Alexandria University,

Faculty of Engineering,

Computer and Systems Engineering Dept.

DataStructue2

*Assignment3*

BTree Implementation

&

Search Engine

Names:

1. El-Sayed Abdel Naseer El-Sayed. (12)
2. Ziad Hisham Ali Abdel-Hameed. (19)
3. Youssef Ahmed Sayed Gabriel. (61)
4. Problem Statement:

Implementing Btree and use it in the search engine

1. Main Modules ”Classes” :

* **BtreeNode**

Implement the attributes of the node in Btree

* setNumOfKeys
* getNumOfKeys
* setLeaf
* getLeaf
* setKeys
* getKeys
* setValues
* getValues
* setChildren
* getChildren
* **Btree**

Implement Operations that done on the Btree

Has a constructor that take minDegree “t” as a parameter.

* getMinDegree
* getRoot
* insert
* insertNonFull
* search
* searchHelper
* find
* findHelper
* delete
* remove
* getSuccessor
* getPredecessor
* **SearchEngine**

Implement the Search & indexing web pages

Has a constructor that take minDegree “t” as a parameter.

* indexWebPage
* parseFile
* listFiles
* indexDirectory
* deleteWebPage
* searchByWordWithRanking
* searchByMultipleWordWithRanking
* **SearchResult**

Implement the return value of the search engine

Has a constructor that take “Id” & “Rank” as a parameter.

* setId
* getId
* setRank
* getRank

1. Code Design “SearchEngine”:

* There’s two first calls of function either **indexDirectory** or **indexWebPage** & other operations as **deleteWebPage , SearchByWordWithRanking, SearchByMultipleWordByRanking**

1. **indexDirectory :**

function take directory path as a parameter first, check if the directory exists or not if exist, take all files in the given directory and list it in array, then call **listFiles** function that was a recursive function that open each folder in the given directory recursively using DFS till reach to an xml file , Once it reach to this file , calling **indexWebPage** function that first of all check if the given path of the file is correct and parse it using **parseFile** function that check if the given file exist & parse it’s content , After all these checks , inserting all the words that present in the given file in the HashTable then put them in Btree using **insert** function in the “Btree” Class to increase the speed of the operation as the word it’s self as a key And the id of the document of webpage & the rank of the word stored together as a Arraylist of ResultSet as a value .

1. **indexWebPage :**

It’s considered as a part of **indexDirectory,** so it’s completely explained in **indexDirectory** case .

1. **deleteWebPage :**

Function takes file path as a parameter , First parse the give

file using **parseFile** function that check if the given file

exist & parse it’s content , then search for all words that

exist in this file on the Btree using **search** function in

“Btree” Class then decrease it’s Rank by 1 and

remove the Id of the given path & if the Rank reach

to zero , delete the key itself using **delete**

function in “Btree” Class.

1. **SearchByWordWithRanking:**

First, Check if the given Word not null , calling **search** function in “Btree” Class , it returns an ArrayList of SearchResult that contain all documents & Rank of each word in these docs , then sort it by Id using BubbleSort and return it as a return value .

1. **SearchByMultipleWordByRanking:**

First, Check if the given Sentence not null , then split it to take each word individual , then take intersection of Id between first two words and the minimum rank between them , then take intersection between this resulted intersection & the next word and so on …

Finally , return Arraylist of SearchResult as a returnValue .

1. Time & Space Complexity :

* **Btree**
* Search

**Time:** As in the TREE-SEARCH procedure for binary search

trees, the nodes encountered

during the recursion form a simple path downward from the root of the tree, The BTREE\_SEARCH procedure therefore accesses O(h) = O(log n) to the base t disk pages, where h is the height of the Btree and n is the number of keys in the Btree & t is MinDegree.

Since no of keys in each node< 2t, the **while** loop takes O(t) within each node .

So, Total Order is O(t\*logn).

**Space:** It’s just point to the stored Btree

So, There’s no space used.

* Insert

**Time:** For a Btree of height h, BTREE\_INSERT performs

O(h) = O(log n) disk accesses, since only O(1)

DISK\_READ & DISK\_WRITE operations

occur between calls to BTREE\_INSERTNONFULL.

So, Total Order is O(t\*logn).

**Space:** Use ArrayLists to store each node with it’s keys

& Values And Another Arraylist for each node

children

* Delete

**Time:** practice, deletion operations are most often used to delete keys from leaves. The

B-TREE-DELETE procedure then acts in one downward

pass through the tree,

without having to back up. When deleting a key in

an internal node, however,

the procedure makes a downward pass through the

tree but may have to return to

the node from which the key was deleted to replace

the key with its predecessor or

successor (cases 2a and 2b).

Although this procedure seems complicated, it

involves only O(h) disk operations

for a B-tree of height h, since only O(1) calls to

DISK-READ and DISKWRITE

are made between recursive invocations of the

procedure.

So, Total Order is O(t\*logn).

**Space:** It’s just deleting the nodes and it’s Arraylist,

But in Merge case it’s needed new Arraylist to

merge the two Nodes with its children ,then

delete them

* **SearchEngine**
* indexDirectory

**Time:** As it first lists all files recursively using DFS

So, its O(V+E) , where V is number of files & E is

Number of paths between folders.

In Addition to, to calling of **indexWebPage** function that takes O(n\*m) , where n is number of words in the given file & m is number of Id of this word “number of documents that this word present on it

In Addition to, to calling of **parseFile** function that takes O(u) , where u is number of documents in the give file

So, Total Order is O(V\*(u)\*(n\*m)+E).

**Space:** Its need 1D array to list files in the directory

& 2D array to store each document with its

attributes “Id, URL, Title” & ArrayList of

ResultSet

* indexWebPage

**Time:** As it first calls **parseFile** function that takes O(u) ,

where u is number of documents in the give file ,

In Addition to function that takes O(n\*m) , where n is

number of words in the given file & m is number of Id of

this word “number of documents that this word present

on it

So, Total Order is O(u+(n\*m)).

**Space:** It need 2D array to store each document with its

attributes “Id, URL, Title” & ArrayList of

ResultSet & HashTable to insert words with their

Attributes “Word, Id, Rank”

* DeleteWebPage

**Time:** As it first calls **parseFile** function that takes O(u) ,

where u is number of documents in the give file ,

In Addition to function that takes O(n\*m), where n is

number of words in the given file & m is number of Id of

this word “number of documents that this word present

on it

So, Total Order is O(u+(n\*m)).

**Space:** It need 2D array to store each document with its

attributes “Id, URL, Title” & ArrayList of

ResultSet

* SearchByWordWithRanking

**Time:** As it calls **search** function in “Btree” Class

So, its O(t\*logn).

In Addition to, the Bubble Sort to sort SearchResult according to Id that takes O(n^2)

So, Total Order is O(t\*logn + n^2).

**Space:** Its Only need ArrayList of SearchResult to return

it as a returnValue

* SearchByMultipleWordByRanking

**Time:** As it calls **search** function in “Btree” Class

So, its O(t\*logn).

In Addition to, taking the intersection of Id

between given words that takes O(n^3)

So, Total Order is O(t\*logn + n^3).

**Space:** Its Only need ArrayList of SearchResult to return

it as a returnValue

1. Sample Runs:

