Advances Data Structures (COP 5536)

Fall 2016

Programming Project Report

Karan Acharekar

UFID 3868-9483

karanacharekar@ufl.edu

PROJECT DESCRIPTION

This project involves implementing a system to find the n most popular hashtags appeared on social media such as Facebook or Twitter.

This project requires a max priority structure to find out the most popular hashtags.
Implementation requires following structure:

1. Max Fibonacci heap: use to keep track of the frequencies of hashtags.
2. Hash table: Key for the hash table is hashtag and value is pointer to the corresponding node in the Fibonacci heap.

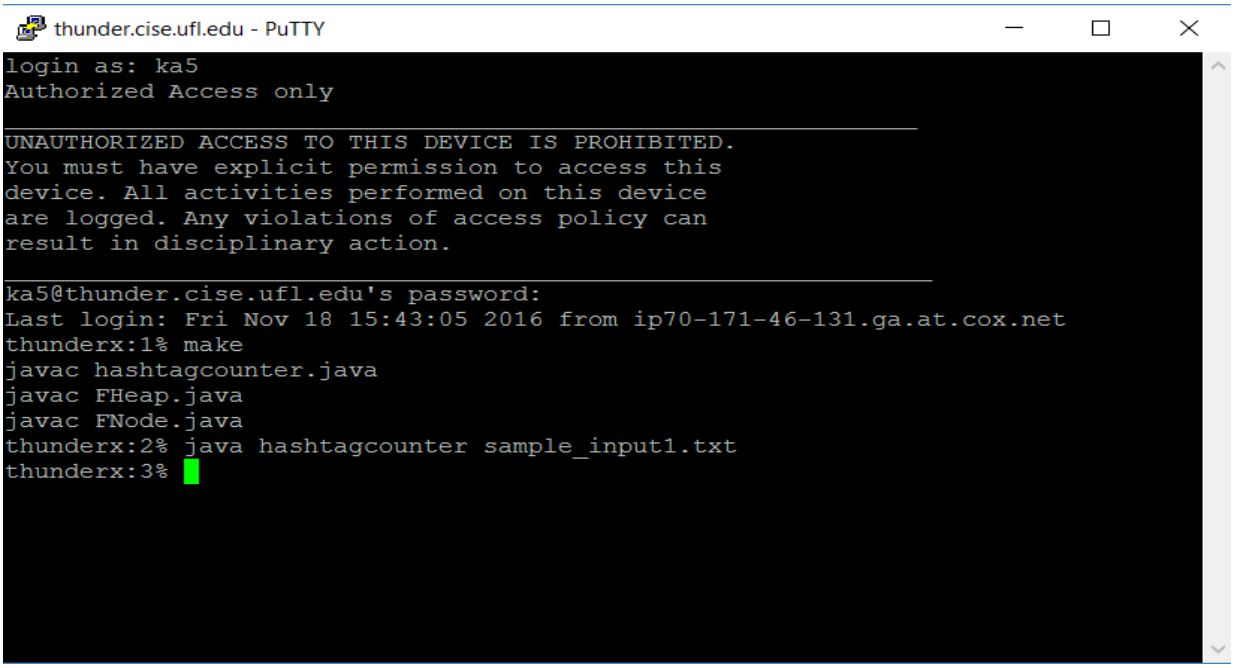
MACHINE DETAILS

1. Processor
    - Intel Core i7-6500U CPU @ 2.50 GHz 2.60 GHz
2. Memory
    - 16GB DDR4 (RAM)
    - 256 GB SSD (Internal Memory)
3. Graphics
    - Intel HD Graphics 520

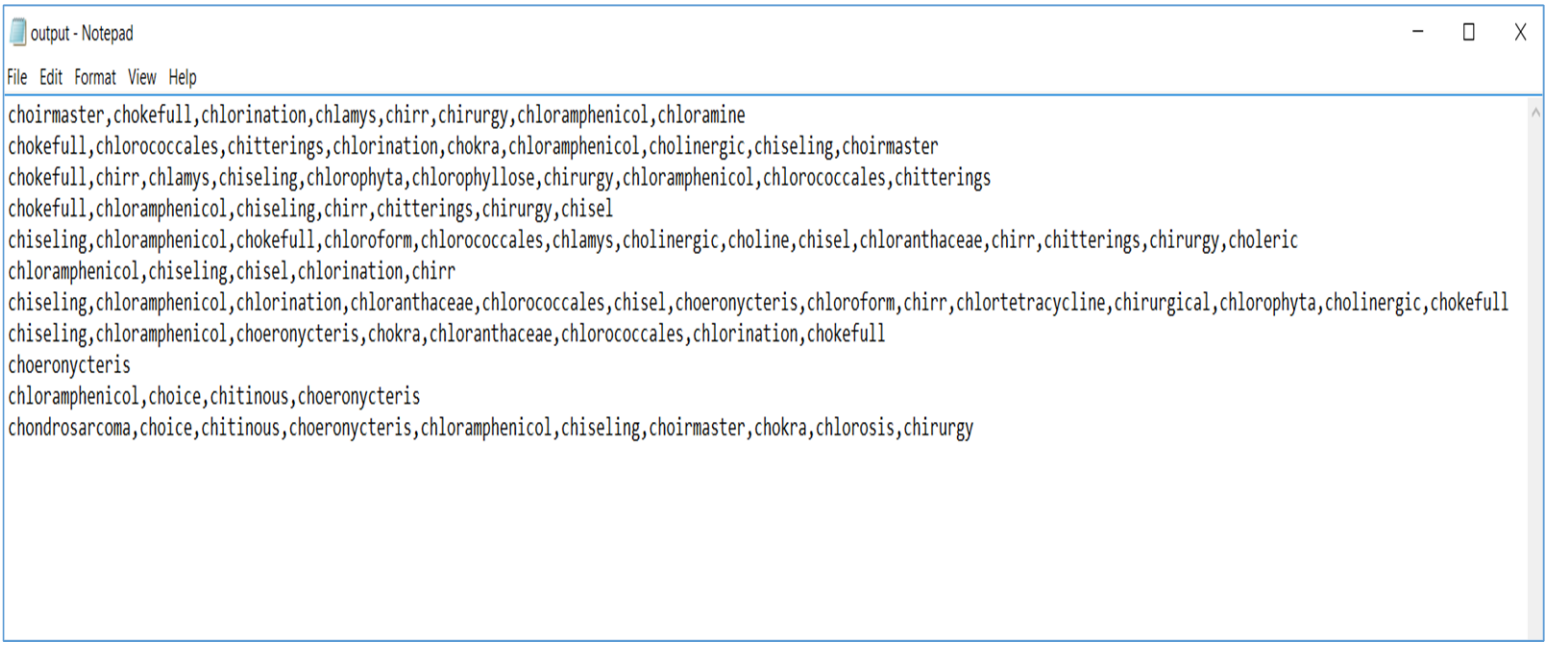
COMPILER USED

Java Compiler version: 1.8.0_111
Java Runtime Environment version: 1.8.0_111-b14

COMPILING AND RUNNING



OUTPUT



STRUCTURE OF THE PROGRAM

Three classes have been used to implement the project. They are FNode.java, FHeap.java and HTCounter.java

I. Class FNode

This class describes the structure of the Node. It contains definitions and constructors to create a node for the Fibonacci Heap.

- Variables
  Left - stores left child of node
  Right - stores right child of node
  Parent - stores parent of node
  Child - stores child of node
  Degree - stores degree of the node
  Hashname - stores the hashname corresponding to the node
  Data - stores the count of the hashtag
  Childcut - stores child cut value of the node (true or false)
- FNode() - Constructor used to initialize to default values

II. Class FHeap

This class contains all the Heap operations

- **insert(FNode fn)**
  Return type: void
  This function inserts a node into the Fibonacci heap. If it is the first node to be inserted, it assigns the maxpointer to it. For every successive node, it inserts and checks whether it greater than the node pointer by maxpointer. If yes, it assigns the maxpointer to this newly inserted node.

- **increaseKey(FNode fn, int increment)**

  Return type: void
  This function is used the increase the count of the hashtag that has already occurred once by the amount specified by the variable increment. If after increment, the key of the node becomes greater than its parent, it calls the cut() function. If the parent of the removed node is not null, it also calls the cascadingCut() function.

- **cascadingCut(FNode fnparent, FNode fnpp)**

  Return type: void
  This function checks the child cut of node it has been passed. If it is false, it sets it to true. If it is true, it performs cut() and cascadingCut() recursively until it reaches a parent whose childcut is false, upon which it is set to true or until it reaches the root level.

- **cut(FNode fn, FNode fnparent)**

  Return type: void
  This function when called, it detaches the node from the doubly linked-list. If it is the only child, it is just inserted at the root level by calling the insert() function. If it has siblings, it adjusts the left, right, parent and parent's child link before inserting it at the root level.

- **removeMax()**

  Return type: maxnode
  This function is used to remove the max node i.e. the one pointer by the maxpointer. It first traverses all the root level nodes in the doubly linked-list using the exchange pointer and performs pairwise combine recursively. This is done until only the node pointer pointed by maxpointer remains at the root level. It then checks if the node pointed by the maxpointer has children. If no, this node is returned. If yes, then the child of maxpointer becomes the right child of maxpointer and the last sibling becomes the left child of maxpointer. This creates another root level doubly linked-list which again traversed using exchange pointer and combined using pairwisecombine(). The function then iterates over combinedtreehashtable and inserts all the nodes at the root level by calling the insert() function. Now we have a new maxpointer.

- **pairwisecombine(FNode fnchild, Hashtable<Integer, FibonacciNode> combinedtreehashtable)**

  Return type: void
  This function combines trees of similar degree by inserting the smaller node below the greater node. It first checks if the combinedtreehashtable contains a node with same degree as node which is passed to it as argument. If not, it just inserts it into the hash table using its degree as key. If yes, it removes the node from the hash table, combines it with the node passed to it. The node with smaller key becomes the child of node with larger key. It then reinserts it into the hash table with its new degree as the key. This is done recursively.

III. Class hashtagcounter

This class contains the public static void main. The program flow starts from here.

- It makes use of Buffered Reader and Buffered Writer to read from and write to the file.
- It reads the input file one line at a time and checks if it is a hashtag, a query or a stop sign. It makes use of a switch case to handle all the three events.
- If it is a new hashtag it inserts it into the hash table as well as the Fibonacci heap. If it is a repeating hashtag it calls the increaseKey() function.
- If it is a query, it calls the removeMax() function for the number of times specified in the query. The max nodes are then printed to the output file and then reinserted into the Fibonacci heap.
- If it is a 'stop' we halt the program.