객체지향프로래밍과 자료구조 (실습)

Lab 10 (보충설명) Re-Balancing Binary Search Tree

영남대학교 정보통신공학과

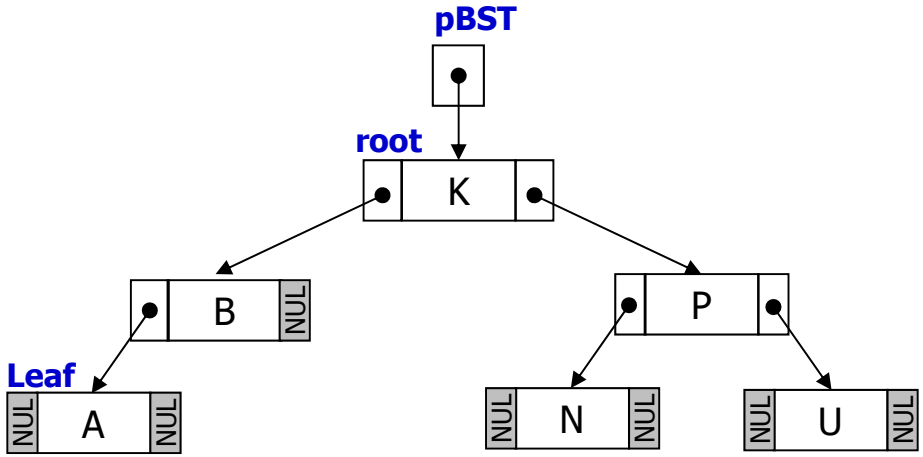
교수 김 영 탁

(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

이진탐색트리 (Binary Search Tree)

이진 트리 (Binary Trees)

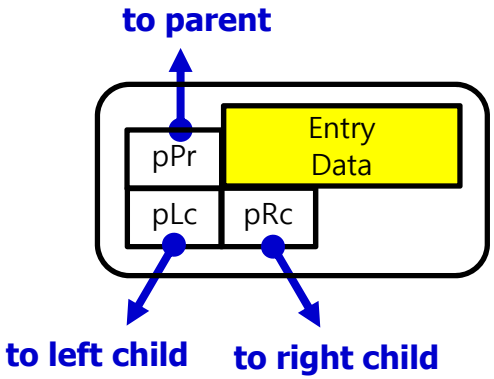
- **Binary tree (이진트리)**
  - 비 선형 자료 구조이며, 각 노드는 0, 1, 또는 2개의 다른 노드를 가리킴
- **Parent(부모) 노드, Children (자식) 노드**
  - (부모) 노드 N이 가리키는 하위 노드들은 그 (부모) 노드 N의 children (자식) 노드들임
- **Root(루트) 노드**
  - 이진 트리의 최상위 부모노드
- **Leaf(리프) 노드**
  - 이진 트리의 최하단 노드로서 자식이 없는 트리노드



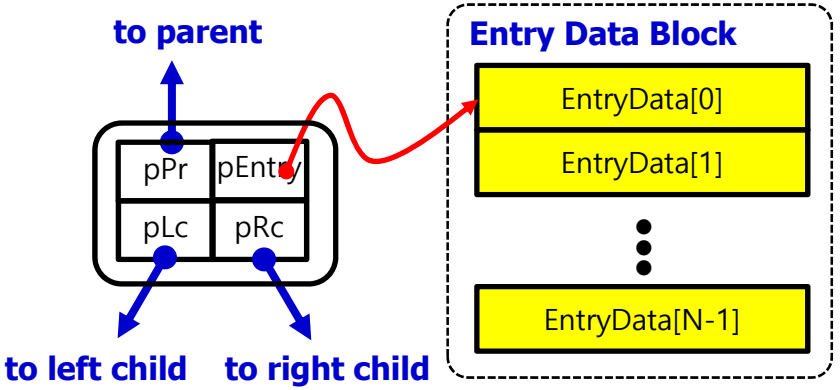
Binary Tree Node

◆ Binary Tree Node

- three pointers: to left child(pLc), to right child(pRc), to parent (pPr)
- data or pointer to data (pD)



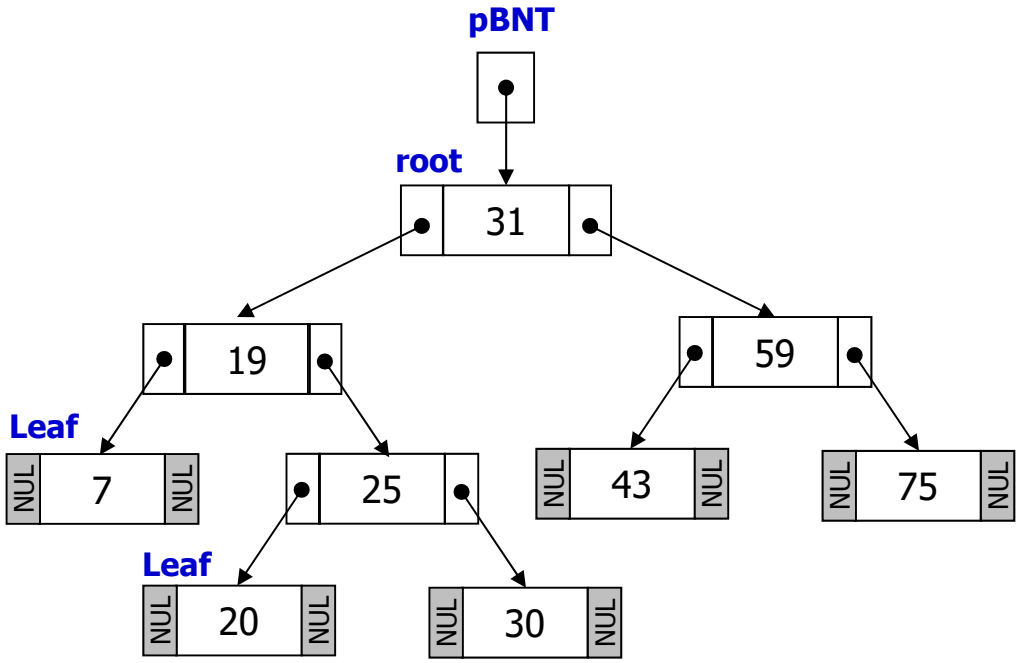
(a) Binary Tree Node with Data



(b) Binary Tree Node with pointer to Entry Data

이진 탐색 트리 (Binary Search Tree)

- 이진 탐색 트리 (Binary search tree): 데이터 탐색을 쉽게 수행할 수 있도록 구성된 이진 트리
- 각 노드의 왼쪽 서브 트리는 그 노드의 값보다 작은 데이터를 가지는 노드들로만 구성
- 각 노드의 오른쪽 서브 트리는 그노드이 데이터 값보다 큰 데이터 값을 가지는 노드 들로만 구성



T_Entry<K, V>

```
template<typename K, typename V>
class T_Entry
{
   friend ostream& operator<<(ostream& fout, T_Entry<K, V>& entry)
   {
         fout << "(" << entry.getKey() << ": " << *(entry.getValue()) << ")";
         return fout;
   }
public:
   T_Entry(K key, V value): _key(key), _value(value) {} // constructor
   T_Entry() { } // default constructor
   ~T_Entry() {}
   void setKey(const K& key) { _key = key; }
   void setValue(const V& value) { _value = value; }
   K getKey() const { return _key; }
   V getValue() const { return _value; }
   bool operator>(const T_Entry<K, V>& right) const { return (_key > right.getKey()); }
   bool operator>=(const T_Entry<K, V>& right) const { return (_key >= right.getKey()); }
   bool operator<(const T_Entry<K, V>& right) const { return (_key < right.getKey()); }
   bool operator<=(const T_Entry<K, V>& right) const { return (_key <= right.getKey()); }
        bool operator==(const T_Entry<K, V>& right) const
        { return ((_key == right.getKey()) && (_value == right.getValue())); }
   T_Entry<K, V>& operator=(T_Entry<K, V>& right);
   void fprint(ostream fout);
private:
   K _key;
   V _value;
};
```

class T_BSTN<K, V>

```
/** Template Binary Search Tree Node. h */
#ifndef T_BSTN_H
#define T_BSTN_H
#include "T_Entry.h"
template<typename K, typename V>
class T_BSTN    //  binary search tree node
{
public:
      T_BSTN() : entry(), pPr(NULL), pLc(NULL), pRc(NULL) { } // default constructor
      T_BSTN(T_Entry<K, V> e) : entry(e), pPr(NULL), pLc(NULL), pRc(NULL) { } // constructor
      K getKey() { return entry.getKey(); }
      V getValue() { return entry.getValue(); }
      T_Entry<K, V>& getEntry() { return entry; }
      void setEntry(T_Entry<K, V> e) { entry = e; }
      T_BSTN<K, V>* getpPr() { return pPr; }
      T_BSTN<K, V>* getpLc() { return pLc; }
      T_BSTN<K, V>* getpRc() { return pRc; }
      T_BSTN<K, V>** getppLc() { return &pLc; }
      T_BSTN<K, V>** getppRc() { return &pRc; }
      void setpPr(T_BSTN<K, V>* pTN) { pPr = pTN; }
      void setpLc(T_BSTN<K, V>* pTN) { pLc = pTN; }
      void setpRc(T_BSTN<K, V>* pTN) { pRc = pTN; }
      T_Entry<K, V>& operator*() { return entry; }
private:
      T_Entry<K, V> entry;      // element value
      T_BSTN<K, V>*   pPr;     // parent
      T_BSTN<K, V>*   pLc;     // left child
      T_BSTN<K, V>*   pRc;     // right child
};
#endif
```

Class T_BST<K, V>

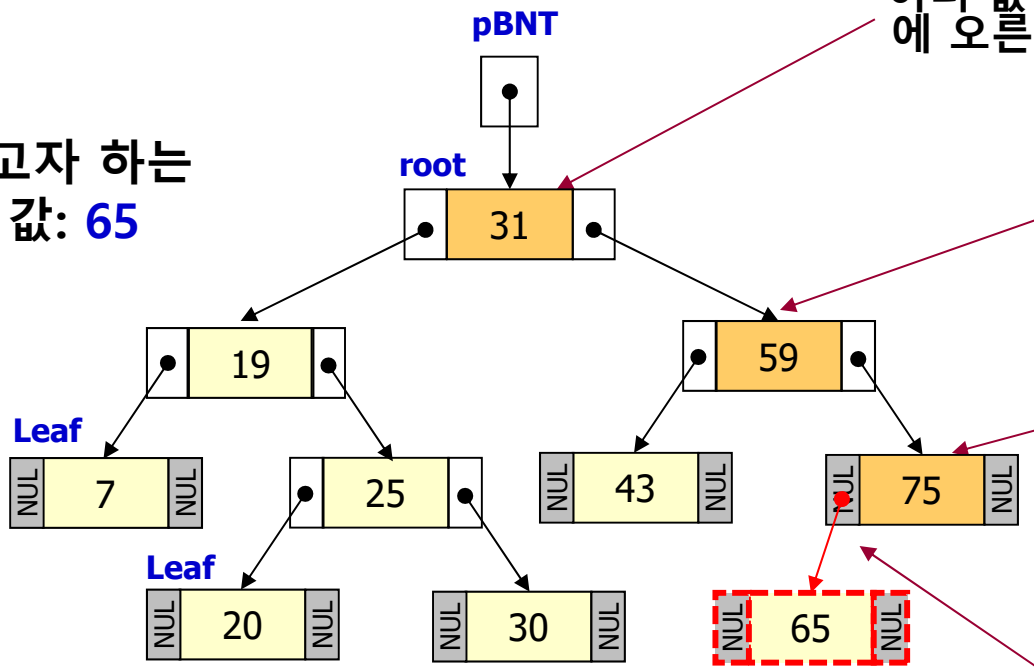
```
/** Template Binary Search Tree.h  (1) */
#ifndef T_BST_H
#define T_BST_H

#include <iostream>
#include <fstream>
#include "T_BSTN.h"
#include "T_Array.h"

template<typename K, typename V>
class T_BST
{
public:
    T_BST(string nm) : _root(NULL), num_entry(0), name(nm) {}  // constructor
    string getName() { return name;  }
    int size() const { return num_entry; }
    bool empty() const { return  (num_entry == 0); }
    void clear();

    T_BSTN<K, V>* getRoot() { return _root;  }
    T_BSTN<K, V>** getRootAddr() { return &_root; }
    T_Entry<K, V>& getRootaEntry() { return _root->getEntry(); }
```

```
/** Template Binary Search Tree.h  (1) */

    T_BSTN<K, V>* eraseBSTN(T_BSTN<K, V>** pp);
    void insertInOrder(const T_Entry<K, V> entry);
    void insertAndRebalance(T_Entry<K, V> e);
    void traversal_inOrder(T_BSTN<K, V>* p, T_Array<V>& array_value);
    void traversal_preOrder(T_BSTN<K, V>* pos, T_Array<V>& array_value);
    void traversal_postOrder(T_BSTN<K, V>* pos, T_Array<V>& array_value);

    T_BSTN<K, V>* searchBSTN(K k);
    T_Entry<K, V>& minEntry();
    T_Entry<K, V>& maxEntry();
    void fprint_with_Depth(ostream& fout);
    void fprint_inOrder(ostream& fout);

protected:
    T_BSTN<K, V>* _maxBSTN(T_BSTN<K, V>* subRoot);
    T_BSTN<K, V>* _minBSTN(T_BSTN<K, V>* subRoot);
```

```
/** Template Binary Search Tree.h  (1) */

protected:
    T_BSTN<K, V>*
        _insertInOrder(T_BSTN<K, V>** p, T_BSTN<K, V>* parenPos, const T_Entry<K, V> e);
    T_BSTN<K, V>*
        _insertAndRebalance(T_BSTN<K, V>** ppTN, T_BSTN<K, V>* pPr, T_Entry<K, V> e);
    T_BSTN<K, V>* _rotate_LL(T_BSTN<K, V> *pCurSubRoot);
    T_BSTN<K, V>* _rotate_RR(T_BSTN<K, V> *pCurSubRoot);
    T_BSTN<K, V>* _rotate_RL(T_BSTN<K, V> *pCurSubRoot);
    T_BSTN<K, V>* _rotate_LR(T_BSTN<K, V> *pCurSubRoot);
    int _getHeight(T_BSTN<K, V>* pTN);
    int _getHeightDiff(T_BSTN<K, V>* pTN);
    T_BSTN<K, V>* _reBalance(T_BSTN<K, V>** ppTN);
    T_BSTN<K, V>* _searchBSTN(T_BSTN<K, V>* pos, K k);
    void _fprint_with_Depth(T_BSTN<K, V>* pTN, ostream& fout, int depth);
    void _fprint_inOrder(T_BSTN<K, V>* pTN, ostream& fout);

private:
     T_BSTN<K, V>* _root; // pointer to the root
     int num_entry; // number of tree nodes
     string name;
};  // end of class T_BST
```

이진 탐색 트리 (Binary Search Tree)에 새로운 Entry (65 데이터 값)의 추가

입력하고자 하는 노드 값: 65



Step 1: 65가 root 노드의 데이터 값 (31) 보다 크기 때문에 오른쪽 서브 트리에 삽입해야 함.

Step 2: 65가 서브트리 root 노드의 데이터 값 (59) 보다 크기 때문에, 오른쪽 서브 트리에 삽입해야 함.

Step 3: 65가 서브트리 root 노드의 데이터 값 (75) 보다 작기 때문에, 왼쪽 서브 트리에 삽입해야 함.

Step 4: 왼쪽 서브 트리가 NULL 이므로 데이터 65의 이진 트리 노드를 이곳에 삽입

이진 탐색 트리에서의 탐색 (Searching)

1: Root 노드 부터 시작

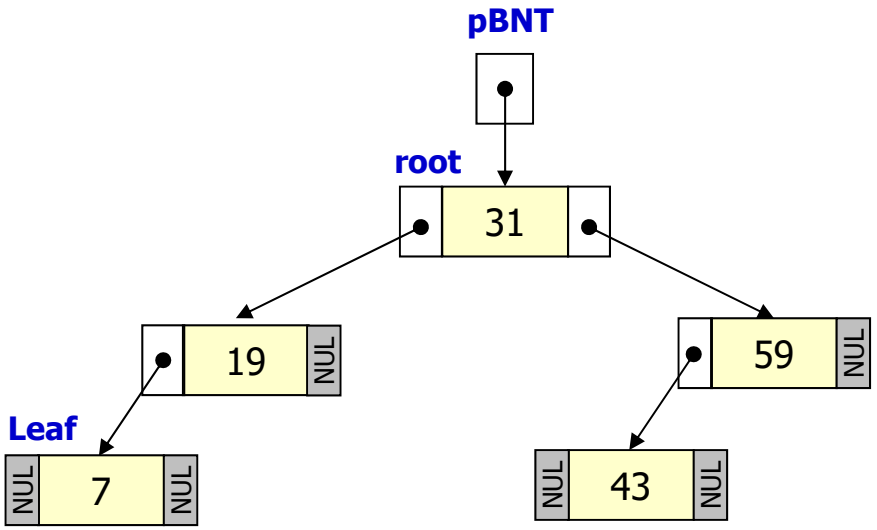
2: 노드 데이터를 검사 :

2.1: 만약 찾고자 하는 데이터라면 탐색 성공/종료

2.2: 만약 탐색 데이터가 노드 데이터 보다 작다면, 왼쪽 서브 트리에서 step 2를 수행

2.3: 만약 탐색 데이터가 노드 데이터 보다 크다면, 오른쪽 서브 트리에서 step 2를 수행

3: 탐색 데이터를 찾을 때 까지 계속 탐색하거나, NULL 포인터에 도달 할 때 까지 수행

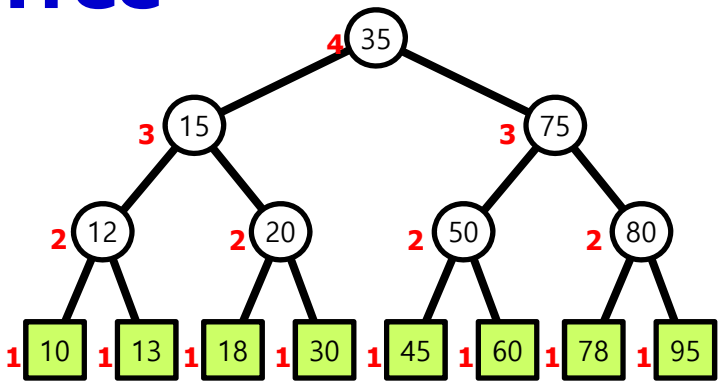


이진탐색트리의 균형화
(Balanced Binary Search Tree)

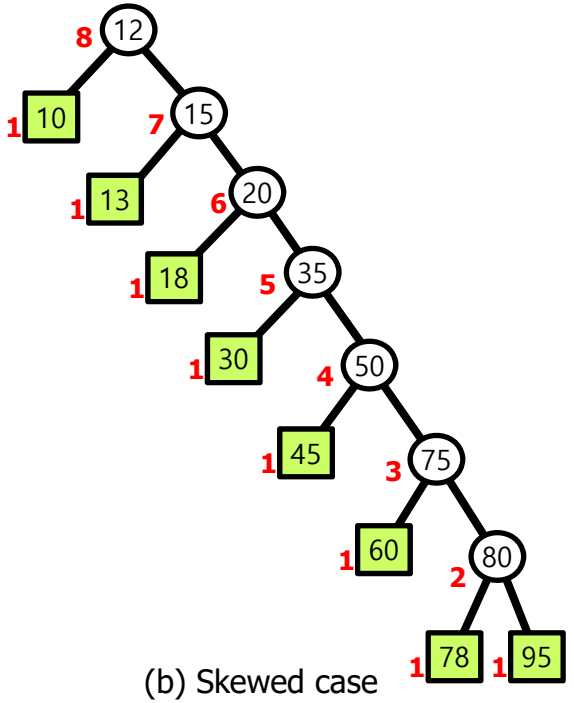
Binary Search Tree

Basic operations

- tree node contains element and key
- insert in order according to the given key
- binary search tree maintains the order among tree nodes
- search the element (node) with the given key



(a) Well-balanced case



(b) Skewed case
(without re-balancing)

Performance of Binary Search Tree

- when binary search tree is balanced, search takes $O(\log_2 N)$
- when binary search tree is skewed in one direction, search takes $O(N)$

Binary Tree의 문제점

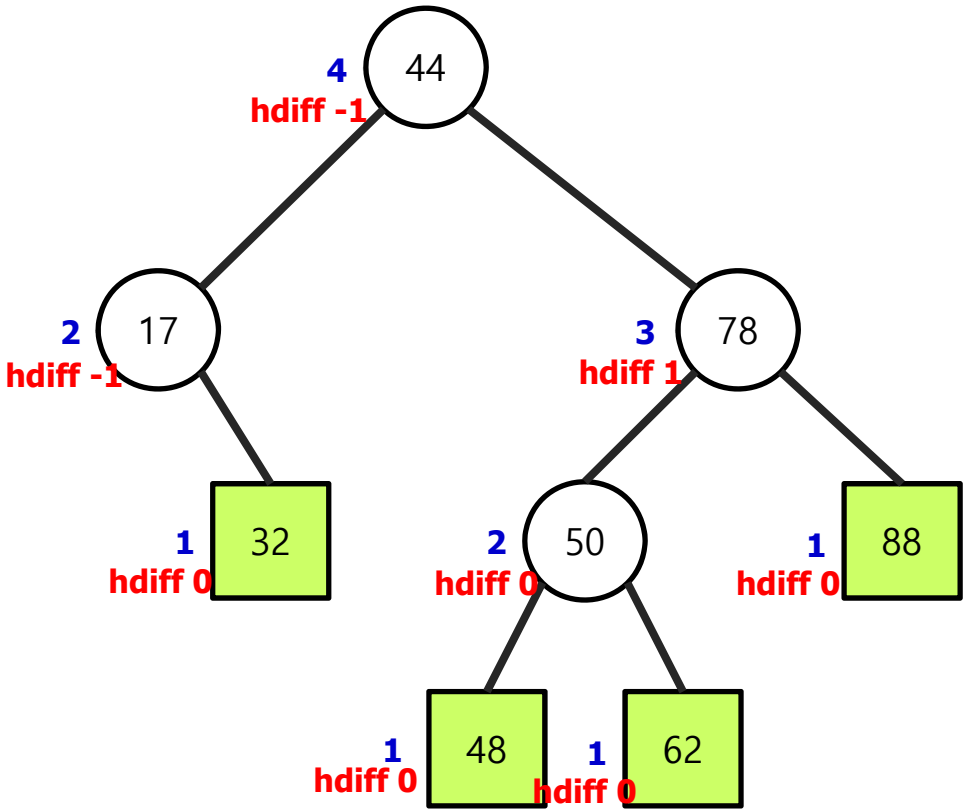
◆ Binary Tree의 편중/편향

- Binary Tree의 좌우 균형이 잘 잡혀있는 경우, 탐색 (search) 시간은 $O(\log_2 N)$
- Root 노드의 데이터 값이 전체 데이터의 중간 값을 가지지 못하는 경우, 이진 트리가 한쪽으로 편중/편향될 수 있으며, 이 경우 binary search tree 기반의 search 성능이 저하됨
- 최악의 경우로 root 노드가 전체 데이터 값 중 최소값 또는 최대값을 가지는 경우, linked list와 같은 형태가 발생할 수 도 있음: 탐색 시간이 평균 N/2임 (i.e., O(N))
- Binary Tree에 새로운 데이터 노드를 추가할 때, 전체 binary tree의 균형을 잡을 수 있도록 재 조정하여야 함

균형 이진 탐색 트리 - AVL Tree

◆AVL Tree

- Proposed by Adelson, Velskii, Landis in 1962
- AVL trees are balanced
- An AVL Tree is a binary search tree such that for every internal node v of T, the heights of the children of v can differ by at most 1



An example of an AVL tree
(with **height and height difference**)

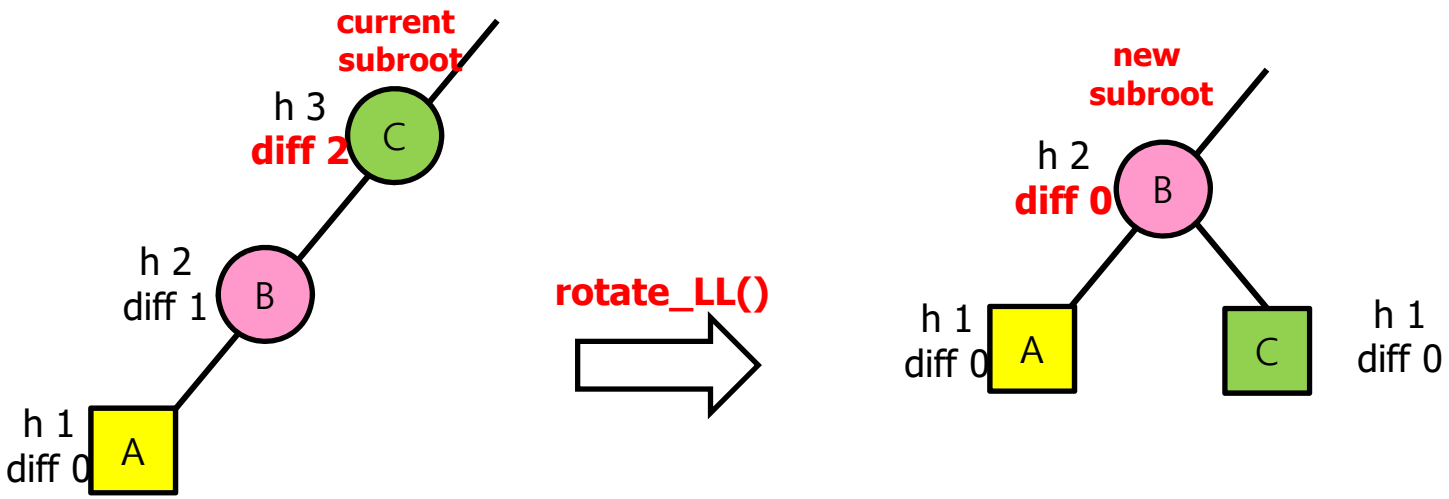
◆ getHeight(), getHeightDiff()

```
template<typename K, typename V>
int T_BST<K, V>::_getHeight(T_BSTN<K, V>* pTN)
{
    int height = 0;
    int height_Lc, height_Rc;

    if (pTN != NULL)
    {
        height_Lc = _getHeight(pTN->getpLc());
        height_Rc = _getHeight(pTN->getpRc());
        if (height_Lc > height_Rc)
            height = 1 + height_Lc;
        else
            height = 1 + height_Rc;
    }
    return height;
}
```

```
template<typename K, typename V>
int T_BST<K, V>::
    _getHeightDiff(T_BSTN<K, V>* pTN)
{
    int heightDiff = 0;

    if (pTN == NULL)
        return 0;
    heightDiff = _getHeight(pTN->getpLc())
                - _getHeight(pTN->getpRc());

    return heightDiff;
}
```

Tri-node Restructuring – rotate_LL

◆ Single rotation LL of subtree

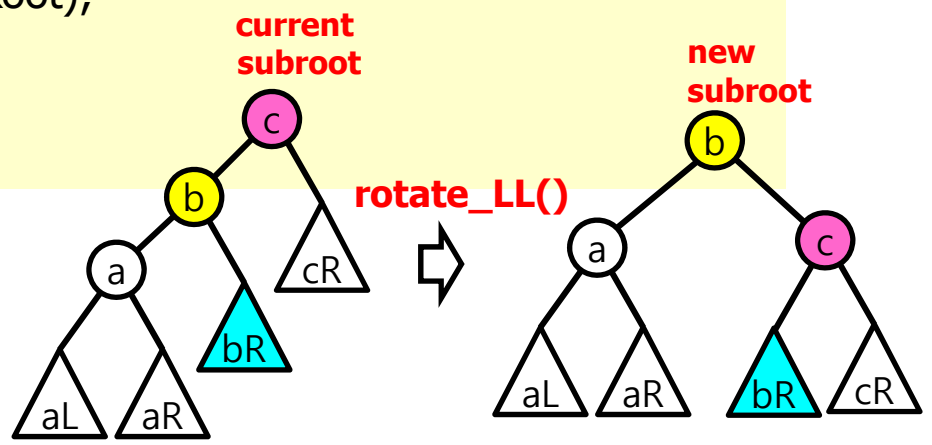


(a) Left-left skewed

(b) After rotate_LL()

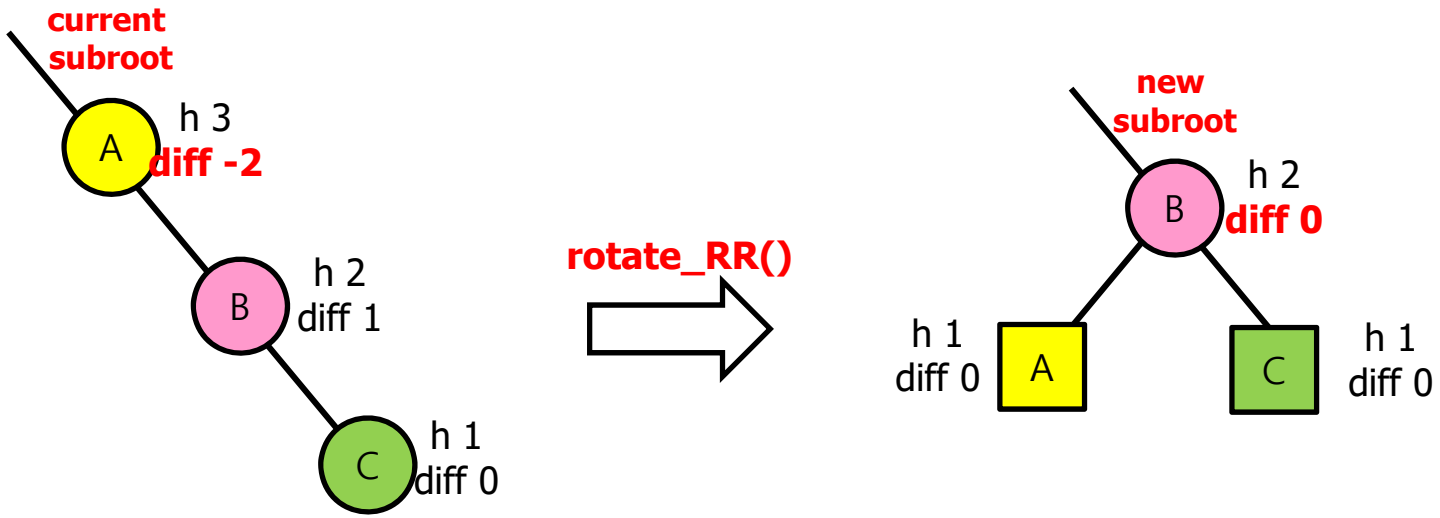
◆ rotate_LL()

```
template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::_rotate_LL(T_BSTN<K, V> *pCurSubRoot)
{
    T_BSTN<K, V> *pNewSubRoot, *pBR, *pCurParent;

    pCurParent = pCurSubRoot->getpPr();
    pNewSubRoot = pCurSubRoot->getpLc();
    pBR = pNewSubRoot->getpRc();
    pCurSubRoot->setpLc(pBR);
    if (pBR != NULL)
        pBR->setpPr(pCurSubRoot);
    pNewSubRoot->setpRc(pCurSubRoot);
    pNewSubRoot->setpPr(pCurParent);
    pCurSubRoot->setpPr(pNewSubRoot);

    return pNewSubRoot;
}
```



Tri-node Restructuring – rotate_RR (1)

◆ Single rotation RR of subtree

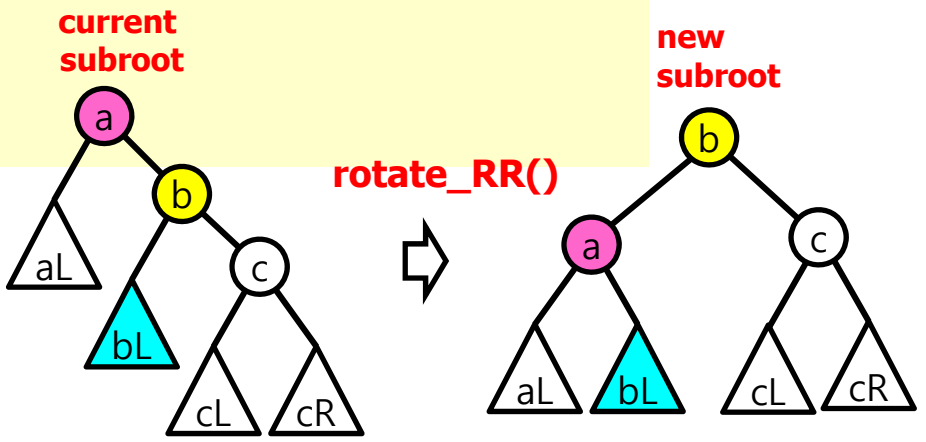


(a) Right-right skewed

(b) After rotate_RR()

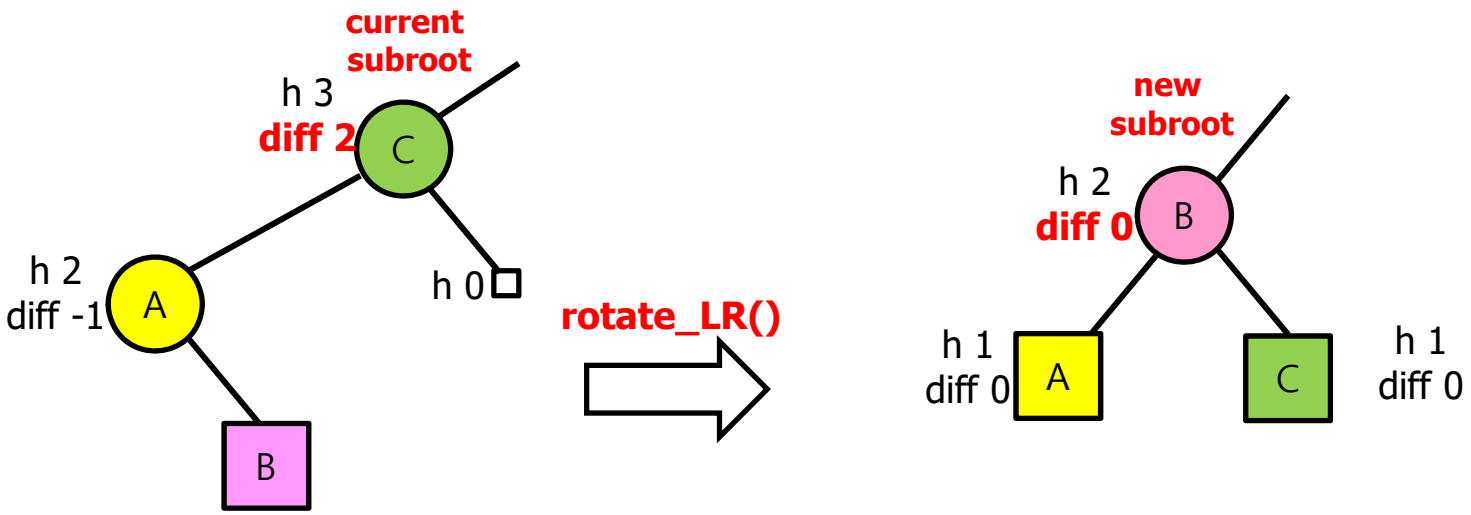
◆ rotate_RR(),

```
template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::_rotate_RR(T_BSTN<K, V> *pCurSubRoot)
{
    T_BSTN<K, V> *pNewSubRoot, *pBL, *pCurParent;

    pCurParent = pCurSubRoot->getpPr();
    pNewSubRoot = pCurSubRoot->getpRc();
    pBL = pNewSubRoot->getpLc();
    pCurSubRoot->setpRc(pBL);
    if (pBL != NULL)
        pBL->setpPr(pCurSubRoot);
    pNewSubRoot->setpLc(pCurSubRoot);
    pNewSubRoot->setpPr(pCurParent);
    pCurSubRoot->setpPr(pNewSubRoot);

    return pNewSubRoot;
}
```



Tri-node Restructuring – rotate_LR (1)

◆ Double rotation LR of subtree

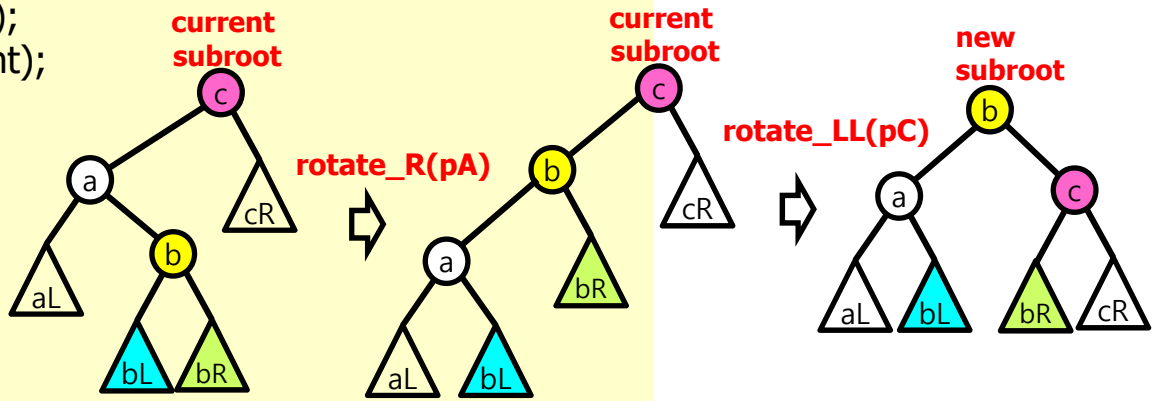


(a) Left-right skewed

(b) After rotate_LR()

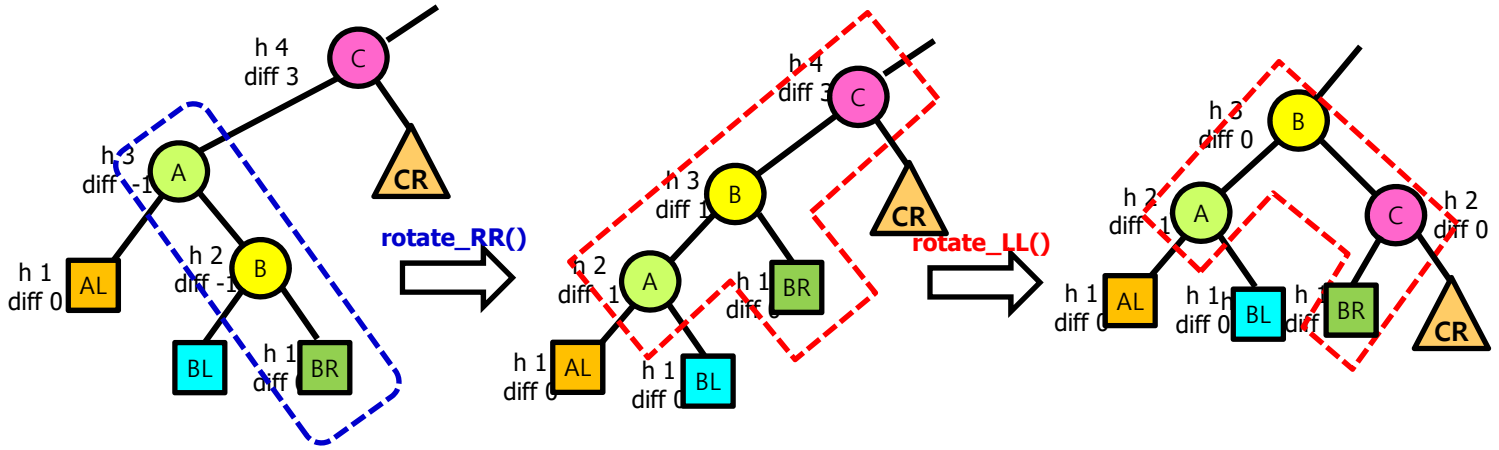
◆ Algorithm rotate_LR()

```
template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::_rotate_LR(T_BSTN<K, V> *pCurSubRoot)
{
    T_BSTN<K, V> *pSubRoot, *pNewSubRoot, *pCurParent;
    T_BSTN<K, V> *pA, *pB, *pC, *pBL, *pBR;

    pC = pCurSubRoot;
    pCurParent = pCurSubRoot->getpPr();
    pA = pC->getpLc();
    pB = pA->getpRc();
    pBL = pB->getpLc();
    pBR = pB->getpRc();
    pSubRoot = _rotate_RR(pA);
    pCurSubRoot->setpLc(pSubRoot);
    pNewSubRoot = _rotate_LL(pC);
    pNewSubRoot->setpPr(pCurParent);
    pA->setpPr(pNewSubRoot);
    pC->setpPr(pNewSubRoot);
    if (pBL != NULL)
        pBL->setpPr(pA);
    if (pBR != NULL)
        pBR->setpPr(pC);

    return pNewSubRoot;
}
```



(a) LR- skewed (b) after rotate_RR(pA) (c) after rotate_LR()

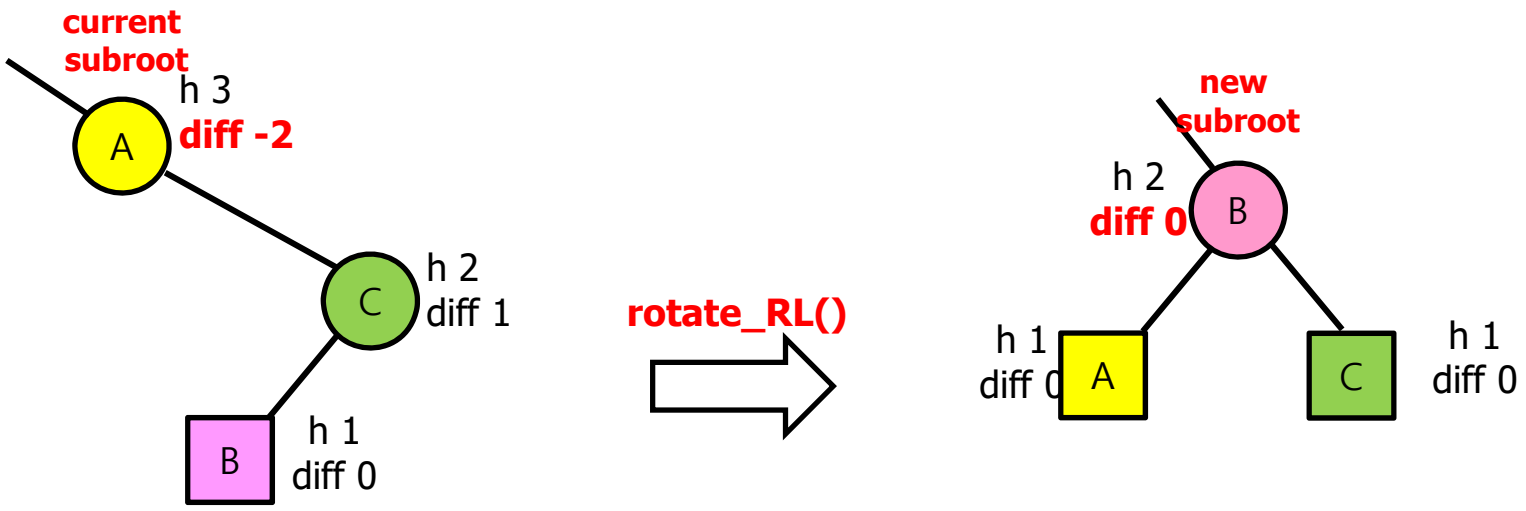
◆ Detailed Operations in rotateLR()



(a) Initial Status (b) after rotate_RR() (c) after rotate_LL()

Tri-node Restructuring – rotate_RL (1)

◆ Double rotation RL of subtree

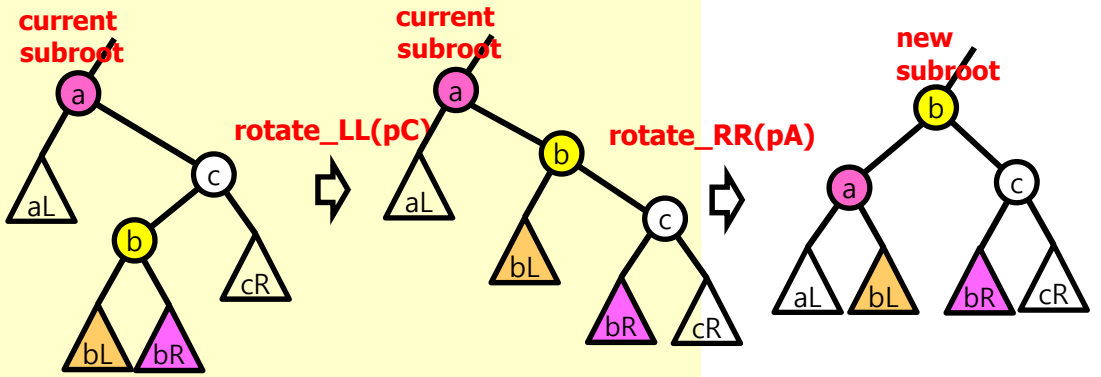


(a) Right-Left skewed

(b) After rotate_RL()

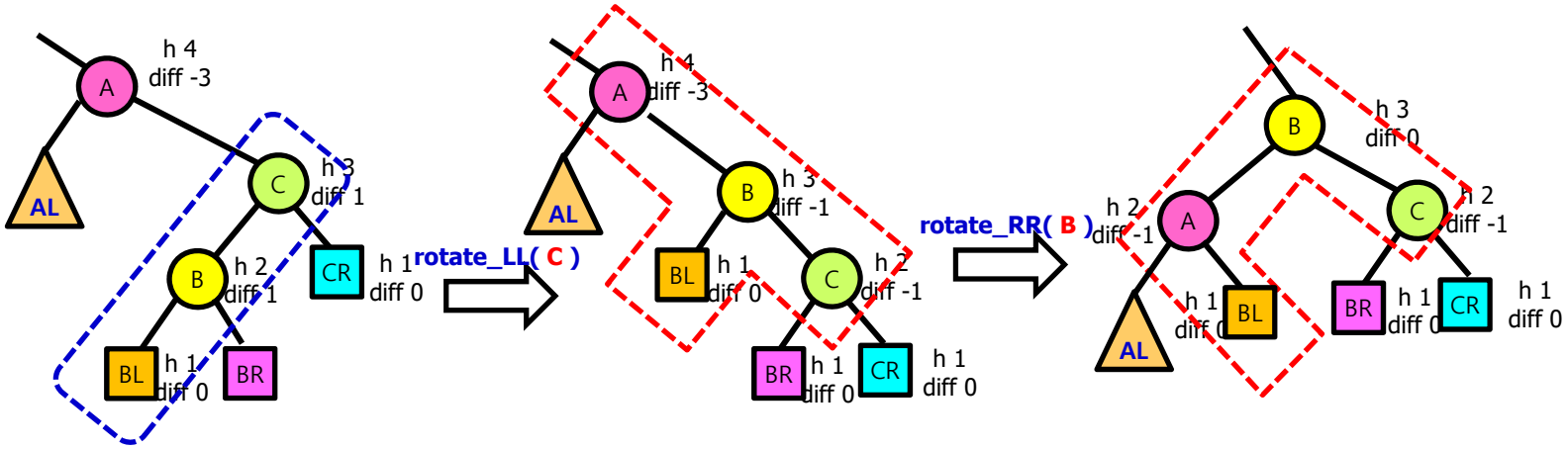
◆ rotate_RL()

```
template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::_rotate_RL(T_BSTN<K, V> *pCurSubRoot)
{
    T_BSTN<K, V> *pSubRoot, *pNewSubRoot, *pCurParent;
    T_BSTN<K, V> *pA, *pB, *pC, *pBL, *pBR;

    pA = pCurSubRoot;
    pCurParent = pCurSubRoot->getpPr();
    pC = pA->getpRc();
    pB = pC->getpLc();
    pBL = pB->getpLc();
    pBR = pB->getpRc();
    pSubRoot = _rotate_LL(pC);
    pCurSubRoot->setpRc(pSubRoot);
    pNewSubRoot = _rotate_RR(pA);

    pNewSubRoot->setpPr(pCurParent);
    pA->setpPr(pNewSubRoot);
    pC->setpPr(pNewSubRoot);
    if (pBL != NULL)
        pBL->setpPr(pA);
    if (pBR != NULL)
        pBR->setpPr(pC);

    return pNewSubRoot;

}
```



(a) Right-Left skewed (b) Right-Left skewed (c) after rotate_RL()

◆ Detailed Operations in rotateRL()



(a) Initial status (b) After rotate_LL (c) After rotate_RR

_reBalance()

```
template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::_reBalance(T_BSTN<K, V>** ppTN)
{
	int heightDiff = 0;

	heightDiff = _getHeightDiff(*ppTN);
	if (heightDiff > 1) // left subtree is higher
	{
		if (_getHeightDiff((*ppTN)->getpLc()) > 0)
			*ppTN = _rotate_LL(*ppTN);
		else
			*ppTN = _rotate_LR(*ppTN);
	}
	else if (heightDiff < -1)  // right subtree is higher
	{
		if (_getHeightDiff((*ppTN)->getpRc()) < 0)
			*ppTN = _rotate_RR(*ppTN);
		else
			*ppTN = _rotate_RL(*ppTN);
	}

	return *ppTN;
}
```

insertAndRebalance()

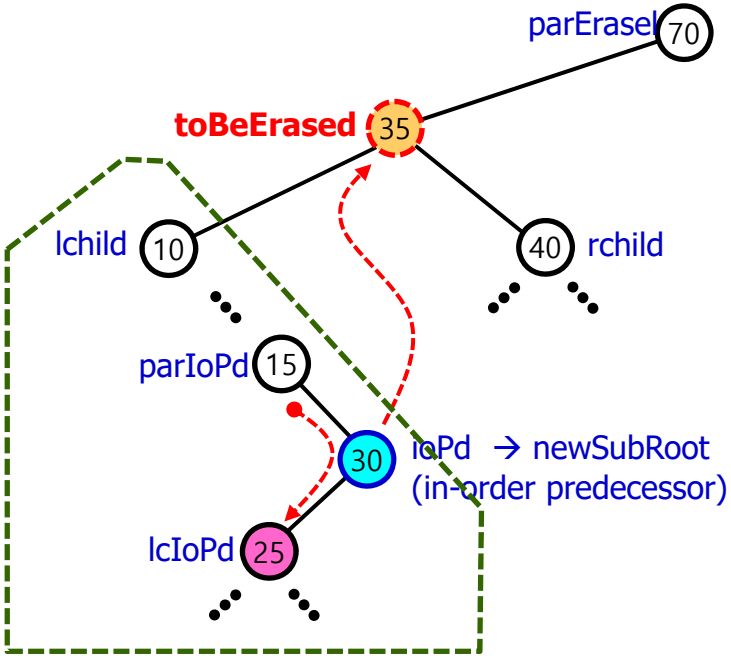
```
template<typename K, typename V>
void T_BST<K, V>::insertAndRebalance(T_Entry<K, V> entry)
{
    _insertAndRebalance(&_root, NULL, entry);
}

template<typename K, typename V>
T_BSTN<K, V>* T_BST<K, V>::_insertAndRebalance(T_BSTN<K, V>** ppTN,
    T_BSTN<K, V>* pPr, T_Entry<K, V> entry)
{
     T_BSTN<K, V> *pTN, **ppLc, **ppRc;

     if (*ppTN == NULL) // attach a new tree node at the currently external node
     {
         pTN = new T_BSTN<K, V>(entry);
         *ppTN = pTN;
         if (pPr != NULL) // if not root
             pTN->setpPr(pPr);
         (*ppTN)->setpLc(NULL);
         (*ppTN)->setpRc(NULL);
         num_entry++;
         return *ppTN;
     }
```

```
    T_Entry<K, V> bstn_entry;
    bstn_entry = (*ppTN)->getEntry();
    if (entry < bstn_entry)  // T_Entry<K, V> must provide '<' operator overloading !!
    {
        ppLc = (*ppTN)->getppLc();
        pTN = _insertAndRebalance(ppLc, *ppTN, entry);
        if (ppTN != NULL)
        {
            (*ppTN)->setpLc(pTN);
            *ppTN = _reBalance(ppTN);
        }
    }
    else // entry >= bstn_entry
    {
        ppRc = (*ppTN)->getppRc();
        pTN = _insertAndRebalance(ppRc, *ppTN, entry);
        if (ppTN != NULL)
        {
            (*ppTN)->setpRc(pTN);
            *ppTN = _reBalance(ppTN);
        }
    }
    return *ppTN;
}
```

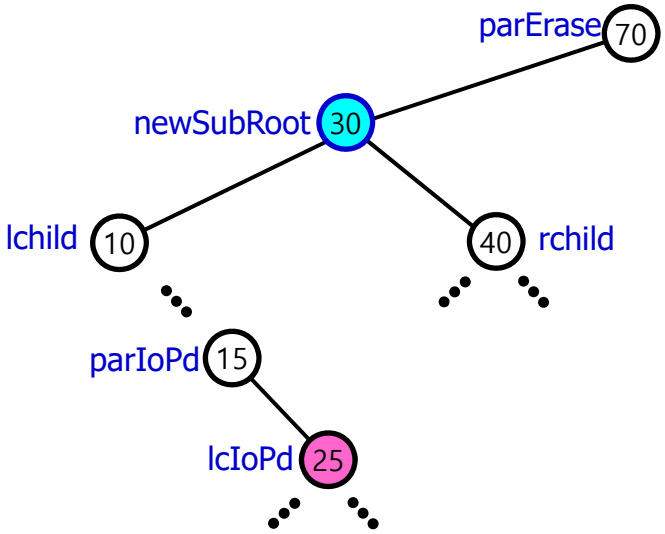
Balance를 고려한 deleteBSTN() (1)

◆ 왼쪽 서브 트리의 height가 더 높은 경우

- 왼쪽 서브 트리에서 가장 큰 값을 가진 in-order predecessor를 찾아 삭제 대상 노드 위치에 배치



(a) Before erase of a tree node
(delete 35)

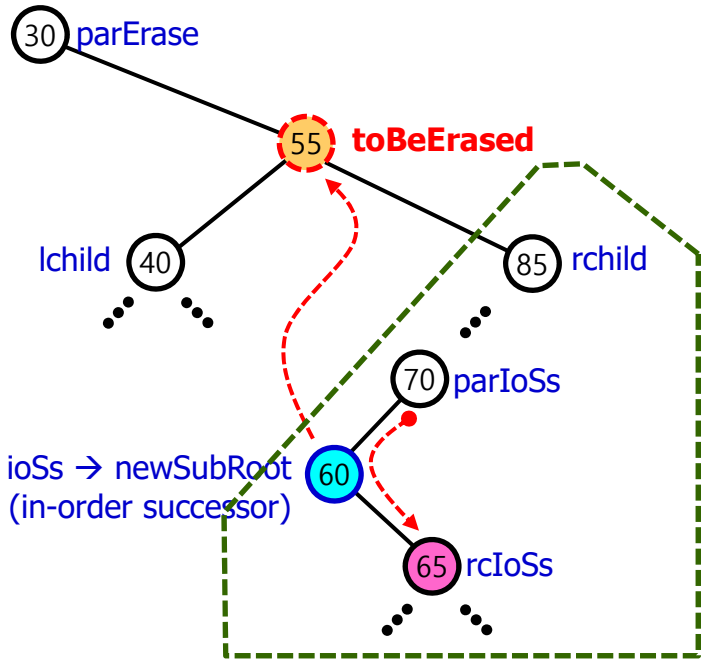


(b) After erase and substitution with
in-order predecessor

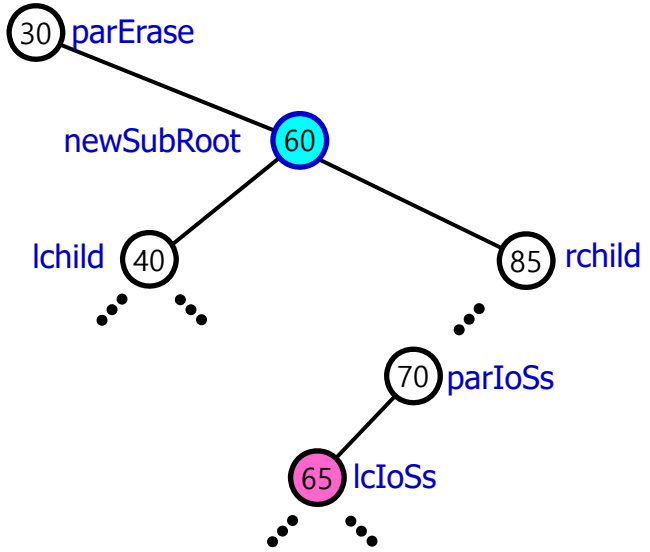
Balance를 고려한 eraseBSTN() (2)

◆ 오른쪽 서브 트리의 height가 더 높은 경우

- 오른쪽 서브 트리에서 가장 작은 값을 가진 in-order successor를 찾아 삭제 대상 노드 위치에 배치



(c) Before erase of a tree node (delete 55)



(d) After erase and substitution with in-order successor

균형잡힌 이진탐색트리의 응용

도서관에 소장된 서적 (book) 정보를 관리하기 위한 균형화 이진 탐색 트리

◆ 기본 기능

(1) 서적의 정보를 표현하기 위하여 다음과 같은 항목이 포함 되는 class Book을 구현하라: 도서명 (string), ISBN (international standard book number) (int), 저자명 (string), 출판연월일(class Date).

(2) class Book을 사용하여 15권의 서적 정보를 담는 배열 books[]를 준비하라.

(3) 서적의 모든 정보를 출력하는 fprintBook() 함수를 operator<<() 연산자 오버로딩 함수로 구현하라.

(4) class T_Entry<K, V>를 저장하는 균형화 이진 탐색 트리 (Balanced Binaray Search Tree, BBST)를 구현하라. 이 균형화 이진 탐색 트리는 지정된 key 값에 따라 정렬이 되도록 하며, 자동적으로 균형을 잡을 수 있도록 구현되어야 한다.

(5) 위에서 만든 도서관 서적 정보에서 도서명을 기준으로 정렬이 되도록 이진 탐색 트리 (BST_BK_title)를 구현하고, 이진 탐색 트리의 각 노드가 class Book 배열 books[]의 해당 서적 레코드를 가리키도록 하라. 서적명을 입력 받아 이진 탐색 트리를 검색하고, 그 서적의 모든 정보를 출력하는 프로그램을 작성하라.

(6) 위에서 만든 도서관 서적 정보에서 저자명을 기준으로 정렬이 되도록 이진 탐색 트리 (BST_BK_author)를 구현하고, 이진 탐색 트리의 각 노드가 book 객체 배열의 해당 서적 레코드를 가리키도록 하라. 이진 탐색 트리에서 동일 저자가 저술한 다수의 서적들이 함께 관리될 수 있도록 이진 탐색 트리를 구현하라.
동일 저자가 저술한 서적들인 경우, 먼저 입력된 순서로 저장될 수 있게 하라.
저자명을 입력 받아 이진 탐색 트리를 검색하고, 그 저자가 저술한 모든 서적들의 모든 정보를 출력하는 프로그램을 작성하라.

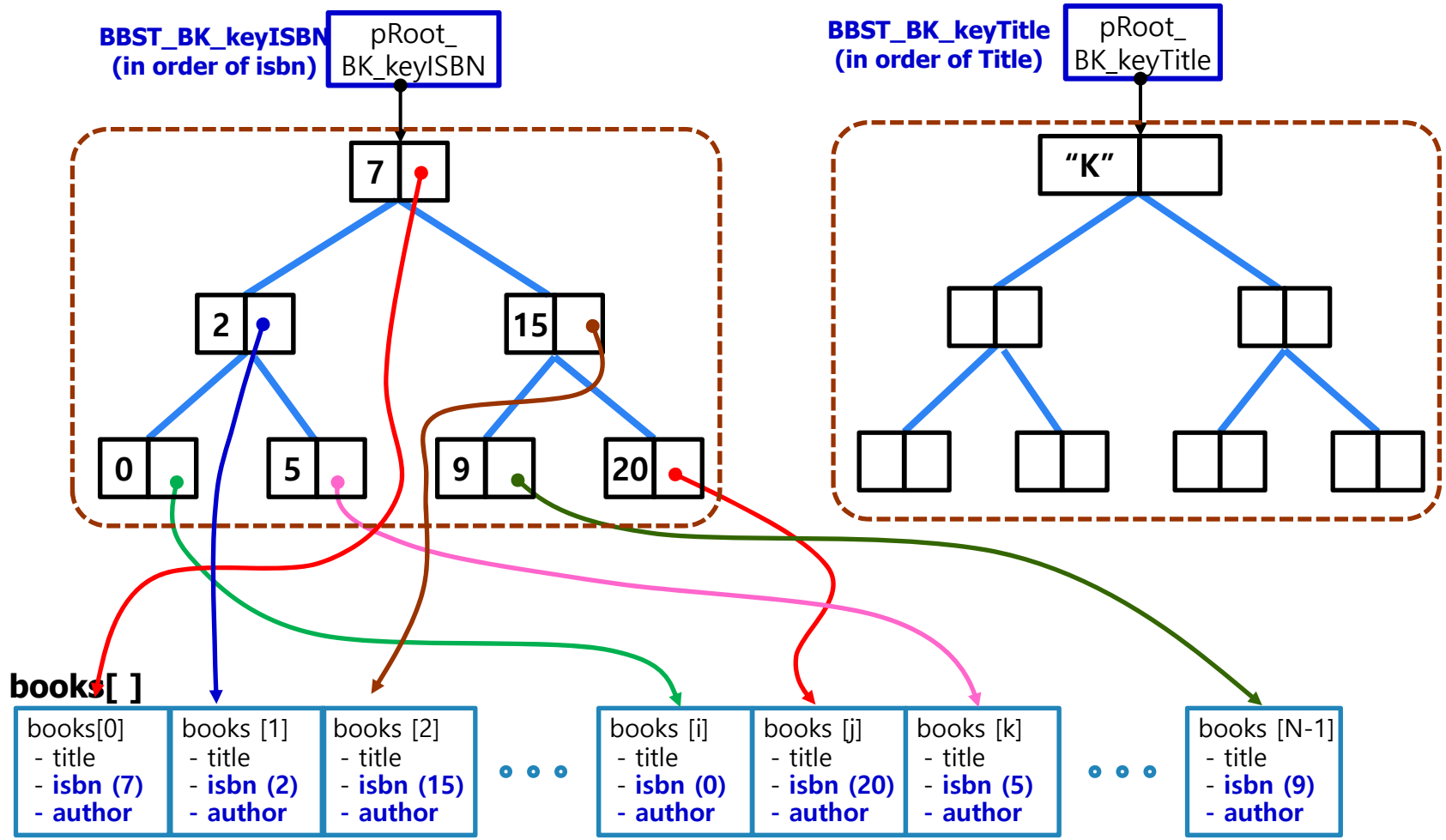
(7) 위에서 만든 도서관 서적 정보에서 출판연월일을 기준으로 정렬이 되도록 이진 탐색 트리 (BST_BK_publishDate)를 구현하고, 이진 탐색 트리의 각 노드가 객체 배열의 해당 서적 레코드를 가리키도록 하라. 출판연월일은 두개 입력받아, 지정된 두 날짜 사이에 출판되었던 서적들을 모두 출력하는 프로그램을 작성하라.

(8) 저자명과 출판연월일을 입력받아, 그 저자가 지정된 출판연월일 이후에 저술한 모든 서적을 출력하는 프로그램을 작성하라.

```
class Book
{
    friend ostream& operator<<(ostream& fout, Book& bk)
    {
            fout.setf(ios::left);
            fout << "["  << setw(8) << bk.title << ", " << setw(8) << bk.author;
            fout << ", " << bk.pubDate  << "]" ;
            return fout;
    }
public:
      Book(string bk_title, string bk_author, Date dt) :
      title(bk_title), author(bk_author), pubDate(dt){}
      string& getTitle() { return title; }
      string getAuthor() { return author; }
      Date getPubDate() { return pubDate; }
      void setTitle(string bk_title) { title = bk_title; }
      void setAuthor(string bk_author) { author = bk_author; }
private:
    string title;
    string author;
    Date pubDate;
};
```

Balanced Binary Search Tree for Books

◆ Total configuration



main()

```
/** main.cpp  (1) */
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include "T_BST.h"
#include "T_Entry.h"
#include "T_Array.h"
#include "Book.h"
#include "Date.h"

using namespace std;
#define NUM_BOOKS 15

void main()
{
	Book books[NUM_BOOKS] =
	{
		//Book(string bk_title, string bk_author, Date dt)
		Book(string("Book_01"), string("Kim"), Date(2020, 1, 1)),
		Book(string("Book_02"), string("Kim"), Date(2010, 1, 1)),
		Book(string("Book_03"), string("Kim"), Date(2013, 1, 1)),
		Book(string("Book_04"), string("Lee"), Date(2011, 1, 1)),
		Book(string("Book_05"), string("Hwang"), Date(2001, 1, 1)),
		Book(string("Book_06"), string("Choi"), Date(2003, 1, 1)),
		Book(string("Book_07"), string("Park"), Date(2009, 1, 1)),
		Book(string("Book_08"), string("Brown"), Date(2012, 1, 1)),
```

```
/** main.cpp  (2) */

            Book(string("Book_09"), string("Alpha"), Date(1980, 1, 1)),
            Book(string("Book_10"), string("Chalie"), Date(1970, 1, 1)),
            Book(string("Book_11"), string("Tango"), Date(1985, 1, 1)),
            Book(string("Book_12"), string("Yankee"), Date(1977, 1, 1)),
            Book(string("Book_13"), string("Zulu"), Date(2018, 1, 1)),
            Book(string("Book_14"), string("Foxtrot"), Date(2015, 1, 1)),
            Book(string("Book_15"), string("Delta"), Date(2000, 1, 1)),
    };

    ofstream fout("output.txt");
    if (fout.fail())
    {
        cout << "Fail to create output.txt for results !!" << endl;
        exit;
    }


    fout << "Input books[] array : " << endl;
    for (int i = 0; i < NUM_BOOKS; i++)
    {
        fout << books[i] << endl;
    }
    fout << endl;
```

```
/** main.cpp  (3) */

    fout << endl << "Balanced Binary Search Tree (BBST) with key book-title" << endl;
    T_Entry<string, Book*> entry_title_pBK;
    T_BST<string, Book*> BBST_BK_keyTitle("BBST_BK_keyTitle");
    T_BSTN<string, Book*> *pRoot, **ppBBST_BK_root;
    ppBBST_BK_root = BBST_BK_keyTitle.getRootAddr();
    for (int i = 0; i < NUM_BOOKS; i++)
    {
        entry_title_pBK.setKey(books[i].getTitle());
        entry_title_pBK.setValue(&books[i]);
        //fout << "Insert inOrder (" << setw(3) << books[i] << ") into " << BBST_BK_keyTitle.getName()
        << endl;
        BBST_BK_keyTitle.insertAndRebalance(entry_title_pBK);
    }
    fout << "\nEntries in " << BBST_BK_keyTitle.getName() << " (in order of Book Title) : " << endl;
    //BBST_BK_keyTitle.fprint_inOrder(fout);
    BBST_BK_keyTitle.fprint_with_Depth(fout);
```

```
/** main.cpp  (4) */

    fout << endl << "Balanced Binary Search Tree (BBST) with key book-author" << endl;
    T_Entry<string, Book*> entry_Author_pBK;
    T_BST<string, Book*> BBST_BK_keyAuthor("BBST_BK_keyAuthor");
    T_BSTN<string, Book*> **ppRoot_BBST_BK_keyAuthor, *pBBST_BK_keyAuthor;
    ppRoot_BBST_BK_keyAuthor = BBST_BK_keyAuthor.getRootAddr();
    for (int i = 0; i < NUM_BOOKS; i++)
    {
        entry_Author_pBK.setKey(books[i].getAuthor());
        entry_Author_pBK.setValue(&books[i]);
        //fout << "Insert inOrder (" << setw(3) << books[i] << ") into " << BBST_BK_keyTitle.getName()
        << endl;
        BBST_BK_keyAuthor.insertAndRebalance(entry_Author_pBK);
    }
    fout << "\nEntries in " << BBST_BK_keyAuthor.getName() << " (in order of Book Author) : " << endl;
    //BBST_BK_keyAuthor.fprint_inOrder(fout);
    BBST_BK_keyAuthor.fprint_with_Depth(fout);
```

```
/** main.cpp  (5) */

    // Testing Search on Binary Search Tree
    string author = books[0].getAuthor();
    Date d1, d2;
    Book *pBk;
    T_Array<Book *> array_pBook(1, string("Array_Book"));
    d1.setDate(2010, 1, 1);
    d2.setDate(2015, 12, 31);
    pBBST_BK_keyAuthor = BBST_BK_keyAuthor.searchBSTN(author);
    BBST_BK_keyAuthor.traversal_inOrder(pBBST_BK_keyAuthor, array_pBook);
    fout << "Books of author (" << author << ") published during " << d1 << " ~ " <<d2 << ":" << endl;
    for (int i = 0; i < array_pBook.size(); i++)
    {
        if (array_pBook[i]->getAuthor() == author)
        {
        pBk = array_pBook[i];
        if ((pBk->getPubDate() >= d1) && (pBk->getPubDate() <= d2))
        fout << *(array_pBook[i]) << endl;
        }
    }
```

```
/** main.cpp  (6) */

    fout << endl << "Balanced Binary Search Tree (BBST) with key publication-date" << endl;
    T_Entry<Date, Book*> entry_PubDate_pBK;
    T_BST<Date, Book*> BBST_BK_keyPubDate("BBST_BK_keyPubDate");
    T_BSTN<Date, Book*> **ppRoot_BBST_BK_keyPubDate;
    ppRoot_BBST_BK_keyPubDate = BBST_BK_keyPubDate.getRootAddr();
    for (int i = 0; i < NUM_BOOKS; i++)
    {
        entry_PubDate_pBK.setKey(books[i].getPubDate());
        entry_PubDate_pBK.setValue(&books[i]);
        //fout << "Insert inOrder (" << setw(3) << books[i] << ") into " << BBST_BK_keyTitle.getName()
        << endl;
        BBST_BK_keyPubDate.insertAndRebalance(entry_PubDate_pBK);
    }
    fout << "\nEntries in " << BBST_BK_keyPubDate.getName() << " (in order of Book Publication
    Date) : " << endl;
    //BBST_BK_keyPubDate.fprint_inOrder(fout);
    BBST_BK_keyPubDate.fprint_with_Depth(fout);
```

```
/** main.cpp  (7) */

      fout << "\nRemoving the root entry in sequence ..." << endl;
      for (int i = 0; i < NUM_BOOKS; i++)
      {
            pRoot = BBST_BK_keyTitle.getRoot();
            entry_title_pBK = pRoot->getEntry();
            fout << "\nremove " << entry_title_pBK << endl;
            BBST_BK_keyTitle.eraseBSTN(&pRoot);
            BBST_BK_keyTitle.fprint_with_Depth(fout);
      }

      fout << "\nClearing BBST_BKs . . . " << endl;
      BBST_BK_keyTitle.clear();
      BBST_BK_keyAuthor.clear();
      BBST_BK_keyPubDate.clear();
      fout << "All BBST_BKs cleared !! " << endl;

      fout.close();
}
```

```
Input books[] array :
[Book_01 , Kim     , (2020.1 .1 ) ]
[Book_02 , Kim     , (2010.1 .1 ) ]
[Book_03 , Kim     , (2013.1 .1 ) ]
[Book_04 , Lee     , (2011.1 .1 ) ]
[Book_05 , Hwang   , (2001.1 .1 ) ]
[Book_06 , Choi    , (2003.1 .1 ) ]
[Book_07 , Park    , (2009.1 .1 ) ]
[Book_08 , Brown   , (2012.1 .1 ) ]
[Book_09 , Alpha   , (1980.1 .1 ) ]
[Book_10 , Chalie  , (1970.1 .1 ) ]
[Book_11 , Tango   , (1985.1 .1 ) ]
[Book_12 , Yankee  , (1977.1 .1 ) ]
[Book_13 , Zulu    , (2018.1 .1 ) ]
[Book_14 , Foxtrot , (2015.1 .1 ) ]
[Book_15 , Delta   , (2000.1 .1 ) ]


Balanced Binary Search Tree (BBST) with key book-title

Entries in BBST_BK_keyTitle (in order of Book Title) :
BBST_BK_keyTitle : current size (15)
            (Book_15 : [Book_15 , Delta   , (2000.1 .1 ) ])
        (Book_14 : [Book_14 , Foxtrot , (2015.1 .1 ) ])
            (Book_13 : [Book_13 , Zulu    , (2018.1 .1 ) ])
    (Book_12 : [Book_12 , Yankee  , (1977.1 .1 ) ])
            (Book_11 : [Book_11 , Tango   , (1985.1 .1 ) ])
        (Book_10 : [Book_10 , Chalie  , (1970.1 .1 ) ])
            (Book_09 : [Book_09 , Alpha   , (1980.1 .1 ) ])
(Book_08 : [Book_08 , Brown   , (2012.1 .1 ) ])
            (Book_07 : [Book_07 , Park    , (2009.1 .1 ) ])
        (Book_06 : [Book_06 , Choi    , (2003.1 .1 ) ])
            (Book_05 : [Book_05 , Hwang   , (2001.1 .1 ) ])
    (Book_04 : [Book_04 , Lee     , (2011.1 .1 ) ])
            (Book_03 : [Book_03 , Kim     , (2013.1 .1 ) ])
        (Book_02 : [Book_02 , Kim     , (2010.1 .1 ) ])
            (Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])
```

```
Balanced Binary Search Tree (BBST) with key book-author

Entries in BBST_BK_keyAuthor (in order of Book Author) :
BBST_BK_keyAuthor : current size (15)
            (Zulu    : [Book_13 , Zulu    , (2018.1 .1 ) ])
        (Yankee  : [Book_12 , Yankee  , (1977.1 .1 ) ])
    (Tango   : [Book_11 , Tango   , (1985.1 .1 ) ])
            (Park    : [Book_07 , Park    , (2009.1 .1 ) ])
        (Lee     : [Book_04 , Lee     , (2011.1 .1 ) ])
            (Kim     : [Book_03 , Kim     , (2013.1 .1 ) ])
(Kim     : [Book_02 , Kim     , (2010.1 .1 ) ])
            (Kim     : [Book_01 , Kim     , (2020.1 .1 ) ])
        (Hwang   : [Book_05 , Hwang   , (2001.1 .1 ) ])
            (Foxtrot : [Book_14 , Foxtrot , (2015.1 .1 ) ])
                (Delta   : [Book_15 , Delta   , (2000.1 .1 ) ])
    (Choi    : [Book_06 , Choi    , (2003.1 .1 ) ])
            (Chalie  : [Book_10 , Chalie  , (1970.1 .1 ) ])
        (Brown   : [Book_08 , Brown   , (2012.1 .1 ) ])
            (Alpha   : [Book_09 , Alpha   , (1980.1 .1 ) ])
Books of author (Kim) published during (2010.1 .1 )  ~ (2015.12.31) :
[Book_02 , Kim     , (2010.1 .1 ) ]
[Book_03 , Kim     , (2013.1 .1 ) ]

Balanced Binary Search Tree (BBST) with key publication-date

Entries in BBST_BK_keyPubDate (in order of Book Publication Date) :
BBST_BK_keyPubDate : current size (15)
            ((       2020.1 .1 ) : [Book_01 , Kim     , (2020.1 .1 ) ])
        ((       2018.1 .1 ) : [Book_13 , Zulu    , (2018.1 .1 ) ])
            ((       2015.1 .1 ) : [Book_14 , Foxtrot , (2015.1 .1 ) ])
    ((       2013.1 .1 ) : [Book_03 , Kim     , (2013.1 .1 ) ])
            ((       2012.1 .1 ) : [Book_08 , Brown   , (2012.1 .1 ) ])
        ((       2011.1 .1 ) : [Book_04 , Lee     , (2011.1 .1 ) ])
((       2010.1 .1 ) : [Book_02 , Kim     , (2010.1 .1 ) ])
            ((       2009.1 .1 ) : [Book_07 , Park    , (2009.1 .1 ) ])
        ((       2003.1 .1 ) : [Book_06 , Choi    , (2003.1 .1 ) ])
    ((       2001.1 .1 ) : [Book_05 , Hwang   , (2001.1 .1 ) ])
                ((       2000.1 .1 ) : [Book_15 , Delta   , (2000.1 .1 ) ])
            ((       1985.1 .1 ) : [Book_11 , Tango   , (1985.1 .1 ) ])
        ((       1980.1 .1 ) : [Book_09 , Alpha   , (1980.1 .1 ) ])
                ((       1977.1 .1 ) : [Book_12 , Yankee  , (1977.1 .1 ) ])
            ((       1970.1 .1 ) : [Book_10 , Chalie  , (1970.1 .1 ) ])
```

```
Removing the root entry in sequence ...

remove (Book_08 : [Book_08 , Brown   , (2012.1 .1 ) ])
BBST_BK_keyTitle : current size (14)
            (Book_15 : [Book_15 , Delta   , (2000.1 .1 ) ])
        (Book_14 : [Book_14 , Foxtrot , (2015.1 .1 ) ])
            (Book_13 : [Book_13 , Zulu    , (2018.1 .1 ) ])
    (Book_12 : [Book_12 , Yankee  , (1977.1 .1 ) ])
            (Book_11 : [Book_11 , Tango   , (1985.1 .1 ) ])
        (Book_10 : [Book_10 , Chalie  , (1970.1 .1 ) ])
(Book_09 : [Book_09 , Alpha   , (1980.1 .1 ) ])
            (Book_07 : [Book_07 , Park    , (2009.1 .1 ) ])
        (Book_06 : [Book_06 , Choi    , (2003.1 .1 ) ])
            (Book_05 : [Book_05 , Hwang   , (2001.1 .1 ) ])
    (Book_04 : [Book_04 , Lee     , (2011.1 .1 ) ])
            (Book_03 : [Book_03 , Kim     , (2013.1 .1 ) ])
        (Book_02 : [Book_02 , Kim     , (2010.1 .1 ) ])
            (Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])

remove (Book_09 : [Book_09 , Alpha   , (1980.1 .1 ) ])
BBST_BK_keyTitle : current size (13)
            (Book_15 : [Book_15 , Delta   , (2000.1 .1 ) ])
        (Book_14 : [Book_14 , Foxtrot , (2015.1 .1 ) ])
            (Book_13 : [Book_13 , Zulu    , (2018.1 .1 ) ])
    (Book_12 : [Book_12 , Yankee  , (1977.1 .1 ) ])
        (Book_11 : [Book_11 , Tango   , (1985.1 .1 ) ])
(Book_10 : [Book_10 , Chalie  , (1970.1 .1 ) ])
            (Book_07 : [Book_07 , Park    , (2009.1 .1 ) ])
        (Book_06 : [Book_06 , Choi    , (2003.1 .1 ) ])
            (Book_05 : [Book_05 , Hwang   , (2001.1 .1 ) ])
    (Book_04 : [Book_04 , Lee     , (2011.1 .1 ) ])
            (Book_03 : [Book_03 , Kim     , (2013.1 .1 ) ])
        (Book_02 : [Book_02 , Kim     , (2010.1 .1 ) ])
            (Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])

remove (Book_13 : [Book_13 , Zulu    , (2018.1 .1 ) ])
BBST_BK_keyTitle : current size (5)
    (Book_15 : [Book_15 , Delta   , (2000.1 .1 ) ])
(Book_14 : [Book_14 , Foxtrot , (2015.1 .1 ) ])
        (Book_03 : [Book_03 , Kim     , (2013.1 .1 ) ])
    (Book_02 : [Book_02 , Kim     , (2010.1 .1 ) ])
        (Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])

remove (Book_14 : [Book_14 , Foxtrot , (2015.1 .1 ) ])
BBST_BK_keyTitle : current size (4)
    (Book_15 : [Book_15 , Delta   , (2000.1 .1 ) ])
(Book_03 : [Book_03 , Kim     , (2013.1 .1 ) ])
    (Book_02 : [Book_02 , Kim     , (2010.1 .1 ) ])
        (Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])

remove (Book_03 : [Book_03 , Kim     , (2013.1 .1 ) ])
BBST_BK_keyTitle : current size (3)
    (Book_15 : [Book_15 , Delta   , (2000.1 .1 ) ])
(Book_02 : [Book_02 , Kim     , (2010.1 .1 ) ])
    (Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])

remove (Book_02 : [Book_02 , Kim     , (2010.1 .1 ) ])
BBST_BK_keyTitle : current size (2)
(Book_15 : [Book_15 , Delta   , (2000.1 .1 ) ])
    (Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])

remove (Book_15 : [Book_15 , Delta   , (2000.1 .1 ) ])
BBST_BK_keyTitle : current size (1)
(Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])

remove (Book_01 : [Book_01 , Kim     , (2020.1 .1 ) ])
BBST_BK_keyTitle is empty now !!

Clearing BBST_BKs . . .
All BBST_BKs cleared !!
```

Oral Test 10

Oral Test 10

10.1 이진 탐색 트리에서의 새로운 트리노드 입력에서 재 균형화 (re-balancing)를 실행하지 않는 경우 어떤 문제점이 있는가에 대하여 설명하라.

<Key Points>

(1) 탐색, 삽입, 삭제 동작의 성능 관점에서 비교하고, Big Oh 방식으로 표시 할 것

항목	균형화를 하지 않을 때	균형화를 할 때
탐색 (search)	- 평균: - 최대:	- 평균: - 최대:
트리노드 삽입 (insert)	- 평균: - 최대:	- 평균: - 최대:
트리노드 삭제 (erase)	- 평균: - 최대:	- 평균: - 최대:

10.2 이진 탐색 트리에서의 트리노드 삭제에서 재 균형화 (re-balancing)를 실행하는 세부 기능에 대하여 설명하라. 트리노드들의 관계를 그림으로 표현하고, psedo code로 설명하라.

<Key Points>

(1) 삭제 대상 노드의 자식이 없는 경우
(2) 삭제 대상 노드의 자식이 하나만 있는 경우
(3) 삭제 대상 노드의 자식이 모두 있는 경우, 노드들의 서브트리 height 계산 및 height-difference 계산
(4) 왼쪽 자식 노드의 height가 더 높은 경우, in-order predecessor (ioPd) 찾기 및 노드 재배치
(5) 오른쪽 자식 노드의 height가 같거나 더 높은 경우, in-order successor(ioSs) 찾기 및 노드 재배치

Oral Test 10

10.3 이진 탐색 트리에서의 새로운 트리노드 입력에서 재 균형화를 실행하는 세부 절차인 rotate_LL, rotate_RR 에 대하여 설명하라. 트리노드들의 관계를 그림으로 표현하고 (rotate를 수행하기 전과, 수행한 후 비교), psedo code로 설명하라.

<Key Points>

(1) 왼쪽으로 연속 편중된 경우에 대한 rotate_LL()

(2) 오른쪽으로 연속 편중된 경우에 대한 rotate_RR()

10.4 이진 탐색 트리에서의 새로운 트리노드 입력에서 재 균형화를 실행하는 세부 절차인 rotate_LR, rotate_RL 에 대하여 설명하라. 트리노드들의 관계를 그림으로 표현하고 표현하고 (rotate를 수행하기 전과, 수행한 후 비교), psedo code로 설명하라.

<Key Points>

(1) 왼쪽으로 편중되어 있고, 내부에서 오른쪽으로 편중되어 있는 경우에 대한 rotate_LR()

(2) 오른쪽으로 편중되어 있고, 내부에서 왼쪽으로 편중되어 있는 경우에 대한 rotate_RL()