6주차 1차시 기본적인 탐색법

【학습목표】

1. 탐색의 기본개념 및 용어를 이해할 수 있다.
2. 이진 탐색을 이해할 수 있다.

지난 강의 정리

*** 정렬 알고리즘 비교표**

종류	실행시간	안정적	제자리
선택	$O(n^2)$	X/O	O
버블	$O(n^2)$	O	O
삽입	$O(n^2)$	O	O
쉘	$O(n^{1.5})$	X	O
퀵	O(nlogn), $O(n^2)$	X	O
합병	O(nlogn)	O	X
히프	O(nlogn)	X	O
계수	O(n)	O	X
기수	O(n)	O	X
버킷	O(n)	O	X

*** 탐색**

여러 레코드에서 주어진 키를 갖는 레코드를 찾아내는 것을 의미한다.

- **순차 탐색**
- **이진 탐색**
- **이진 탐색 나무**
- **균형나무(AVL 트리, 2-3-4나무, 흑적나무)**

초기화, 삽입, 삭제 정렬 조인 등의 연산도 함께 고려해야 한다.

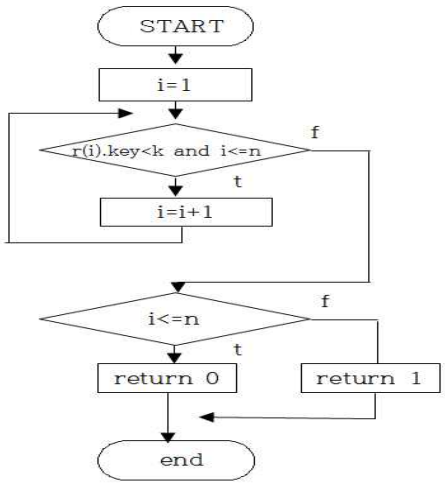
학습내용1 : 순차 탐색

*** 순차 탐색(sequential search)**
서로 이웃한 데이터를 하나씩 차례차례로 탐색키와 비교하면서 찾는 방법이다.

비순서 파일의 탐색에 적합하다.
⊙ 키 값이 순서에 관계없이 무순서로 연속해서 저장된 경우.

n개 데이터의 순차탐색
⊙ 탐색 실패 시 : n번 비교
⊙ 탐색 성공 시 : 최소 1번, 평균 (n+1)/2번 ☞ O(n)

1. 비순서 파일의 선형 탐색



위의 알고리즘은 while 문에 의해서 첫 번째 레코드부터 차례로 비교하여 i 값이 n+1이 되면 지정된 키를 가진 레코드는 파일 상에 존재하지 않는다.
위의 알고리즘 복잡도는 while 문을 반복하는 횟수에 비례하는데 최선의 경우 첫 번째 비교로 탐색이 가능하며 탐색이 실패 하는 경우에는 끝까지 비교해야 하므로 n번 반복한다. 또한 탐색이 성공되었을 때의 평균 탐색길이는 $L=\sum_{i=1}^{n} i \cdot Pi$ (n : 레코드수, Pi : 각 레코드가 탐색될 확률 $Pi=\frac{1}{n}$) $=\frac{1}{n} \cdot \sum_{i=1}^{n} i \frac{1}{n} \cdot \frac{n(n+1)}{2}=\frac{n+1}{2}$ 이 된다. 그러므로 알고리즘의 복잡도는 O(n)이다.

학습내용2 : 이진 탐색

1. 탐색 방법

정렬된 레코드의 선형 리스트에 분할 정복을 적용한 방법이다.

탐색 방법
⦿ 탐색 범위의 중간 위치의 키와 비교
◎ 탐색키 = 중간키 ☞ (성공)
◎ 탐색키 < 중간키 ☞ (앞부분 탐색)
◎ 탐색키 > 중간키 ☞ (뒤부분 탐색)
탐색을 수행할 때 마다 탐색 원소의 개수가 1/2씩 감소

먼저 배열 표현의 이진 탐색 과정을 살펴보자. 여기서는 360을 탐색하는 예를 보여주고 있다.
<초기단계>

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	10	25	40	70	90	100	120	200	300	360	400	500	550	600
low						mid=(low+ high)/2								high

<1단계>

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	10	25	40	70	90	100	120	200	300	360	400	500	550	600
								low			mid			high

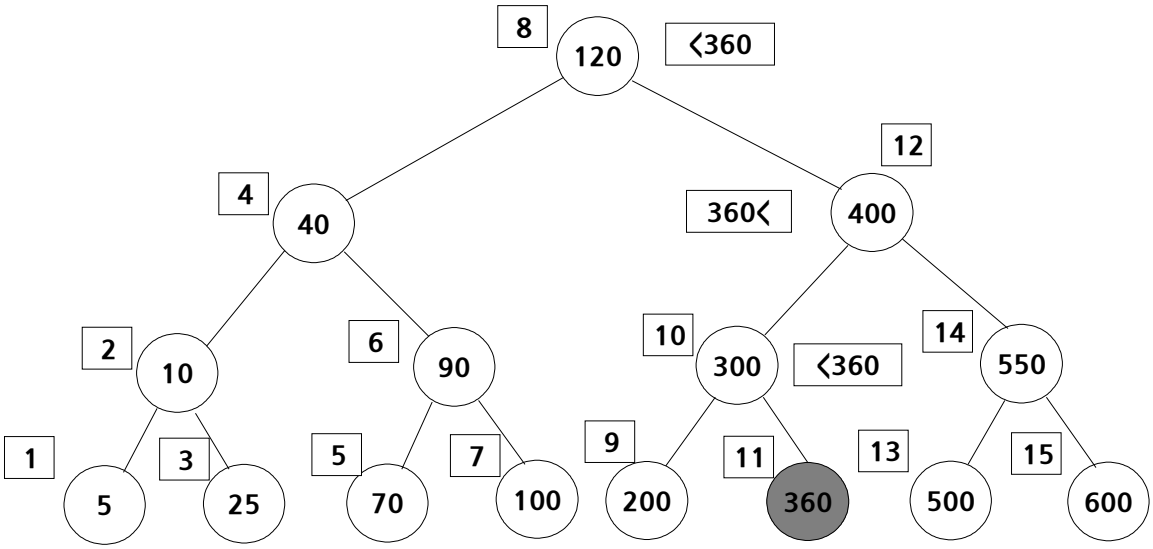
<2단계>

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	10	25	40	70	90	100	120	200	300	360	400	500	550	600
								low	mid	high				

<3단계>

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	10	25	40	70	90	100	120	200	300	360	400	500	550	600
								low=mid=high						

지금까지의 과정을 트리를 이용하여 표현하면 다음의 그림과 같이 된다.



2. 이진 탐색법의 분석

⦿ 삽입이나 삭제 시 정렬 상태 유지를 위해서 자료의 이동이 필요하다.
  ◎ 최악의 경우 n번
  ◎ 평균적 경우 n/2번
  ◎ 삽입이나 삭제가 빈번한 경우 좋지 않다.
⦿ 시간 복잡도
  ◎ 이진 탐색법의 시간 복잡도 O(logn) 이다.

3. 이진 탐색 나무(Binary Search Tree)

이진 탐색 나무란 나무에 존재하는 각 노드들이 K_1, K_2, ..., K_n의 키 값을 갖는 n 개의 레코드들 중 하나를 갖는 나무를 말한다. 여기서 키 들은 노드의 식별자 이므로 이진 나무에서 어떤 노드를 찾는다는 것은 그 노드의 키 값을 찾는다는 것과 같은 의미가 된다.

* 사전을 구현하는 데 좋은 구조

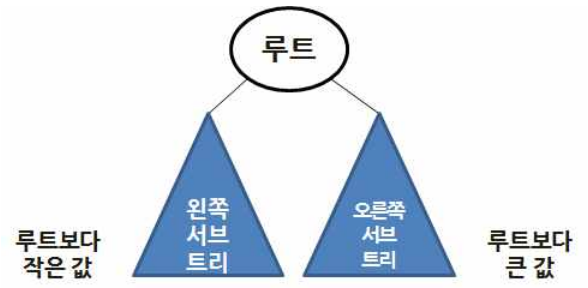
⦿ 사전(dictionary)

◎ 탐색, 삽입, 삭제가 행해질 수 있는 동적인 데이터의 집합.

⦿ 이진 나무 구조

◎ 각 노드 X의 왼쪽 부분 나무의 모든 키는 X 키보다 작다.

◎ 각 노드 X의 오른쪽 부분 나무의 모든 키는 X 키보다 크다.



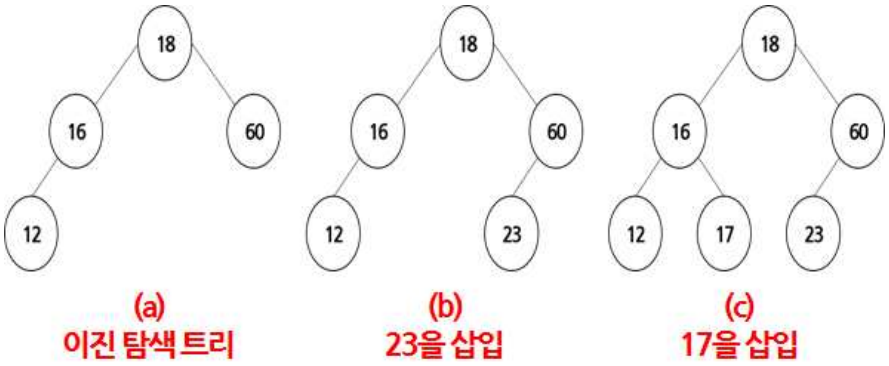
◎ 연산 : 초기화(생성), 탐색, 삭제.

4. 이진 탐색 트리의 삽입

특정 노드를 이진 탐색 트리에 삽입하려면 같은 키 값을 가지는 노드가 없어야 한다. 그러므로 다음과 같은 방법으로 동일한 키를 가지고 노드의 존재여부와 삽입할 위치를 알아낸다.

① 현재 서브 트리의 루트와 키 값을 비교한다.
② 루트의 키 값과 검색 키가 같으면 동일 키를 가진 노드가 존재하는 것이므로 삽입이 불가능하다.
③ 루트의 키보다 검색 키가 작으면 왼쪽 서브 트리를 검사한다.
④ 루트의 키보다 검색 키가 크면 오른쪽 서브 트리를 검사한다.
⑤ 더 이상 검사할 노드가 없을 때, 즉 말단 노드에 도달한 경우 그 위치가 삽입할 위치이다.
⑥ 삽입할 위치를 찾았으면 새로운 노드를 삽입한다.
⑦ 현재 삽입위치에 있는 노드의 키 값과 새로운 키를 비교한다.
⑧ 삽입위치 노드의 키 값이 새로운 키 값보다 크면 새로운 노드는 left child가 된다.
⑨ 삽입위치 노드의 키 값이 새로운 키 값보다 작으면 새로운 노드는 right child 가 된다.

예) 이진 탐색 트리에(a)에 키 값이 23인 노드와 17인 노드를 삽입하는 과정

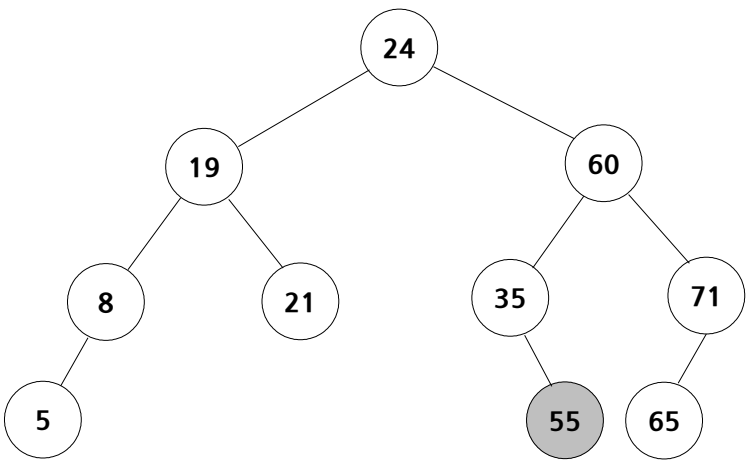


5. 이진 탐색 트리의 삭제

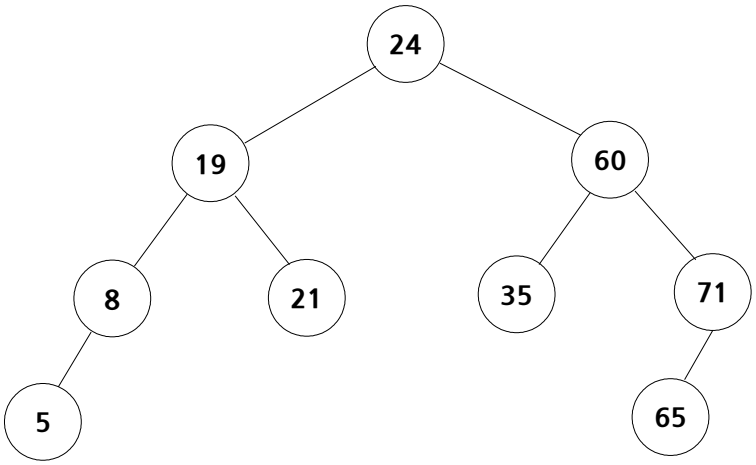
이진 탐색 트리에서 특정 킷값을 가지는 노드를 삭제하려면 먼저 검색 연산을 하여 삭제할 노드를 찾아낸다. 특정 노드를 삭제하는 것은 다음의 3가지 경우중 하나에 해당된다.

① 삭제하려는 노드가 말단 노드인 경우
② 삭제하려는 노드가 왼쪽이나 오른쪽 서브 트리중 하나만 가지고 있는 경우
③ 삭제하려는 노드가 두 개의 서브 트리를 모두 가지고 있는 경우

만일 삭제하려는 노드가 말단 노드인 경우에는 삭제하려는 노드가 말단 노드인 경우는 해당 노드만 삭제해준다.

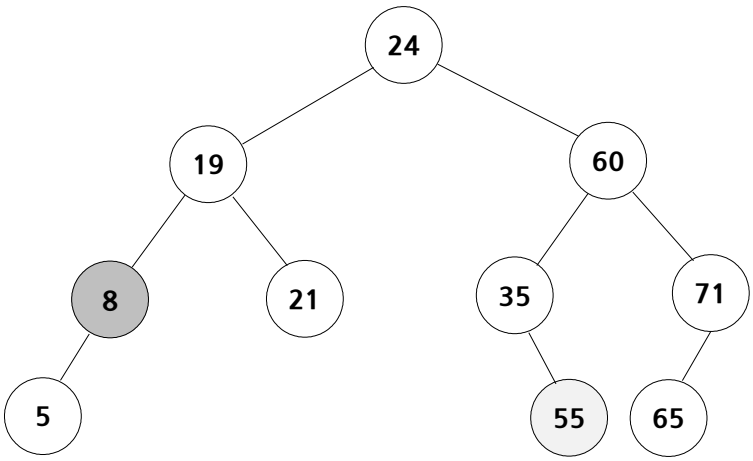


(a) 삭제 전

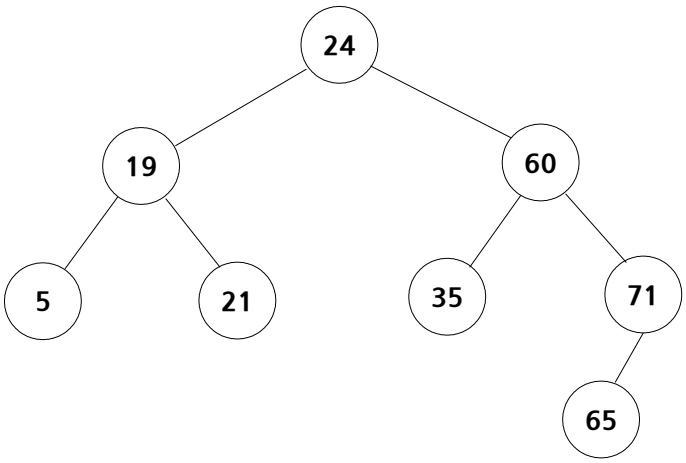


(b) 55 삭제 후

삭제하려는 노드가 왼쪽이나 오른쪽 서브 트리중 하나만 갖고 있을 경우에는 자기 노드는 삭제하고 서브 트리는 자기 노드의 부모 노드에 붙여준다.

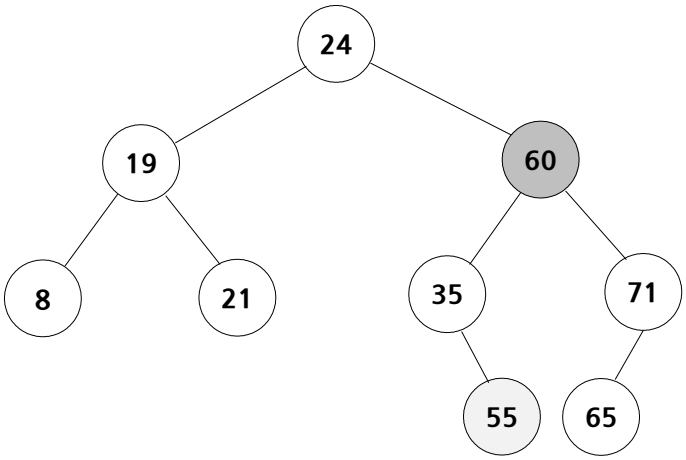


(a) 삭제 전

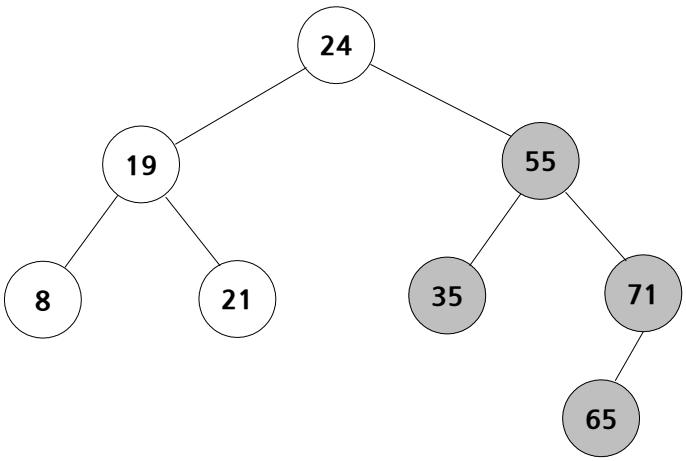


(b) 8 삭제 후

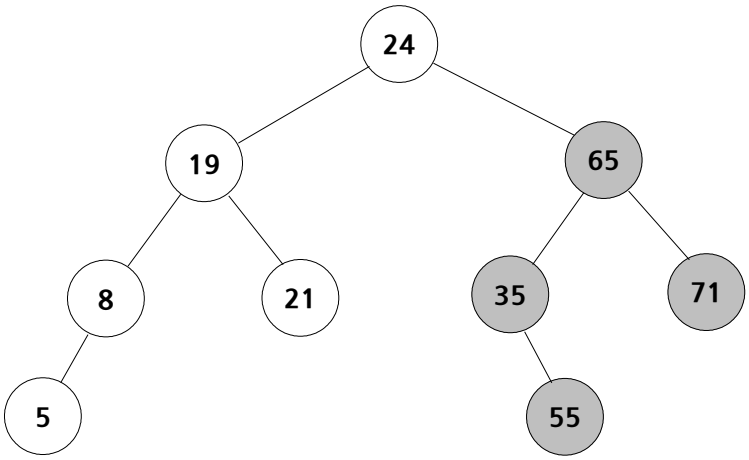
마지막으로 삭제하려는 노드가 두 개의 서브 트리를 모두 가지고 있는 경우에는 자기 노드를 삭제한 후 두 개의 서브 트리를 하나로 합하여 부모 노드에 붙여준다. 합한 서브 트리의 루트 노드는 왼쪽 서브 트리에서 가장 큰 키 값을 가지는 노드나, 오른쪽 서브 트리에서 가장 작은 키 값을 가지는 노드가 된다.



(a) 삭제 전



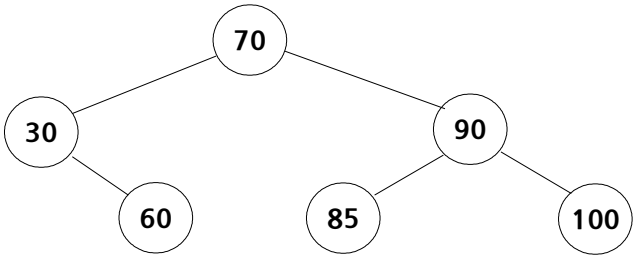
(b) 60 삭제 후



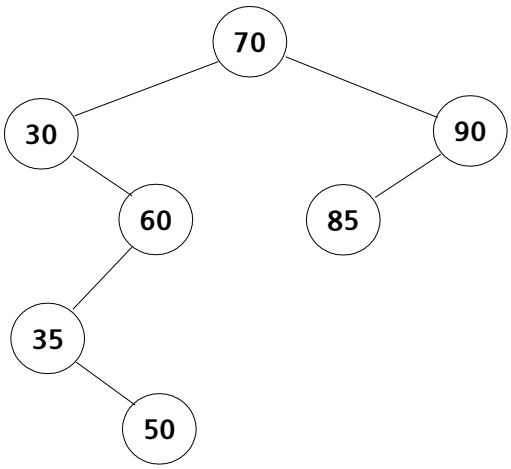
(c) 트리 재구성

6. 이진 탐색 트리의 특징

이진 탐색 트리의 탐색 속도는 O(log n+1)이며, 삽입/삭제 시 자료 이동 횟수는 O(log n)이다. 따라서 이진 탐색 트리 알고리즘에 적합한 것은 주기억 장소에서의 탐색이다. 이진 탐색 트리에서 삽입/삭제 조작 시 문제점이 있는데, 불균형 된 이진 탐색 트리로 변화 시 탐색 속도가 늘어나는 것이 단점이다.

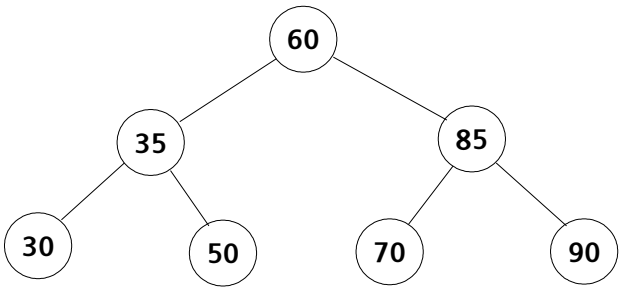


(a) 초기 상태



(b) 35, 50 삽입 후, 100 삭제

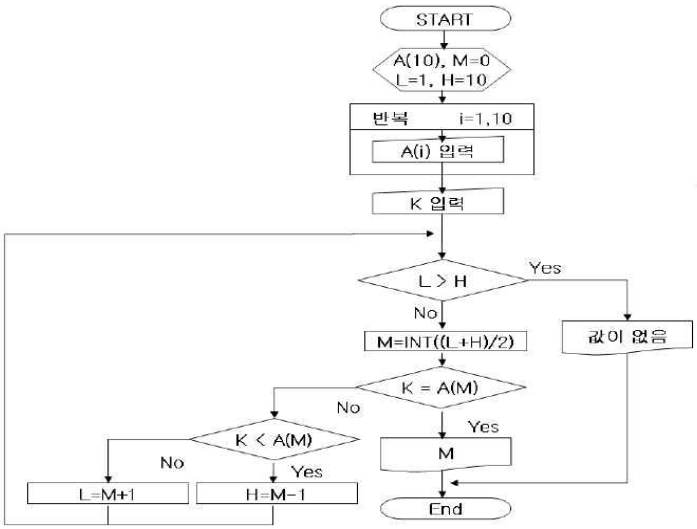
최악의 경우 경사진 이진 트리(skewed binary tree)를 생성하게 되는데 이 경우에는 탐색 속도가 O(n)으로 최악의 효율을 나타낼 수 있다.
위의 트리를 균형이진 탐색 트리로 나타내면 다음과 같다.



* 이진 탐색 알고리즘

1) 문제 : 입력 받은 값이 배열의 몇 번째에 기억되어 있는지 알아보기 위해 출력하는 순서도를 작성하시오.
(단, 배열에 저장된 값은 오름차순 정렬된 10개 데이터이며, 이분검색을 이용하라.)

2) 조건처리 :
-A(10) : 숫자가 저장된 배열
-K : 검색할 값
-L : 배열 첨자이며, 시작 위치
-H : 배열 첨자이며, 마지막 위치
-M : 배열 첨자이며, 중간 위치
- i : 인덱스 변수



* 이진 탐색법(Binary Search)의 예
n=17일 때 주어진 레코드에서 키 값 46을 이용하여 이진 탐색을 수행한 과정

인덱스	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
레 코 드 키	2	3	6	7	8	12	14	16	20	25	28	40	41	46	60	66	69

① 1회전 : lo=1, hi=17, m=(1+17)/2 =9
k=46, R[m].key=20 이므로 k>R[m].key가 되어 lo=m+1 =9+1=10가 된다.

② 2회전 lo=10, hi=17, m=(10+17)/2 =13
k=46, R[m].key=41 이므로 k>R[m].key가 되어 lo=m+1 =13+1=14가 된다.

③ 3회전 lo=14, hi=17, m=(14+17)/2 =15
k=46, R[m].key=60 이므로 k<R[m].key가 되어 hi=m-1 =15-1=14가 된다.

④ 4회전 lo=14, hi=14, m=(14+14)/2 =14
k=46, R[m].key=46 이므로 k=R[m].key가 되어 탐색은 성공이 된다.

* 이진 탐색에서 레코드의 수와 키의 비교 횟수

키의 비교 횟수	탐색될 레코드 수
1	2^0
2	2^1
3	2^2
.	.
.	.
K	2^{k-1}

【학습정리】

1. 순차 탐색(sequential search)
- 가장 간단한 탐색 방법이 바로 순차 탐색 방법이다. 이 방법은 배열이나 연결 리스트에서 사용할 수 있는 방법으로, 첫 번째 레코드부터 차례차례로 비교하여 가면서 해당 키 값을 가지고 있는 레코드를 찾아내는 방법으로 다음과 같은 특성을 같는다.
① 비정렬 자료의 탐색에 적절하다.
② n개의 비정렬 자료의 순차탐색은 탐색에 실패할 경우 n+1번의 비교가, 탐색에 성공할 경우에는 평균적으로 (n+1)/2번의 비교가 요구된다.
③ 순차탐색법의 시간복잡도는 O(n)이다.

2. 이진 탐색법
- 반드시 정렬되어 있는 순서 파일에서만 가능한 검색 방법으로 분할 정복(divide and conquer) 개념에 기본을 둔 방법이다. 레코드 수가 많은 큰 파일에서 효율적으로 전체 파일을 두 부분으로 나누어 찾아야 되는 레코드가 어느 부분에 있는 가를 결정한 후 그 부분으로 가서 원하는 레코드를 찾을 때까지 이 과정을 반복한다.
① 이진탐색 탐색시간은 O(logn) 이다.
② 이진 탐색법은 성능은 우수하나 삽입이나 삭제시 정렬 상태를 유지하기 위해 평균적으로 n/2 레코드를 이동해야 하므로 삽입/삭제가 많은 응용 분야에는 적절치 못하다.
③ 이진 탐색법의 단점은 삽입과 삭제에 시간이 많이 걸린다.

3. 이진 탐색 나무
- 데이터가 고정되어 있지 않고 삽입과 삭제가 빈번한 경우에 적합한 구조
- 이진 탐색 나무는 이진 나무로서 각 노드 x의 왼쪽 서브트리의 모든 키는 x의 키보다 작고 오른쪽 서브트리의 모든 키는 x의 키보다 크다는 조건을 만족시킨다.
- 탐색 연산: 루트로부터 시작해서 크기 관계에 따라 나무의 경로를 따라 내려가면서 수행
- 삽입 연산: 우선 나무를 탐색한 후, 탐색이 실패한 지점에 자식 노드를 생성하여 삽입
- 삭제 연산: 3가지 경우로 나누어 수행 (삭제할 노드를 x라고 가정)

① 삭제하려는 노드가 말단 노드인 경우
삭제하려는 노드가 말단 노드인 경우에는 삭제하려는 노드가 말단 노드인 경우는 해당 노드만 삭제해준다.
② 삭제하려는 노드가 왼쪽이나 오른쪽 서브 트리중 하나만 가지고 있는 경우
삭제하려는 노드가 왼쪽이나 오른쪽 서브 트리중 하나만 갖고 있을 경우에는 자기 노드는 삭제하고 서브 트리는 자기 노드의 부모 노드에 붙여준다.
③ 삭제하려는 노드가 두 개의 서브 트리를 모두 가지고 있는 경우
자기 노드를 삭제한 후 두 개의 서브 트리를 하나로 합하여 부모 노드에 붙여준다. 합한 서브 트리의 루트 노드는 왼쪽 서브 트리에서 가장 큰 키 값을 가지는 노드나, 오른쪽 서브 트리에서 가장 작은 키 값을 가지는 노드가 된다.

- 탐색 시간은 최악의 경우(경사 나무가 생성되는 경우)에는 O(n), 평균 O(logn)이 된다.