ADVANCED DATA STRUCTURES

COP 5536

Fall 2018

Project 1: DuckDuckGo

Name: Amruta Basrur

UFID: 4463 4819

Email Id: amruta.basrur@ufl.edu

PROJECT DESCRIPTION

This project implements a search engine "DuckDuckGo" to find n most popular keywords used in the search engine.

It uses the following data structures:

- Max Fibonacci Heap: Used to keep track of frequency of keywords
- Hash Table: Used to store the keyword as keys in the hash table and value is the pointer to the corresponding node in a Fibonacci heap

This project is implemented in Java and doesn't make use of any internal data structure except Hash Table implementation for creating a Hash Table.

EXECUTION INSTRUCTIONS

1. Extract contents of Basrur_Amruta.zip
2. Enter the below command to compile java files
   Make
3. Enter the below command to execute java file keywordcounter.java
   java keywordcounter filepath/file_name.txt

   Ensure that the file path is entered along with the file name if the file is not in the same folder
4. Open the output_file.txt file to view the generated output

STRUCTURE OF THE PROGRAM

Node.java

Class variables:

Int degree: Integer value of signifying number of children of a node used in Fibonacci heap.

Int data: Integer value signifying frequency of keyword

Node left: Pointer to the element on the left of current element in doubly linked list

Node right: Pointer to the element on the right of current element in doubly linked list

Node parent: Pointer to parent element of the current element in doubly linked list

Node child: Pointer to one of the children of current element in doubly linked list

Boolean childCut: Boolean field indicating whether child of the current element is removed after it has become the root or child of another element.

Class Methods:

List of all getter setters:

Public int getDegree()	To get degree of a Node
public void setDegree(int degree)	To set degree of a Node
public int getData()	To get value from a Node
public void setData(int data)	To set value field of a Node
public Node getLeft()	To get left Node of current Node
public void setLeft(Node left)	To set left Node of current Node
public Node getRight()	To get right Node of current Node
public void setRight(Node right)	To set right Node of current Node
public Node getParent()	To get parent of current Node
public void setParent	To set parent of current Node
public Node getChild()	To get child of current Node
public void setChild(Node child)	To set child of current Node
public boolean isChildCut()	To get childCut flag of Node
public void setChildCut(boolean childCut)	To set childCut flag of Node

Node(int data)		
Description	A constructor called while creating objects of Node class. Here data, i.e. frequency of keyword to be inserted into max Fibonacci heap is given as the input.	
Parameters	Int data	Value signifying frequency of keyword
Return Value	null	

FibonacciHeap.java:

This class has the below parameters and methods to perform basic Fibonacci heap functionalities.

Parameters:

Node maxPointer: Pointer to a Node with the highest data field.

Functions:

public void insert(Node)		
Description	Insert an element in the max Fibonacci heap. Here the element is inserted after maxPointer in the top level doubly linked list. If the value of newly inserted node is greater than the value in maxPointer, maxPointer is updated to pint to the newly inserted node.	
Parameters	Node newNode	A newly created node object with already stored data field
Return Value	null	

public Node removeMax()		
Description	Removes element stored in the maxPointer, i.e. an element with maximum value of data field from Fibonacci heap. It returns null if no element exists in Fibonacci heap else returns the max element. After this, insert() is called for all its children along with pairwiseCombine() for all neighboring elements which returns new value of maxPointer.	
Parameters	maxPointer	Pointer to a Node with the highest data field
Return Value	Pointer to the removed max node	

public Node pairwiseCombine()		
Description	Combines elements with same degree from the top level doubly linked list. This is done by using a degreeTable, where if there is an already existing element with same degree in degreeTable, merge() is called. If not, the element is store in degreeTable corresponding to its degree as key. After there is only one element per degree, all elements from the degreeTable are linked together to form a doubly linked list and root element with max value is returned.	
Parameters	Node start HashMap degreeTable	Pointer to right element of maxPointer (node to be removed) A table storing degree as the key and a pointer to element of that degree as value
Return Value	New maxPointer after removal of previous max element	

public void merge(Node node1, Node node2)	
Merges two elements with same degree making element with greater data field as the parent of other. Also, it increments the degree field of the parent and inserts this element back into the degreeTable calling merge() again if there exists an element with the same degree in degreeTable. If not, this element is directly inserted into the degreeTable.	
Node node1 Node node2	Pointer to an already existing element in degreeTable Pointer to an element with same degree as node2
null	

public void increaseKey(Node node, int data)		
Description	Increases the value of data field of Node with input increment. After increment, if the value is greater than parent's value then cut() is called. If not maxPointer is updated incase increased value is greater than maxPointer's data.	
Parameters	Node node Int data	Pointer to an element whose value is increased Value of the increment
Return Value	null	

public void cut(Node node)		
Description	Removes the given element and inserts it into top level doubly linked list. It updates the parent field of node and child field of node's parent if necessary. Then calls insert() for the node and cascadingCut() for the parent.	
Parameters	Node node	Pointer to an element to be removed and inserted into top level list
Return Value	null	

public void cascadingCut(Node node)		
Description	Removes the given parent element if childCut field is set to true or if parent is not null. It calls cut() for an element which satisfies above criteria which then makes a recursive call to cascadingCut() for its parent. If childCut is false than we just return the call to the caller.	
Parameters	Node node	Pointer to an element on which cascadingCut is to be performed
Return Value	null	

Keywordcounter.java

It takes file name along with the file path given in command line arguments as the input file and creates an output file in the same directory as the java files.

Public static void main (String args[])		
Description	This method reads line by line of input file specified in command line arguments (args[]). It creates a HashTable to store keywords as key and pointer to a Node in Fibonacci heap as value. Until it encounters stop keyword or an exception (for invalid inputs) it keeps scanning each line. It calls insert() upon finding a new keyword or increaseKey() if it's an existing keyword. Upon encountering number of queries, it writes the output file with n most frequently used keyword by calling removeMax() n times and inserts these elements back after execution of n queries.	
Parameters	Int data	Value signifying frequency of keyword
Return Value	null	

EXECUTION OF THE PROGRAM

Input file:

```
$facebook 20
$youtube 1
$amazon 11
$gmail 12
$weather 13
$$facebook 5
$$youtube 6
$ebay 7
$news 8
$stop 9
$playing 4
$drawing 3
1
$$facebook 7
$ebay 2
14
stop
```

```
thunder:15% ls
 FibonacciHeap.java   keywordcounter.java   Node.java
 Input.txt            makefile             'Project report.docx'
thunder:16% make
javac Node.java
javac FibonacciHeap.java
javac keywordcounter.java
thunder:17% ls
 FibonacciHeap.class   keywordcounter.class   Node.class
 FibonacciHeap.java    keywordcounter.java    Node.java
 Input.txt             makefile              'Project report.docx'
thunder:18% java keywordcounter Input.txt
thunder:19% ls
 FibonacciHeap.class    keywordcounter.java   output_file.txt
 FibonacciHeap.java     makefile             'Project report.docx'
 Input.txt              Node.class
 keywordcounter.class   Node.java
thunder:20% cat output_file.txt
facebook,youtube,amazon
facebook,youtube,gmail,ebay,amazonthunder:21%
```