B-Tree and Indexing

Ahmed Yasser Ahmed Sobhy – ID:10

Pierre Maged Mounir Ibrahim – ID:22

Hossam Eldin Abdel-Ghany Elkordi – ID:24

Mina Ashraf Isaac Barsoum – ID:66

- We were required to implement a B-tree and a simple search engine application that utilizes the B-Tree for data indexing.

- B-trees are balanced search trees designed to work well on disks or other direct access secondary storage devices. B-tree nodes can store multiple keys and have many children.

- We were required to implement a simple search engine that given a search query of one or multiple words we should return the matched documents and order them based on the frequency of the query words in each wiki document.

So we've implemented the Search Engine depending on the properties of the B-tree, since the main idea of B-tree is to reduce the number of disk accesses. The height of B-tree is kept low by putting maximum possible keys in a B-tree node. Generally, a B-tree node size is kept equal to the disk block size. Since height is low for B-tree, total disk accesses for most of the operations are reduced significantly compared to balanced Binary Search Trees. All keys of a node are sorted in increasing order. Like other balanced Binary Search Trees, time complexity to search, insert and delete is O (log n).

Time and space analysis:

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

    public int getMinimumDegree();
    // Space complexity: O(1)           Time complexity: O(1)

    public IBTreeNode<K, V> getRoot();
    // Space complexity: O(1)           Time complexity: O(1)

    public void insert(K key, V value);
    // Space complexity: O(1)           Time complexity: O(log n)

    public V search(K key);
    // Space complexity: O(1)           Time complexity: O(log n)

    public boolean delete(K key);
    // Space complexity: O(1)           Time complexity: O(log n)
}
```

```java
public interface ISearchEngine {

    public void indexWebPage(String filePath);
    // Space complexity: O(1)           Time complexity: O(log n)

    public void indexDirectory(String directoryPath);
    // Space complexity: O(m)           Time complexity: O(m * log n)

    public void deleteWebPage(String filePath);
    // Space complexity: O(1)           Time complexity: O(log n)

    public List<ISearchResult> searchByWordWithRanking(String word);
    // Space complexity: O(1)           Time complexity: O(n)

    public List<ISearchResult> searchByMultipleWordWithRanking(String sentence);
    // Space complexity: O(1)           Time complexity: O(n^2)

}
```

- All the test cases have passed. The search word test cases list contained a lot of different ids with the same rank, we edited it so that the output's rank becomes unique, for example: first element is of rank 1, second element is of rank 2 and so on.