Advances Data Structures (COP 5536) FALL 2016

Programming Project Report

DEVYASH SANGHAI UFID

33831-5642

[devyashsanghai@ufl.edu](mailto:devyashsanghai@ufl.edu)

# PROJECT DESCRIPTION

The goal of this project is to implement a system to find the n most popular hashtags appeared on social media such as Facebook or Twitter. For the scope of this project hashtags are taken as an input file. Basic idea for the implementation is to use a max priority structure to find out the most popular hashtags.

The project uses the following data structure.

1. Max Fibonacci heap: Use to keep track of the frequencies of Hashtags.

2. Hash table(Hash Map in java) : Key for the hash table is hashtag and value is pointer to the corresponding node in the Fibonacci heap.

The project is written in JAVA. I have implemented Fibonacci Heap in java and stored the address of all the nodes in a Hash Map (Built in Data Structure). I have written the project using JAVA without any external in build data structure. Max Fibonacci heap is required because it has better theoretical bounds for increase key operation.

A Fibonacci heap is a data structure for priority queue operations, consisting of a collection of heap-ordered trees. It has a better amortized running time than many other priority queue data structures including the binary heap and binomial heap. Michael L. Fredman and Robert E. Tarjan developed Fibonacci heaps in 1984 and published them in a scientific journal in 1987. They named Fibonacci heaps after the Fibonacci numbers, which are used in their running time analysis.[1]

|  |  |
| --- | --- |
| **Fibonacci Max Heap** | |
| Amortised Complexity | |
| Space | O(1) |
| Search | O(1) |
| Insert | O(log n) |
| Delete | O(1) |
| Find Max | O(1) |
| Delete Max | O(log n) |
| Increase Key | O(1) |
| Merge | O(1) |

# WORKING ENVIRONMENT

## HARDWARE REQUIREMENT

Hard Disk space: 4 GB minimum Memory: 512 MB

CPU: x86

## OPERATING SYSTEM

LINUX/UNIX/MAC OS(If using other OS make command wont work)

## COMPILER

Javac

# COMPILING & RUNNING INSTRUCTIONS

The project has been compiled and tested on thunder.cise.ufl.edu and java compiler on local machine.

To execute the program,

You can remotely access the server using ssh [username@thunder.cise.ufl.edu](mailto:username@thunder.cise.ufl.edu)

For running the Hash Tag Counter

1. **Extract the contents of the zip file**
2. **Type ‘make’ without the quotes.**

## Type ‘java hashtagcounter ‘file path/input\_file\_name.txt’ ’ without the quotes and add the file name and pathI’ve included the file sampleInput.txt .

**STRUCTURE OF THE PROGRAM AND FUNCTION DESCRIPTIONS**

The program consists of 3 classes.

1) hashtagcounter - The main class that reads and writes the output.

2) Node – This class is used to instantiate and object of node in memory.

3) FibonacciHeap – This class is used to instantiate the methods and functions of the Heap class.

The basic workflow of the program is the hashtagcounter.class takes input file and reads it creates a hash map and performs insert node operations on an instance of the Fibonacciheap.class by creating new nodes using Node.class. It performs remove max as soon as it finds a single digit in the input file. And then prints it in the output\_file.txt, after which it reinserts the node back into the hashmap.

The detailed overview of all the class functions is given below.

**hashtagcounter.java**

|  |  |  |
| --- | --- | --- |
| Variables | Type | Description |
| 1. pathtofile | String | The input filename take from user. |
| 1. hm | HashMap<String,Node> | The HashMap where node and hashTag are stored. |
| 1. fh | FibonacciHeap | The instance of the Fibonacci heap. |
| 1. file | String | Name of the outputfile. |
| 1. p | Pattern | Stores the pattern of the input given, the format of the input.  ([#])([a-z\_]+)(\\s)(\\d+) |
| 1. p1 | Pattern | Stores the pattern of the digits to be removed from the hashTag |
| 1. m | Matcher | Matcher object for pattern ‘p’ |
| 1. m1 | Matcher | Matcher object for pattern ‘p1’ |
| 1. hashTag | String | HashTag to be stored. |
| 1. key | Integer | Key value to be stored in node. |
| 1. increaseKey | Integer | Old key value+ new key value. |
| 1. removeNumber | Integer | Number of nodes to be removed |
| 1. removedNodes | ArrayList<Node> | Stores all the removed nodes |
| 1. startTime | Time(milliseconds) | Start time of program. |
| 1. totalTime | Time(milliseconds) | End time of the program. |

**Node.java**

|  |  |  |
| --- | --- | --- |
| **Class Variable** | **Type** | **Description** |
| Degree | Integer | Signifies the number of nodes that a node can have in its next level. |
| hashTag | String | Contains the Hash Tag. |
| key | Integer | Signifies the value of integer. |
| left | Node | Points to the left node in the circular list. |
| right | Node | Points to the right node in the circular list. |
| parent | Node | Points to the parent node |
| child | Node | Points to the child node. |
| childCut | Boolean | A Child Cut of false means that no child has ever been removed from that node. |

|  |  |  |
| --- | --- | --- |
| **Function Name** | **Return Type** | **Parameters** |
| 1. **Node(String hashTag, int key)** | - | String hashTag, int key |
| 1. **public String getHashTag()** | String | - |

|  |  |
| --- | --- |
| **Function Name** | **Discription** |
| 1. **Node(String hashTag, int key)** | Constutor that initializes the hashtag and key |
| 1. **public String getHashTag()** | Returns the hash Tag of the node. |

**FibonacciHeap.java**

|  |  |  |
| --- | --- | --- |
| **Class Variable** | **Type** | **Description** |
| 1. maxNode | Node | Points to the maximum node in the heap. |
| 1. numberOfNode | Integer | Signifies the number of nodes that are stored in the heap. |

|  |  |  |
| --- | --- | --- |
| **Function Name** | **Return Type** | **Parameters** |
| 1. **public void insert(Node node)** | Void | Node node |
| 1. **public void cut(Node x, Node y)** | Void | Node x, Node y |
| 1. **public void cascadingCut(Node y)** | Void | Node y |
| 1. **public void increaseKey(Node x, int k)** | Void | Node x , int k |
| 1. **public Node removeMax()** | Node | Null |
| 1. **public void degreewiseMerge()** | Void | Null |
| 1. **public void makeChild(Node y, Node x)** | Void | Node y, Node x |

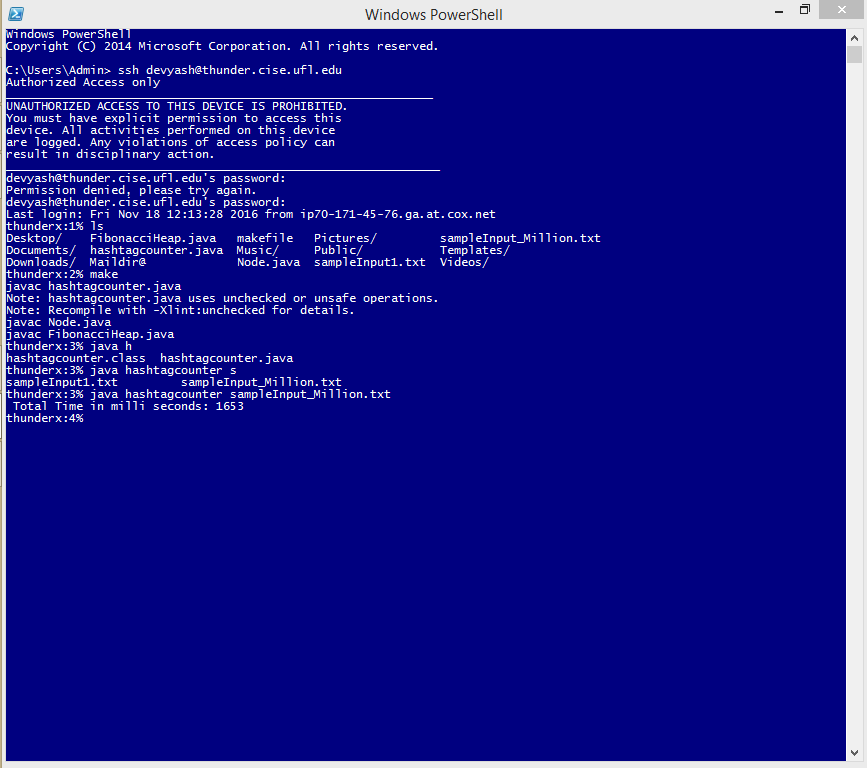
|  |  |
| --- | --- |
| **Function Name** | **Description** |
| 1. public void insert(Node node) | This function inserts a node into the max Fibonacci heap. The function works in two ways. If the max is null i.e if the maxNode is null, maxNode is assigned to be the root of the Fibonacci heap since it is the only node in the Fibonacci heap right now. However, if there is a maxNode , myNode is added to the top level of the Fibonacci heap, to the right of the max root in the doubly linked list of that level. It increases the numberOfNode by 1. |
| 1. public void cut(Node x, Node y) | This function performs cut operation on Node y. It cuts Node x from Node y, then it decreases the degree of Node. |
| 1. public void cascadingCut(Node y) | This function checks the childCut value and makes the necessary changes and performs remove if needed. If the childcut value is false for the parent, make it true. If the childcut is true, keep going up the tree and removing the nodes until it finds a node whose childcut is false. It calls itself recursively till the node has a parent |
| 1. public void increaseKey(Node x, int k) | This function is used to increase the value of the key to k in Node x. The key value of parent is checked, and if it is less than the parent the node is cut and cascading cut is performed on the parent if the parent childCut is true. |
| 1. public Node removeMax() | This function removes the maximum node, from the Fibonacci heap. And while there are children of max node, put them on root. And them call degreewisemerge(). |
| 1. public void degreewiseMerge() | This function performs degreewisemerge. It melds the nodes in the root list of the heap. |
| 1. public void makeChild(Node y, Node x) | This function is used to make Node y a child of Node x. |

**RESULTS**

The code was run for 1 million inputs and the output time required was found to be **1653ms.**

The order of the output can be different as there are many nodes with same values when remove max was called.

# RUNNING THE PROGRAMS

****

**CONCLUSION**

The objective of this assignment has been met. The program successfully creates a implementation of Fibonacci Heap. While correctly performing the removeMax and Increase Key operation on a Max Fibonacci Heap.

REFERENCES:

[1]Fibonacci heap. (n.d.). Retrieved November 18, 2016, from https://en.wikipedia.org/wiki/Fibonacci\_heap