COP 5536 Advanced Data Structures Fall 2016

Programming Project

Report

Implementation of Max Fibonacci heap and Hash Table to find n most popular hashtags

Author: VAGISHA TYAGI UFID: 04289808 UF Email: vagisha@ufl.edu

(1) Working Environment

1. Hardware Requirement:
   - Hard Disk space: Minimum 5GB
   - Memory: 512 MB Minimum
2. Operating System:
   - Win32 and above versions like WINDOWS 98/NT/XP/VISTA/7/8
   - Linux
3. Compiler:
   - Standard Java Compiler (JDK)

(2) Run Process

Open the terminal/cmd and change directory to where the files are located.

On terminal/cmd run the make file.

Write – $java hashtagcounter file_name

(3) Structure of program:

Submission consists of four source files. Each file is having functions/methods.

Below are the classes/file and function defined inside them

(A) **Node.java -**

Node class: Class for the structure of a Fibonacci heap node. Contains the key, tagName, degree, parent, left sibling, right sibling, child, childcut as class variables.

Class variables :

public H count;

public String tagName;

public int degree;

public Node<H> parent;

public Node<H> child;

public Node<H> left;

public Node<H> right;

public boolean childCut;

public boolean isMax;

Methods :

1. *public Node()* : It is a constructor method, when argument is not passed, key of object Node is null.
2. *public Node(H count, String tag)*: It is a constructor method, creates the node with given count and tag.
3. Function*: public H getCount()*

   Arguments: none

   Return value: Pointer to count

   Description: It returns the count of Node that invoked this method.
4. Function: *public String getTag()*

   Arguments: none

   Return value: pointer to tagName

   Description: It returns the tagName of Node that invoked this method.
5. Function: *public int compareTo(Node<H> other)*

   Arguments: Node object

   Return value: 1, 0, -1 depending on comparison of keys

   Description: It compares the object that invoked the method to the argument.

(B) **FibonacciMaxHeap.java**:

Contains FibonacciMaxHeap Class for max Fibonacci heap implementation.

Class variables:

private Node<H> root;

private int size;

There are 3 constructors in it named as-

public FibonacciMaxHeap(): To initialize the heap with root and size as null

public FibonacciMaxHeap(Node<H> root, int size): To initialize the heap with given root and size.

public FibonacciMaxHeap(Node<H> node): To initialize the heap with given node and taking size as 1.

Methods:

1. Function: *public Node<H> getMax()*

Arguments: none

Return value: Pointer to root

Description: It returns the maximum value of heap stored as root in heap

2. Function: *public Node<H> insert(H count, string tag)*

Arguments: count and tag to be inserted

Return value: Pointer to Node created

Description: For given count, tag pair it creates a Node object and adds it to the right of root and accordingly sets the root.

3. Function: *public void increaseKey(Node<H> node, H newKey)*

Arguments: Node to be updated and the new key to be inserted into that node

Return value: void

Description: Updates the key value of a given node with a new key value and applies the cascading cut .

4. Function: *public void cut(Node <H> node, Node<H> parent)*

Arguments: Node objects – node and parent

Return value: void

Description: To remove the node when it becomes larger than parent and add it to root list, decreasing degree of parent and setting its pointer to parent as null and childcut as false.

5. Function: *private void cascadingCut(Node<H> node)*

Arguments: Reference to node

Return value: void

Description: Keeps on cascading cut to parent until root is reached if node is marked as true and makes child cut of parent as true if parent is found to be false.

6. Function: *public Node<H> extractMax ()*

Arguments: none

Return value: reference to maximum Fibonacci Heap node

Description: Deletes the root node i.e the node with maximum count value and returns it. Calls combineSameDegree() method to join two trees of equal degree and updates max element i.e root.

7. Function: *public void combineSameDegree()*

Arguments: none

Return value: void

Description: Traverses the root list, linking the traversed trees' degrees to array list, equal degree trees joined together by making one tree by calling linkSmallToBig() function

subtree of the other

8. Function: *private void removeNodeFromList(Node<H> node)*

Arguments: Reference to node to be removed

Return value: void

Description: Removes the given node and its subtree from the heap and updates pointers

9.Function: *private void linkSmallToBig(Node<H> nodeA, Node<H> nodeB)*

Arguments: Reference to small and big key nodes

Return value: void

Description: Removes small key node from list and makes it child of big key list updating the child cut and parent pointers

10. Function*: public Node<H> merge Node<H> a, Node<H> b)*

Arguments: Reference to nodes to be merged together

Return value: object of Node<H>

Description: Adds two nodes lists together updating their pointers

(C) **Traverse.java**: It has Traverse Class which is used to simplify the *combineSameDegree()* method by implementing iterator and using its functions peek() and poll(). It works by gathering a list of the nodes in the list in the constructor since the nodes can change during consolidation.

Class Variables:

private Queue<Node<H>> elements = new LinkedList<Node<H>>

Methods:

1. public Traverse(Node<H> start): Constructor which adds current node to queue and proceed to its right, it keeps on iterating to right and adding to queue till start node appears.
2. public boolean hasNext(): It returns peek() method which retrieves the value of the first element of the queue without removing it from the queue.
3. public Node<H> next(): It returns the poll() method retrieves the value of the first element of the queue by removing it from the queue.

(D) **HashtagCounter.java**: It has Hashtagcounter Class containing the main method

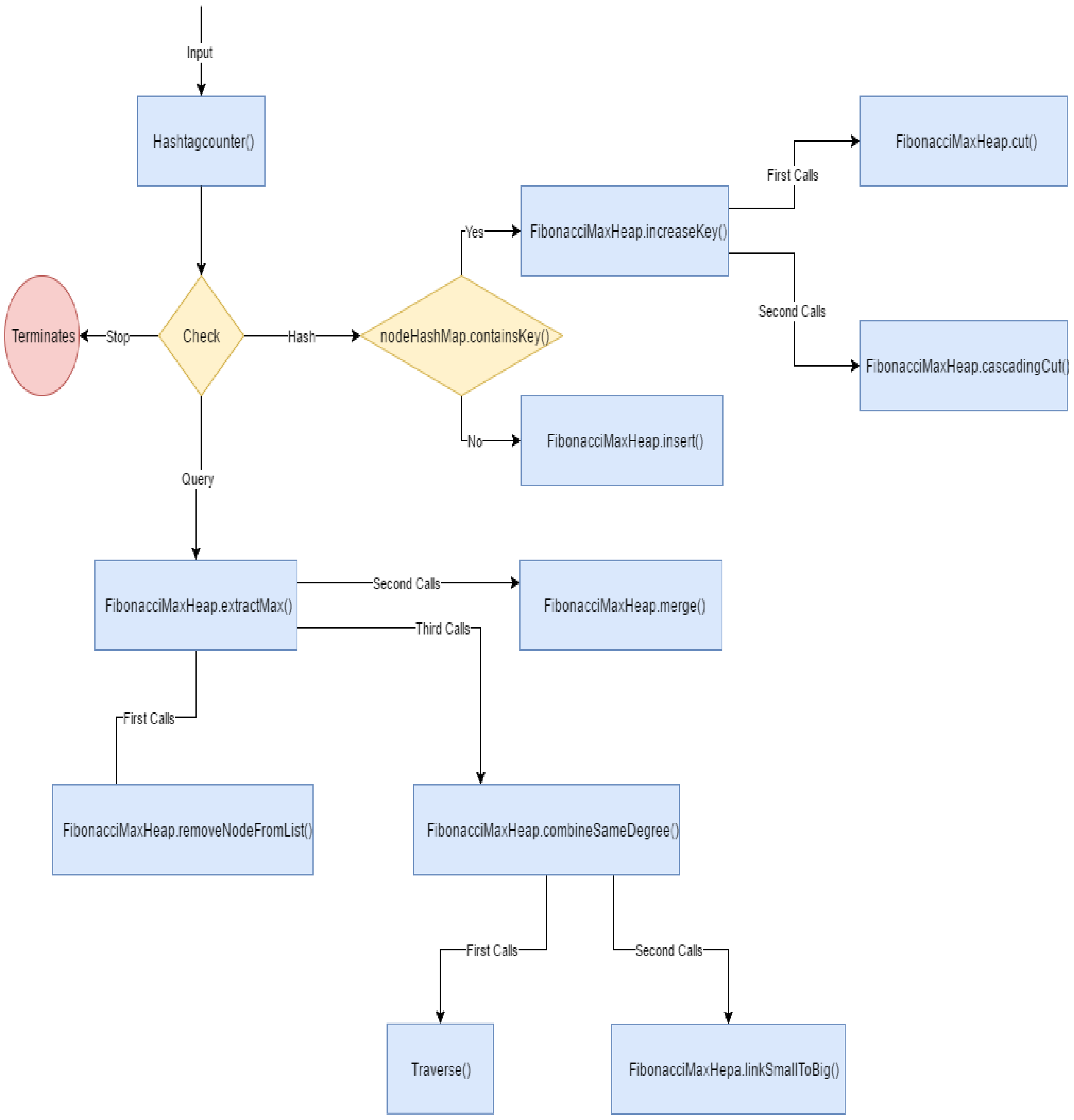
public static void main(String args[]) – This takes inputs as file name and output is returned to output file, also catches IOException.

HashMap<String, Node<Integer>> nodeHashMap = new HashMap<>()

Hash table is created with key as hashtag name and value as reference to node. Different patterns for hash, query and stop is determined and it is matched with the string . When stop appears, loop breaks. When new hashtag appears, its key and value are correspondingly stored in hash map , but when old hashtag appears , value of old count is incremented by new count.

When query is given, method extractMax() is calculated for that number of times and output is written to output file. After performing the query function, extracted values are reinserted into hash table.

Working Structure of program:


Input
Hashtagcounter()
Check
Stop
Terminates
Hash
nodeHashMap.containsKey()
Yes
FibonacciMaxHeap.increaseKey()
First Calls
FibonacciMaxHeap.cut()
Second Calls
FibonacciMaxHeap.cascadingCut()
No
FibonacciMaxHeap.insert()
Query
FibonacciMaxHeap.extractMax()
Second Calls
FibonacciMaxHeap.merge()
Third Calls
First Calls
FibonacciMaxHeap.removeNodeFromList()
FibonacciMaxHeap.combineSameDegree()
First Calls
Second Calls
Traverse()
FibonacciMaxHepa.linkSmallToBig()