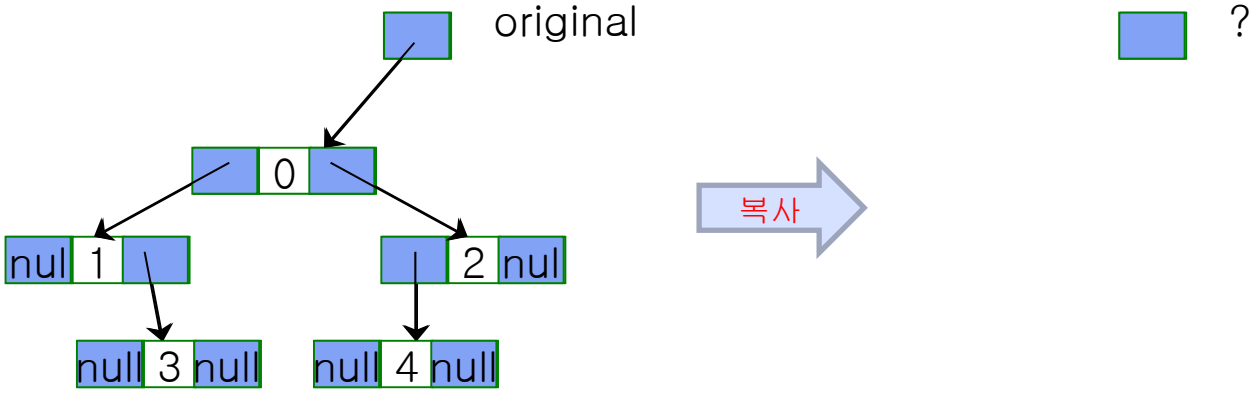
3. 이진트리에 관한 알고리즘

1. 이진트리 복사 : 이진트리가 있을 때 똑같은 모양의 이진트리를 만든다면?

똑같은 이진트리를 한 개 더 만든다면 트리를 탐색하면서 새로운 노드를 만날 때마다 노드를 복사하여야 한다. 후위탐색 방법을 이용하여 복사하여 보도록 하자.

이진트리의 복사 알고리즘

```
/* program copy() */
tree_ptr copy(tree_ptr original)
{   tree_ptr temp;
      if(original) {
            temp = (tree_ptr)malloc(sizeof(node));
            temp->left_child = copy(original->left_child); ①
            temp->right_child = copy(original->right_child); ②
            temp->data = original->data; ③
            return temp;
      }
      return NULL;
}
```

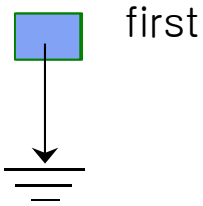

original
?
0
nul
1
2
nul
null
3
null
null
4
null
복사

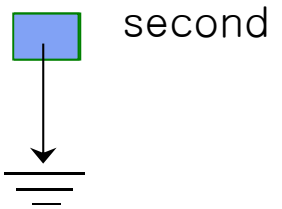
2. 이진트리 동등비교 : 두개의 이진트리가 있을 때 두 이진트리가 모양과 노드에 값이 똑같은지 비교하려면?

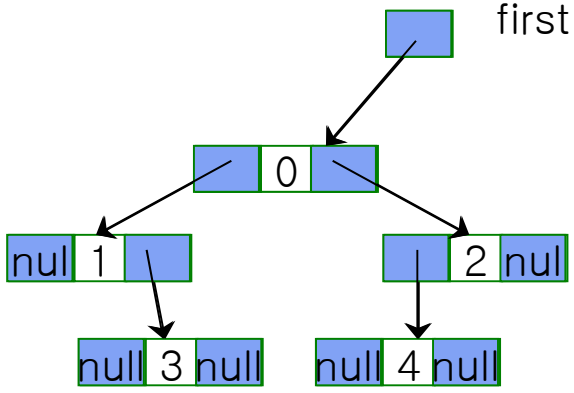
두개의 이진트리를 탐색하면서 노드를 만날 때마다 두 트리의 노드의 값을 비교하면 된다. 탐색 방법은 여러가지 탐색 방법 중 택하면 되지만 전위탐색 방법을 이용하여 비교하도록 하자.

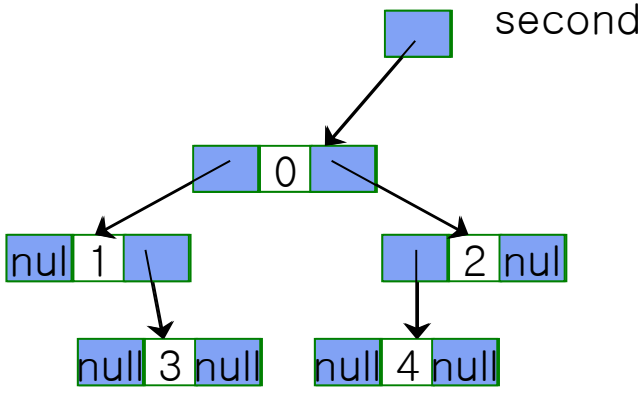
이진트리의 동등 검사 알고리즘

```
/* program equal() */
int equal(tree_ptr first,tree_ptr second) {
        return ((!first && !second)
                        || (first && second
                        && (first->data == second->data)
                        && equal(first->left_child, second->left_child)
                        && equal(first->right_child, second->right_child)));
}
```


first


second


first
0
nul
1
2
nul
null
3
null
null
4
null


second
0
nul
1
2
nul
null
3
null
null
4
null

Review

⦿ 트리는 중요한 자료구조 중의 하나이다. 또 트리에 관한 대부분 프로그램은 트리의 탐색에서 시작한다.

이 장에서는 트리의 탐색 방법으로 중위탐색, 전위탐색, 후위탐색과 레벨 탐색에 대한 알고리즘을 살펴보았다. 탐색은 대부분 순환 알고리즘을 이용하여 작성된다.

⦿ 이진트리를 효율적으로 탐색하는 방법은 트리구조를 쓰레드(thread) 구조로 바꾸는 방법이 있다. 중위탐색을 효율적으로하기 위한 방법으로 노드의 링크 값 중 사용하지 않는 값을 중위탐색의 다음이나, 전 방문 노드를 가리키도록 하는 방법이다. 자료구조를 변경하여 프로그램의 효율성을 높이는 대표적인 예이다.

⦿ 이진트리에 관한 알고리즘이 많이 있지만 이진트리의 복사 알고리즘, 동등성 검사 알고리즘을 통하여 좀 복잡한 알고리즘에 대하여 살펴볼 수 있다.