Advanced Data Structures (COP 5536)

PROJECT REPORT FALL 2018

Anurag Bagalwadi
UFID: 4936-9125
anuragbagalwadi@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.

Project Description

A Fibonacci heap is a data structure for priority queue operations, consisting of a collection of heap-ordered trees. It has a better amortized running time than many other priority queue data structures including the binary heap and binomial heap. Fibonacci heaps are named after the Fibonacci numbers, which are used in their running time analysis. To implement the keyword counter for a search engine, we use such a Fibonacci heap as a maximum priority structure to perform operations like insertion, extraction of maximum element, increase key of any element and pairwise melding of the Fibonacci Heap in case of remove max. At any instance, the Fibonacci Heap will maintain a pointer to the max element in the data structure and we can extract the max element as and when required as expected by the problem statement. Once a set of max keywords are extracted from the Fibonacci Heap based on the input, the max nodes are re-inserted into the heap before the next set of operations. The use of Fibonacci Heaps allow us to implement a solution with a superior amortized runtime complexity.

Instructions and Output:

The following steps can be followed to run the program:

```
anurag@ANURAGPC:~/ADSProject$ ll
total 0
drwxrwxrwx 1 anurag anurag 4096 Nov 15 18:41 ./
drwxr-xr-x 1 anurag anurag 4096 Nov 15 18:41 ../
-rwxrwxrwx 1 anurag anurag  301 Nov 11 21:23 .classpath*
-rwxrwxrwx 1 anurag anurag  386 Nov 11 21:23 .project*
drwxrwxrwx 1 anurag anurag 4096 Nov 15 18:16 .settings/
drwxrwxrwx 1 anurag anurag 4096 Nov 15 18:16 bin/
-rwxrwxrwx 1 anurag anurag   31 Nov 15 18:41 makefile*
drwxrwxrwx 1 anurag anurag 4096 Nov 15 18:42 src/
anurag@ANURAGPC:~/ADSProject$ make
javac src/FibHeap.java
anurag@ANURAGPC:~/ADSProject$ cd src
anurag@ANURAGPC:~/ADSProject/src$ ll
total 20
drwxrwxrwx 1 anurag anurag 4096 Nov 15 18:43 ./
drwxrwxrwx 1 anurag anurag 4096 Nov 15 18:41 ../
-rw-rw-rw- 1 anurag anurag 2898 Nov 15 18:43 FibHeap.class
-rwxrwxrwx 1 anurag anurag 6136 Nov 15 14:06 FibHeap.java*
-rw-rw-rw- 1 anurag anurag 1215 Nov 15 18:43 Node.class
-rwxrwxrwx 1 anurag anurag  262 Nov 14 14:19 input_file.txt*
-rw-rw-rw- 1 anurag anurag 2622 Nov 15 18:43 keywordcounter.class
anurag@ANURAGPC:~/ADSProject/src$ java keywordcounter input_file.txt
anurag@ANURAGPC:~/ADSProject/src$ ll
total 20
drwxrwxrwx 1 anurag anurag 4096 Nov 15 18:43 ./
drwxrwxrwx 1 anurag anurag 4096 Nov 15 18:41 ../
-rw-rw-rw- 1 anurag anurag 2898 Nov 15 18:43 FibHeap.class
-rwxrwxrwx 1 anurag anurag 6136 Nov 15 14:06 FibHeap.java*
-rw-rw-rw- 1 anurag anurag 1215 Nov 15 18:43 Node.class
-rwxrwxrwx 1 anurag anurag  262 Nov 14 14:19 input_file.txt*
-rw-rw-rw- 1 anurag anurag 2622 Nov 15 18:43 keywordcounter.class
-rw-rw-rw- 1 anurag anurag   59 Nov 15 18:43 output_file.txt
anurag@ANURAGPC:~/ADSProject/src$ cat output_file.txt
facebook,youtube,amazon
facebook,youtube,gmail,ebay,amazon
anurag@ANURAGPC:~/ADSProject/src$
```

Class structure
and
Function prototypes

Node.class

Class variables:

Type	Name	Description
Node	parent	Holds the parent node of the current node instance
Node	left	Holds the left sibling node of the current node instance
Node	right	Holds the right sibling node of the current node instance
Arraylist<Node>	children	Holds the list of all children nodes of the current node instance
String	key	Holds the key of the current node instance
int	value	Holds the frequency value of the current node instance
int	degree	Holds the value of number of child nodes connected to current node instance
boolean	isRoot	True if the current node instance is a root node, otherwise false
boolean	childCut	Holds the childCut value of the current node instance

Constructors and Functions:

public Node(String k, int val)		
Description	Parameterized constructor for Node instance	
Parameters	String k	The key of the Node e.g. “facebook”
	Int val	The value for the node i.e. number of instances of occurance of “facebook” e.g. 10

public void setParent(Node p)		
Description	Setter method for setting Parent node of current node instance	
Parameters	Node p	p is the parent node of current node instance that needs to be set
Return type	void	

public void setLeft(Node l)		
Description	Setter method for setting left sibling node of current node instance	
Parameters	Node l	l is the left sibling node of current node instance that needs to be set
Return type	void	

public void setRight(Node r)		
Description	Setter method for setting right sibling node of current node instance	
Parameters	Node r	r is the right sibling node of current node instance that needs to be set
Return type	void	

public void setValue(int v)		
Description	Setter method for setting value of current node instance	
Parameters	Int v	v is the new value of current node instance that needs to be set
Return type	void	

public void setChildCut(boolean cCut)		
Description	Setter method for setting child cut value of current node instance	
Parameters	Boolean cCut	cCut is the true/false value representing the child cut of current node instance
Return type	void	

public void addChild(Node c)		
Description	Adds and sets new child node for the current node instance	
Parameters	Node c	c is the new child node that needs to be added to current node instance
Return type	void	

public void removeChild(Node c)		
Description	Removes the current node and child node link for the given c in the input	
Parameters	Node c	c is the child for which parent/child mapping needs to be removed
Return type	void	

FibHeap.class

Class variables:

Type	Name	Description
List<Node>	circularList	Holds the circular list of all root nodes in the Fibonacci Heap
Node	first	Holds the first node that was inserted in the circular list of root nodes
Node	left	Holds the latest node inserted into the circular list of root nodes
Node	max	Holds the node that has the max value associated with it, at any given time

Functions:

public void insert(Node newNode)		
Description	Inserts new node into the circular list of root nodes and sets its left and right pointers	
Parameters	Node newNode	newNode is the new node that needs to be inserted into the Fibonacci Heap
Return type	void	

public void increaseKey(Node node,int val)		
Description	Increases value of given key node in the input to the value sent in the input and perform cascading cut if necessary	
Parameters	Node node	node is the node whose value we need to increase
	Int val	val is the value to which the current node value needs to be increased
Return type	void	

private void performCascadingCut(Node node)		
Description	Checks value of child cut of parent and performs cascading cut if necessary	
Parameters	Node node	node whose parent's child cut value we need to check to decide if cascading cut needs to be performed or not
Return type	void	

private void meld()	
Description	Performs pairwise combine based on the degree of each node in the circular list of root nodes. All nodes of same degree and melded together iteratively and finally we get a list of heaps that have distinct degrees. This function is called after every remove max operation.
Return type	void

public Node removeMax()	
Description	Removes and returns the max element of the Fibonacci Heap and adds its children to the circular list of root nodes. Calls meld function after to perform pairwise combine
Return type	Node

public Node findMax()	
Description	Finds and assigns the new max value of the Fibonacci Heap after the remove max operation
Return type	Node

keywordcounter.class

Functions:

public static void main(String args[])		
Description	The main function of the program. Reads the input file per line and performs operations based on input values. Writes output to file every time it encounters a query input. Exits program if it encounters a 'stop' command in the input file	
Parameters	String args[]	args[0] expects the name of the input file to be processed e.g. "input_file.txt"
Return type	void	