

12

이진 탐색 트리 ADT

목적

이번 실습에서 여러분은

- 이진 탐색 트리 ADT를 링크드 트리 구조를 사용하여 만든다.
- 인덱스가 데이터베이스 파일에서 어떻게 레코드를 꺼내 오는데 사용되는지 조사하고 계정 데이터베이스(account database)에 대한 색인 프로그램(indexing program)을 구성한다.
- 트리의 높이(height)를 계산하는 기능을 작성하고 지정한 값보다 작은 트리 내의 원소들을 출력한다.
- 여러분이 구현한 이진 탐색 트리 ADT의 효율성을 분석하라.

개요

연구 11에서 여러분은 계층 데이터 구조를 사용하여 산술식의 평가가 표현되어지는 방법을 알아 보았다. 이번 실습에서 여러분은 이진 트리가 이진 검색 알고리즘을 구체화한 계층 검색 프로세스(hierarchical search process)를 표현하는데 사용될 수 있는 방법을 검사한다.

이진 탐색 알고리즘에서 키(key)라는 유일한 구별자를 가지고 정렬되어 적재되는 원소를 배열에 위치시킬 수 있다. 아래에 주어진 키들의 배열에서

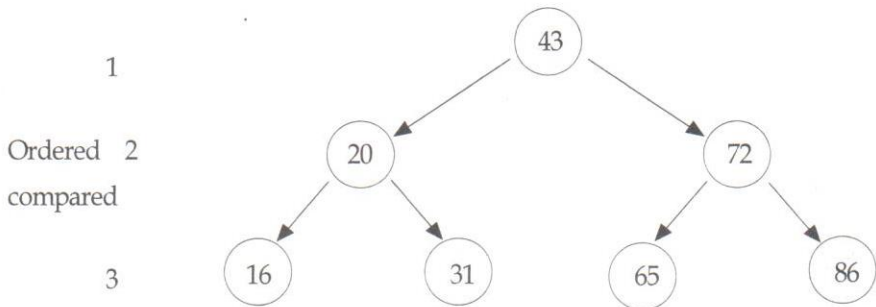
Index	0	1	2	3	4	5	6
Key	16	20	31	43	65	72	86

key값 31을 갖는 원소에 대한 이진 검색은 key 값 31과 배열의 중앙에 있는 배열의 key(43)과 비교를 시작으로 한다. 31이 43보다 작기 때문에 키가 31인 원소는 배열의 절반 아래에 놓인다(순위0-2). 부속 배열의 중앙의 키는 20이다. 31은 20보다 크기 때문에 31을 키로 갖는 원소는 이 부속 배열의 반 이상에 위치해야 한다(순위 2). 이 배열 순서는 키 31을 갖는다. 따라서 검색은 성공적으로 종료한다.

비록 주어진 키에 대한 검색이 키에 의존하는 동안 비교가 이루어져도 비교가 되는 관계 순서는 주어진 배열 원소에 대해 다양하지 않다. 예를 들어 이전의 배열을 탐색할 때 여러분은 키를 20이나 72와 비교하기 전에 검색하고 있는 키를 43과 비교한다. 비슷하게 키를 65나 86과 비교하기 전에 여러분은 항상 키를 72와 비교한다. 아래에 이 배열로 이루어진 비교 순서가 주어진다.

Index	0	1	2	3	4	5	6
Key	16	20	31	43	65	72	86
비교된 순서	3	2	3	1	3	2	3

이진 검색에 의해 수행되는 비교의 계층 상태는 다음 트리에 반영되어진다.



이 트리에서 각 키 K에 대한 관찰에서 K의 왼쪽의 모든 키는 K보다 작고 K의 오른쪽 부트리의 모든 키는 K보다 크다. 이런 성질을 갖는 키들은 이진 탐색 트리(binary search trees)라고 한다.

이진 탐색 트리에서 키에 대해 검색할 때, 여러분은 루트노드에서 시작하고 키를 포함하는 노드를 찾거나 단노드(leaf node)에 도달할 때까지 가지를 따라 아래 방향으로 이동한다. 이진 탐색 알고리즘에서 가지(branch)를 따르는 각 이동은 세분 배

열(arr subdivision)에 대응한다. 각 노드에서 여러분은 검색 중인 키가 노드에 적재된 키보다 작으면 왼쪽 아래로 이동하고 검색 중인 키가 노드에 적재된 키보다 크면 오른쪽 아래로 이동한다.

이진 탐색 트리 ADT

원소

이진 탐색 트리에 있는 원소들은 보통의 TE 타입이다. 각 원소는 유일하게 원소를 구별하는 키(일반적인 KF타입)를 갖는다. 원소들은 보통 추가적인 데이터를 포함한다. KF 타입은 6개의 기본적인 관계연산자를 지원하고 TE 타입은 원소의 키를 반환하는 `key()`를 호출하는 함수를 제공한다.

구조

원소들은 이진 탐색 트리 형태이다. 트리에서 각 원소 E에서 E의 왼편 서브트리에 있는 모든 원소는 E의 키보다 작은 키를 갖고 E의 오른편 서브트리에 있는 모든 원소는 E의 키보다 큰 키를 갖는다.

기능

BSTree ()

요구사항:

없음.

결과:

객체 생성자. 빈 이진 탐색 트리를 만든다.

~BSTree ()

요구사항:

없음.

결과:

객체 파괴자. 이진 탐색 트리를 적재하는데 사용한 메모리를 할당 해제.

void insert (const TE &newElement)

요구사항:

이진 탐색트리는 가득 차 있지 않다.

결과:

newElement를 이진 탐색 트리에 삽입한다. newElement와 같은 키를 갖는 원소가 이미 트리내에 존재하면 원소의 키가 아닌 필드를 newElement의 키가 아닌 필드로 갱신한다.

Int retrieve (KF searchKey, TE &searchElement) const

요구사항:

없음.

결과:

searchkey를 갖는 원소에 대한 이진 탐색 트리를 검색한다. 이 원소가 발견되면 원소를 searchElement로 복사하고 1을 반환한다. 그렇지 않은 경우는 정의되지 않은 searchElement로 0을 반환한다.

int remove (KF deleteKey)

요구사항:

없음.

결과:

이진 탐색 트리에서 키 deleteKey를 갖는 원소를 제거한다. 원소가 발견되면 트리에서 삭제하고 1을 반환, 발견 못하면 0을 반환.

void writeKeys () const

요구사항:

없음.

결과:

이진 탐색 트리에서 원소들의 키를 출력한다. 한 행마다 키는 오름차순으로 출력된다.

void clear ()

요구사항:

없음.

결과:

이진 탐색 트리에서 모든 원소를 삭제한다.

int empty () const

요구사항:

없음.

결과:

이진 탐색 트리가 비어 있으면 1을 반환, 그렇지 않으면 0을 반환.

int full () const

요구사항:

없음.

결과:

이진 탐색 트리가 가득차 있으면 1을 반환 아니면 0을 반환.

void showStructure () const

요구사항:

없음.

결과:

이진 탐색 트리에서 키를 출력한다. 트리는 왼쪽에서 오른쪽으로 향하는 가지로 출력된다. 즉 트리는 기본 위치에서 시계 반대 방향으로 90도 회전되어 출력된다. 트리가 비어 있으면 "Empty tree"를 출력한다. 이 기능은 시험와 디버깅을 목적으로 사용한다.

실습 12 : 표지

날짜_____ 구분_____

성명_____

강사가 여러분에게 할당한 다음에 연습문제들에 할당된 열에 체크표시를 하고 이 결장을 앞으로의 연구에서 여러분이 제출할 과제물 앞에 붙이시오.


	할 일	완 료
실습 전 연습	✓	
연 결 연습	✓	
실습 중 연습 1		
실습 중 연습 2		
실습 중 연습 3		
실습 후 연습 1		
실습 후 연습 2		
	총 점	

실습 12 : 실습 전 연습

날짜_____ 구분_____

성명_____

1단계 : 링크드 트리 구조를 사용하여 이진 탐색 트리 ADT로 기능을 구현하여라. 여러분이 이전 연구에서 개발한 링크드 선형 구조를 가지고 여러분은 링크드 트리 구조 구현에 트리내의 노드(BSTreeNode)에 대한 것과 전체 트리 구조(BSTree)대한 두 개의 클래스를 사용한다. 트리의 각 노드는 한 원소를 가져야 하고 노드의 자식에 대해 두 개의 포인터를 갖는다(왼쪽과 오른쪽). 여러분의 구현은 역시 트리의 루트 노드에 대한 포인터를 유지해야 한다.

 파일 *bstree.hs*에 있는 다음의 선언에 기본으로 하여 구현하라.

 *showStructrue* 연산자의 구현은 파일 *show12.cpp*에 있다.

```
template <class TE, class KF>
class BSTreeNode // Facilitator class for the BSTree class
{
    private:

        // Constructor
        BSTreeNode (const TE &elem, BSTreeNode *leftPtr,
                    BSTreeNode *rightPtr);

        // Data members
        TE element;           // Binary search tree element
        BSTreeNode *left,     // Pointer to the left child
                    *right;   // Pointer to the right child
    friend class BSTree<TE,KF>;
};

template <class TE, class KF> // TE : tree element
class BSTree                // KF : key field
{
    public:
```

```

// Constructor
BSTree ( );

// Destructor
~BSTree ( );

// Binary search tree manipulation operations
void insert (const TE &newElement); // Insert element
int retrieve (KF searchKey, TE &searchElement) const; // Retrieve element
int remove (KF deleteKey); // Remove element
void writeKeys ( ) const; // Output keys
void clear ( ); // Clear tree

// Binary search tree status operations
int empty ( ) const; // Tree is empty
int full ( ) const; // Tree is full
// Output the tree structure -- used in
// testing/debugging
void showStructure ( ) const;

private:
// Recursive partners of the public member functions --
// insert prototypes of these functions here.
void showSub (BSTreeNode<TE, KF> *p, int level) const;

// Data member
BSTreeNode<TE, KF> *root; // Pointer to the root node
};

```



2단계 : 파일 `bstree.hs`에 있는 `BSTree` 클래스의 선언은 여러분의 이진 탐색 트리 ADT구현에서 필요로 하는 재귀 `private` 멤버 함수를 포함하지 않는다. 프로토타입을 추가하고 클래스 선언의 결과를 파일 `bstree.h`에 저장하라.

3단계 : 여러분의 이진 탐색 트리 ADT구현을 파일 `bstree.cpp`에 저장하라. 여러분의 코드를 문서로 확인하라.

실습 12 : 연결 연습문제

날짜 _____ 구분 _____

성명 _____

여러분의 강사와 함께 여러분이 이 연습을 연구 기간 이전이나 연구 기간 중에 마칠 수 있는지 체크하라.

파일 `test12.cpp`에 있는 시험 프로그램에서 여러분은 다음 명령어를 사용하는 이진 탐색 트리 ADT의 구현을 상호적으로 시험할 수 있다.

명령어	기 능
+key	지정된 키로 원소를 삽입(또는 갱신).
?key	지정된 키를 갖는 원소를 꺼내고 출력한다.
-key	지정된 키를 갖는 원소를 삭제한다.
K	오름차순으로 키들을 출력한다.
E	트리가 비어 있는지 조사한다.
F	트리가 가득 차 있는지 조사한다.
C	트리를 clear한다.
Q	테스트 프로그램을 종료한다.

1단계 : 이진 탐색 트리 ADT의 구현에 대한 시험 계획을 준비하라. 여러분의 시험 계획은 단 가지(single-branch) 트리, 단일 원소 트리, 빈 트리를 포함해서 다양한 모양과 크기를 모두 다루어야 한다.

2단계 : 여러분의 시험 계획을 실행하라. 여러분의 구현에서 실수를 발견하면 수정하고 다시 실행하라.

이진 탐색 트리 ADT 시험 계획			
시험 항목	명령어	예상 결과	검사