BTree

Name: Hamza Hassan Mohammed ID : 26

Contents

1- Objective

The objective of the assignment is implementing BTree and dealing efficiently with its basic operations with preserving the properties of Btree and – as an application – implementing a simple search engine to deal with files in the hard disk

2- Introduction

B-Tress are balanced search trees designed to work well on disks or other direct-access secondary storage devices.

It’s used frequently to store information mainly in database systems

3- Main Method Time Complexity

	Best case	Average	Worst Case
Insertion	O(1)	$O(tLog_tn)$	$O(tLog_tn)$
deletion	O(1)	$O(tLog_tn)$	$O(tLog_tn)$
search	O(1)	$O(tLog_tn)$	$O(tLog_tn)$
indexWebPage	$O(cnt * tLog_tn)$	$O(cnt * tLog_tn)$	$O(cnt * tLog_tn)$
SearchWordRanking	$O(tLog_tn)$	$O(tLog_tn)$	$O(tLog_tn)$
DeleteWebPage	$O(cnt * tLog_tn)$	$O(cnt * tLog_tn)$	$O(cnt * tLog_tn)$

Note

n: the number of nodes in the tree

t: is the minimum degree of the tree

cnt: is the number of words in the file

4- Space Complexity

All these methods use Btree to get and maintain data so we aren't considering the space complexity of it (**$O(tLog_t n)$**).

Method	**Complexity**
Insertion	**O(1)**
deletion	**O(1)**
search	**O(1)**
indexWebPage	**O(cnt)**
SearchWordRanking	**O(1)**
DeleteWebPage	**O(cnt)**

Note

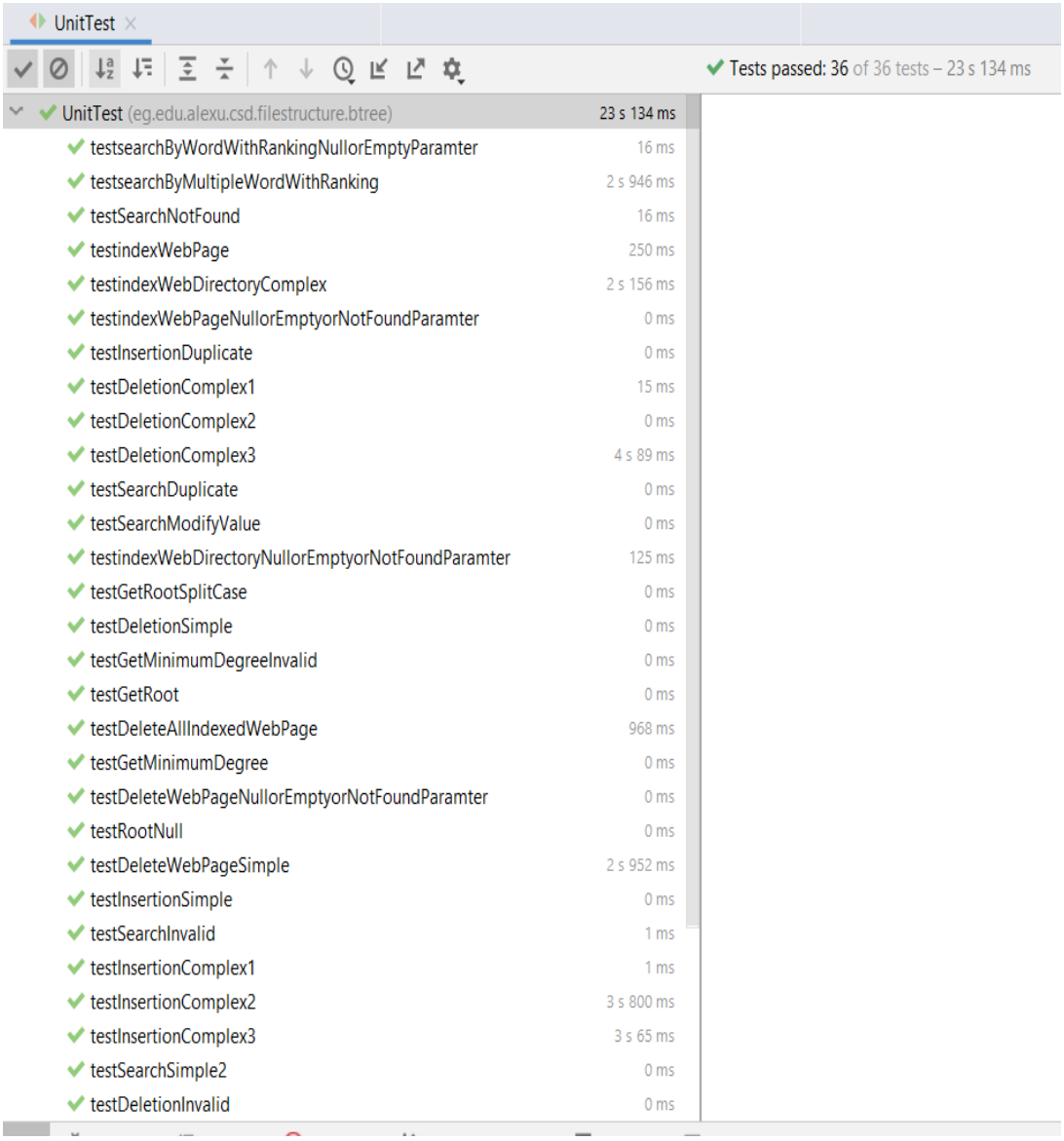
n: the number of nodes in the tree

t: is the minimum degree of the tree

cnt: is the number of words in the file

5-Screen shots

I- Tests



II- Merge Subroutine

```
void merge(IBTreeNode<K, V> parent, int idx) {

    List<IBTreeNode<K, V>> children = parent.getChildren();
    IBTreeNode<K, V> ch1 = children.get(idx);
    IBTreeNode<K, V> ch2 = children.get(idx + 1);

    List<K> parentKeys = parent.getKeys();
    List<V> parentValues = parent.getValues();
    K key = parentKeys.get(idx);
    V value = parentValues.get(idx);
    parentKeys.remove(idx);
    parentValues.remove(idx);

    List<K> mergedKeysList = ch1.getKeys();
    List<V> mergedValuesList = ch1.getValues();
    List<IBTreeNode<K, V>> mergedChildrenList = ch1.getChildren();

    mergedKeysList.add(key);
    mergedValuesList.add(value);
    /* add keys and values to the merged list */
    for (int i = 0; i < ch2.getKeys().size(); i++) {
        mergedKeysList.add(ch2.getKeys().get(i));
        mergedValuesList.add(ch2.getValues().get(i));
    }
    /*add the children of the sibling to the merged list */
    if (mergedChildrenList != null) {
        for (int i = 0; i < ch2.getChildren().size(); i++) {
            IBTreeNode<K, V> child = ch2.getChildren().get(i);
            mergedChildrenList.add(child);
        }
    }
    children.remove( index: idx + 1);
    children.remove(idx);
    children.add(idx, ch1);

    parent.setChildren(children);
    parent.setKeys(parentKeys);
    parent.setValues(parentValues);
    parent.setNumOfKeys(parentKeys.size());


}
```

III- Split subroutine

```
private void split(IBTreeNode<K, V> parent, int idx) {
    if (parent == null || parent.getChildren() == null || parent.getChildren().get(idx) == null) {
        System.out.println("Unexpected corrupted data in the split function");
        LocalException.throwRunTimeErrorException();
    }
    IBTreeNode<K, V> target = parent.getChildren().get(idx);
    IBTreeNode<K, V> newNode = new BTreeNode<>();
    newNode.setLeaf(target.isLeaf());
    /*
    Fill the new node
     */
    List<K> keys = new ArrayList<>();
    List<V> val = new ArrayList<>();
    List<K> newTargetKeys = new ArrayList<>();
    List<V> newTargetValues = new ArrayList<>();
    K medianKey = target.getKeys().get(minimumDegree - 1);
    V medianValue = target.getValues().get(minimumDegree - 1);

    for (int i = 0; i < minimumDegree - 1; i++) {
        keys.add(target.getKeys().get(i + minimumDegree));
        val.add(target.getValues().get(i + minimumDegree));

        newTargetKeys.add(target.getKeys().get(i));
        newTargetValues.add(target.getValues().get(i));
    }
    newNode.setKeys(keys);
    newNode.setValues(val);
    newNode.setNumOfKeys(keys.size());

    target.setKeys(newTargetKeys);
    target.setValues(newTargetValues);
    target.setNumOfKeys(keys.size());
    if (!newNode.isLeaf()) {
        List<IBTreeNode<K, V>> children = new ArrayList<>();
        List<IBTreeNode<K, V>> newTargetChildren = new ArrayList<>();
        for (int i = 0; i < minimumDegree; i++) {
            children.add(target.getChildren().get(i + minimumDegree));
            newTargetChildren.add(target.getChildren().get(i));
        }
        target.setChildren(newTargetChildren);
        newNode.setChildren(children);
    }
    /*
    Shift parent nodes so that we make the median place ready for the median element
     */
    List<K> parentKeys = parent.getKeys();
    List<V> parentValues = parent.getValues();
    List<IBTreeNode<K, V>> children = parent.getChildren();
    children.add(idx + 1, newNode);
    parentKeys.add(idx, medianKey);
    parentValues.add(idx, medianValue);

    parent.setValues(parentValues);
    parent.setChildren(children);
    parent.setKeys(parentKeys);
    parent.setNumOfKeys(parent.getKeys().size());

}
```