Advanced Data Structures (COP 5536)

Project-1

Fall 2018

Topics-

1. Description
2. Running Instructions
3. Program Structure
4. Test cases

Submitted by-
Shreya Singh
UFID- 79154462
Shreya.singh@ufl.edu

1 DESCRIPTION

The main aim of the project was the creation of a Keyword counter. Problem statement stated an implementation of a new search engine which can count the most popular keywords searched using the search engine. The input is given in the form of n as a number and top n keywords are to be retrieved.

The data structures which would be used are Hashmap for storing the strings of keywords, as a part of the collected data by the search engine. Alongside, a Fibonacci heap data structure is used to obtain the top n high frequency keywords. The complexity description of both the data structures is as below –

	Fibonacci Heap		**HashMap**
Create	O(1)	Search	O(1)
Insert	O(1)	Insert	O(1)
Minimum	O(1)	Delete	O(1)
Delete minimum	O(log n)		
Increase key	O(1)		
Delete	O(log n)		

The project has been implemented in JAVA. The input to the program is in the form of a .txt file containing a sequence of keywords with their frequencies, duplicates being allowed, so the JAVA Scanner class and BufferedWriter class has been used to read input and write the output into a file, respectively. Exception handling in JAVA has been used to throw exceptions whenever input is not as expected and few exceptions which might occur have been caught and handled in the code using the JAVA try catch feature.

2 RUNNING INSTRUCTIONS

To execute the program, unzip the file names “Singh_Shreya.zip”

From command line traverse to the folder where the unzip has been done.

Run the makefile command to compile the java files.

make

The following input syntax is required via command line.

java keywordcounter C:\Users\rainb\IdeaProjects\Keywordcounter\data.txt

In case if the input file is in the same folder as that of the current directory, only file name needs to be entered.

java keywordcounter data.txt

The output file is created in the zip folder itself by the name of ***“output_file.txt”.***

3 PROGRAM STRUCTURE

The program is structured in four separate classes with each class performing a separate function.

Class – Fibonacci Heap

public class FibonacciHeap		
Description–	The class is representing the fields to which an object of Fibonacci Heap points to. The class has variables including next, previous, parent, child, degree, childCut and element which present the values that a Fibonacci heap node can store. The class is initiazed with a parameterized constructor for creating an initial singleton node.	
Parameters	Next	Pointer to the element next to the node in the Fibonacci heap
	Prev	Pointer to the previous element in a Fibonacci heap
	Parent	Pointer to the parent of the node in a heap
	Child	Pointer to the child of the node in the heap
	Degree	Number of children of the node
	ChildCut	Boolean value of the number of children node has lost
	Element	Value of a node
Return Value	Pointer having the maximum value in the heap	

Function – Insert into Root

FibonacciHeap insertIntoRoot (FibonacciHeap obj);		
Description	The function performs insertion into the top level linked list. The input object is in the form of a singleton node and it is added into the linked list of the max pointer. If the heap is empty, the new node becomes the first node and max pointer is updated to the node. The function also updates the max pointer in every step.	
Parameters	Obj	Pointer to be inserted into the top level linked list
	maxPointer	Pointer of the maximum value in heap
Return Value	Pointer having the maximum value in the heap	

Function – Remove Maximum

FibonacciHeap removeMax();		
Description	Wrapper function to perform remove max operation. The function wraps around the steps to perform remove max operation including delete maximum value from the heap and pairwise combine. The function also checks to perform pairwise operation only if the heap is not null.	
Parameters	maxPointer	Pointer of the maximum value in heap
Return Value	Pointer having the maximum value in the heap	

Function – Pairwise Combine

FibonacciHeap pairwiseCombine();		
Description	The function helps in performing pairwise combine of trees with the same degree in the heap starting from left to right. The first part of the function creates a temporary array to store all the root heap values, which are then traversed to find similar degree trees. Another array temporarily stores degree values for each tree. If there exists a duplicate value, the function performs the join operation. The max degree tree and min degree tree are found out from the root nodes. If they are same, the join operation is performed by removing it from the root list and making the max value the parent of the min value.	
Parameters	degreeOfTrees rootValues minTree maxTree maxPointer	An array to store the degrees of each tree at the array index Array to store all the values in the root linked list of heap The node with smaller element The node with larger element Pointer towards the maximum element in the heap
Return Value	Pointer having the maximum value in the heap	

Function – Merge two lists

void mergeTwoLinkedList(FibonacciHeap node1, FibonacciHeap node2);		
Description	The function which takes two nodes as the input and combines them into a single list. The two nodes may be a part of the linked list so the function joins the two linked lists together.	
Parameters	Node1 Node2	Node which is to be joined into node2 linked list Node1 will be added to node2 linked list
Return Value	null	

Function – Increase key

FibonacciHeap increaseKey(FibonacciHeap node, int value);		
Description	The function performs increase key operation. This is performed in the case when the node value has to be increased in the heap and the node is the child of another node. This operation is required since the max property of Fibonacci heap is nullified in case the value of a child is bigger than its parents. The function puts a check on the values of child and parent and performs the cut operation as and when required. It also updates the max pointer if the value in the root list has changed.	
Parameters	node value	Pointer whose value is to be updated New value of the node
Return Value	Pointer having the maximum value in the heap	

Function – Cascading Cut

private void recursiveCascadingCut(FibonacciHeap node);		
Description	The function performs cut and cascading cut recursively. The cut is required when the node value becomes greater than the parent's value and the node has to be removed from being a child to being in the root linked list. The corresponding operations of clearing parent child fields have to be performed. Recursive cascading cut is needed when the parent has already lost a child, on losing the second child the parent also has to move to the root linked list.	
Parameters	node	Cut and cascading cut has to be performed on this pointer.
Return Value	null	

Function – Read input file

void readFile(BufferedWriter bw, String input_name) ;		
Description	The function is used to read the input file and parse its values into string and value pair for further storing into heap or performing operations. The function handles all the test cases which can occur as a part of the incorrect input into the program from the test file. The function checks for undesired spaces, characters, values etc.	
Parameters	bw input_name	Object of the bufferWriter class for writing into the txt file Path of the input file
Return Value	null	

Function – Check values within a map

private boolean checkDuplicateandUpdate(String key, int value);		
Description	The function performs increase key operation. This is performed in the case when the node value has to be increased in the heap and the node is the child of another node. This operation is required since the max property of Fibonacci heap is nullified in case the value of a child is bigger than its parents. The function puts a check on the values of child and parent and performs the cut operation as and when required. It also updates the max pointer if the value in the root list has changed.	
Parameters	key value	String to be checked as key into the map New frequency of the string to be update into map
Return Value	Boolean value if the duplicate exists or not	

Function – Write into the output file

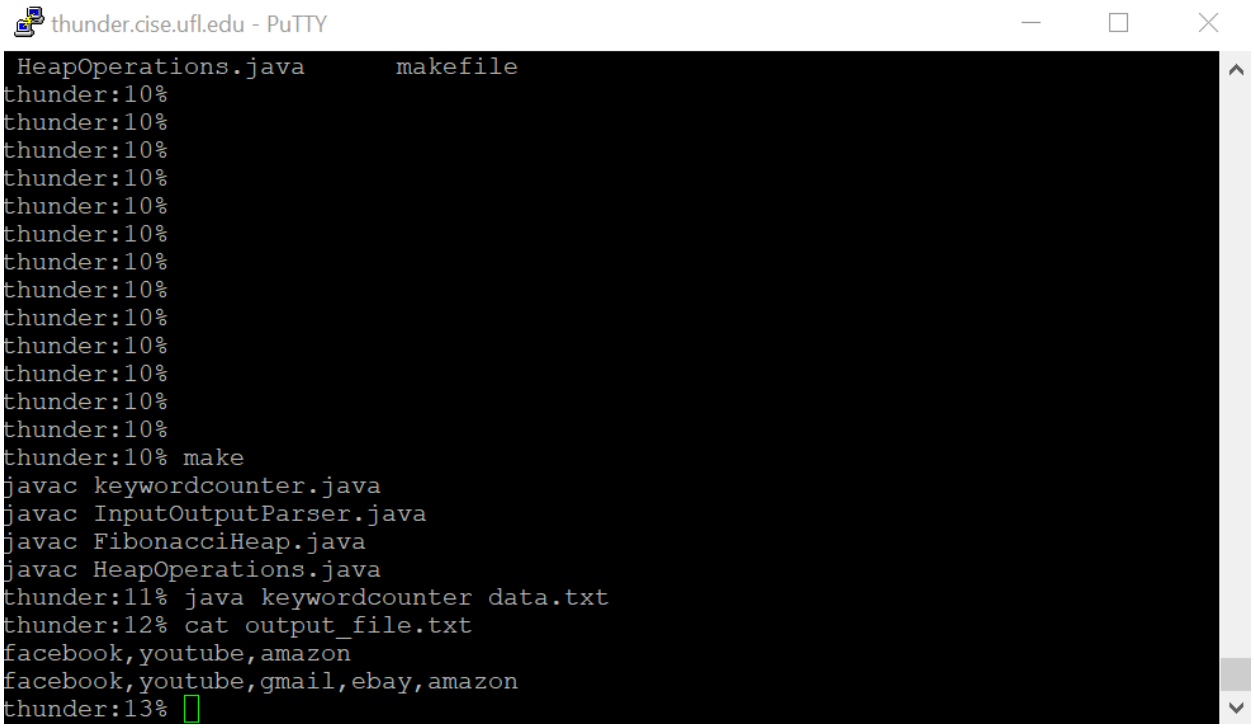
void writeFile(String data, BufferedWriter bw) ;		
Description	The function performs write file operation in which the string of values with maximum frequency as per the query are flushed into the output file.	
Parameters	data bw	String of keywords with the maximum frequency corresponding to query Object of bufferWriter class to write into the file
Return Value	null	

4 TEST CASES

The program has been tested on several inputs including the basic input file given in the project statement pdf.

The program has been tested on Thunder on the input file provided in the project description pdf-

```
$facebook 5
$youtube 3
$facebook 10
$amazon 2
$gmail 4
$weather 2
$facebook 6
$youtube 8
$ebay 2
$news 2
$facebook 12
$youtube 11
$amazon 6
3
$facebook 12
$amazon 2
$stop 3
$playing 4
$gmail 15
$drawing 3
$ebay 12
$netflix 6
$cnn 5
5
stop
```



The program was run using 4 steps-

First, the login was made to the Thunder using PuTTY.

Second, the zip was copied using pscp and extracted using unzip in Ubuntu.

Thirdly, after navigating to the extracted folder, the make file was run using ‘make’

Lastly, the command to run java file was given. Since the file was in the current directory, only the file name was provided.

The output can be seen using ‘cat’ command in Ubuntu.

Test cases executed –

Following test cases have been checked for and outputs can be validated from the outputs folder.

1. Data_1.txt – Contains the given test case as per the problem statement.
2. Input_million.txt – Contains over a million entries to be tested.
3. ADSTest.txt – Contains about 2 million entries.
4. testData.txt – Contains test cases like query > number of present elements. The code also has been tested for the following within the same file –
    a. n > 20
    b. Alphanumeric keywords
    c. Negative frequencies or queries
    d. Single integer in any query or other format manipulations
    e. 1 million entries
    f. Blank spaces in between inputs
    g. Multiple stop statements