COP 5536 Spring 2020 - Programming Project

Name: Divyareddy Surapa Reddy | **UFID**: 1372-7408 | **Email id**: dsurapareddy@ufl.edu

Problem Statement

To implement a system that finds the most ‘n’ popular hashtags appearing on social media platforms like facebook or twitter.

Requirements

- *Input File* containing the mix of the following:
    1. Hashtags one per line, starting with # sign. Also the count of the corresponding hashtag is mentioned.
    2. An integer (n), where n top hashtags must be printed to the output file.
    3. Line with “stop” which indicates that the program should terminate.
- *Output File*
    1. Each time an integer (n) occurs in the input file, the top n most popular hashtags calculated will be written to the output file in descending order.
    2. The format in which it should be written to the output file is a comma separated list of n hashtags with no space in between them.

Data Structures

The below data structures are used for the execution of project

- *Max Fibonacci Heap*
    1. Heap stores the details of each hashtag and it’s respective frequencies count
    2. At every level of the heap, each node is linked to each other using a circular doubly linked list.
    3. Max pointer at the top level points to the hashtag with the most count.
- *Hash Table*
    1. Each unique hashtag is stored as the key
    2. Object the respective hashtags in the heap is stored as it’s value.

Program Execution

1. Unzip **Surapa_Reddy_Divyareddy.zip**
2. cd **Surapa_Reddy_Divyareddy** | run **make** command
3. Execute **./hashtagcounter <input_filename> <output_filename>**
4. Output will be in **<output_filename>** or on stdout as executed

Structure of Project

- hashTagCounter.cpp
- Makefile

Execution details

hashTagCounter.cpp

1. The node structure containing details of the hashtag is represented using a class called "**HashTagNode**"

class HashTagNode { public:	Description of each member of the class:
int data;	1. data : Frequency of each hashtag (count)
HashTagNode *leftSibling;	2. leftSibling : Pointer to left sibling
HashTagNode *rightSibling;	3. rightSibling : Pointer to right sibling
HashTagNode *parentNode;	4. parentNode : Pointer to parent node
HashTagNode *childNode;	5. childNode : Pointer to child node with max value
int degree;	6. degree : Number of children
bool cascadeCut;	7. cascadeCut : Bool to check if cascade cut is possible or not
string name;	8. name : HashTag name
};	

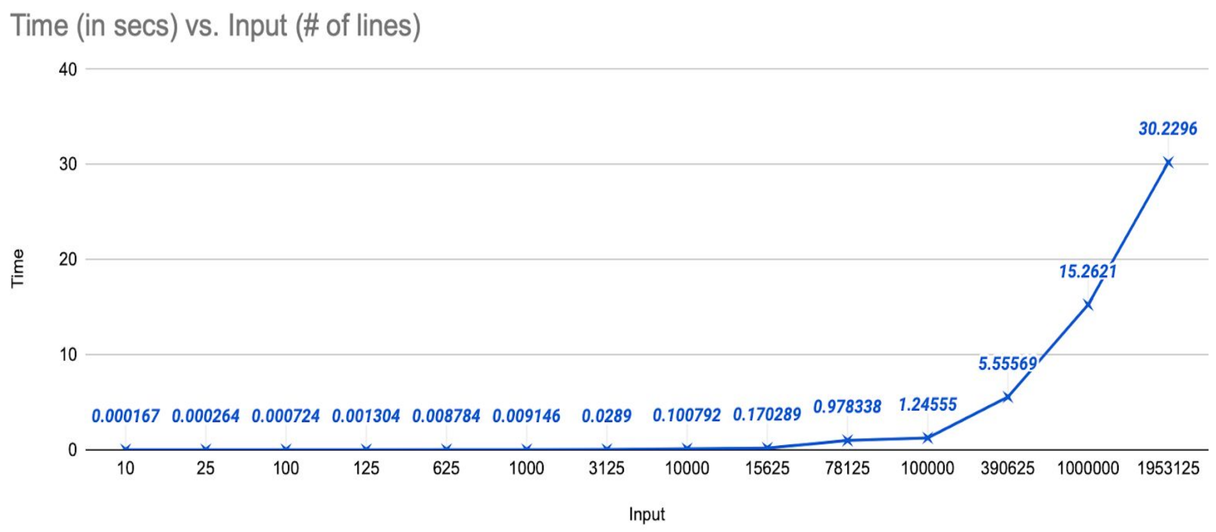
2. Implementing all Fibonacci heap operations in class called "**CDLkdListFibHeap**".
   The below are the member functions of the class:

- *void createHashTagNode(HashTagNode** htMaxPtr, HashTagNode* newHTNode)*
  Creates a new hashtag node if not present and max pointer points to the maximum valued node.
- *void cascadeCutHashTagNode(HashTagNode** htMaxPtr, HashTagNode* htcNode*)
  Cascade cut operations are performed when the child node data is more than the parent node. Operation continues one level up, if it's been set to true already till root.
- *void increaseHashTagValue(HashTagNode** htMaxPtr, HashTagNode* htcNode, int incValue*)
  Increases the count of the corresponding hashnode by incValue and inserts the node in appropriate position
- *HashTagNode* combinePairWise(HashTagNode* alreadyNode, HashTagNode* newNode)*
  Pairwise combine the nodes based on degree of each node and return parent node with it's updated children
- *HashTagNode* pickTopNHashTags(HashTagNode** htMaxPtr)*
  Extracts the max pointer node (top hashtag) by then and returns the same.

Program Flow

1. File handlers to read the input file and write to the output file are created.
2. New object of the CDLkdListFibHeap class is created, so could access it's member functions.
3. A hashmap is created to maintain the hashtag as key and it's object when created as it's respective value.
4. Will read each line of the input file of hashtags to process them and this loop continues to run till we get to stop.
5. If the line starts with a '#' sign, it performs two functionalities
    a. Creates new node in the fibonacci heap and inserts it's details in the hashmap
        i. If the circular doubly linked is empty, the new node created will be the maxpointer.
        ii. If the new node is greater than the max pointer node, it is inserted to it's left and the max pointer is changed to the new node.
        iii. If the new node is smaller than the max pointer node, it is inserted to it's right in the appropriate position (descending order) by parsing through the list.
    b. If the node already exists, it increases its value and gets sorted in the heap.
        i. The current node value is increased as per the amount given.
        ii. If the current node's value is greater than the parent's node value, cascadeCut operation is performed and parent's node cascadeCut when false is changed to true later.
        iii. If cascadeCut is true for the parent's node even before the operation is performed, it continues to perform the operation on all the nodes each level up till it reaches to the point where it's parent's cascadeCut is false or when it reaches root.
6. If the line starts with a number (n), it extracts and prints the top n hashtags into the output_file.txt/stdout and later re-inserts the nodes picked before the next line is read.
    a. While each time it extracts the max pointer node, it assigns the max pointer to the next right sibling until the tree becomes empty and returns the current.
    b. Before returning each time the below operations are performed:
        i. If the current max pointer has children, they are extracted and reinserted into the heap at the top level.
        ii. Pairwise combine operation is performed on all the nodes at top level.
            1. Degree table of max-degree size is initialized to keep track of each node at top level by degree and do pairwise combine
            2. Re-insert all nodes from the table and reset all entries to null.
    c. If 'n' is greater than the actual number of hashtags present, it will return only the number of hashtags present in the hashtable in descending order.
7. If the line is 'stop', the program terminates.

Performance


Time (in secs) vs. Input (# of lines)
Time
Input