NAME MEMEBR 1 : FEBRONIA ASHRAF TAWFIK ID : 29

NAME MEMEBR 2 : MERNA MUSTAFA EL_REFAE ID : 54

NAME MEMEBR 3 : NEVEEN SAMIR NAGY ID : 58

LAB 3
B TREE AND INDEXING

problem statement :

implementing a B-tree and a simple search engine application that utilizes the B-Tree for data indexing.

B-Tree

B-trees are balanced search trees designed to work well on disks or other direct access secondary storage devices. Unlike the red-black trees, B-tree nodes can store multiple keys and have many children. If an internal B-tree node x contains x.n keys, then x has x.n + 1 children. The keys in node x serve as dividing points separating the range of keys handled by x into x.n + 1 sub-ranges, each handled by one child of x.
Please check the reference[1] for more details about the B-Trees.

Simple Search Engine

You will be given a set of Wikipedia documents in the XML format and you are required to implement a simple search engine that given a search query of one or multiple words you should return the matched documents and order them based on the frequency of the query words in each wiki document, please check the requirements section for more details.

---

code design for the simple search engine application :

1) INDEX WEB PAGE :

Id ← id from the file
ArrayList Words ← words from this id
List<ISearchResult> list ← btree.search(every element in words)
If(list = null) → then btree.insert(word,list<ISearchResult>(id,Rank))
Else → add in this list the new element (ISearchResult (id,Rank))
END

```java
    public void indexWebPage(String filePath) {
        if (filePath == null || filePath == "" || !new File(filePath).exists()) {
            throw new RuntimeErrorException(null);
        }
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = null;
        try {
            builder = factory.newDocumentBuilder();
        } catch (ParserConfigurationException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        try {
            Document document = builder.parse(filePath);
            document.getDocumentElement().normalize();
            // Element root = document.getDocumentElement();
            NodeList nList = document.getElementsByTagName("doc");
            for (int temp = 0; temp < nList.getLength(); temp++) {
                Node node = nList.item(temp);
                // System.out.println(""); //Just a separator
                if (node.getNodeType() == Node.ELEMENT_NODE) {
                    // Print each employee's detail
                    HashMap<String, Integer> map = new HashMap<String, Integer>();
                    Element eElement = (Element) node;
                    String id = eElement.getAttribute("id");
                    String value = eElement.getTextContent();
                    value = value.toLowerCase();
                    ArrayList<String> words = new ArrayList<String>();
                    StringTokenizer t = new StringTokenizer(value);
                    String word = "";
                    while (t.hasMoreTokens()) {
                        word = t.nextToken();
                        words.add(word.toLowerCase());
                    }
                    for (int j=0; j<words.size();j++) {
                        Integer i = map.get(words.get(j));
                        if (i == null) {
                            map.put(words.get(j), 1);
```

```java
                    }
                    for (int j=0; j<words.size();j++) {
                        Integer i = map.get(words.get(j));
                        if (i == null) {
                            map.put(words.get(j), 1);
                        } else {
                            map.put(words.get(j), i + 1);
                        }
                    }
                    for (Entry<String, Integer> entry : map.entrySet()) {
                        String k = entry.getKey();
                        Integer v = entry.getValue();
                        List<ISearchResult> list = btree.search(k);
                        ISearchResult e = new SearchResult(id, v);
                        if (list == null) {
                            List<ISearchResult> newlist = new ArrayList<ISearchResult>();
                            newlist.add(e);
                            btree.insert(k, newlist);
                        } else {
                            list.add(e);
                        }
                    }

                }
            }

        } catch (SAXException | IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }
```

2) INDEX DIRECTORY :

Files[] files = Directory.listFiles()

indexWebPage(every element in files)

```java
@Override
public void indexDirectory(String directoryPath) {
    if (directoryPath == null || directoryPath == "" || !new File(directoryPath).exists()) {
        throw new RuntimeErrorException(null);
    }
    File folder = new File(directoryPath);
    File[] listOfFiles = folder.listFiles();
    for (File file : listOfFiles) {
        if (file.isFile()) {
            indexWebPage(directoryPath+"\\"+file.getName());
        }
    }
}
```

3) DELETE WEB PAGE :

Id ← id from the file

ArrayList Words ← words from this id

List<ISearchResult> list ← btree.search(every element in words)

If(list = null) → then it not found in btree.

Else → list.remove(ISearchResult which ID = id)

END

```java
    public void deleteWebPage(String filePath) {
        if (filePath == null || filePath == "") {
            throw new RuntimeErrorException(null);
        }
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = null;
        try {
            builder = factory.newDocumentBuilder();
        } catch (ParserConfigurationException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        try {
            Document document = builder.parse(filePath);
            document.getDocumentElement().normalize();
            // Element root = document.getDocumentElement();
            NodeList nList = document.getElementsByTagName("doc");
            for (int temp = 0; temp < nList.getLength(); temp++) {
                Node node = nList.item(temp);
                // System.out.println(""); //Just a separator
                if (node.getNodeType() == Node.ELEMENT_NODE) {
                    // Print each employee's detail
                    Element eElement = (Element) node;
                    String id = eElement.getAttribute("id");
                    String value = eElement.getTextContent();
                    value = value.toLowerCase();
                    ArrayList<String> words = new ArrayList<String>();
                    StringTokenizer t = new StringTokenizer(value);
                    String word = "";
                    while (t.hasMoreTokens()) {
                        word = t.nextToken();
                        words.add(word.toLowerCase());
                    }
                    for(int i=0;i<words.size();i++) {
                        List<ISearchResult> list = btree.search(words.get(i));
                        if(list!=null) {
                            for(int j=0;j<list.size();j++) {
                                if(list.get(j).getId().equals(id)) {
```

```java
                Node node = nList.item(temp);
                // System.out.println(""); //Just a separator
                if (node.getNodeType() == Node.ELEMENT_NODE) {
                    // Print each employee's detail
                    Element eElement = (Element) node;
                    String id = eElement.getAttribute("id");
                    String value = eElement.getTextContent();
                    value = value.toLowerCase();
                    ArrayList<String> words = new ArrayList<String>();
                    StringTokenizer t = new StringTokenizer(value);
                    String word = "";
                    while (t.hasMoreTokens()) {
                        word = t.nextToken();
                        words.add(word.toLowerCase());
                    }
                    for(int i=0;i<words.size();i++) {
                        List<ISearchResult> list = btree.search(words.get(i));
                        if(list!=null) {
                            for(int j=0;j<list.size();j++) {
                                if(list.get(j).getId().equals(id)) {
                                    list.remove(j);
                                    break;
                                }
                            }
                        }
                    }
                }
            }
        }catch (SAXException | IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

4) Search by word with ranking :

List<ISearchResult> list = btree.search(word)

```java
@Override
public List<ISearchResult> searchByWordWithRanking(String word) {
    if (word == null) {
        throw new RuntimeErrorException(null);
    } else if (word == "") {
        return new ArrayList<>();
    }
    return btree.search(word.toLowerCase());
}
```

5) Search by multiple word with ranking :

ArrayList Words ← words from sentence

List<ISearchResult> List = btree.search(for every element in wods)

If(list = null) → then return emty list

Else → put in map word and its list

Then compare the ids in every list in map with the ids in the other lists if its equal then put in list this id and the minimum rank between them

And repeat the operation until the list is finish

```java
@Override
public List<ISearchResult> searchByMultipleWordWithRanking(String sentence) {
    ArrayList<String> words = new ArrayList<String>();
    Map<String, List<ISearchResult>> map = new HashMap<String, List<ISearchResult>>();
    SearchResult search = new SearchResult("", 0);
    if (sentence == null) {
        throw new RuntimeErrorException(null);
    } else if (sentence == "") {
        return new ArrayList<>();
    }
    StringTokenizer t = new StringTokenizer(sentence);
    String word = "";
    List<ISearchResult> mulitySearch = new ArrayList<ISearchResult>();
    int count = 0;
    while (t.hasMoreTokens()) {
        word = t.nextToken();
        words.add(word.toLowerCase());
    }
    for (int i = 0; i < words.size(); i++) {
        map.put(words.get(i), searchByWordWithRanking(words.get(i)));
        if (map.get(words.get(i)).size() == 0) {
            return new ArrayList<ISearchResult>();
        }
    }
    for (int j = 0; j < map.get(words.get(0)).size(); j++) {
        count = 1;
        search.setId(map.get(words.get(0)).get(j).getId());
        search.setRank(map.get(words.get(0)).get(j).getRank());
        for (int i = 1; i < words.size(); i++) {
            for (int k = 0; k < map.get(words.get(i)).size(); k++) {
                if (search.getId().equals(map.get(words.get(i)).get(k).getId())) {
                    count++;
                    if (search.getRank() > map.get(words.get(i)).get(k).getRank()) {
                        search.setRank(map.get(words.get(i)).get(k).getRank());
                    }
                    break;
                }
            }
        }
    }
```

The time complexity for the provided interfaces functions of both the B-Tree and the simple search engine :

1) B_TREE :

a) Insertion →

Time : O (M log N) where M is the maximum number of elements in node of B_tree & N is the number of elements in B_Tree.

b) Search →

Time : O (M log N) where M is the maximum number of elements in node of B_tree & N is the number of elements in B_Tree.

c) Deletion →

Time : O (M log N) where M is the maximum number of elements in node of B_tree & N is the number of elements in B_Tree.

d) Get_Root →

Time : O(1)

e) Get_Minimum →

Time : O(1)

2) SEATCH ENGINE →

a) Index Web Page →

Time : O(I*W*M log N) where I is the number of IDs in file & W is the number of words in one documents & M log N is the time of insertion in B_Tree.

b) Index Directory →

Time : O(F*I*W*M log N) where F is the number of files in folder & I is the number of IDs in file & W is the number of words in one documents & M log N is the time of insertion in B_Tree.

c) Delete Web Page →

Time : O(I*W*M log N) where I is the number of IDs in file & W is the number of words in one documents & M log N is the time of deletion in B_Tree.

d) Search by Word →

Time : O(M log N) where M log N is the time of search in B_Tree.

e) Search by multiple word →

Time : O(L^2*W*M log N) where L is the size of list of word & W is the number of words in sentence & M log N is the time of search in B_Tree.