COP 5536 Fall 2018

Programming Project

Name: Sanchit Deora

UFID: 8909-4939

Email: sanchitdeora@ufl.edu

STRUCTURE OF THE PROGRAM

The program uses the following:

1. Node.java
   Creates the Node class which defines the structure of the node to be used in the Max Fibonacci heap.
   It uses the following fields:
   - Child and Parent field of type Node is used to point to other nodes to form a tree in the Fibonacci Heap.
   - Left and Child field of type Node is used to form a circular doubly linked list with their sibling nodes.
   - Degree field of Integer type to keep track of the number of children the node has.
   - ChildCut field is of Boolean type. It is set to True if the node has lost a child since it became a child of its current parent. It is set to false when that node becomes a child to a new node.
   - Keyword of type String and Frequency of type Integer to store the keyword in the node and the frequency at which it has appeared in the input file.

2. FibonacciHeap.java
   Creates the FibonacciHeap class which is used to define various operations and functions of the Fibonacci Heap that will be used to implement storing the keywords used in the search engine and count and computing the most popular keywords.
   It uses the following class variables:
   - maxNode is used to keep track of the node that has the keyword with the most searched frequency in the search engine.
   - countNodes is used to keep track of the number of nodes in the root list of the Fibonacci Heap.

3. keywordcounter.java
   Creates the keywordcounter class which is used to define the main class to read from the input file and write to output file.

The program uses the keywordcounter.class to take input file as an argument and perform operations on them. It creates an instance of the class FibonacciHeap. It also uses Hashmap to store the keywords as keys in the table and stores the pointer to the corresponding node in the Max Fibonacci Heap as the value of the key. The class reads one line at a time from the input file to check whether the input is a keyword and its frequency or a query.

- If the keyword is appearing for the first time in the input file, it calls the insert function of the class FibonacciHeap which in turn creates and initializes a new node using the class Node and inserts this node into the Max Fibonacci Heap.
- If the keyword is re-appearing in the input file, it calls the increaseKey function of the class FibonacciHeap to increase the frequency of the keyword in that node.
- If the input file reads a query, say number ‘n’, then it calls removeMax ‘n’ times to return the ‘n’ most popular keyword. After returning those keywords, it calls the insert function to re-insert those elements back in the Max Fibonacci Heap.

It stops the program once the class reads stop in the input file.

FUNCTION PROTOTYPES

1.	public void insert(Node)	
Description	This function is used to insert a new node to the root list. It first checks for the node associated with the maxNode in the Fibonacci heap. If maxNode is equal to null, then it means the heap is empty. Hence, it makes the new node, i.e. Node a as the maxNode since it is the only element in the heap. If the maxNode is not equal to null, then it adds Node a to the right of the maxNode to the circular doubly linked list of all the siblings i.e. the root nodes of all the trees in the heap (or Root list). After inserting the element, it increases the countNodes variable by one to keep the count of the number of nodes in the root list and calls the function findMax(a) to find the maxNode in the root list.	
Parameters	Node a	New node to be inserted
Return Value	Void	

2.	private void findMax(Node)	
Description	This function is used to find the node with the maximum frequency and set that node as maxNode. It first checks the value of countNodes. If countNodes is equal to 0, that means the heap is empty and maxNode is set to null. If countNodes is not equal to 0, then it traverses the root list by moving from Node a to its right till it reaches back to the node it started from (since it is a circular doubly linked list) and compares its frequency to find the node with the maximum frequency. The node with the maximum frequency is set as the maxNode.	
Parameters	Node a	Node used to traverse the root list
Return Value	Void	

3.	private void putChild(Node, Node)	
Description	This function is used to make Node a child to the parent Node b. Since Node a has a new parent, its childCut field is set to False. The degree of Node b is increased by one. Node a is deleted from the root list and countNodes is decreased by one. If Node b doesn't have a child, Node a is set as the child of Node b. If Node b already has a child, then Node a is inserted to the right of the child of Node b into the circular doubly linked list.	
Parameters	Node a	Node made a child of Node b
	Node b	Node made parent of Node a
Return Value	Void	

4.	private void cutNode(Node, Node)	
Description	This function is used to cut Node a from the parent Node b. The degree of Node b is decreased by one. If Node b has a degree of value 0, that means Node b has no child and hence the child field of Node b is set to null. If Node a was set as the child of Node b, then change the child to the node on the right of Node a in the child list of Node b. The child Node a is then removed from the child list of Node b and is inserted in the root list by calling the function insert(a).	
Parameters	Node a	Node cut from parent Node b
	Node b	Node cut from child Node a
Return Value	Void	

5.	private void cascadingCut(Node)	
Description	This function performs cascading cut in the tree by checking the value of childCut for Node a and assessing if any further cuts are required in the tree. It stores the parent of Node a in Node b. Cascading cut occurs only on non-root nodes and hence the Node b is checked if its equal to null or not (Node b is equal to null if node a is a root node). If the childCut value of Node a is false, it is set to true. If the childCut value of Node a is true, it calls cut(a,b). And now it makes the parent Node b call cascadingCut(b). It occurs recursively till the root node till it finds a node with childCut set to false.	
Parameters	Node a	Node on which cascading cut is performed.
Return Value	Void	

6.	public void increaseKey(Node, int)	
Description	This function is used to increase the frequency of the Node a. It adds val to the current frequency. It sets Node b to be the parent of Node a. It checks if the parent node is not equal to null and new frequency is greater than the Node b. If it is true, then it cuts Node a from its parent by calling cutNode(a,b) and calls cascadingCut(b);	
Parameters	Node a	Node for which the frequency is to be increased
	Int val	Value by which the frequency increases
Return Value	Void	

7.	public Node removeMax()
Description	This function is used to find and return the maxNode in the heap. It sets Node b as the maxNode. If Node b has a degree greater than 0, it calls cutNode(a,b) iteratively where Node a is the child of Node b. This cuts all the nodes in the child list of Node b and re-inserts them in the root list. Node b is removed from the root list and countNodes is decreased by one. If Node b was the only node in the heap, maxNode is set to null. Else, it calls findMax(b.right) to find the new maxNode and then calls pairwiseCombine().
Parameters	Null
Return Value	Node with the maximum frequency

8.	private void pairwiseCombine()
Description	This function is used to perform Pairwise combine operation for an enhanced remove max operation in the heap. It creates a Hash table known as the tree table to store the node with their degree. The objective of this function is for all the tree to be unique in terms of their degree in the root list. It sets Node a as the maxNode and is used to traverse in the root list. It checks if the node for its degree exists in the tree table. If there exist another node in the tree table with the same degree, the maximum frequency of the two nodes are found. Then it iteratively calls putChild(q,p) to make Node q the child of Node p where Node q is the node with the smaller frequency than Node p. It increases the degree of Node p by one. It then adds the node to the table with the Node being the value and the key being its degree. It calls findMax(p.right) to find the new maxNode in the root list in the Max Fibonacci Heap.
Parameters	Null
Return Value	Void

CONCLUSION

The program has been implemented correctly to obtain the ‘n’ most popular keywords at any given time using Max Fibonacci Heap using the Fibonacci Heap operations on a given input file.