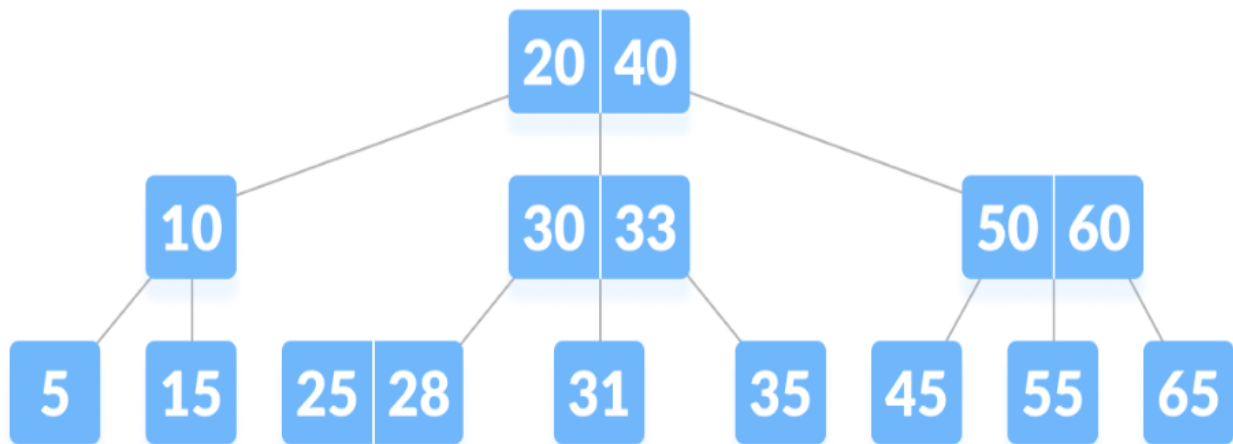


# Data structures and Algorithms



## B-Tree and Indexing



**ID**

19016063  
19015941  
19017359

**NAME**

Omar Khairat Mohamed  
Abdelaziz Mohamed Abdelaziz  
Abdelmoneim Hany Abdelmoneim

## Time & Space Complexity

### IBTreeNode

Function	Time Complexity	Space Complexity
GetNumOfKeys()	O(1)	O(1)
setNumOfKeys()	O(1)	O(1)
isLeaf()	O(1)	O(1)
setLeaf()	O(1)	O(1)
getKeys()	O(n)	O(n)
setKeys()	O(n)	O(n)
getValues()	O(1)	O(n)
setValues()	O(n)	O(n)
getChildren()	O(n)	O(n)
setChildren()	O(n)	O(n)

### IBTree

Function	Time Complexity	Space Complexity
getMinimumDegree()	O(1)	O(1)
getRoot()	O(1)	O(1)
insert()	O(b logn)	O(n)
Search()	O(logn)	O(n)
Delete()	O(logn)	O(n)

## ISearchEngine

Function	Time Complexity	Space Complexity
indexWebPage(String filePath)	$O(N)$	$O(N)$
indexDirectory(String directoryPath)	$O(N)$	$O(N)$
deleteWebPage(String filePath)	$O(N)$	$O(N)$
searchByWordWithRanking(String word)	$O(N)$	$O(N)$
searchByMultipleWordsWithRanking(String sentence)	$O(N)$	$O(N)$

## ISearchResult

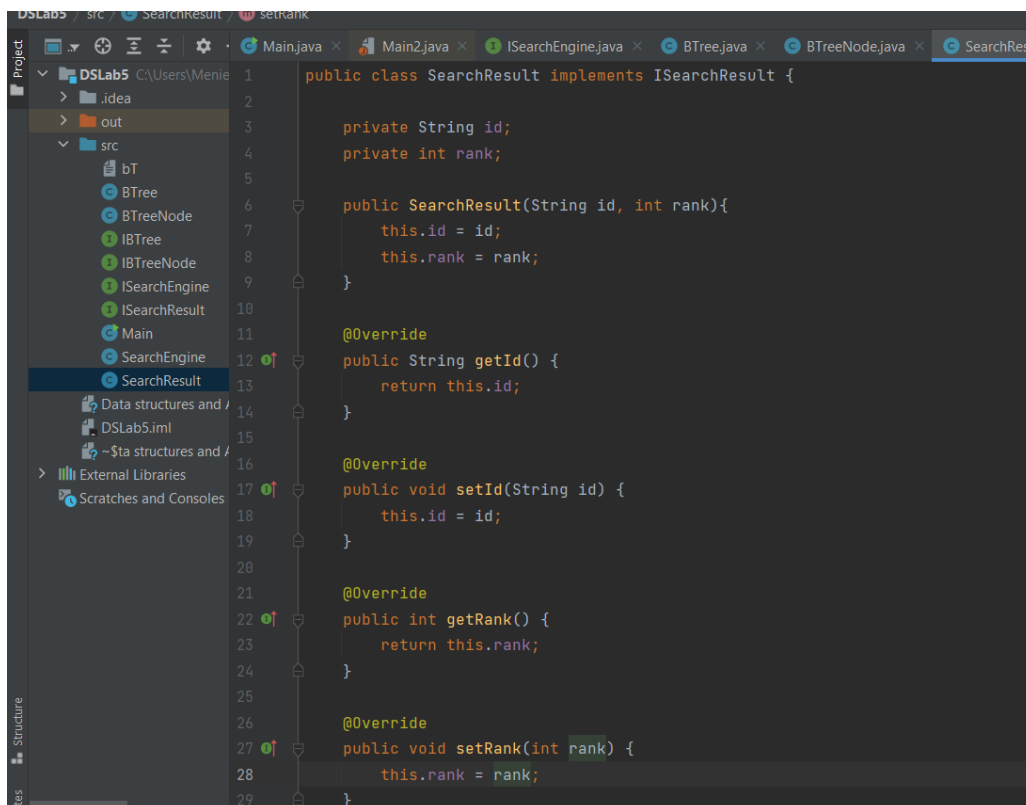
Function	Time Complexity	Space Complexity
getId()	$O(1)$	$O(1)$
getRank()	$O(1)$	$O(1)$
setId()	$O(1)$	$O(1)$
setRank()	$O(1)$	$O(1)$

## Search Engine Code Design

We indexed the Web Page by having B-Tree nodes with key = Document ID and value = Text Context of Document, saving Space Complexity. Then we search the tree for a Sentence by traversing the tree and accessing the value of the node and counting the occurrences of word and setting the rank of the node. If it's a single word then the rank is the number of occurrences of word in text context, if we search for multiple words then we return the minimum number of occurrences of the words as the rank.

## Code Snippets

Search Result class that implements ISearchResult.

A screenshot of an IDE window showing the implementation of the SearchResult class. The left sidebar displays a project structure with folders like 'src' and 'out', and a list of files including 'BTree', 'BTreeNode', 'IBTree', 'IBTreeNode', 'ISearchEngine', 'ISearchResult', 'Main', 'SearchEngine', and 'SearchResult'. The 'SearchResult' file is selected. The main editor area shows the code for the SearchResult class, which implements the ISearchResult interface. The code includes private fields for 'id' and 'rank', a constructor, and four methods: getId(), setId(), getRank(), and setRank().

```
1 public class SearchResult implements ISearchResult {
2
3     private String id;
4     private int rank;
5
6     public SearchResult(String id, int rank){
7         this.id = id;
8         this.rank = rank;
9     }
10
11     @Override
12     public String getId() {
13         return this.id;
14     }
15
16     @Override
17     public void setId(String id) {
18         this.id = id;
19     }
20
21     @Override
22     public int getRank() {
23         return this.rank;
24     }
25
26     @Override
27     public void setRank(int rank) {
28         this.rank = rank;
29     }
30 }
```

Initialize Global B-Tree and list of search results.

Use Java Dom XML Parser to get meta data and insert the document as node in B-Tree with key: ID and value: Text Context.

```
public class SearchEngine implements ISearchEngine{

    private BTree indexedGlobalTree;
    List<ISearchResult> search= new ArrayList<>();

    @Override
    public void indexWebPage(String filePath) {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = null;
        try {
            builder = factory.newDocumentBuilder();
            Document document = builder.parse(new File(filePath));
            NodeList nodeList = document.getElementsByTagName("doc");
            int nodeListLength = nodeList.getLength();
            BTree indexedTree = new BTree((int) Math.floor(nodeListLength/70));
            for(int i = 0; i < nodeListLength; i++){
                Node docNode = nodeList.item(i);
                Element docElement = (Element) docNode;
                indexedTree.insert(docElement.getAttribute( "name: id"), docElement.getTextContent());
            }
            this.indexedGlobalTree = indexedTree;
        } catch (ParserConfigurationException | SAXException | IOException e) {
            e.printStackTrace();
        }
    }
}
```

Same as before but we index the whole directory and check for XML files. (Design Choice) => xml files must end with .xml or .XML . Then we add all Document Nodes in Global B-Tree.

```
public void indexDirectory(String directoryPath) {
    File directory = new File(directoryPath);
    FileFilter xmlFilter = new FileFilter() {

        public boolean accept(File f)
        {
            return (f.getName().endsWith(".xml") || f.getName().endsWith(".XML"));
        }

    };
    File[] listOfXmlFiles = directory.listFiles(xmlFilter);
    if(listOfXmlFiles.length == 0){
        System.out.println("There is no Xml files in this directory.");
        return;
    }
    BTree indexedBigTree = new BTree(100);
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = null;
    for (int i = 0; i < listOfXmlFiles.length; i++) {
        try {
            builder = factory.newDocumentBuilder();
            Document document = builder.parse(listOfXmlFiles[i]);
            NodeList nodeList = document.getElementsByTagName("doc");
            int nodeListLength = nodeList.getLength();
            for(int j = 0; j < nodeListLength; j++){
                Node docNode = nodeList.item(j);
                Element docElement = (Element) docNode;
                indexedBigTree.insert(docElement.getAttribute("id"), docElement.getTextContent());
            }
        } catch (ParserConfigurationException | SAXException | IOException e) {
            e.printStackTrace();
        }
    }
}
```

Same as before but we delete from B-Tree the indices of the doc elements in given file.

```
@Override
public void deleteWebPage(String filePath) {

    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = null;
    try {
        builder = factory.newDocumentBuilder();
        Document document = builder.parse(new File(filePath));
        NodeList nodeList = document.getElementsByTagName("doc");
        int nodeListLength = nodeList.getLength();
        for(int i = 0; i < nodeListLength; i++){
            Node docNode = nodeList.item(i);
            Element docElement = (Element) docNode;
            this.indexedGlobalTree.delete(docElement.getAttribute( name: "id"));
        }
    } catch (ParserConfigurationException | SAXException | IOException e) {
        e.printStackTrace();
    }
}
```

We use this method to count the number of occurrences in Text Context.

```
private static int Count_Occurrences(String searchString, String textContext){
    String[] word = searchString.split( regex: " ");
    int[] counts = new int[word.length];
    for (int i = 0; i < word.length; i++){
        int count = 0;
        int index_from = 0;
        while ((index_from = textContext.indexOf(word[i], index_from)) != -1 ){
            count++;
            index_from++;
        }
        counts[i] = count;
    }
    Arrays.sort(counts);
    return counts[0];
}
```

We use this function to traverse the tree and add the search result element to the list.

```
private void traverse_btree(BTreeNode x,String word) {
    int rank;
    assert (x == null);
    for (int i = 0; i < x.getNumOfKeys(); i++) {
        rank = Count_Occurrences(word, x.getValue(i).toString());
        if (rank != 0) {
            SearchResult I = new SearchResult(x.getKey(i).toString(), rank);
            search.add(I);
        }
    }
    if (!x.isLeaf()) {
        for (int i = 0; i < x.getNumOfKeys() + 1; i++) {
            traverse_btree(x.getChild(i), word);
        }
    }
}
```

Here we use the previous functions to get our search results

```
@Override
public List<ISearchResult> searchByWordWithRanking(String word) {
    traverse_btree((BTreeNode) indexedGlobalTree.getRoot(),word);
    return search;
}

@Override
public List<ISearchResult> searchByMultipleWordWithRanking(String sentence) {
    traverse_btree((BTreeNode) indexedGlobalTree.getRoot(),sentence);
    return search;
}
```