[제 14 주 실습]

BTree

강 지 훈
jhkang@cnu.ac.kr

충남대학교 컴퓨터공학과

[문제 14]
BTREE

문제 개요

- BTree를 실제로 구현하여 본다.

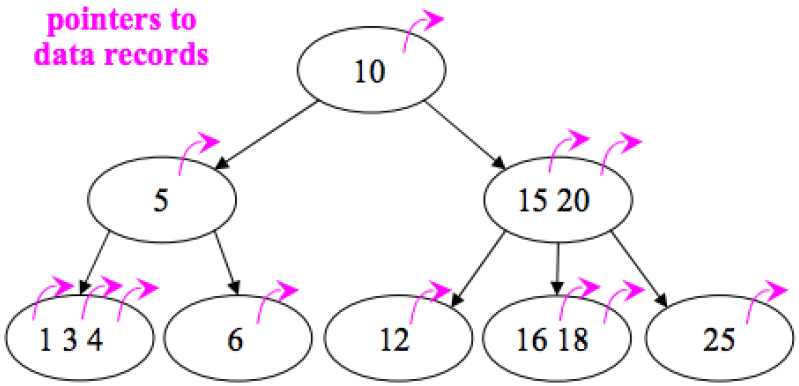
B-Tree

- 특징
  - 각 노드가 여러 개의 엔트리(value)를 가지고 엔트리 + 1 개의 자식 노드를 가질 수 있다.
  - 모든 leaf 노드의 depth는 동일하다.
  - 분할(split) 및 병합(merge) 작업으로 인하여 항상 트리의 균형을 유지한다.
  - 새로운 노드의 삽입/삭제는 leaf node에서만 수행한다.
- 장점
  - 노드의 삽입/삭제 후에도 균형 트리 유지하므로 균등한 응답속도를 보장한다.
- 단점
  - 삽입/삭제 후에 균형을 맞추기 위하여 복잡한 연산이 수행된다. 또한 순차 탐색 시에 Inorder 순회 수행으로 비효율적이다.

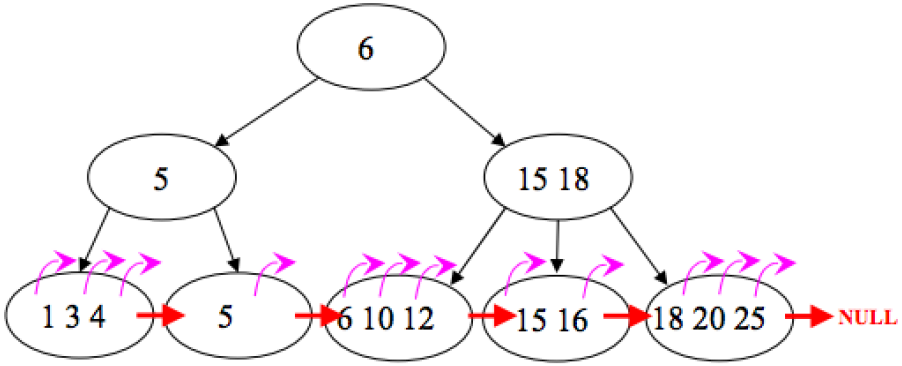
B-Tree

- A B⁺-tree can be viewed as a B-tree in which each node contains only keys (not pairs), and to which an additional level is added at the bottom with linked leaves

B-tree of order 4



B⁺-tree of order 4



❑출력화면

```
<<프로그램을 시작합니다>>
수행 하려고 하는 메뉴를 선택하세요
(add : 1, remove : 2, search : 3, debug : 4, exit : -1) : 1
Key를 입력하세요 : 1
Value를 입력하세요 : 1
수행 하려고 하는 메뉴를 선택하세요
(add : 1, remove : 2, search : 3, debug : 4, exit : -1) : 1
Key를 입력하세요 : 3
Value를 입력하세요 : 3
수행 하려고 하는 메뉴를 선택하세요
(add : 1, remove : 2, search : 3, debug : 4, exit : -1) : 1
Key를 입력하세요 : 5
Value를 입력하세요 : 5
수행 하려고 하는 메뉴를 선택하세요
(add : 1, remove : 2, search : 3, debug : 4, exit : -1) : 1
Key를 입력하세요 : 7
Value를 입력하세요 : 7
수행 하려고 하는 메뉴를 선택하세요
(add : 1, remove : 2, search : 3, debug : 4, exit : -1) : 1
Key를 입력하세요 : 2
Value를 입력하세요 : 2
수행 하려고 하는 메뉴를 선택하세요
(add : 1, remove : 2, search : 3, debug : 4, exit : -1) : 1
Key를 입력하세요 : 4
Value를 입력하세요 : 4
수행 하려고 하는 메뉴를 선택하세요
(add : 1, remove : 2, search : 3, debug : 4, exit : -1) : 4
1, 2, 3, 4, 5, 7,
수행 하려고 하는 메뉴를 선택하세요
(add : 1, remove : 2, search : 3, debug : 4, exit : -1) : 2
Key를 입력하세요 : 2
수행 하려고 하는 메뉴를 선택하세요
(add : 1, remove : 2, search : 3, debug : 4, exit : -1) : 4
1, 3, 4, 5, 7,
수행 하려고 하는 메뉴를 선택하세요
(add : 1, remove : 2, search : 3, debug : 4, exit : -1) : -1
```

이 과제에서 필요한 객체는?

- AppView
- AppController
  - BTree
- BTree
  - Node

AppControllor의 공개 함수는?

- 사용자에게 필요한 함수 (Public Functions)
  - public void run()

AppView의 공개 함수는?

- 사용자에게 필요한 함수 (Public Functions)
  - public AppView()
  - public void outputMsg(String aString)
  - public void outputMsg(BTree aTree)
  - public String inputString()
  - public int inputInt()

BTree의 공개 함수는?

- 사용자에게 필요한 함수 (Public Functions)
  - public BTree()
  - public String toString()
  - public void addKeyandObejct(int key, Object object)
  - public void removeObjectForKey(int key)
  - public Object search(int key)
  - InnerClass Node가 존재

Node의 공개 함수는?

- 사용자에게 필요한 함수 (Public Functions)
  - public int binarySearch(int key)
  - public boolean contains(int key)
  - public void remove(int index, int leftOrRightChild)
  - public void shiftRightByOne()
  - public int subtreeRootNodeIndex(int key)

Class “AppControllor“

AppControllor– 비공개 인스턴스 변수

```
public class AppControllor {
    private AppView _appView;
    private BTree _tree;
```

AppControllor 의 공개 함수 run()의 구현

```
public void run() throws Exception {
        _appView = new AppView();
        _tree = new BTree();
        int command;

        _appView.outputMsg(_appView.MSG_StartProgram);
        _appView.outputMsg(_appView.MSG_Menu);
        command = _appView.inputInt();
        while (command != -1) {
            Integer key, value;
            switch (command) {
            case 1:
                _appView.outputMsg(_appView.MSG_InputKey);
                key = _appView.inputInt();
                _appView.outputMsg(_appView.MSG_InputValue);
                value = _appView.inputInt();
                _tree.addKeyandObejct(key, value);
                break;
```

AppControllor 의 공개 함수 run()의 구현

```
            case 2:
                _appView.outputMsg(_appView.MSG_InputKey);
                key = _appView.inputInt();
                _tree.removeObjectForKey(key);
                break;
            case 3:
                _appView.outputMsg(_appView.MSG_InputKey);
                key = _appView.inputInt();
                _appView.outputMsg(key + _appView.MSG_Search +
_tree.search(key) + "₩n");
                break;
            case 4:
                _appView.outputMsg(_tree);
                break;
            }
            _appView.outputMsg(_appView.MSG_Menu);
            command = _appView.inputInt();
        }
    }
```

Class “AppView“

AppView – 비공개 인스턴스 변수

```
import java.util.Scanner;

public class AppView {
    private Scanner _scanner;
```

AppView의 Public Method

- AppView 의 Public Member의 선언
  - public String MSG_StartProgram = "<<프로그램을 시작합니다>>₩n";
  - public String MSG_EndProgram = "<<프로그램을 종료합니다>>₩n";
  - public String MSG_Menu = "수행 하려고 하는 메뉴를 선택하세요₩n(add : 1, remove : 2, search : 3, debug : 4, exit : -1) : ";
  - public String MSG_InputKey = "Key를 입력하세요 : ";
  - public String MSG_InputValue = "Value를 입력하세요 : ";
  - public String MSG_Remove = " 삭제된 Value : ";
  - public String MSG_Search = " 검색된 Value : ";
  - public String MSG_Debug = "[DEBUG] PRINT AVL TREE₩n";
  - public String MSG_Error = "[Error]₩n";

AppView의 Public Method

- AppView 의 Public Member function의 사용법과 구현
  - public AppView()
    - 생성자
  - public void outputMsg(String aString)
    - aString을 출력
  - public void outputMsg(BTree aTree)
    - BTree를 출력
  - public String inputString()
    - String을 하나 입력 받아 반환
  - public int inputInt()
    - Int를 하나 입력 받아 반환

Class “BTree”

비공개 인스턴스 변수

```
public class BTree {
    private static final int T = 2;
    private Node _root;
    private static final int LEFT_NODE = 0;
    private static final int RIGHT_NODE = 1;
```

Public Method

- BTree의 Public Member function의 사용법
  - public BTree()
    - BTree 생성자
  - public String toString()
    - BTree를 출력
    - 자바에서 생성한 class를 String 형태로 출력 하기 위하여 사용하는 메소드
      - System.out.print(BTree)
  - public void addKeyandObejct(int key, Object object)
    - Key와 Object를 삽입
  - public void removeObjectForKey(int key)
    - Key를 삭제
  - public Object search(int key)
    - Key로 Object를 찾음

Public Method

- BTree의 Private Member function의 사용법
  - private void addIntoNonFullNode(Node node, int key, Object object)
    - 가득 차지 않은 node에 값을 삽입
  - private Object search(Node node, int key)
  - private void remove(Node node, int key)
    - 삭제를 위한 aKey가 있는 aNode를 찾아서 삭제
  - private int merge(Node dstNode, Node srcNode)
    - 두 개의 Node를 maerge
  - private void moveKey(Node srcNode, int srcKeyIndex, int childIndex, Node dstNode,int medianKeyIndex)
    - Key를 srcNode의 index부터 dstNode의 medianKeyIndex로 이동
  - private void splitChildNode(Node parentNode, int i, Node node)
    - 자식 노드를 나눔
  - private boolean update(Node node, int key, Object object)
    - key를 가진 노드를 새로운 object로 update
  - private String printBTree(Node node)
    - BTree 출력을 하기 위한 String 형태로 변환

Public Method

- BTree의 Public Member function의 구현
  - public BTree()
    - _root를 새로 생성
    - _root의 _isLeafNode를 true로 초기화
  - public String toString()
    - printBTree를 _root부터 출력 할 수 있도록 호출

Public Method

- BTree의 Public Member function의 구현

```
public void addKeyandObejct(int key, Object object) {
    Node rootNode = _root;
    if (!update(_root, key, object)) {
        if (rootNode._sizeOfKey == (2 * T - 1)) {
            Node newNode = new Node();
            _root = newNode;
            newNode._isLeafNode = false;
            _root._childNodes[0] = rootNode;
            // Split rootNode and move its median (middle) key up into
            // newRootNode.
            splitChildNode(newNode, 0, rootNode);
            // add the key into the B-Tree with root newRootNode.
            addIntoNonFullNode(newNode, key, object);
        } else {
            // add the key into the B-Tree with root rootNode.
            addIntoNonFullNode(rootNode, key, object);
        }
    }
}
```

Public Method

- BTree의 Public Member function의 구현
  - public void removeObjectForKey(int key)
    - Remove함수를 _root와 key를 인자로 하여 호출
  - public Object search(int key)
    - search함수를 _root와 key를 인자로 하여 호출

Private Method

- BTree의 Private Member function의 구현

```
private void addIntoNonFullNode(Node node, int key, Object object) {
    int i = node._sizeOfKey - 1;
    if (node._isLeafNode) {
        while (i >= 0 && key < node._keys[i]) {
            node._keys[i + 1] = node._keys[i];
            node._Objs[i + 1] = node._Objs[i];
            i--;
        }
        i++;
        node._keys[i] = key;
        node._Objs[i] = object;
        node._sizeOfKey++;
    } else {
        while (i >= 0 && key < node._keys[i]) {
            i--;
        }
        i++;
        if (node._childNodes[i]._sizeOfKey == (2 * T - 1)) {
            splitChildNode(node, i, node._childNodes[i]);
            if (key > node._keys[i]) {
                i++;
            }
        }
        addIntoNonFullNode(node._childNodes[i], key, object);
    }
}
```

Private Method

- BTree의 Private Member function의 구현

```
private Object search(Node node, int key) {
    int i = 0;
    while (i < node._sizeOfKey && key > node._keys[i]) {
        i++;
    }
    if (i < node._sizeOfKey && key == node._keys[i]) {
        return node._Objs[i];
    }
    if (node._isLeafNode) {
        return null;
    } else {
        return search(node._childNodes[i], key);
    }
}
```

Private Method

- BTree의 Private Member function의 구현

```
private int merge(Node dstNode, Node srcNode) {
    int medianKeyIndex;
    if (srcNode._keys[0] < dstNode._keys[dstNode._sizeOfKey - 1]) {
        int i;
        if (!dstNode._isLeafNode) {
            dstNode._childNodes[srcNode._sizeOfKey + dstNode._sizeOfKey + 1] =
                    dstNode._childNodes[dstNode._sizeOfKey];
        }
        for (i = dstNode._sizeOfKey; i > 0; i--) {
            dstNode._keys[srcNode._sizeOfKey + i] = dstNode._keys[i - 1];
            dstNode._Objs[srcNode._sizeOfKey + i] = dstNode._Objs[i - 1];
            if (!dstNode._isLeafNode) {
                dstNode._childNodes[srcNode._sizeOfKey + i] = dstNode._childNodes[i - 1];
            }
        }
        medianKeyIndex = srcNode._sizeOfKey;
        dstNode._keys[medianKeyIndex] = 0;
        dstNode._Objs[medianKeyIndex] = null;

        for (i = 0; i < srcNode._sizeOfKey; i++) {
            dstNode._keys[i] = srcNode._keys[i];
            dstNode._Objs[i] = srcNode._Objs[i];
            if (!srcNode._isLeafNode) {
                dstNode._childNodes[i] = srcNode._childNodes[i];
            }
        }
        if (!srcNode._isLeafNode) {
            dstNode._childNodes[i] = srcNode._childNodes[i];
        }
    }
```

Private Method

- BTree의 Private Member function의 구현

```
        else {
            medianKeyIndex = dstNode._sizeOfKey;
            dstNode._keys[medianKeyIndex] = 0;
            dstNode._Objs[medianKeyIndex] = null;

            int offset = medianKeyIndex + 1;
            int i;
            for (i = 0; i < srcNode._sizeOfKey; i++) {
                dstNode._keys[offset + i] = srcNode._keys[i];
                dstNode._Objs[offset + i] = srcNode._Objs[i];
                if (!srcNode._isLeafNode) {
                    dstNode._childNodes[offset + i] = srcNode._childNodes[i];
                }
            }
            if (!srcNode._isLeafNode) {
                dstNode._childNodes[offset + i] = srcNode._childNodes[i];
            }
        }
        dstNode._sizeOfKey += srcNode._sizeOfKey;
        return medianKeyIndex;
    }
```

Private Method

- BTree의 Private Member function의 구현

```
private void moveKey(Node srcNode, int srcKeyIndex, int childIndex, Node dstNode,
        int medianKeyIndex) {
    dstNode._keys[medianKeyIndex] = srcNode._keys[srcKeyIndex];
    dstNode._Objs[medianKeyIndex] = srcNode._Objs[srcKeyIndex];
    dstNode._sizeOfKey++;

    srcNode.remove(srcKeyIndex, childIndex);

    if (srcNode == _root && srcNode._sizeOfKey == 0) {
        _root = dstNode;
    }
}
```

Private Method

- BTree의 Private Member function의 구현

```
private void splitChildNode(Node parentNode, int i, Node node) {
    Node newNode = new Node();
    newNode._isLeafNode = node._isLeafNode;
    newNode._sizeOfKey = T - 1;
    for (int j = 0; j < T - 1; j++) {
        newNode._keys[j] = node._keys[j + T];
        newNode._Objs[j] = node._Objs[j + T];
    }
    if (!newNode._isLeafNode) {
        for (int j = 0; j < T; j++) {
            newNode._childNodes[j] = node._childNodes[j + T];
        }
        for (int j = T; j <= node._sizeOfKey; j++) {
            node._childNodes[j] = null;
        }
    }
    for (int j = T; j < node._sizeOfKey; j++) {
        node._keys[j] = 0;
        node._Objs[j] = null;
    }
    node._sizeOfKey = T - 1;
```

Private Method

- BTree의 Private Member function의 구현

```
    for (int j = parentNode._sizeOfKey; j >= i + 1; j--) {
        parentNode._childNodes[j + 1] = parentNode._childNodes[j];
    }
    parentNode._childNodes[i + 1] = newNode;
    for (int j = parentNode._sizeOfKey - 1; j >= i; j--) {
        parentNode._keys[j + 1] = parentNode._keys[j];
        parentNode._Objs[j + 1] = parentNode._Objs[j];
    }
    parentNode._keys[i] = node._keys[T - 1];
    parentNode._Objs[i] = node._Objs[T - 1];
    node._keys[T - 1] = 0;
    node._Objs[T - 1] = null;
    parentNode._sizeOfKey++;
}
```

Private Method

- BTree의 Private Member function의 구현

```
private boolean update(Node node, int key, Object object) {
    while (node != null) {
        int i = 0;
        while (i < node._sizeOfKey && key > node._keys[i]) {
            i++;
        }
        if (i < node._sizeOfKey && key == node._keys[i]) {
            node._Objs[i] = object;
            return true;
        }
        if (node._isLeafNode) {
            return false;
        } else {
            node = node._childNodes[i];
        }
    }
    return false;
}
```

Private Method

- BTree의 Private Member function의 구현

```
private void remove(Node node, int key) {
    if (node._isLeafNode) {
        int i;
        if ((i = node.binarySearch(key)) != -1) {
            node.remove(i, LEFT_NODE);
        }
    } else {
        int i;
        if ((i = node.binarySearch(key)) != -1) {
            Node leftChildNode = node._childNodes[i];
            Node rightChildNode = node._childNodes[i + 1];
            if (leftChildNode._sizeOfKey >= T) {
                Node predecessorNode = leftChildNode;
                Node erasureNode = predecessorNode;
                while (!predecessorNode._isLeafNode) {
                    erasureNode = predecessorNode;
                    predecessorNode = predecessorNode._childNodes[node._sizeOfKey - 1];
                }
                node._keys[i] = predecessorNode._keys[predecessorNode._sizeOfKey - 1];
                node._Objs[i] = predecessorNode._Objs[predecessorNode._sizeOfKey - 1];
                remove(erasureNode, node._keys[i]);
            }
            else if (rightChildNode._sizeOfKey >= T) {
                Node successorNode = rightChildNode;
                Node erasureNode = successorNode;
                while (!successorNode._isLeafNode) {
                    erasureNode = successorNode;
                    successorNode = successorNode._childNodes[0];
                }
                node._keys[i] = successorNode._keys[0];
                node._Objs[i] = successorNode._Objs[0];
                remove(erasureNode, node._keys[i]);
```

Private Method

- BTree의 Private Member function의 구현

```
        } else {
            int medianKeyIndex = merge(leftChildNode,
                    rightChildNode);
            moveKey(node, i, RIGHT_NODE, leftChildNode, medianKeyIndex);
            remove(leftChildNode, key);
        }
    }
    else {
        i = node.subtreeRootNodeIndex(key);
        Node childNode = node._childNodes[i];

        if (childNode._sizeOfKey == T - 1) {
            Node leftChildSibling = (i - 1 >= 0) ? node._childNodes[i - 1]: null;
            Node rightChildSibling = (i + 1 <= node._sizeOfKey) ?
                    node._childNodes[i + 1]: null;
            if (leftChildSibling != null
                    && leftChildSibling._sizeOfKey >= T) {
                childNode.shiftRightByOne();
                childNode._keys[0] = node._keys[i - 1];
                childNode._Objs[0] = node._Objs[i - 1];
                if (!childNode._isLeafNode) {
                    childNode._childNodes[0] =
                            leftChildSibling._childNodes[leftChildSibling._sizeOfKey];
                }
                childNode._sizeOfKey++;
                node._keys[i - 1] =
                        leftChildSibling._keys[leftChildSibling._sizeOfKey - 1];
                node._Objs[i - 1] =
                        leftChildSibling._Objs[leftChildSibling._sizeOfKey - 1];
                leftChildSibling.remove(
                        leftChildSibling._sizeOfKey - 1, RIGHT_NODE);
            }
```

Private Method

- BTree의 Private Member function의 구현

```
                else if (rightChildSibling != null
                        && rightChildSibling._sizeOfKey >= T) {
                    childNode._keys[childNode._sizeOfKey] = node._keys[i];
                    childNode._Objs[childNode._sizeOfKey] = node._Objs[i];
                    if (!childNode._isLeafNode) {
                        childNode._childNodes[childNode._sizeOfKey + 1] = rightChildSibling._childNodes[0];
                    }
                    childNode._sizeOfKey++;
                    node._keys[i] = rightChildSibling._keys[0];
                    node._Objs[i] = rightChildSibling._Objs[0];
                    rightChildSibling.remove(0, LEFT_NODE);
                } else {
                    if (leftChildSibling != null) {
                        int medianKeyIndex = merge(childNode, leftChildSibling);
                        moveKey(node, i - 1, LEFT_NODE, childNode,
                                medianKeyIndex);
                    } else if (rightChildSibling != null) {
                        int medianKeyIndex = merge(childNode, rightChildSibling);
                        moveKey(node, i, RIGHT_NODE, childNode, medianKeyIndex);
                    }
                }
            }
            remove(childNode, key);
        }
    }
}
```

Private Method

- BTree의 Private Member function의 구현

```
private String printBTree(Node node) {
    String string = "";
    if (node != null) {
        if (node._isLeafNode) {
            for (int i = 0; i < node._sizeOfKey; i++) {
                string += node._Objs[i] + ", ";
            }
        } else {
            int i;
            for (i = 0; i < node._sizeOfKey; i++) {
                string += printBTree(node._childNodes[i]);
                string += node._Objs[i] + ", ";
            }
            string += printBTree(node._childNodes[i]);
        }
    }
    return string;
}
```

Inner Class “Node”

비공개 인스턴스 변수

```
public class BTree {
    ....
    class Node {
        private int _sizeOfKey = 0;
        private int[] _keys = new int[2 * T - 1];
        private Object[] _Objs = new Object[2 * T - 1];
        private Node[] _childNodes = new Node[2 * T];
        private boolean _isLeafNode;
```

Public Method

- BTreeNode의 Public Member function의 구현

```
public int binarySearch(int key) {
    int leftIndex = 0;
    int rightIndex = _sizeOfKey - 1;

    while (leftIndex <= rightIndex) {
        final int middleIndex = leftIndex + ((rightIndex - leftIndex) / 2);
        if (_keys[middleIndex] < key) {
            leftIndex = middleIndex + 1;
        } else if (_keys[middleIndex] > key) {
            rightIndex = middleIndex - 1;
        } else {
            return middleIndex;
        }
    }

    return -1;
}

public boolean contains(int key) {
    return binarySearch(key) != -1;
}
```

Public Method

- BTreeNode의 Public Member function의 구현

```
private void remove(int index, int leftOrRightChild) {
    if (index >= 0) {
        int i;
        for (i = index; i < _sizeOfKey - 1; i++) {
            _keys[i] = _keys[i + 1];
            _Objs[i] = _Objs[i + 1];
            if (!_isLeafNode) {
                if (i >= index + leftOrRightChild) {
                    _childNodes[i] = _childNodes[i + 1];
                }
            }
        }
        _keys[i] = 0;
        _Objs[i] = null;
        if (!_isLeafNode) {
            if (i >= index + leftOrRightChild) {
                _childNodes[i] = _childNodes[i + 1];
            }
            _childNodes[i + 1] = null;
        }
        _sizeOfKey--;
    }
}
```

Public Method

- BTreeNode의 Public Member function의 구현

```
private void shiftRightByOne() {
    if (!_isLeafNode) {
        _childNodes[_sizeOfKey + 1] = _childNodes[_sizeOfKey];
    }
    for (int i = _sizeOfKey - 1; i >= 0; i--) {
        _keys[i + 1] = _keys[i];
        _Objs[i + 1] = _Objs[i];
        if (!_isLeafNode) {
            _childNodes[i + 1] = _childNodes[i];
        }
    }
}

public int subtreeRootNodeIndex(int key) {
    for (int i = 0; i < _sizeOfKey; i++) {
        if (key < _keys[i]) {
            return i;
        }
    }
    return _sizeOfKey;
}
```

[문제 14] 요약

- BTree에 대하여 이해한다.
- 제공된 소스에 대하여 Btree를 이해한 내용을 바탕으로 주석을 달아 제출하시오.

- 각 요약에 대한 내용을 보고서에 작성하여 제출하세요.

과제 제출

과제 제출

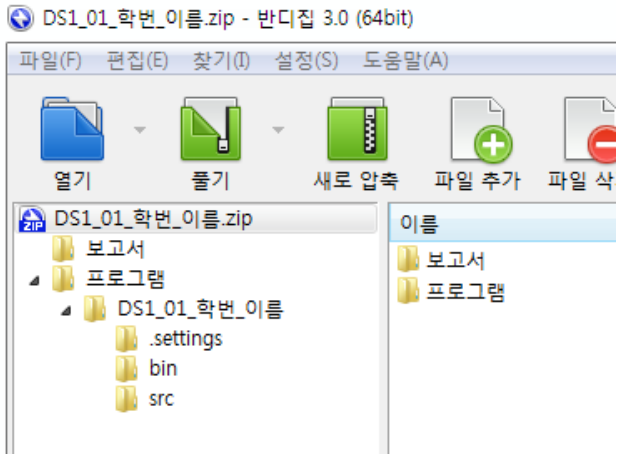
- pineai@cnu.ac.kr
  - 메일 제목 : [0X]DS2_14_학번_이름
    - 양식에 맞지 않는 메일 제목은 미제출로 간주됨
    - 앞의 0X는 분반명 (오전10시 : 00반 / 오후4시 : 01반)

- 제출 기한
  - 12월 15일(일) 23시59분까지
  - 시간 내 제출 엄수
  - 모든 실습 과제는 12월 22일까지 제출되어야함

  - 제출을 하지 않을 경우 0점 처리하고, 숙제를 50% 이상 제출하지 않으면 F 학점 처리하며, 2번 이상 제출하지 않으면 A 학점을 받을 수 없다.

과제 제출

- 파일 이름 작명 방법
  - DS2_14_학번_이름.zip
  - 폴더의 구성
    - DS2_14_학번_이름
      - 프로그램
        - 프로젝트 폴더 / 소스
        - 메인 클래스 이름 : DS2_14_학번_이름.java
      - 보고서
        - 이곳에 보고서 문서 파일을 저장한다.
        - 입력과 실행 결과는 화면 image로 문서에 포함시킨다.
        - 문서는 pdf 파일로 만들어 제출한다.



보고서 작성 방법

- 겉장
  - 제목: 자료구조 실습 보고서
  - [제xx주] 숙제명
  - 제출일
  - 학번/이름
- 내용
  1. 프로그램 설명서
     1. 주요 알고리즘 /자료구조 /기타
     2. 함수 설명서
     3. 종합 설명서 : 프로그램 사용방법 등을 기술
  2. 구현 후 느낀 점 : 요약의 내용을 포함하여 작성한다.
  3. 실행 결과 분석
     1. 입력과 출력 (화면 capture : 실습예시와 다른 예제로 할 것)
     2. 결과 분석

     -------- 표지 제외 3장 이내 작성 --------
  4. 소스코드 : 화면 capture가 아닌 소스를 붙여넣을 것

     소스는 장수 제한이 없음.

[제 14 주 실습] 끝