제 6 장

6.1 차수가 15인 B-트리에서 각 노드에는 최고 얼마나 많은 키값과 포인터를 포함할 수 있는가?

답) 키값 : 최대 14개
포인터 : 최대 15개

6.2 차수가 15인 B-트리에서 한 노드는 최소 몇 개의 키값과 몇 개의 포인터를 유지해야 하는가?

답) 키값 : 최소 15/2 -1 = 7(개)
포인터 : 최소 15/2 = 8(개)

6.3 차수가 3인 B-트리에서 키값이 242개일 때 하나의 키를 검색하기 위하여 최대 몇 개의 노드를 순회해야 하는가? 키값이 1023개일 때는 어떠한가?

답) 루트의 레벨을 1이고 각 노드에서 최소 2개의 키값을 가져야 한다고 할때 각 레벨에서 최소의 노드수는 다음과 같다.

레벨 2에서의 최소 노드수 : 2
레벨 3에서의 최소 노드수 : 2 * 2
.....
레벨 (h+1)에서의 최소 노드수 : 2*2(h-1)

트리의 정의에서 모든 널 포인터는 리프 노드에 존재하며 그 수는 트리의 키의 수보다 1이 크다. 그러므로 존재하지 않는 h+1 레벨에서는 다음과 같은 식이 성립한다.

$$N+1 \geq 2 \times 2(h-1)$$
$$h \leq 1 + \log_2[(N+1)/2]$$

위의 식에 의해 키값이 242개일 때는 위의 식에 대입하면 최대 7개를 순회해야 한다. 1023인 경우는 대입하면 최대 10개를 순회해야 한다.

6.4 차수가 9인 B-트리가 n개의 키를 가지고 있을 때 하나의 키를 검색하기 위하여 최대 몇 개의 노드를 순회해야 하는가?

답) 루트의 레벨을 1이고 각 노드에서 최소 5(9/2)개의 키값을 가져야 한다고 할때 각 레벨에서 최소의 노드수는 다음과 같다.

레벨 2에서의 최소 노드수 : 2

레벨 3에서의 최소 노드수 : 2 * 5

.....

레벨 (h+1)에서의 최소 노드수 : 2* 5(h-1)

트리의 정의에서 모든 널 포인터는 리프 노드에 존재하며 그 수는 트리의 키의 수보다 1이 크다. 그러므로 존재하지 않는 h+1 레벨에서는 다음과 같은 식이 성립한다.

$$n+1 \geq 2 \times 5(h-1)$$
$$h \leq 1 + \log_5[(n+1)/2]$$

위의 식에 의해 키값이 n개일 때는 최대 $1+\log_5[(n+1)/2]$ 개를 순회해야 한다.

6.5 차수가 m인 B-트리를 중위(inorder) 순회하는 알고리즘을 작성하라.

답)

```
procedure TRAVERSE(tree,node,nodestack)
(* node = root와 nodestack = null로 초기화한다. *)
i = 1;
while i ≤ NUMBER-OF-KEYS(node) DO
        BEGIN
        push(nodestack,node);
        READ-DIRECT(tree,node,TREE-POINTER(node,i));
        TRAVERSE(tree,node,nodestack);
        (* recursive call *)
        node = pop(nodestack);
        PROCESS(DATA-POINTER(node,i));
        i = i + 1;
        END;
READ-DERECT(tree,node,TREE-POINTER(node,i));
TRAVERSE(tree,node,nodestack);
```

6.6 키를 순차적으로 입력하여 차수 3인 B-트리를 구성하는 프로그램을 작성하라.

답)

```
procedure BUILD-BTREE(tree,entry)
find appropriate leaf for key(entry);
INSERT(entry,leaf);
CHECK-OVERFLOW(tree,leaf);

        procedure CHECK-OVERFLOW(tree,leaf)
        if NUMBER-OF-KEYS(node) > 2d then
                if NUMBER-OF-KEYS(left-brother) < 2d or
                   NUMBER-OF-KEYS(right-brother) < 2d
                then redistribution
                else
                        begin
                        SPLIT(node);
                        if node = root then
                                begin
                                GET(new-root);
                                INSERT(middle-entry,new-root);
                                end;
                        else begin
                                INSERT(middle-entry,father-node);
                                CHECK-OVERFLOW(tree,father-node);
        end;
end;
```

6.7 차수 3인 B-트리에 주어진 키값을 검색하는 알고리즘을 작성하라.

답)

```
procedure SEARCH(tree,node,keyvalue)
(* node = root 로 초기화한다. *)
i = 1;
while KEY(node,i) < keyvalue and i < NUMBER-OF-KEYS(node) DO
        i = 1 + 1;
```

```
if KEY(node,i) = keyvalue
        then return DATA-POINTER(node,i);
else begin
        if KEY(node,i) > keyvalue
                then p = TREE-POINTER(node,i);
        else p = TREE-POINTER(node,i+1);
        if p == null then return null;
        else begin
                READ-DIRECT(tree,nextnode,p);
                return SEARCH(tree,nextnode,keyvalue);
        end;
end;
```

6.8 차수가 3인 B-트리에서 키값을 순차로 검색하는 알고리즘을 작성하라.

답)

```
procedure TRAVERSE(tree,node,nodestack)
(* node = root와 nodestack = null로 초기화한다. *)
i = 1;
while i ≤ NUMBER-OF-KEYS(node) DO
        BEGIN
        push(nodestack,node);
        READ-DIRECT(tree,node,TREE-POINTER(node,i));
        TRAVERSE(tree,node,nodestack);
        (* recursive call *)
        node = pop(nodestack);
        PROCESS(DATA-POINTER(node,i));
        i = i + 1;
        END;
READ-DERECT(tree,node,TREE-POINTER(node,i));
TRAVERSE(tree,node,nodestack);
```

6.9 B-트리와 AVL 트리, B^*-트리, B^+-트리의 차이점을 설명하라.

답) AVL 트리는 B-트리와는 달리 트리의 모든 노드에 대해 단지 오른편 서브트리와 왼편 서브트리의 높이차가 1이거나 같게 유지해서 전체 트리를 재균형시키지 않고도 트리가 균형을 유지하도록 하는 것이다. 반면, B-트리는 모든 리프의 레벨이 같도록 항상 균형을 유지하고 있으며 트리의 각 노드가 적어도 반 이상이 차 있어서 높이가 높아지므로 탐색 속도가 늦어지는 것을 방지한다.

B^*-트리는 B-트리에서 삽입과 삭제에 필요한 부수적 연산 작업을 줄이면서 특정 키값의 직접 탐색 성능을 더욱 향상시킬 수 있다. 즉, 각 노드가 최소한 2/3가 채워지도록 B-트리를 변경한 것으로 노드 분열의 빈도를 줄이려는 것이다.

B^+-트리는 두 부분으로 구성되는데 하나는 리프가 아닌 노드로 된 인덱스 세트이고 다른 하나는 리프 노드로만 구성된 순차 세트이다. B-트리와 다른점은 인덱스 부분에 있는 키값은 리프 노드에 있는 키값을 직접 신속하게 찾아갈 수 있도록 경로만 제공하며 사실상 모두 리프 노드에 다시 존재한다. 또한 순차 세트의 모든 노드가 순차적으로 서로 연결되어 있어서 순차 접근시 효율적인 처리가 가능하다.

6.13 균형 트리와 높이 균형 트리(AVL 트리)의 차이점을 설명하라.

답) 균형 트리는 모든 노드에 대해 왼쪽 서브트리와 오른쪽 서브트리의 노드수가 가능한 같게 만들어서 트리의 최대 경로 길이를 최소화하는 것이고, 높이 균형 트리는 그 서브 트리의 높이로 균형을 유지하는 트리를 말한다.

6.14 2원 탐색 트리에서 키값을 검색하는 프로시저와 새로운 키값을 삽입하는 프로시저를 작성하라. 키값이 논리적으로 삭제되어 표시만 되어 있는 키값이 있다고 가정하라.

답) 검색 프로시저

```
struct node {
        int key,info;
        struct node *l,*r;
        };
struct node *t, *head, *z;
int binarysearch(int v)
{
        struct node *x = head->r;
        z->key = v;
        while(v != x->key)
```

```
                x = (v < x-> key) ? x->l : x->r;
        return x->info;
}
```

삽입 프로시저

```
binaryinsert(int v, int info)
{
        struct node *p, *x;
        p = head;
        x = head->r;
        while(x != z)
        {
                p = x;
                x = (v < x->key) ? x->l : x->r;
        }
        x = (struct node *) malloc(sizeof *x);
        x->key = v;
        x->info = info;
        x->l = z;
        x->r = z;
        if(v < p->key)
                p->l = x;
        else p->r = x;
}
```

6.15 2원 탐색 트리에서 키값을 물리적으로 삭제하는 프로시저를 작성해 보라.

답) 삭제 프로시저

```
binarydelete(int v)
{
        struct node *c, *p, *x;
        z->key = v;
        p = head;
        x = head->r;
        while(v != x->key)
```

```
        {
                p = x;
                x = (v < x->key) ? x->l : x->r;
        }
        t = x;
        if ( t->r == z) x = x->l;
        else if (t->r->l == z)
        {
                x = x->r;
                x->l = t->l;
        }
        else
        {
                c = x->r;
                while (c->l->l != z) c = c->l;
                x = c->l; c->l = x->r;
                x->l = t->l; x->r = t->r;
        }
        free(t);
        if (v < p->key) p->l = x;
        else p->r = x;
}
```

제는 노드의 첨가나 삭제가 수반될 수 있어도 노드의 병합이나 분할은 일어나지 않는다.

6.20 AVL 트리가 균형 2원 탐색 트리라고 한다면 균형 m-원 탐색 트리는 어떻게 정의할 수 있겠는가?

답) 각 노드에서 서브트리에 대한 임의의 포인터 하나의 높이가 다른 모든 포인터의 높이와 같거나 차이가 1이 되도록 해서 전체 트리를 재균형시키지 않고도 트리가 균형을 유지하도록 탐색 트리의 형태를 제어하는 것을 균형 m-원 탐색 트리라고 한다.

6.21 높이가 4이고 차수가 5인 B-트리에서 노드의 수는 최대로 몇 개가 있을 수 있는가? 또 키값의 수는 최대 몇 개가 있을 수 있는가?

답) 최대의 노드의 수를 가지려면

레벨 1 : 1 개 노드,

레벨 2 : 5 개 노드,

레벨 3 : 52 개 노드,

레벨 4 : 53 개 노드

이므로 전체 노드수는 1 + 51 + 52 + 53 = 156 (개)이다.

또, 최대 키값의 수는 노드마다 4개의 키값을 가지고 있을 때이므로 156 * 4 = 624 (개)이다.