Alexandria University
Computer and Systems Engineering
Department.

Faculty of Engineering
Data Structures 2
Due: 28 May, 2022

Data Structures| | Lab 5

...Bassant Yasser Salah

>>19017262

...Engy Ibrahim Mahmoud

>> 19015478

...Heba-tallah Mostafa

>> 19016836

Problem Statement

… 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.

>> 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: IBTreeNode && IBTree

>> Search Engine

You will be given a set of Wikipedia documents in the XML format , 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: ISearchEngine && ISearchResult .

Search Engine Design

The search engine depends on a database which is a BTree where each node has a key and a value. Each key is a word from the documents words and its corresponding value is a HashMap containing the ids of the documents containing this word and the frequency of this word in that document.

The search engine moreover contains an XML document parser to be able to index a webpage and parse its text content into the database.

The operations connected with the search engine:

1. <u>indexWebPage():</u>
   This function takes a file path containing XML documents and parse its text content into the database to be able to search for it later.
   It divides the file into its documents storing each document ID and text content. Then, it loops through these words and counts the number of occurrences of each word inside the document. Next, it loops through every unique word and searches for it in the database. It adds the entry containing that specific ID and the number of occurrences of this word to the HashMap associated with this word updating its value.
2. <u>indexDirectory():</u>
   This function takes a directory path and parses all of its XML files and the XML files in its subdirectories into the database to be able to search for their text content later.
   It does this by listing all the files in the specified directory looping through them and if the current file is a directory, it indexes this

directory recursively. Otherwise, it indexes the current file using the previous function.

3. **<u>deleteWebPage():</u>**
   This function takes a file path and deletes the IDs of all its documents from the database.
   It divides the file into its documents storing each document ID and text content. Then, it loops through these words searching for each one in the database and removing the current ID from its corresponding HashMap if it exists. If the corresponding HashMap becomes empty, it deletes the word from the database.
4. **<u>searchByWordWithRanking():</u>**
   This function takes a single word and returns a list of entries consisting of the ID of the document containing this word and the frequency of this word in that document. The returned list is ordered based on the frequency of occurrence of this word.
   It takes the word and searches for it in the database storing its corresponding HashMap. It stores the HashMap entries inside a list and sorts this list based on the frequency of occurrence such that the document with the highest frequency of the given word is the first in the list.
5. **<u>searchByMultipleWordWithRanking():</u>**
   This function takes a sentence and returns a list of entries consisting of the ID of the document containing the words of the given sentence and the minimum frequency of the frequencies of these words in that document. The returned list is ordered based on the frequency and the words of the sentence can be of any order.
   It takes the sentence and breaks it into its words. Then, it searches for each word in the database and stores the intersection of the resulting HashMaps in another HashMap. At last, it stores this HashMap entries inside a list and sorts it based on the minimum frequency of all words such that the document with the highest minimum frequency is the first in the list.

>> space complexity

In function insert and delete sometimes we use list to store the children of a node …… let k is the number of children of a node Space Complexity: O(k)

>> B-Tree Insertion ::

Insertion takes on average O(log n) "where n is

the number of inserted elements "

n	10	100	1000	10000	100000	1000000
Time (ms)	1	4	17	72	322	1997

>> B-Tree Deletion ::

Deletion takes on average O(log n) "where n is

the number of inserted elements " to delete the whole tree .

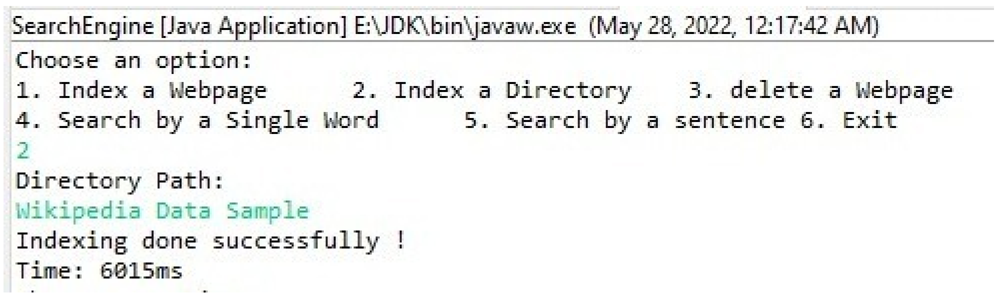
n	10	100	1000	10000	100000	1000000
Time (ms)	1	2	10	13	545	2261

>> >> B-Tree Search ::

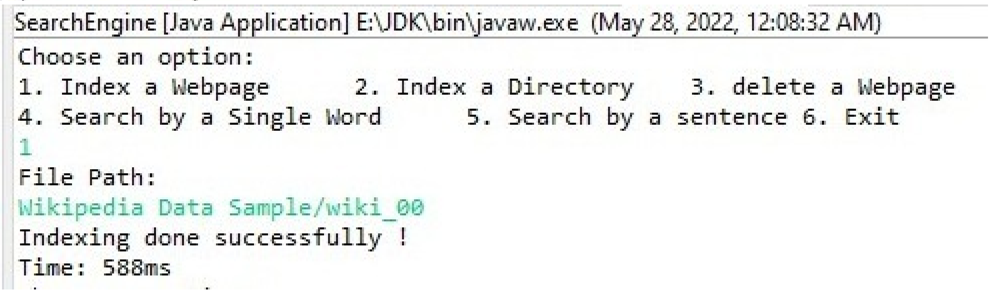
Search takes on average O(log n) also.

Implementation Analysis for Search Engine

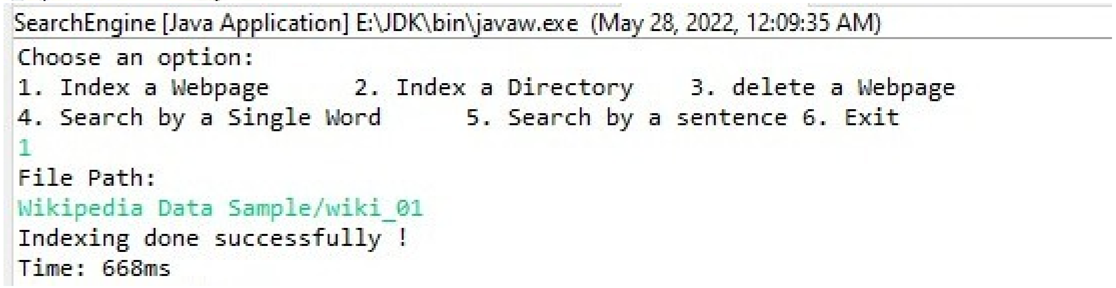
>> Search Engine Insertion



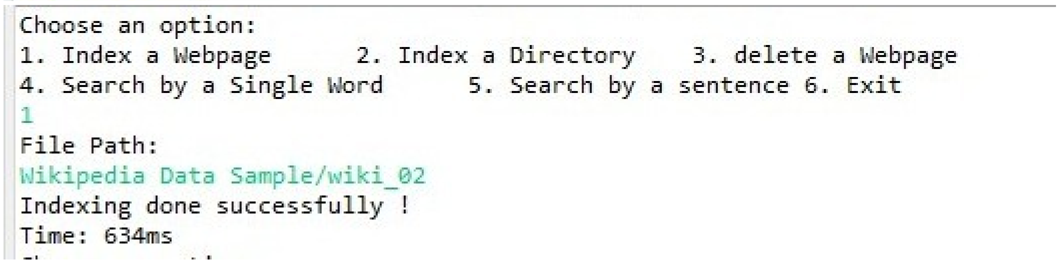
```
SearchEngine [Java Application] E:\JDK\bin\javaw.exe  (May 28, 2022, 12:17:42 AM)
Choose an option:
1. Index a Webpage      2. Index a Directory    3. delete a Webpage
4. Search by a Single Word      5. Search by a sentence 6. Exit
2
Directory Path:
Wikipedia Data Sample
Indexing done successfully !
Time: 6015ms
```



```
SearchEngine [Java Application] E:\JDK\bin\javaw.exe  (May 28, 2022, 12:08:32 AM)
Choose an option:
1. Index a Webpage      2. Index a Directory    3. delete a Webpage
4. Search by a Single Word      5. Search by a sentence 6. Exit
1
File Path:
Wikipedia Data Sample/wiki_00
Indexing done successfully !
Time: 588ms
```

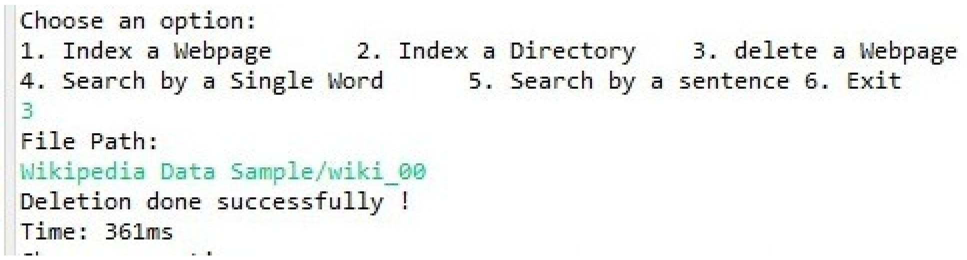


```
SearchEngine [Java Application] E:\JDK\bin\javaw.exe  (May 28, 2022, 12:09:35 AM)
Choose an option:
1. Index a Webpage      2. Index a Directory    3. delete a Webpage
4. Search by a Single Word      5. Search by a sentence 6. Exit
1
File Path:
Wikipedia Data Sample/wiki_01
Indexing done successfully !
Time: 668ms
```

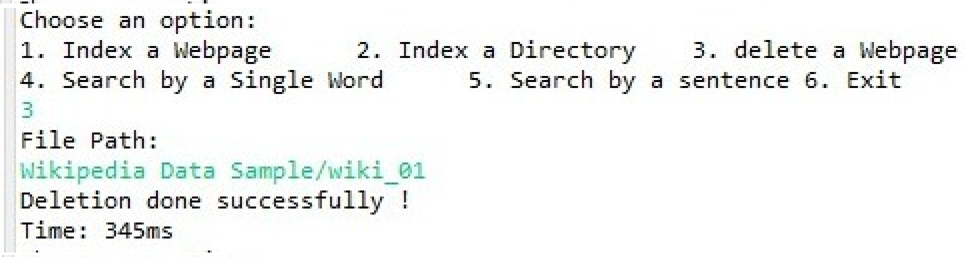


```
Choose an option:
1. Index a Webpage      2. Index a Directory    3. delete a Webpage
4. Search by a Single Word      5. Search by a sentence 6. Exit
1
File Path:
Wikipedia Data Sample/wiki_02
Indexing done successfully !
Time: 634ms
```

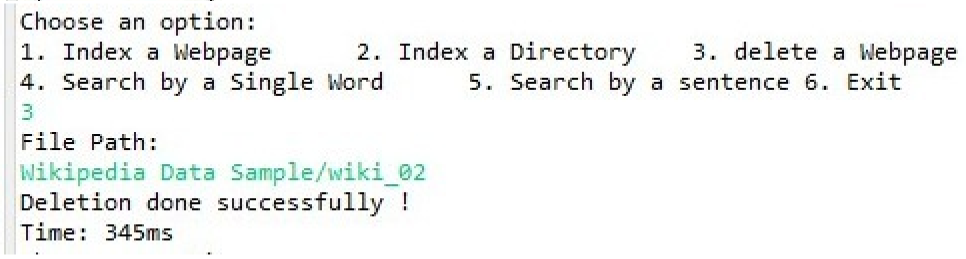
>> Search Engine Deletion



```
Choose an option:
1. Index a Webpage      2. Index a Directory    3. delete a Webpage
4. Search by a Single Word      5. Search by a sentence 6. Exit
3
File Path:
Wikipedia Data Sample/wiki_00
Deletion done successfully !
Time: 361ms
```

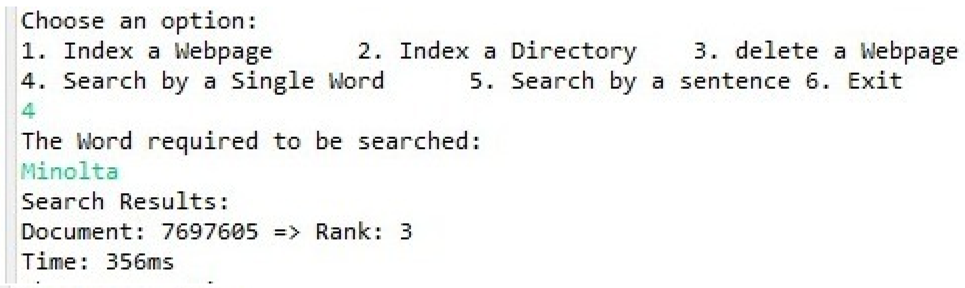


```
Choose an option:
1. Index a Webpage      2. Index a Directory    3. delete a Webpage
4. Search by a Single Word      5. Search by a sentence 6. Exit
3
File Path:
Wikipedia Data Sample/wiki_01
Deletion done successfully !
Time: 345ms
```

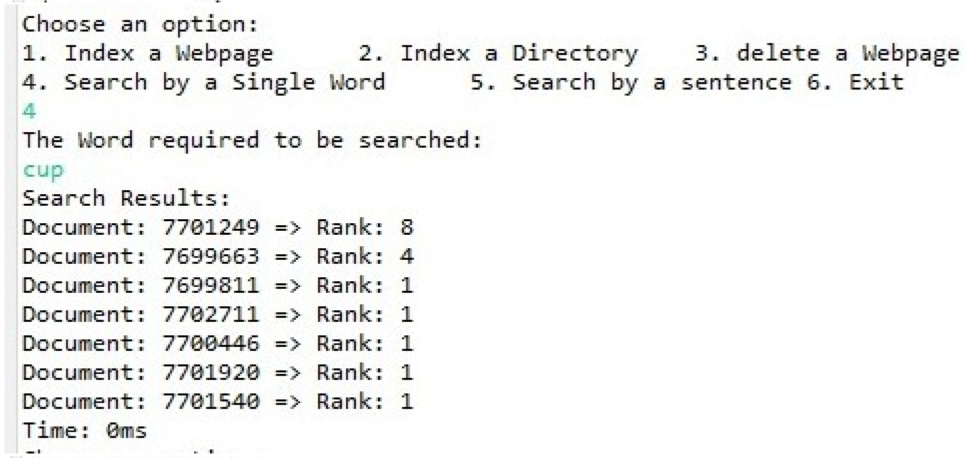


```
Choose an option:
1. Index a Webpage      2. Index a Directory    3. delete a Webpage
4. Search by a Single Word      5. Search by a sentence 6. Exit
3
File Path:
Wikipedia Data Sample/wiki_02
Deletion done successfully !
Time: 345ms
```

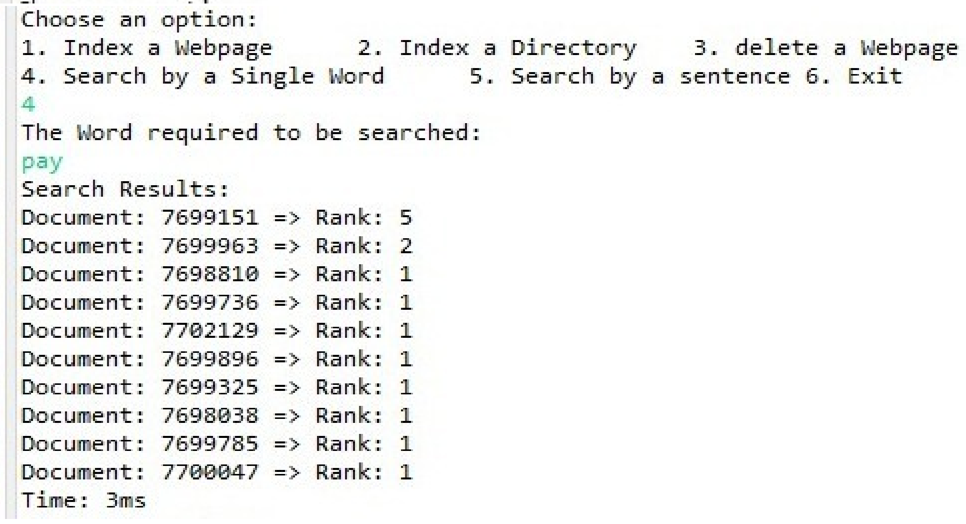
>> Search Engine Searching for a word



```
Choose an option:
1. Index a Webpage      2. Index a Directory    3. delete a Webpage
4. Search by a Single Word      5. Search by a sentence 6. Exit
4
The Word required to be searched:
Minolta
Search Results:
Document: 7697605 => Rank: 3
Time: 356ms
```

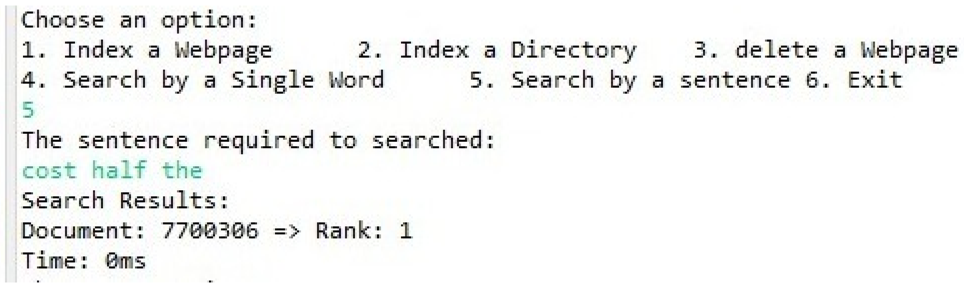


```
Choose an option:
1. Index a Webpage      2. Index a Directory    3. delete a Webpage
4. Search by a Single Word      5. Search by a sentence 6. Exit
4
The Word required to be searched:
cup
Search Results:
Document: 7701249 => Rank: 8
Document: 7699663 => Rank: 4
Document: 7699811 => Rank: 1
Document: 7702711 => Rank: 1
Document: 7700446 => Rank: 1
Document: 7701920 => Rank: 1
Document: 7701540 => Rank: 1
Time: 0ms
```

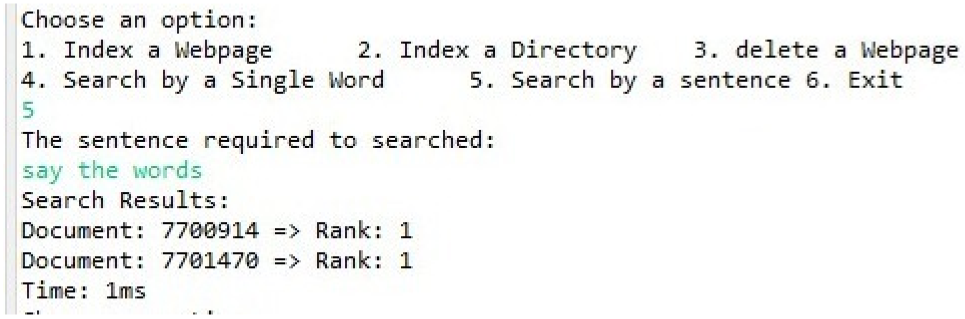


```
Choose an option:
1. Index a Webpage      2. Index a Directory    3. delete a Webpage
4. Search by a Single Word      5. Search by a sentence 6. Exit
4
The Word required to be searched:
pay
Search Results:
Document: 7699151 => Rank: 5
Document: 7699963 => Rank: 2
Document: 7698810 => Rank: 1
Document: 7699736 => Rank: 1
Document: 7702129 => Rank: 1
Document: 7699896 => Rank: 1
Document: 7699325 => Rank: 1
Document: 7698038 => Rank: 1
Document: 7699785 => Rank: 1
Document: 7700047 => Rank: 1
Time: 3ms
```

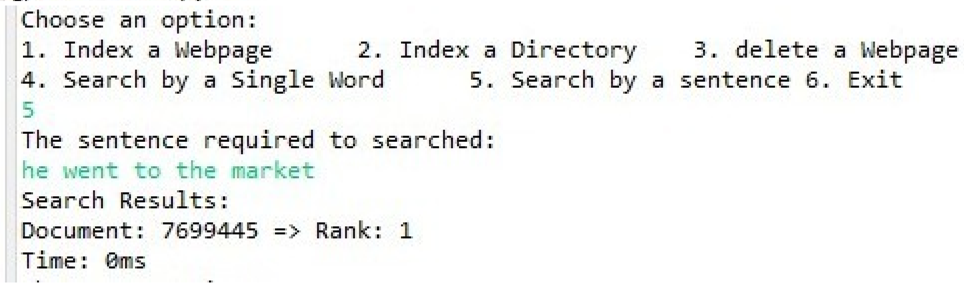
>> Search Engine Searching for a statement



```
Choose an option:
1. Index a Webpage      2. Index a Directory    3. delete a Webpage
4. Search by a Single Word      5. Search by a sentence 6. Exit
5
The sentence required to searched:
cost half the
Search Results:
Document: 7700306 => Rank: 1
Time: 0ms
```



```
Choose an option:
1. Index a Webpage      2. Index a Directory    3. delete a Webpage
4. Search by a Single Word      5. Search by a sentence 6. Exit
5
The sentence required to searched:
say the words
Search Results:
Document: 7700914 => Rank: 1
Document: 7701470 => Rank: 1
Time: 1ms
```



```
Choose an option:
1. Index a Webpage      2. Index a Directory    3. delete a Webpage
4. Search by a Single Word      5. Search by a sentence 6. Exit
5
The sentence required to searched:
he went to the market
Search Results:
Document: 7699445 => Rank: 1
Time: 0ms
```

Time & space complexity in Search Engine

1. <u>indexWebPage():</u>
   Let n be the number of documents in the given file.
   Let k be the number of unique words in each document.
   Let d be the number of words in the BTree.
   Time Complexity: O(nklogd)
   Space Complexity:
   NodeList tags: O(n)
   HashMap wordMap: O(k)
   Space Complexity: O(k)

2. indexDirectory():
   Let m be the number of files in the given directory.
   Let n be the number of documents in each file.
   Let k be the number of unique words in each document.
   Let d be the number of words in the BTree.
   Time Complexity: O(mnklogd)
   Space Complexity: O(k)

3. <u>deleteWebPage():</u>
   Let n be the number of documents in the given file.
   Let k be the number of words in each document.
   Let d be the number of words in the BTree.
   Time Complexity: O(nklogd)
   Space Complexity:
   NodeList tags: O(n)
   String[] words: O(k)
   Space Complexity: O(k)

4. <u>searchByWordWithRanking():</u>
   Let d be the number of words in the BTree.
   Let n be the number of search results.
   searching takes: O(logd)
   adding search results to list takes: O(n)
   sorting search results takes: O(nlogn)
   Time Complexity: O(nlogn)
   Space Complexity:
   HashMap resMap: O(n)
   List searchResList: O(n)

5. <u>SortByMultipleWordWithRanking():</u>
   Let d be the number of words in the BTree.
   Let n be the number of search results.
   Let s be the number of words in the given sentence.
   searching takes: O(slogd)
   determining the minimum frequency takes: O(sn)
   adding search results to list takes: O(n)
   sorting search results takes: O(nlogn)
   Time Complexity: O(sn)
   Space Complexity:
   HashMap totalResMap: O(n)
   List searchResList: O(n)
   Space Complexity: O(n)