Report of Project

Name: Xi Yang

UFID: 16216573

UF email: alexgre@ufl.edu

Details, function prototypes and analysis:

Totally three classes are implemented as three source files:

1. **HashtagCounter.java**
2. **FrequencyStorage.java**
3. **HashtagFreq.java**

The **HashtagFreq** class:

This class is the node class. It represents the node in the Fibonacci heap. 7 private fields are included as 4 pointers as child, left, right, parent, two int types as degree, actual data(freq) and a Boolean type marked(define if a cut child has been performed before).

Prototypes:

public int getFreq()

When getFreq() method is called

It returns the data(freq) of the current node.

The **FrequencyStorage** class:

This class is basically a max Fibonacci heap which store the frequency data obtained by input_file. Basic operations of Fibonacci heap are implemented including insert, get max, delete max, increase key. These methods implemented based on the algorithms covered in class. Therefore, insert runs O(1), get max runs O(1), delete max runs O(logn) and decrease key runs O(1).

Prototypes:

public void insert(HashtagFreq node)

private void removeNode(HashtagFreq node)

private void addNode(HashtagFreq root, HashtagFreq newNode)

private void recombine( )

public HashtagFreq getMax( )

private void cascadeCut(HashtagFreq parent)

private HashtagFreq popMax( )

private void decreaseFreq(HashtagFreq node, int freq)

public void changeFreq(HashtagFreq node, int newFreq)

private void linkTwoHeaps(HashtagFreq child, HashtagFreq root)

private void increaseFreq(HashtagFreq node, int newFreq)

private void cut(HashtagFreq child, HashtagFreq parent)

public void deleteMax( )

public HashtagFreq getHashtagFreqInstance(int freq)

When function insert(node) is called:

The parms node will be added as the left node of the max by rearranging the pointers of max node, original left node of max node and parms node.

When changeFreq(node, newFreq) is called:

The newFreq will be compared with the current value of the node(oldFreq). If newFreq > oldFreq, increaseFreq() function is called to increase the current value to new value for the node; If newFreq < oldFreq, decreaseFreq() function is called to increase the current value to new value for the node; If newFreq = oldFreq, do nothing.

When getMax() is called:

The max node will be returned by using the max pointer.

When deleteMax() is called:

All of sub-trees of max node are detached from it and added to max left one by one using similar method as insert(). Then, the current max node will be detached from the heap, max pointer temporarily moves to the node on its right (if has). Then a private function recombine() is called to iteratively combine all the sub-trees (rooted at top level) with same degree(degree of its root) until no two trees has the same degree.

When increaseFreq(node, newFreq) is called:

The current Freq value of the node will be updated to newFreq value. Then if the new freq value of the node is larger than that of the parent, a cut() and cascadeCut() operations will be performed on the parent of the node. The node will be cut from the current tree and added to the left of the max. If the parent lost child before, the parent also will be cut from its parent and added to the left of the max. This process will be continue until either a parent node never lost child before or the parent node is root. Then update the max pointer if the increased value is larger the that of the max.

When cut(child, parent) is called:

The child node will be detached from parent node and added to the left of the max node. Parent degree decrease by 1.

When cascadeCut(parent) is called:

If the parent node marked value is true and it is not a root, it will be cut from its parent and added to the left of the max node. Its parent node degree decrease by 1.

When decreaseFreq(node, newFreq) is called:

All the child of the node will be detached and added to the left of the max. The node value will be updated to newFreq and detached from its parent and added to the left of the max. If the node has parent, cut() and cascadeCut() methods will be called. If the node is the max node, the max pointer will be updated if necessary after apply the new value.

When recombine() is called:

Create an array with length equals logn. At each index position, a sub-tree with degree same as index will be added into the position. If the position is not empty, the current tree and the tree in this position will be merged by calling linkTwoHeaps() method. Then update the max node to the node with the largest frerq value.

When linkTwoheaps(child, root) is called:

The freq values of child and root are compared in method recombine(), the node who has the larger freq will be root. The child will be added as a child node of root and the marked value of child will be set to false.

When gethashtagFreqInstance(freq) is called:

A new object of HashtagFreq class with freq value will be created and returned to user.

The **HashtagCounter** class:

Prototypes:

Public static void main(String[] args)

When the main(args) is called:

The input_file.txt will be the argument of the main method. The data in input file will be read into the memory. If the read-in data is a hashtag-frequency pair, a search is performed on a hashmap which stores such pairs of data. Using the hashtag as the key, if the key is not exist in the hashmap, the data pair will be added; if the key is exist in the hashmap, the frequency of it will be updated by adding the new data to the original data. If the read-in data is a number n, then the hashtags of the first nth frequencies will be written into an output file. If the read-in data is “stop”, the program will be terminated.

The overall flowchart of the structure of the program: