Advanced Data Structures(COP 5536)

Fall 2016

Programming Project Report

Raktima Das

UFID 4311 -6109

raktimadas@ufl.edu

PROJECT DESCRIPTION

In this project, we are asked to find out the n most popular hash tags that appear on social media such as Facebook and Twitter. A list of hash tags has been provided to us for the scope of our project. The basic idea of the project is to use a max priority structure to find the most popular hash tags. In this project we will use a max Fibonacci heap to keep track of the frequencies of the hash tags. The goal of this project is to make it space efficient.

The concept of Fibonacci heap was given by Fredman and Tarjan in 1986. The advantages of Fibonacci heap are:

- Decrease key and join takes O(1) time.
- Better theoretical bounds for increase key operation.

The project consists of two parts. The first part is to keep track of the frequencies of the hash tags using a max Fibonacci heap. We take an input file from command line. We read the hash tags and the query values. If it's a hash tag we will add it to our hash map and to our heap as a root node. When we get a query we will extract the max node from the heap and then we do a pair wise merge on the elements of the heap. Similarly, there are other operations that are to be performed such as increase key and delete max and so on.

In the second part, we will get the maximum frequency node from our Fibonacci heap. We will fetch the key which is the hash tag and we will print it accordingly.

We were to assume that there are many hash tags appearing in the stream and we need to perform increase key operation several times.

OPERATION	Fibonacci heap
MAKE -HEAP	1
INSERT	1
FIND-MAX	1
DELETE-MAX	Log N
JOIN	1
INCREASE- KEY	1
DELETE	Log N
IS EMPTY	1

(Table 1: Amortized Complexity of Fibonacci heap)

Working Environment

COMPILING INSTRUCTIONS

The project consists of 3 class files and has been done is Java. The main class is in Hash HashTagCounter.java

The project has been compiled and tested on IntelliJ and thunder.cise.ufl.edu.

To execute the program you can remotely access the server using SSH.

username@thunder.cise.ufl.edu

For compiling we have to run the following:

javac HashTagCounter.java

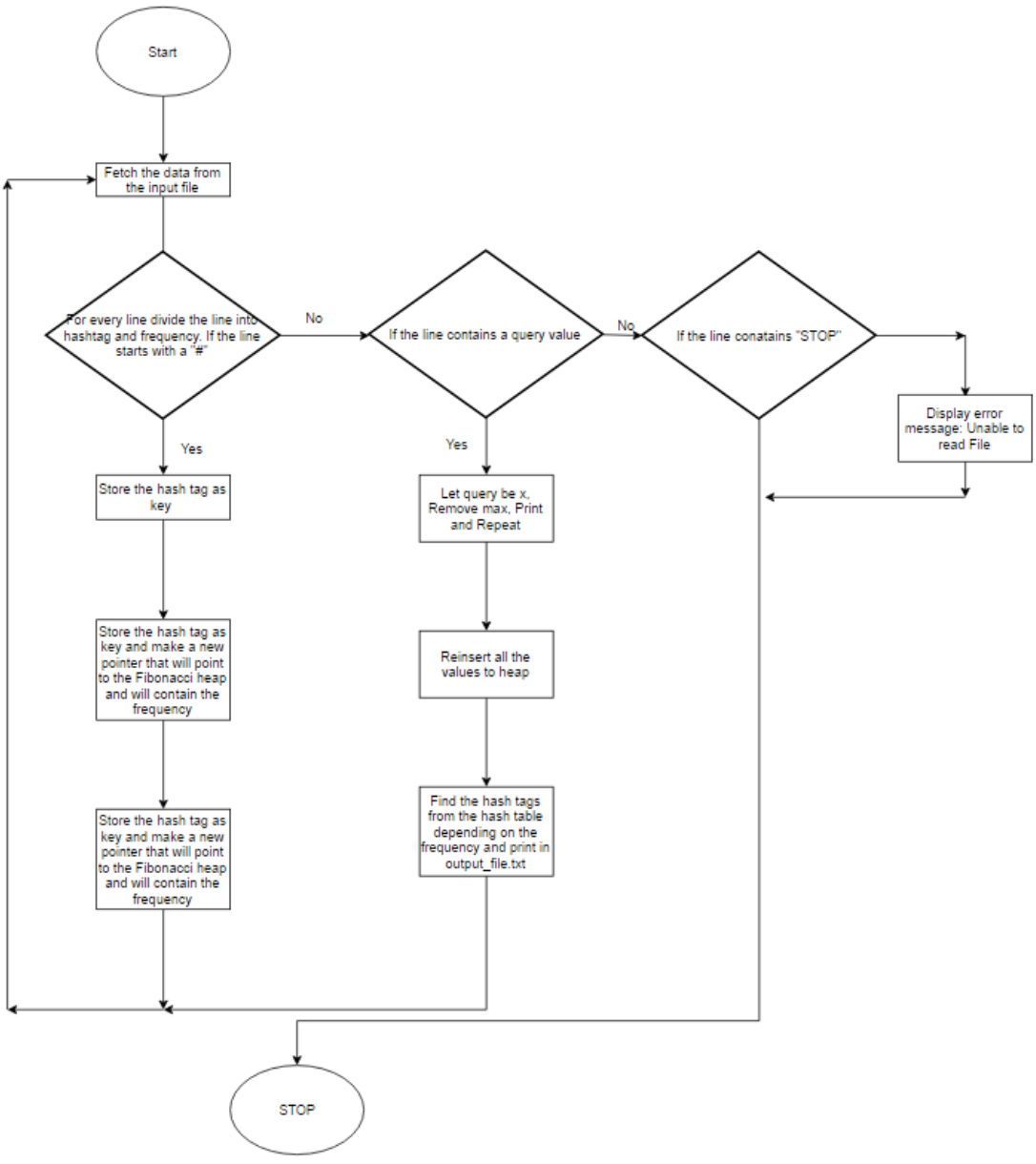
For running the program we have to do:

java HashTagCounter sampleInput.txt

A make file has been provided with source code. “make” will compile the project.

STRUCTURE OF THE PROGRAM AND FUNCTION DESCRIPTIONS

I have used 3 classes in this project. Namely, FibonacciHeapNode.java, FibonacciHeap.java and



HashTagCounter.java. Here's how the functions are defined in each class and a short description of how they work.

FibonacciHeapNode.java

This class mainly describes the structure of a node that is used in a Fibonacci heap.

The data members are:

FREQUENCY	Denotes the number of times a hash tag has appeared in the input file
DEGREE	It is an integer value that denotes the number of child pointers that is the number of nodes a parent can have.
HASH TAG	It is a string value that contains the hash tag
CHILDCUT	Boolean value which is true if the child node has been removed and false if no child has been removed.
CHILDPOINTER	This is of type of Fibonacci heap node which holds reference to the leftmost child of the parent.
PARENT	This is of type Fibonacci heap node which is used to point to its parent.
LEFTPOINTER	This is of type Fibonoacci heap node. It points to the node on the left. If there is only one node the left pointer will point to itself as it is a doubly linked list
RIGHTPOINTER	This is of type Fibonacci heap node. It points to the node on the right. If there is only one node, the right pointer will point to itself as it is a doubly linked list.

(Table 2: Data members of FibonacciHeapNode)

For the implementation of FibonacciHeapNode we have the class as follows:

```
public class FibonacciHeapNode {

        int frequency;
        int degree;
        String hashtag;
        boolean childcut;
        FibonacciHeapNode childpointer;
        FibonacciHeapNode parent;
        FibonacciHeapNode leftpointer;
        FibonacciHeapNode rightpointer;
        public FibonacciHeapNode getRightpointer()
        public void setRightpointer(FibonacciHeapNode rightpointer)
        public FibonacciHeapNode getLeftpointer()
        public void setLeftpointer(FibonacciHeapNode leftpointer)
        public int getDegree()
        public void setDegree(int degree)
        public String getHashtag()
        public void setHashtag(String hashtag)
        public int getFrequency()
        public void setFrequency(int frequency)
        public boolean isChildcut()
        public void setChildcut(boolean childpointercut)
        public FibonacciHeapNode getChild()
        public void setChild(FibonacciHeapNode childpointer)
        public FibonacciHeapNode getParent()
        public void setParent(FibonacciHeapNode parent)

}

Each node has a left pointer, right pointer and pointer to a child.
```

FibonacciHeap.java

In this class all the functions related to the Fibonacci heap have been defined. The functions defined in this class are:

- `public void insert(String hashTag, int cnt)`

  Return type: void

  Parameters: `String hashtag, int cnt`

  This function inserts a node into the hash map named map. If the map does not contain the hash key then this method will insert the hash tag as the key and the value will be a node pointer of the type FibonacciHeapNode. If the hashMap already contains the hash tag value then the frequency of the node will be increased without making any changes to the FibonacciHeapNode pointer.

- `private void increaseKey(FibonacciHeapNode node)`

  Return type: void

  Parameters: FibonacciHeapNode node

  Method increaseKey will take take the node, increase the key and if the heap property is violated that is if the frequency of the child node is greater than the frequency of the parent node then the child is cut from its parent. If the parent is not a root node then the child cut field of the parent is made to true. If it was true already then it is cut and joined to the root level and the child cut value of the parent is also marked. We continue upwards until we reach either the root or a node whose childcut is set to false. Then we set the maximum pointer to the increased value if it is the new maximum.

- `private void insertNode(FibonacciHeapNode node)`

  Return type: void

  Parameters: `FibonacciHeapNode node`

  This function inserts the nodes in the Fibonacci heap. If the heap is null then the new node is added in the heap and both the left and right pointers are pointing to itself. However, if there is a max node present already the new node is added to the right of the max node and the man pointer is updated accordingly.

- `public List<String> getMaxFrequencies(int n)`

  Return type: List<String>

  Parameters: int n

  `private FibonacciHeapNode removeMax()`

  Return type: FibonacciHeapNode

  Parameters: None

`private void join(FibonacciHeapNode children)`

Return type: void

Parameters: FibonacciHeapNode

The function getMaxFrequencies() which calls FibonacciHeapNode removeMax removes the maximum node from the Fibonacci heap. Since max is the root of the tree. On removing the root we will just have to adjust the nodes on the next level of the root. The list of maximum occurring frequency is then returned. The class removeMax calls consolidate() function to meld the subtrees with same degree.

The function removeMax() performs the task of removing the larger node from the doubly linked list.

- `private void consolidate()`

  Return type: void

  Parameters: None

  This function checks which trees have the same degree. If it finds two subtrees with the same degree then it melds the two subtrees of equal degree by making the largest of the root of the two subtrees the root and makes the smaller root and its subtree the child of the tree with larger root.

- `private FibonacciHeapNode meld(FibonacciHeapNode node1, FibonacciHeapNode node2)`

  Return type: FibonacciHeapNode

  Parameters: `FibonacciHeapNode node1, FibonacciHeapNode node2`

  `Combines one subtree with another to form a valid Fibonacci heap.`

- `private void increaseKey(FibonacciHeapNode node)`

  Return type: void

  Parameters: `FibonacciHeapNode node`

  This value increases the frequency value of the node that is passed as an argument. If the frequency of the node goes beyond the frequency of its parent the method will remove the node and will add it as a root node in the linked list using the insertNode function. It will also check the child cut values of the parent and its ancestors.

  Structure of FibonacciHeap.java

```
public class FibonacciHeap {
    private void increaseKey(FibonacciHeapNode node) {
    private void insertNode(FibonacciHeapNode node)
    private FibonacciHeapNode removeMax()
    private void join(FibonacciHeapNode children)
    private void consolidate()
```

```
    private FibonacciHeapNode meld(FibonacciHeapNode node1, FibonacciHeapNode
node2)
}
```

HashTagCounter.java

This class contains the main method which reads the input file given by the user as an argument. It is also used to insert nodes in the Fibonacci heap by calling the insert() method. This is done by making an object of the FibonacciHeap class. It scans the input file for hash tags. If it has found one, it will add it in the doubly linked list. If it encounters a STOP then the processing stops. If it is a query, then it will call getMaxFrequencies() to fetch the required number of hashtags that have appeared in the text file.

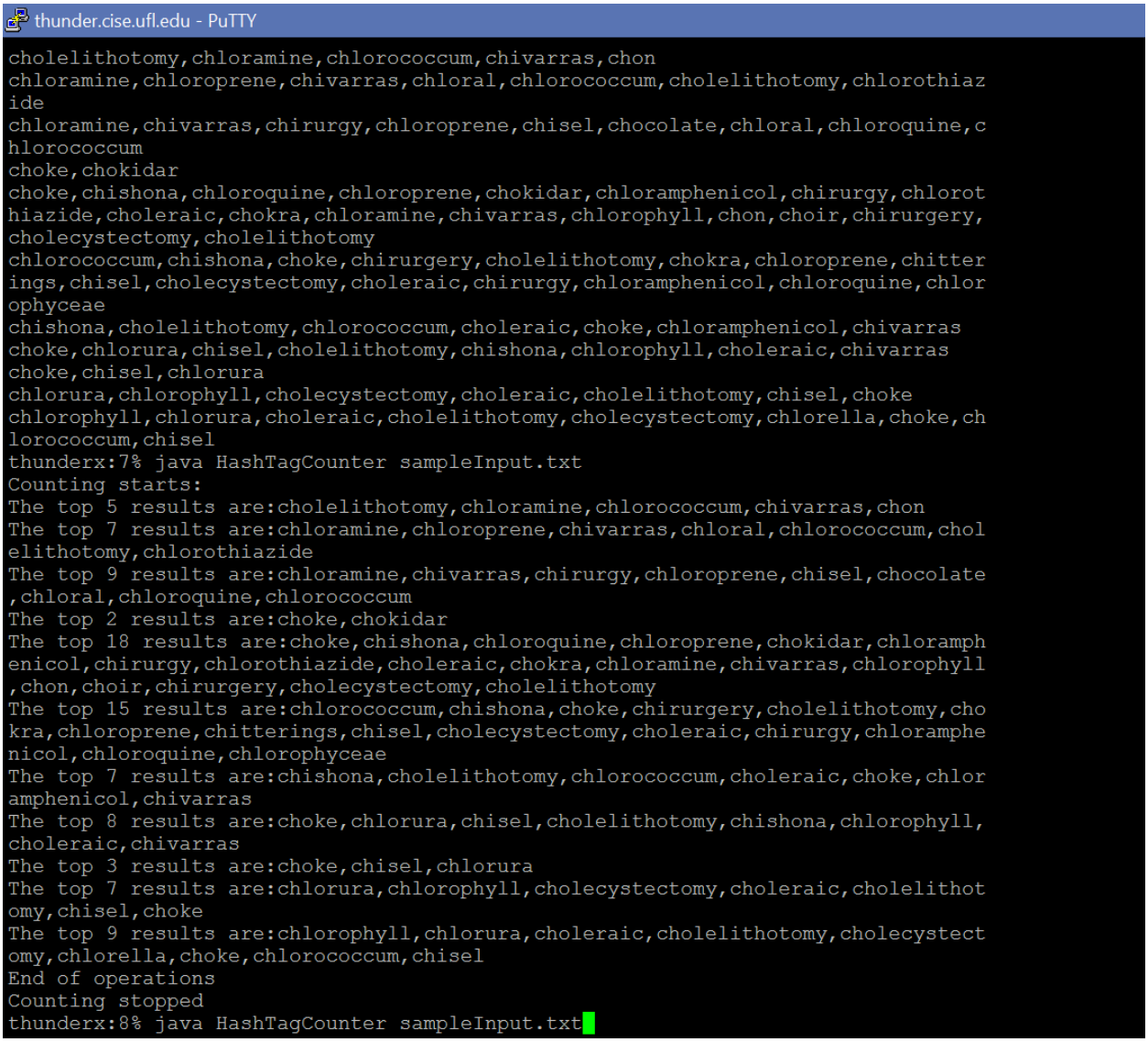
Structure of HashTagCounter:

public class HashTagCounter {

public static void main(String[] args)

}

RUNNING THE PROGRAM



CONCLUSION

The project has been implemented successfully using Fibonacci Heap within acceptable time limits. Further, the output file generated matches perfectly the with ones that have been provided for testing purposes.