Advanced Data Structures
COP5536

Siddhant Gupta
UF ID: 2421-2658
siddhant.gupta@ufl.edu

Problem Statement:

A new search engine "DuckDuckGo" is implementing a system to count the most popular keywords used in their search engine. They want to know what the n most popular keywords are at any given time. You are required to undertake that implementation. Keywords will be given from an input file together with their frequencies. You need to use a max priority structure to find the most popular keywords.

You must use the following data structures for the implementation.

Max Fibonacci heap : To keep track of the frequencies of keywords
Hash table: Keywords should be used as keys for the hash table and value is the pointer to the corresponding node in the Fibonacci heap.

You will need to perform the increase key operation many times as keywords appear in the input keywords stream. You are only required to implement the Fibonacci heap operations that are required to accomplish this task.

Input and Output Requirements:

Your program is required to take the input file name as an argument. Following is an example of a program that read from a file named file_name.

For Java
java keywordcounter file_name

Input Format:

Keywords appear one per each line in the input file and starts with $ sign. In each line, an integer will appear after the keyword and that is the count (frequency) of the keyword (There is a space between keyword and the integer). You need to increment the keyword by that count. Queries will also appear in the input file and once a query (for most popular keywords) appears, you should append the output to the output file. Query will be an integer number (n) without $ sign in the beginning. You should write the top most n keyword to the output file. When "stop" (without $ sign) appears in the input stream, program should end. Following is an example of an input file.

Output Format

Once a query appears, you need to write down the most popular keywords to the output file in descending order. Output for a query should be comma separated list without any new lines. Once the output for a query is finished you should put a new line to write the output for the next query. You should produce all the outputs in the output file named “output_file.txt”. Following is the output file for the above input file

Execution Instructions

- Unzip the file Gupta_Siddhant.zip
- Navigate to the unzipped folder location using ‘cd’ command on the terminal
- Enter make command, ‘$make’, to execute the make file. All the java files required to run this program will be compiled using this command
- Run the program using java command:

    $java keywordcounter <input_file_path_and_name>

Class Structure and Description:

Project consists of 4 java classes :

1. keywordcounter
2. MaxFibonacciHeap
3. Node
4. TestClass

keywordcounter**:**

- This is where the program starts. It contains the main method.
- This class is responsible for reading the input from file, and executing the corresponding query.
- Following are the methods defined in this class
    - **Main -** This is where the code starts
    - **excuteQueries** - This function is responsible for executing the queries
    - **printResult** - This function was coded for testing purposes. It is used to print the results after execution
    - **readKeywordsAndQueriesFromFile** - This function takes file as input, parse that file, and finally save all the queries in a java ArrayList. Later executeQuery function iterates over this list, and executes the corresponding query.

MaxFibonacciHeap:

- This is the Max Finobacci Heap class. Where all the operations of Max Fibonacci heap are carried out.
- Following are the methods defined in this class
    - **insertIntoFibHeap** : This function is used to insert a new node in the fibonacci heap.
    - **deleteMaxNodeFromFibHeap :** This function is responsible for deleting a node from Fib heap.
    - **increaseKey :** This function is used to increment the frequency of the keyword already present in the Fib heap.
    - **pairWiseMergeOfSameDegree :** This function pairwise merges the nodes of the same degree after the delete operation is performed.
    - **singleNode :** This function is used to create a single node in Fib heap.
    - **getListOfNodes :** This function is responsible for getting the list of Nodes present in the any circular doubly linked list.
    - **parentChild :** To make a node child of another node.
    - **deleteNode** : Delete a node from circular doubly linked list.

- **insertBegin :** To insert a node at the beginning of the circular doubly linked list.
- **insertEnd :** To insert a node at the end of the circular doubly linked list.
- **insertListOfNodesAtRootLevel :** To insert list of nodes at the root level.

Node:

- This pojo class contains the basic structure of a Fibonacci node.
- Following are the fields defined in this class
  - **degree**: degree of the Node
  - **child**: pointer to child node
  - **parent**: pointer to parent node
  - **Data :** containing the frequency of the keyword
  - **childCut**: flag for the cascading cut check.
  - **leftSibling :** This contains the pointer to the left sibling of the current node
  - **rightSibling :** This contains the pointer to the right sibling of the current node
  - **Keyword :** keyword of the current node

TestClass:

- This class was created for testing purposes. It checks if the output returned by the program is correct or not. And if the output is wrong, then at which test case it is failing.
- Since this was not required as a part of the project so I have commented out the code which makes a call to this class. On uncommenting, it will generate a report containing the correctness analysis of my program.

max_node: contains pointer to maximum frequency node

Function Prototypes:

Class : keywordcounter

<table>
<tr><th colspan="3">public static void main(String args[])</th></tr>
<tr><td>Description</td><td colspan="2">This is the main function where the program starts.

1. It makes a call to readKeywordsAndQueriesFromFile function to parse the input file and save the queries in a ArrayList.
2. Then calls executeQueries function to execute the queries saved in the ArrayList.
3. Later the results are written to output_file.txt</td></tr>
<tr><td>Parameters</td><td>String[]</td><td>Contains the environment variable, such as the input file name in this case.</td></tr>
<tr><td>Return Value</td><td colspan="2">Void</td></tr>
</table>

public static List<String> readKeywordsAndQueriesFromFile (String fileName)

Description	This function parse the input file. Saves the result to an arrayList.	
Parameters	String	fileName
Return Value	List<String>	Queries from input saved in the list

public void executeQueries (List<String> input, PrintWriter out)

Description	1. This function takes as input the list of queries to be executed 2. It iterates over the queries, executing them one by one in the following way a. If the query starts with '$' then insert/increase the key with the frequency given beside the keyword. b. If it starts with an integer (n), then extract top n entries from Fib heap and save them in output file. c. If it contains stop, then stop the execution of the program.	
Parameters	List<String> PrintWriter out	Queries from input saved in the list This is used to write output in the output file
Return Value	List<String>	

Class : MaxFibonacciHeap

public Node insertIntoFibHeap(Node node)

Description	1. This function is used to perform insert in the Fibonacci heap. 2. Insert the node in the root level and do the following a. If there is no node present in the Fibonacci heap or the node to be inserted have frequency greater than max frequency , then update the max pointer to point to this node.	
Parameters	Node node	Node to be inserted in the Fibonacci Heap
Return Value	Node	Returns the node inserted in the current tree

public Node deleteMaxNodeFromFibHeap(int testCaseNumber)

Description	1. This function performs the delete of a node from the Max Fibonacci Heap. 2. First check if this is the only node present in the heap, if so then delete this node and update the max pointer to null 3. If this is not the only node then do the following a. First insert all the children of the given node to be deleted at the root level. b. Then delete this node c. Perform pairwise merge operation. d. Update the max pointer.	
Parameters	Integer	testCaseNumber
Return Value	Node	Node that was deleted

public void increaseKey (Node node, int count)

Description	1. To increase the frequency of the node given in the input. 2. Algorithm a. First increase the frequency of the node by the specified quantity. b. If the modified frequency of the node is greater than the frequency of its parent, then delete that node and insert it back at the root level. c. Check the childCut field of the parent node and do the following i. If the child cut of parent is true, then keep on deleting the node till its parent has child cut true and insert it back into the root level. ii. If the child cut of parent is false, then delete the node and set child cut of parent true	
Parameters	Node count	Node for which the frequency has to be increased. Count by which the frequency of the node is increased.
Return Value	void	

private void pairWiseMergeOfSameDegree(Node startNode, int testCaseNumber)

Description	1. Pairwise merge of same trees. 2. Start traversing from this node till all the nodes at the root level are being traversed 3. We maintain a hash which keeps the degree of tree traversed till now 4. For each node do the following

	a. First check if any same degree node is present in the hash b. If yes, then merge the 2 nodes by checking which among the both of them have higher frequency c. The higher frequency node becomes the parent and the other node the child. d. Put the parent node with the updated degree in hash	
Parameters	Node testCaseNumber	Node from where the pairwise merge will start The test case number
Return Value	Void	

private Node getSingleNode (Node node)

Description	To get a single node, node having left and right child pointing to itself.	
Parameters	Node node	Node for which a single node has to be created
Return Value	Node node	Returns the created single node

private List<Node> getListOfNodes (Node startNode)

Description	This function returns the list of nodes present in the circular doubly linked list of the start node.	
Parameters	Node	startNode
Return Value	List<Node>	List of nodes present in the circular doubly linked list

private Node parentChild(Node parent, Node child)

Description	This function create parent child relationship between parent and child node which were taken as input.	
Parameters	Node Node	Parent Child
Return Value	Node	Parent

private Node deleteNode(Node node)

Description	This function deletes the node given in the parameter from the cicular doubly linked list.	
Parameters	Node	Node to be deleted
Return Value	Node	Returns the deleted node

private Node insertBegining(Node nodeToInsert, Node startNode)

Description	This function inserts the given node at the start of circular doubly linked list of the startNode.	
Parameters	Node Node	Node To Insert at the starting of the circular doubly linked list Start node of the circular doubly linked list
Return Value	Node	Inserted node

private Node insertEnd(Node nodeToInsert, Node startNode)

Description	This function inserts the given node at the end of circular doubly linked list of the startNode.	
Parameters	Node Node	Node To Insert at the end Start node of the circular doubly linked list

Return Value	Node	Inserted node

private Node insertListOfNodesAtRootLevel (List<Node> nodesList, Node afterThisNode)

Description	This function inserts the list of nodes at the root level after the given node "**afterThisNode".**	
Parameters	List<Node>	List of nodes to be inserted at the start
Return Value	Node	After this node, the list of the nodes has to be inserted

Class : TestClass

public int compare(String keyA, String keyB)

Description	This function is used to compare the frequencies of the keywords specified in the function parameters. This function is used by TreeMap to sort the entries of the map by frequencies of keywords	
Parameters	String String	Keyword Keyword
Return Value	integer	Returns an integer value depending upon which keyword have bigger value

public Map<String, Node> sortByValue(Map<String, Node> unsortedMap)

Description	This function sorts the entries of the map given in the input by the frequency of the keyword.	
Parameters	Map<String, Node>	Unsorted map
Return Value	Map<String, Node>	The sorted by frequency of the keywords map

private Node insertListOfNodesAtRootLevel (List<Node> nodesList, Node afterThisNode)

Description	This function inserts the list of nodes at the root level after the given node "**afterThisNode".**	
Parameters	List<Node>	List of nodes to be inserted at the start
Return Value	Node	After this node, the list of the nodes has to be inserted

public void testTheCorrectness (List<Node> outputByProgram, Map<String, Node> keywordToReferenceMapping, int testCaseNumber, int topNKeywordsCount, PrintWriter out)

Description	This function tests the correctness of the program 1. Compares the frequency of keywords my program thinks have the maximum frequency with the sorted entries of map containing the keywords. 2. If the frequencies are the same that means output of the program is correct 3. Otherwise, it tells at which keyword my program is failing.	
Parameters	(List<Node> outputByProgram	Output of the query which my program computed.
	Map<String, Node> keywordToReferenceMapping	This map contains the node and its corresponding frequency.
	int testCaseNumber	This is the test case number.
	int topNKeywordsCount	Number of top entries to be extracted
	PrintWriter out	To write the output in the file
Return Value	void	