Lab 3
B Tree and Indexing

1 Overview

In this assignment, you're required to implement a B-tree and a simple search engine application that utilizes the B-Tree for data indexing.

2 Introduction

2.1 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.

2.2 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.

3 Requirements

3.1 B-Tree

You are required to implement a generic B-Tree where each node stores key-value pairs and maintains the properties of the B-Trees. The following interfaces should be implemented:

```
package eg.edu.alexu.csd.filestructure.btree;

import java.util.List;


public interface IBTreeNode<K extends Comparable<K>, V> {

        /**
         * @return the numOfKeys return number of keys in this node.
         */
        public int getNumOfKeys();

        /**
         * @param numOfKeys
         */
        public void setNumOfKeys(int numOfKeys);

        /**
         * @return isLeaf if the node is leaf or not.
         */
        public boolean isLeaf();

        /**
         * @param isLeaf
         */
        public void setLeaf(boolean isLeaf);

        /**
         * @return the keys return the list of keys of the given
            node.
         */
        public List<K> getKeys();

        /**
         * @param keys the keys to set
         */
        public void setKeys(List<K> keys);

        /**
         * @return the values return the list of values for the
            given node.
```

```
         */
        public List<V> getValues();

        /**
         * @param values the values to set
         */
        public void setValues(List<V> values);

        /**
         * @return the children return the list of children for the
            given node.
         */
        public List<IBTreeNode<K, V>> getChildren();

        /**
         * @param children the children to set
         */
        public void setChildren(List<IBTreeNode<K, V>> children);
}
```

```
package eg.edu.alexu.csd.filestructure.btree;

public interface IBTree<K extends Comparable<K>, V> {

        /**
         * Return the minimum degree of the given Btree.
         * The minimum degree of the Btree is sent as a parameter t
            the constructor.
         * @return
         */
        public int getMinimumDegree();
        /**
         * Return the root of the given Btree.
         * @return
         */
        public IBTreeNode<K, V> getRoot();
        /**
         * Insert the given key in the Btree. If the key is already
            in the Btree, ignore the call of this method.
         * @param key
         * @param value
         */
        public void insert(K key, V value);
        /**
         * Search for the given key in the BTree.
         * @param key
         * @return
         */
        public V search(K key);
        /**
         * Delete the node with the given key from the Btree.
         * Return true in case of success and false otherwise.
         * @param key
         * @return
         */
        public boolean delete(K key);

}
```

3.2 Simple Search Engine

You will be given a set of Wikipedia documents in the XML format (you can download the Wikipedia data sample from here), and you are required to parse them (using Java DOM XML parser is recommended) and maintain an index of these documents content using the B-Tree to be able to search them efficiently. The following interface should be implemented:

```
package eg.edu.alexu.csd.filestructure.btree;

import java.util.List;

public interface ISearchEngine {

        /**
         * Given a file path that contains one or more Wikipedia
            documents in the same XML format of the provided sample
            data,
         * you are required to index these documents content to be
            able to search through them later.
         * @param filePath
         */
        public void indexWebPage(String filePath);

        /**
         * Given a root directory, you are required to index all the
             files in the given directory/sub directories.
         * The files will be in the same format as described in the
            previous function,
         * you are required to index these documents content to be
            able to search through them later.
         * @param directoryPath
         */
        public void indexDirectory(String directoryPath);

        /**
         * Given a file path that contains one or more Wikipedia
            documents in the same format as described in the
            previous function,
         * you are required to delete these documents IDs from the B
            -Tree index if they were indexed before
         * @param filePath
         */
```

```
    public void deleteWebPage(String filePath);

    /**
     * Given a search query of one word, you are required to
        return a list of SearchResult interface that contains
        the documents IDs where the given word appears in, along
         with each document's rank.
     * The rank of a Wikipedia document is the frequency of the
        given search word in this document.
     * Please note that the search words are not case sensitive.
     * @param word
     * @return
     */
    public List<ISearchResult> searchByWordWithRanking(String
       word);

    /**
     * Similar to the previous function but the search query
        will consist of multiple words
     * you are required to return the Wikipedia documents that
        have this set of word and they can appear in ANY order.
     * The search query words are not case sensitive.
     * The rank of a document is the minimum frequency of the
        given words appearance in this document
     * @param sentence
     * @return
     */
    public List<ISearchResult> searchByMultipleWordWithRanking(
       String sentence);

}
```

```
package eg.edu.alexu.csd.filestructure.btree;

public interface ISearchResult {

        /**
         * Return the document ID which is an attribute provided
            with each Wikipedia document, please check the sample
            data files for more info.
         * @return
         */
        public String getId();
        public void setId(String id);
        /**
         * Return the frequency of the word in the given document.
         * @return
         */
        public int getRank();
        public void setRank(int rank);
}
```

3.3 Report

You are required to deliver a report that contains

- The problem statement
- Your code design for the simple search engine application
- The time and space complexity for the provided interfaces functions of both the B-Tree and the simple search engine.

4 Notes

- You can work in teams of two.
- The Wikipedia data sample can be downloaded from this link.
- The code interfaces can be downloaded from this link.
- Each team should submit the code and report online to this email csed.datastructure.2019@gmail.com with the following subject for the email BTree_NumOfFirstMember_NumOfSecondMember (e.g. BTree_21_53)

5 References

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms.