**COP 5536: Advanced Data Structures**

**Fall 2018**

Programming Project Report – Max Fibonacci Heap

Name: Prahlad Misra

UFID: 0048-9999

[prahlad.misra@ufl.edu](mailto:prahlad.misra@ufl.edu)

# PROJECT DESCRIPTION

**Problem statement:** A new search engine “DuckDuckGo” is implementing a system to count the most popular keywords used in their search engine. They want to know what the 𝑛 most popular keywords are at any given time. You are required to undertake that implementation. Keywords will be given from an input file together with their frequencies. You need to use a max priority structure to find the most popular keywords.

The entire project has been implemented in Java and for the scope of this project, keywords are taken as an input file. Basic idea for the implementation is to use a max priority structure to find out the most popular keywords. No in-built functions or complex data structures have been used.

The project uses the following data structure -

* **Max Fibonacci heap**: to keep track of the frequencies of keywords.
* **Hash table**: keywords should be used as keys for the hash table and value is the pointer to the corresponding node in the Fibonacci heap.

|  |  |
| --- | --- |
| **Fibonacci Max Heap** | |
| Amortized 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) |

# COMPILING & RUNNING INSTRUCTIONS

The project has been compiled and tested on thunder.cise.ufl.edu and compatible with the compilers and run on the following environment:

- Java on the thunder.cise.ufl.edu server

The makefile document has been provided, which creates an executable file named keywordcounter.

To execute the program, you can remotely access the server using ssh [username@thunder.cise.ufl.edu](mailto:username@thunder.cise.ufl.edu)

Steps to run the keywordcounter -

1. **Extract the contents of Misra\_Prahlad.zip.**
2. **Type ‘make’ without the quotes.**

## Type ‘java keywordcounter ‘input\_file\_name.txt’ without the quotes.

**STRUCTURE AND FUNCTION DESCRIPTIONS**

The program consists of 3 classes.

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

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

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

**keywordcounter.class** reads the input file, 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 extract max when a single digit is encountered in the input file and it writes all the ***n*** popular keywords, in the output\_file.txt, after which it reinserts the node back into the hashmap.

**keywordcounter.java**

|  |  |  |
| --- | --- | --- |
| **Variable** | **Type** | **Description** |
| filePath | String | The input filename take from user. |
| hm | HashMap<String, Node> | The HashMap where node and hashTag are stored. |
| fh | FibonacciHeap | The instance of the Fibonacci heap. |
| file | String | Name of the outputfile. |
| hashTag | String | HashTag to be stored. |
| key | Integer | Key value to be stored in node. |
| increaseKey | Integer | Old key value + new key value. |
| removeNumber | Integer | Number of nodes to be removed |
| removedNodes | ArrayList<Node> | Stores all the removed nodes |
| startTime | Time(milliseconds) | Start time of program. |
| totalTime | Time(milliseconds) | End time of the program. |

**Node.java**

|  |  |  |
| --- | --- | --- |
| **Variable** | **Type** | **Description** |
| degree | Integer | Signifies the number of nodes that a node can have in its next level. |
| hash | 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(marked) | Boolean | A Child Cut of false means that no child has ever been removed from that node. |

|  |  |  |
| --- | --- | --- |
| **Function Name** | **Return Type** | **Description** |
| Node (String hash, int key) | - | Constructor to initialize hashtag and key |
| public String getHashTag() | String | Returns hashtag of the node |
| public int getKey() | Integer | Returns key of the node |

**FibonacciHeap.java**

|  |  |  |
| --- | --- | --- |
| **Variable** | **Type** | **Description** |
| maxNode | Node | Points to the maximum node in the heap. |
| numberOfNodes | Integer | Number of nodes that are stored in the heap. |

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

|  |  |
| --- | --- |
| **Function Name** | **Description** |
| 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. |
| 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. |
| 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 |
| 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. |
| public Node extractMax() | 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 pairwiseCombine(). |
| public void pairwiseCombine() | This function performs a degree-wise merge. It melds the nodes in the root list of the heap. |
| 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 a test input of **1 million** keywords. Time taken was **1023 ms**.

As described in the project description, the order of the output may be different if multiple keywords contain the same frequency.

The objective of this assignment has been met. The program successfully implementats a Max Fibonacci Heap, whilst performing the extractMax and Increase Key operation on a Max Fibonacci Heap, based exactly on the algorithm. There are no exceptions, or test cases on which the program won’t run, if the input file contains all keywords, based on the “assumptions” provided.

It does not throw any exception for a wrong input format.