Jaganmohan M
UFID# : 6161-9287, Email: jazzycrazzy@ufl.edu

PROBLEM DESCRIPTION

Implementing a system to find n most popular hashtags given from an input file using a max priority structure.

INTRODUCTION

The cases that are possible with respect to a system while finding top n hashtags are

- System encounters a new hashtag
- System encounters hashtag which it has seen before

To handle the above situations the system must implement an efficient data structure which can insert a new hashtag, search for an old hashtag to increase its frequency, and return top hashtag at a certain instant in optimal time. Fibonacci heap gives an optimal performance to this problem where amortized complexities for insert is O(1), remove max is O(log n) and increase key is O(1).

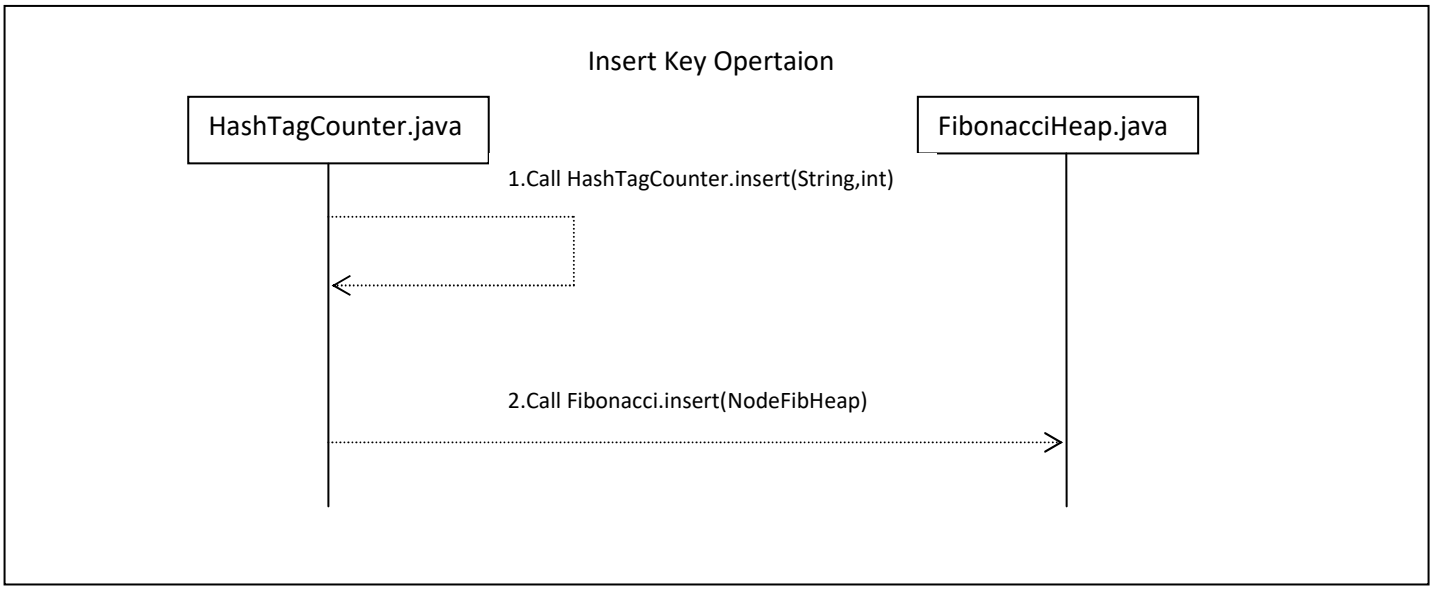
PROGRAMMING ENVIRONMENT

I have used Java to implement Fibonacci heap and jdk 1.8 as the compiler version for this project. Eclipse IDE was the chosen tool to code and debug the source on Windows 10 platform. All the classes are under "adsproject" package. The directory structure for .java files is "HashTagCounter/src/adsproject". A makefile is included which will compile all the .java files in "src" directory and place the .class files in "HashTagCounter/bin/adsproject", after which an executable jar will be generated by makefile with the name "hashtagcounter" which can be executed. Steps to execute the program after unzipping are as following:

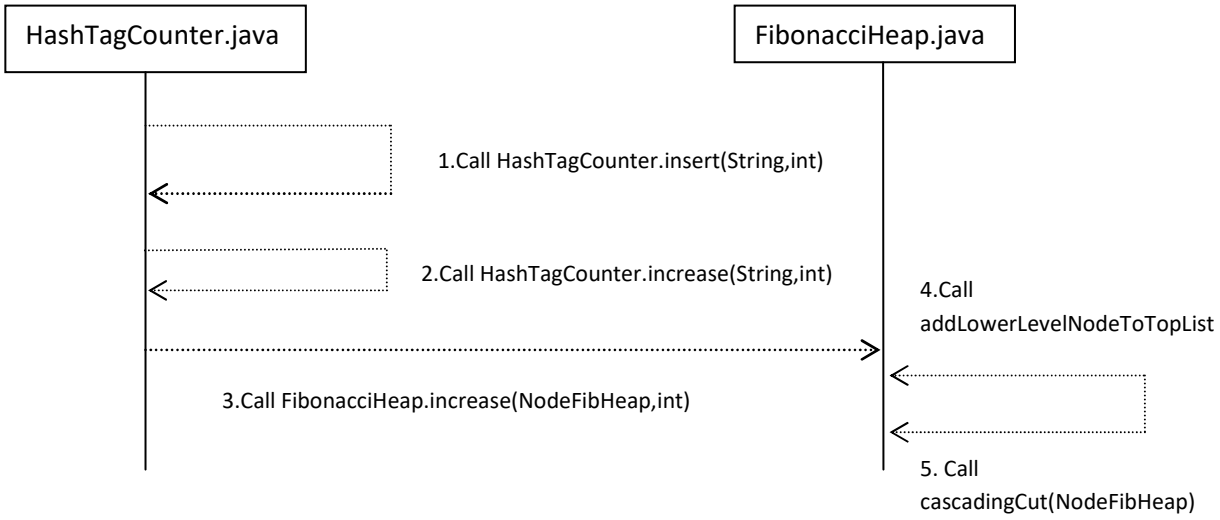
#> cd <Name_Of_Unzipped_Directory>
#> make
#> ./hashtagcounter <INPUT_FILE>

The output of the above command can be found in "output_file.txt"

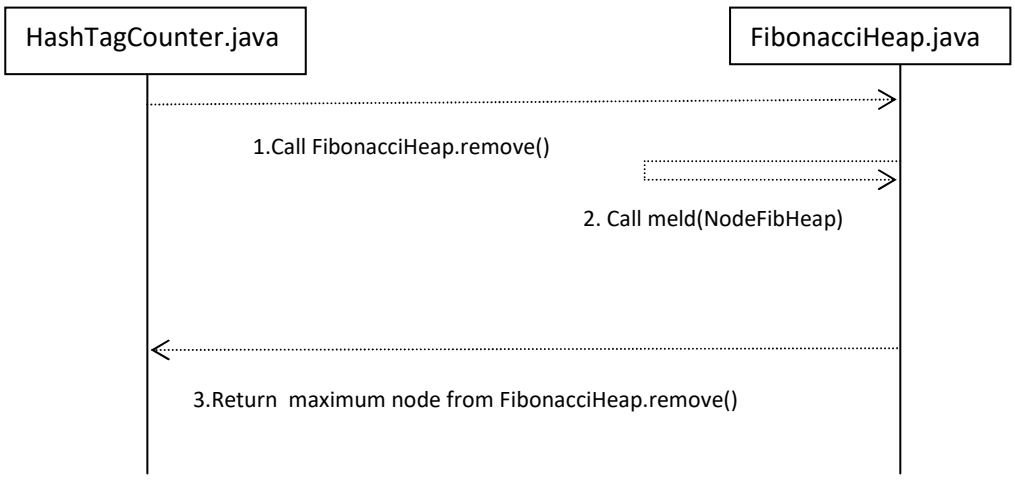
PROGRAM STRUCTURE



Increase Key Operation



Remove Key Operation



IMPLEMENTATION

The project has been implemented using three classes – NodeFibHeap.java, FibonacciHeap.java and HashTagCounter.java. Used Java's java.util.HashMap to store all the hashtags whose references are used while constructing Fibonacci heap. Javadoc has been generated on all the class files and can also be used to see the program implementation details. The classes and function prototypes are presented in a tabular format below.

All Classes:

Class	Description
FibonacciHeap	Implementation of Fibonacci Heap
HashTagCounter	Reads hashtags from an input file and outputs the top 'N' hashtags when it encounters a request for top hashtags
NodeFibHeap	**Node Structure:** Attributes: Key, Data, Degree |Pointers: Parent, Child, Left Sibling, Right Sibling |Boolean: ChildCut

Class NodeFibHeap

java.lang.Object
adsproject.NodeFibHeap

public class **NodeFibHeap**
extends java.lang.Object

Node Structure: Attributes: Key, Data, Degree |Pointers: Parent, Child, Left Sibling, Right Sibling |Boolean: ChildCut

Author: Jagan

Field Summary

Modifier and Type	Field and Description
NodeFibHeap	child
Boolean	childCut
java.lang.Integer	data
Int	degree
java.lang.String	key
NodeFibHeap	leftSibling
NodeFibHeap	parent
NodeFibHeap	rightSibling

Constructor Summary

Constructor and Description
NodeFibHeap()
NodeFibHeap(NodeFibHeap obj)
NodeFibHeap(java.lang.String key, int data)

Method Summary

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

degree

public int degree

data

public java.lang.Integer data

key

public java.lang.String key

parent

public NodeFibHeap parent

child

public NodeFibHeap child

leftSibling

public NodeFibHeap leftSibling

rightSibling

public NodeFibHeap rightSibling

childCut

public boolean childCut

Constructor Detail

NodeFibHeap

public NodeFibHeap()

NodeFibHeap

public NodeFibHeap(java.lang.String key,int data)

NodeFibHeap

public NodeFibHeap(NodeFibHeap obj)

Class FibonacciHeap

java.lang.Object
adsproject.FibonacciHeap

---

public class **FibonacciHeap**
extends java.lang.Object

Implementation of Fibonacci Heap

Author: Jagan

Field Summary

Modifier and Type	Field and Description
private **NodeFibHeap**	**max**
private int	**maxDegree**
private int	**N**

Constructor Summary

Constructor and Description
FibonacciHeap()

Method Summary

Modifier and Type	Method and Description
private Boolean	**addLowerLevelNodeToTopLevellist**(**NodeFibHeap** temp) Since nodes from lower level will be removed during both increase key and cascading cuts modularized to separate function
Void	**arbitaryRemove**(**NodeFibHeap** node)
Void	**cascadingCut**(**NodeFibHeap** temp) It checks whether the increased node in increaseKey is larger than its parent node, if so it will remove the increased node from its parent and add to top level circular list.
NodeFibHeap	**getMax**()
Int	**getMaxDegree**()
Int	**getNoOfNodes**()
Void	**increaseKey**(**NodeFibHeap** node, int increase) Increases the key of a node and updates it in Fibonacci Heap and max if necessary
Void	**insert**(**NodeFibHeap** newNode) Inserts a new node into the fibonacci heap
private **NodeFibHeap**	**meld**(**NodeFibHeap** node1, **NodeFibHeap** node2) Melds two trees Removes the to be child tree from top-level circular list and adds it to the to be parent's child circular list, updating the sibling and parent pointers.
NodeFibHeap	**remove**() Removes maximum node from the Fibonacci heap Updates new maximum and consolidates top circular list trees until no trees of equal rank are left Uses meld operation internally
private void	**updateMax**(**NodeFibHeap** node) Updates the max pointer to maximum node after max removal

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

max

private NodeFibHeap max

maxDegree

private int maxDegree

N

private int N

Constructor Detail

FibonacciHeap

public FibonacciHeap()

Method Detail

getMax

public NodeFibHeap getMax()

getMaxDegree

public int getMaxDegree()

getNoOfNodes

public int getNoOfNodes()

insert

public void insert(NodeFibHeap newNode)

Inserts a new node into the fibonacci heap

Parameters:

newNode – new node to be inserted

updateMax

private void updateMax(NodeFibHeap node)

Updates the max pointer to maximum node after max removal

Parameters:

node - traversal of top-list starts from this node while updating max

remove

public NodeFibHeap remove()

Removes maximum node from the Fibonacci heap Updates new maximum and consolidates top circular list trees until no trees of equal rank are left Uses meld operation internally

Returns:

removed node from Fibonacci heap

arbitaryRemove

public void arbitaryRemove(NodeFibHeap node)

meld

private NodeFibHeap meld(NodeFibHeap node1, NodeFibHeap node2)

Melds two trees Removes the to be child tree from top-level circular list and adds it to the to be parent's child circular list, updating the sibling and parent pointers. Since meld happens only during remove node, to bypass redundant traversal of top-level root nodes we remove the root of the melded tree from its position and put it at "current" position as defined by "current" variable in remove method

Parameters:

node1 - current node in top-level circular list traversal where meld is called

node2 node with equal degree as current node

Returns:

root of the melded trees

increaseKey

public void increaseKey(NodeFibHeap node, int increase)

Increases the key of a node and updates it in Fibonacci Heap and max if necessary

Parameters:

node – node whose frequency needs to be increased

increase – increase in frequency

cascadingCut

public void cascadingCut(NodeFibHeap temp)

It checks whether the increased node in increaseKey is larger than its parent node, if so it will remove the increased node from its parent and add to top level circular list. While doing so to maintain the flat structure of the heap it will also remove the parent whose cascadingCut flag is marked true or in other sense lost a child in the past

Parameters:

temp – node whose cascadingCut field needs to be checked, if true is removed and added to top level list

addLowerLevelNodeToTopLevellist

private boolean addLowerLevelNodeToTopLevellist(NodeFibHeap temp)

Since nodes from lower level will be removed during both increase key and cascading cuts modularized to separate function

Parameters:

temp – node to be removed and added to top level list

Returns:

boolean result for successful addition of a child to top level list

Class HashTagCounter

java.lang.Object

adsproject.HashTagCounter

---

public class **HashTagCounter**

extends java.lang.Object

Reads hashtags from an input file and outputs the top 'N' hashtags when it encounters a request for top hashtags

Author: Jagan

Field Summary

Modifier and Type	Field and Description
private **FibonacciHeap**	**heap**
private java.util.HashMap<java.lang.String,**NodeFibHeap**>	**nodes**
private java.util.ArrayList<**NodeFibHeap**>	**stack**

Constructor Summary

Constructor and Description

HashTagCounter()

Method Summary

Modifier and Type	Method and Description
Void	**addBackRemovedNodes**() Add back the nodes removed while printing top hashtags
Void	**increase**(java.lang.String hashtag, int increase) Increases the count of hashtags on encountering same hashtag
Void	**insert**(**NodeFibHeap** node) Inserts node representing a hashtag
Void	**insert**(java.lang.String hashtag, int data) Inserts a hashtag into Fibonacci heap by calling @FibonacciHeap.insert method
static void	**main**(java.lang.String[] ar)
NodeFibHeap	**remove**() Temporary remove of top 'N' hashtags for outputting trending hashtags
Void	**traverse**(java.util.ArrayList<java.lang.String> list, java.io.PrintWriter printw) Traverses through each line from input file

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

nodes

private java.util.HashMap<java.lang.String,NodeFibHeap> nodes

stack

private java.util.ArrayList<NodeFibHeap> stack

heap

private FibonacciHeap heap

Constructor Detail

HashTagCounter

public HashTagCounter()

Method Detail

insert

public void insert(NodeFibHeap node)

Inserts node representing a hashtag

Parameters:

node – node to be inserted into heap

insert

public void insert(java.lang.String hashtag, int data)

Inserts a hashtag into Fibonacci heap by calling @FibonacciHeap.insert method

Parameters:

hashtag - new tag to insert into heap

data - initial count of the hashtag

remove

public NodeFibHeap remove()

Temporary remove of top 'N' hashtags for outputting trending hashtags

Returns:

top hashtag

increase

public void increase(java.lang.String hashtag, int increase)

Increases the count of hashtags on encountering same hashtag

Parameters:

hashtag - hashtag whose value needs to be increased

increase - increase value

addBackRemovedNodes

public void addBackRemovedNodes()

Add back the nodes removed while printing top hashtags

traverse

public void traverse(java.util.ArrayList<java.lang.String> list, java.io.PrintWriter printw)

Traverses through each line from input file

Parameters:

list - takes list which contains lines from an input file
printw – PrintWriter object to print output to file

main

public static void main(java.lang.String[] ar)

INPUT AND OUTPUT

Takes an input file with hashtags accompanied with frequency in the format “#hashtag freq” where frequency is assumed to be an integer. Output is a sequence of lines with top N hashtags printed to a file named output_file.txt.

CONCLUSION

Compared to other heap structures like Binary and Binomial Heaps, Fibonacci heap has better time complexity in terms of amortized complexity. The time taken to execute a million sample file is 0.8s