# COP 5536 Project Report

# Hashtag Counter

**Name: Zirun Lin** **UFID:[redacted]** **UF Email:[redacted]@ufl.edu**

## 1. Project Structure

For this hashtag counter project, the package MaxFibonacciHeap is the implementation of a max Fibonacci heap which contains two files (Node and Implement). Node is built for creating a single node in a max Fibonacci heap while Implement is the implementation of the structure and functions of a max Fibonacci heap. I will explain the detail of my submission.

## 2. MaxFibonacciHeap

### (1) Node

Node is built for creating a single node in a max Fibonacci heap, it has several attributes including keyword, frequency, childCut, degree, parent, leftmosetChild, leftSibling and rightSibling. When we insert a new node, it will be inserted into the top list of a max Fibonacci heap, therefore, I set childCut = null, degree = 0, parent = null, leftmostChild = null, leftSibling = null and rightSibling = null. Obviously, when we initialize a node, we must set keyword and frequency of the hashtag.

### (2) Implement

Implement is built for the implementation of a max Fibonacci heap, it has several functions including insertNode(Node), cascadingCut(Node), combineTwoNodes(Node1, Node2), increaseKey(Node, int), remove(Node), removeMax(), etc.

- insertNode (inNode):
  This is the function for inserting a new node into a max Fibonacci heap. This new node will be inserted into the top list. if its frequency is larger than the current max node, max pointer will point to this new node.

| Function Name | insertNode |
|---|---|
| Params | inNode |
| Return | void |

- lostOneChild (child, parent):
  This function simulates the condition when a node loses a child. If the node who lost a child has only one child, I set its leftmostChild to null. If the parent node has more than 1 node, the lost child's rightSibling will be the parent node's leftmostSibling

| Function Name | lostOneChild |
|---|---|
| Params | child, parent |
| Return | void |

- removeOneNode (rmNode, parent):
  This function simulates the condition when we remove one node from its parent. I assume that the removed node has a parent, therefore, it is not the remove function and just for supporting other functions like cascadingCut and increaseKey. When a child is removed from its parent, parent's degree will decrease by 1. Removed node will be reinserted into heap, parent will execute cascadingCut function.

| Function Name | removeOneNode |
|---|---|
| Params | rmNode, parent |
| Return | void |

- cascadingCut (ccNode):
  This function is the implementation of the cascading operation. If ccNode's childCut is false, set it to true. If ccNode's childCut is true, ccNode's parent will execute cascadingCut.

| Function Name | cascadingCut |
|---|---|
| Params | ccNode |
| Return | void |

- increaseKey (ikNode, icValue):
  This function is the implementation of the increaseKey operation. The icValue will be added to ikNode's frequency. If ikNode has a parent and has a larger frequency, it will execute removeOnenode. If ikNode has no parent and has the max value of frequency, max pointer will point to ikNode.

| Function Name | increaseKey |
|---|---|
| Params | ikNode, icValue |
| Return | void |

- combineTwoNodes (node1, node2):
  This is the function for combining two nodes and supporting removeMax function. I assume that node1 has a larger frequency value. node2 will be a child of node1.

| Function Name | combineTwoNodes |
|---|---|
| Params | node1, node2 |
| Return | void |

- remove (rmNode):
  This is the implementation of removeNode function in a max Fibonacci heap. If the node is max node, return removeMax(). Else if the rmNode has children, I reinsert its children to the top list of this heap. If rmNode has a parent, its parent will execute cascadingCut.

| Function Name | remove |
|---|---|
| Params | rmNode |
| Return | Node |

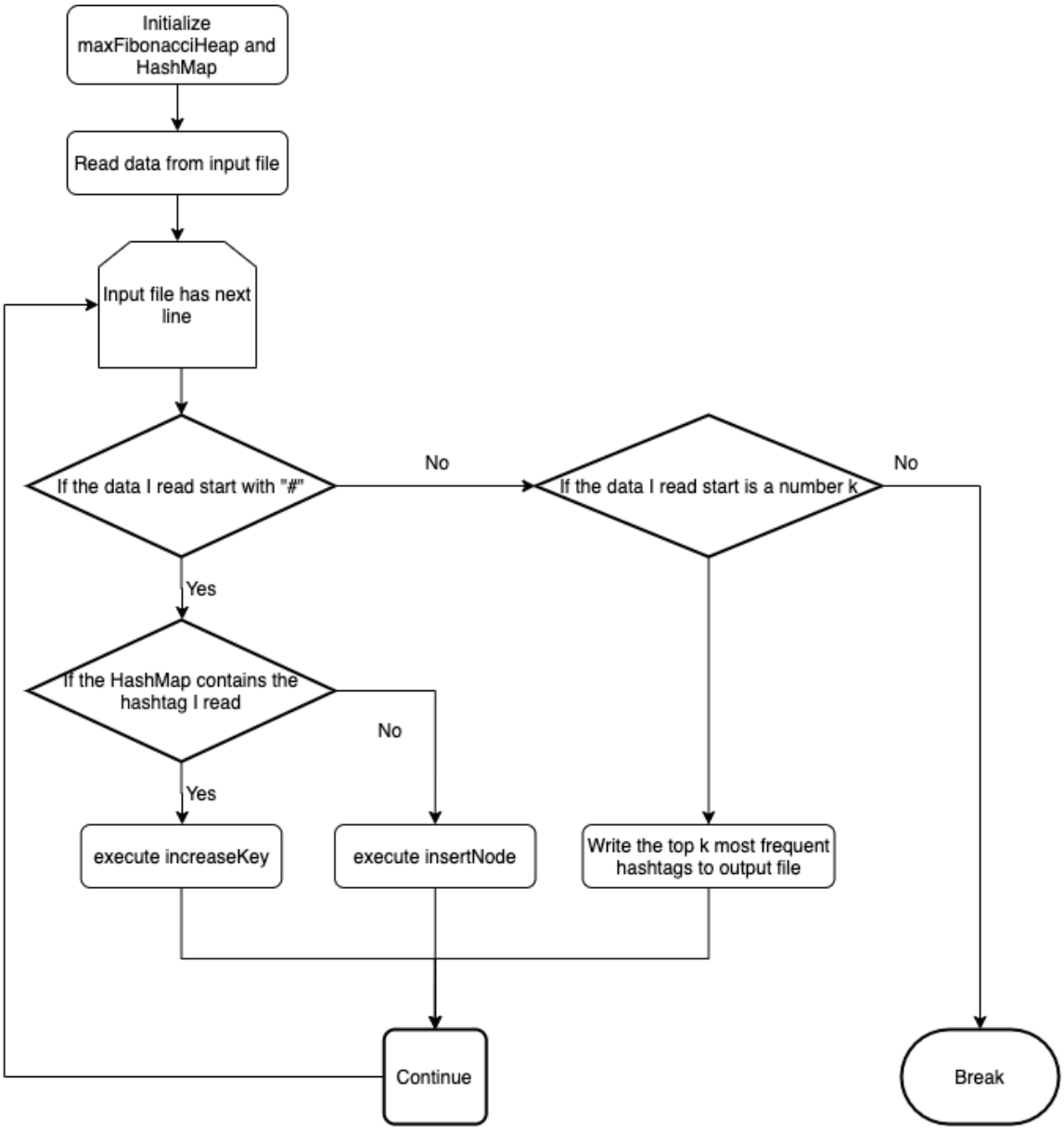
- removeMax():
  This is the implementation of remove max in a max Fibonacci heap. If max node has children, they will be reinserted into this heap and max pointer will point to the current node with largest frequency. I use a HashMap to record the degree of nodes in the top list and meld them.

| Function Name | removeMax |
|---|---|
| Params | rmNode |
| Return | Node |

## 3. hashtagcounter

For this hashtagcounter file, I will initialize IO streams first then start to read data from input file and create the output file. If we specify the output file ($java hashtagcounter <input_file> output_file.txt), it will only write the result to output file. If we do not specify the output file ($java hashtagcounter <input_file>), it will write result to both output file and console.
During the process of reading file and counting hashtags, if there is no next line, break. I use a HashMap to store the hashtag and the corresponding node in the max Fibonacci heap. If the data we read is a number k, I will output top k most frequent hashtags and use comma to separate these hashtags. Finally, I will reinsert these hashtag nodes back to the max Fibonacci heap.

Flowchart is shown below:


Initialize maxFibonacciHeap and HashMap
Read data from input file
Input file has next line
If the data I read start with "#"
No
If the data I read start is a number k
No
Yes
If the HashMap contains the hashtag I read
No
Yes
execute increaseKey
execute insertNode
Write the top k most frequent hashtags to output file
Continue
Break