Lab 3 data
B-Tree

Names: Ahmed Talaat Noser ID: 05
Salma Ragab Gad 32

Code design for search engine:

- The XML files are parsed using Java DOM XML parser then the data is stored in B-tree.
- indexWebPage: stores the data in B-tree, keys are the words of the file in lower case and values are hash maps store the id’s of the documents in which the word exists and the rank of the word in each id.
- indexDirectory: gets all files of the directory and stores their data using “indexWebPage”.
- deleteWebPage: parses the file, loops on the documents -which are indexed before- of the file, gets the words of each document and removes this doc’s id from the tree.
- searchByWordWithRanking: searches in the tree by keys “the words” ,gets the values ”id’s and ranks” and stores them in a list of “searchResult”.
- searchByMultipleWordWithRanking: searches for each word and find the documents contain this word then finds the intersection for all words. This intersection is the required result list.

Time analysis:

- B-tree:
    - getMinimumDegree: $O(1)$
    - getRoot: $O(1)$
    - insert: $O(t*\log_t(n))$
    - search: $O(\log_2(t)*\log_t(n))$
    - delete: $O(t*\log_t(n))$

- Search engine:
    - indexWebPage: $O(n*t*\log_t(m))$

      where: - n -> number of words in all the documents in the file

      - m -> number of words currently in the b-tree.
    - indexDirectory: $O(n*t*\log_t(m))$

      where: - n -> number of words in all the documents in the files of the directory

      - m -> number of words currently in the b-tree.
    - deleteWebPage: $O(n*t*\log_t(m))$

      where: - n -> number of words in all the documents in the file

      - m -> number of words currently in the b-tree.

    - searchByWordWithRanking: $O(d+t*\log_t(m))$

      where: - m -> number of words currently in the b-tree

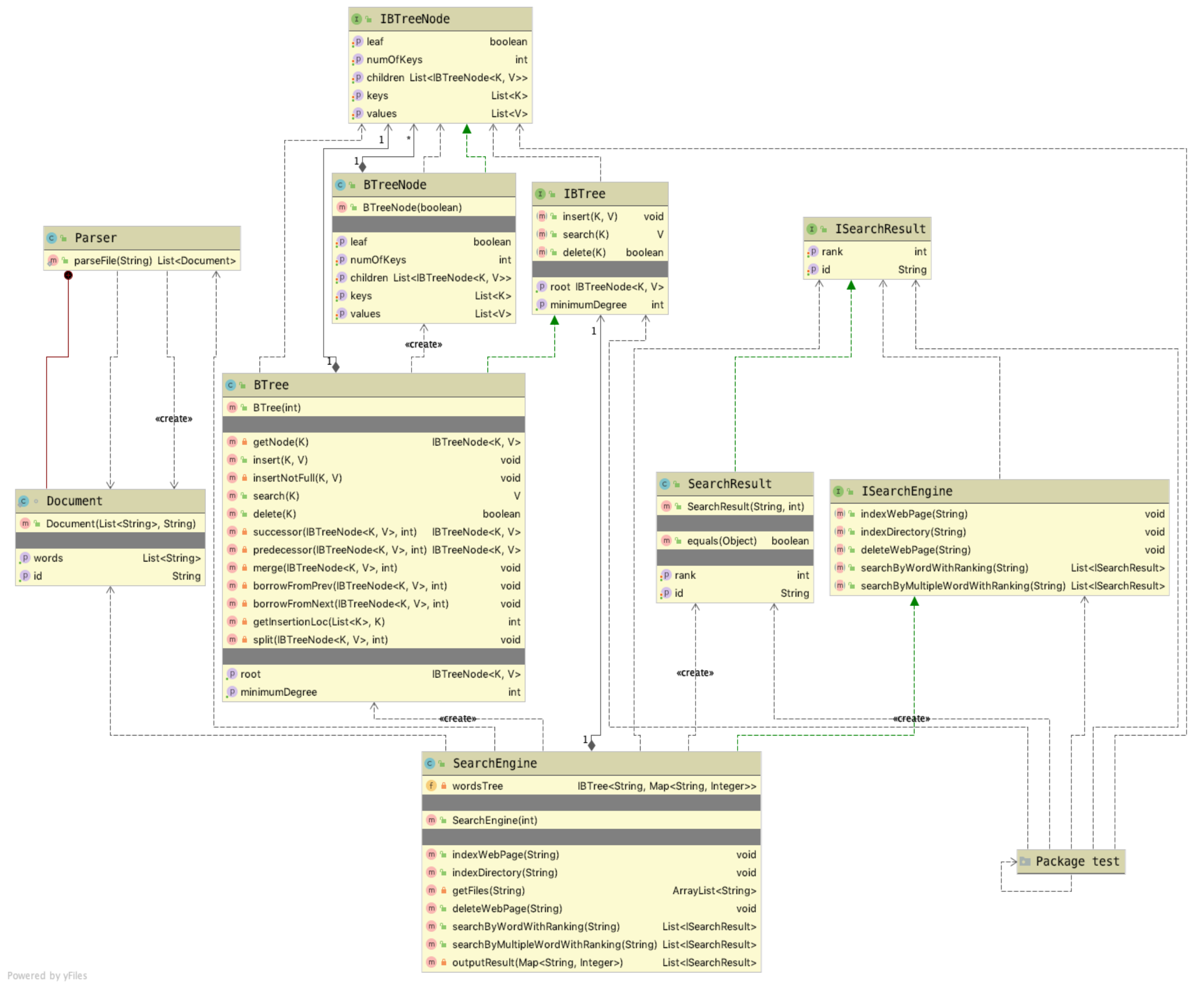
      - d -> number of documents.

- searchByMultipleWordWithRanking:
  $O(d*q*t*\log_t(m))$
  where: - m -> number of words currently in the b-tree
  - d -> number of documents.
  - q -> number of words in query.

Space analysis:

- The memory complexity of B-tree is O(n) where n is the number of keys stored in each node.
- Search engine stores the data in B-tree “words are the keys”.

UML diagram:



IBTreeNode
leaf boolean
numOfKeys int
children List<IBTreeNode<K, V>>
keys List<K>
values List<V>
BTreeNode
BTreeNode(boolean)
IBTree
insert(K, V) void
search(K) V
delete(K) boolean
root IBTreeNode<K, V>
minimumDegree int
ISearchResult
rank int
id String
Parser
parseFile(String) List<Document>
Document
Document(List<String>, String)
words List<String>
BTree
BTree(int)
getNode(K) IBTreeNode<K, V>
insert(K, V) void
insertNotFull(K, V) void
search(K) V
delete(K) boolean
successor(IBTreeNode<K, V>, int) IBTreeNode<K, V>
predecessor(IBTreeNode<K, V>, int) IBTreeNode<K, V>
merge(IBTreeNode<K, V>, int) void
borrowFromPrev(IBTreeNode<K, V>, int) void
borrowFromNext(IBTreeNode<K, V>, int) void
getInsertionLoc(List<K>, K) int
split(IBTreeNode<K, V>, int) void
SearchResult
SearchResult(String, int)
equals(Object) boolean
ISearchEngine
indexWebPage(String) void
indexDirectory(String) void
deleteWebPage(String) void
searchByWordWithRanking(String) List<ISearchResult>
searchByMultipleWordWithRanking(String) List<ISearchResult>
SearchEngine
wordsTree IBTree<String, Map<String, Integer>>
SearchEngine(int)
getFiles(String) ArrayList<String>
outputResult(Map<String, Integer>) List<ISearchResult>
Package test
«create»
1
*
Powered by yFiles