ADS Project Spring 2020

COP5536

Shrishail Zalake

UFID – 0746 6343

shrishailzalake@ufl.edu

Problem Statement:

You are required to implement a system to find the n most popular hashtags that appear on social media such as Facebook or Twitter. For the scope of this project hashtags will be given from an input file. Basic idea for the implementation is to use a max priority structure to find out the most popular hashtags.

You must use the following structures for the implementation.

1. Max Fibonacci heap: use to keep track of the frequencies of hashtags.

2. Hash table: The key for the hash table is the hashtag, and the value is the pointer to the corresponding node in the Fibonacci heap.

Input and Output requirements:

Your program is required to take the input file and output file names as arguments. Below is the command for running the program.

For C/C++ :

$hashtagcounter <input_file_name> [output_file_name]

Your program should check if there is an output file specified in the command line. If it is, like below.

$hashtagcounter <input_file_name> <output_file_name>

Then your program should create a NEW output file in the current directory with the name output_file_name, and write all output to that file. However, if no output file is specified, like below. $hashtagcounter input_file_name Then your program should write all output to console or stdout.

However, if no output file is specified, like below.

$hashtagcounter input_file_name

Then your program should write all output to console or stdout.

Input Format:

Hashtags appear one per line in the input file and start with # sign. After the hashtag an integer will appear and that is the count of the hashtag (There is a space between hashtag and the integer). You need to increment the hashtag frequency by that count. Queries will also appear in the input file and once a query appears you should append the answer to the query to the output file. A query appears as an integer number (n) without # sign in the beginning. The answer to the query n is n hashtags with the highest frequency. These should be written to the output file. An input line with the word “stop” (without hashtag symbol) causes the program to terminate. The following is an example of an input file.

#saturday 5

#sunday 3

#saturday 10

#monday 2

#reading 4

#playing_games 2

#saturday 6

#sunday 8

#friday 2

#tuesday 2

#saturday 12

#sunday 11

#monday 6

3

#saturday 12

#monday 2

#stop 3

#playing 4

#reading 15

#drawing 3

#friday 12

#school 6

#class 5

5

stop

Output Format:

For each query n, you need to write the n most popular hashtags (i.e., highest frequency) to the output file in descending order of frequency (ties may be broken arbitrarily). The output for a query should be a comma separated list occupying a single line in the output “output_file.txt”. There should be no space character after the commas.

Following is the output file for the above input file.

saturday,sunday,monday
saturday,sunday,reading,friday,monday

Execution instructions:

1. Extract the zip file “Zalake_Shrishail.zip”
2. Navigate inside the folder and run the command “make”, this will compile all the cpp files required to the run the program.
3. Run the command “hashtagcounter input_file_name [output_file_name]”. Here output file name is optional. If provided, the output is written in the filename given in command line, else, the output is displayed on console.
   Note: the input and output file name has to have the extention, example : “input.txt”
4. To test with other inputs, use the command “make clean”, and then repeat the steps from step 2.

Program structure and description:

Program execution begins from “hashTagCounter.cpp” file which contains ‘main’ function.

Node structure:

- Degree
- leftSibling
- rightSibling
- data

- child
- parent
- hashtag
- childCut

Fibonacci Heap Structure:

- max – pointer to maximum node
- Total number of nodes in the root list.

Project contains 4 classes,

1. "FibHeap.h" with function declarations, and its corresponding child class "FibHeap.cpp" where the functions are defined so that it can be used in main program.
2. "Node.h" with function declarations, and its corresponding child class "Node.cpp" where necessary functions are defined so that it can be used in main program.

Functions:

- "hashTagCounter.cpp" contains main function where the program execution starts. The logic for reading and writing hashtags from given input file to output file is written in this function. It calls functions from FibHeap class and Node class for the program to work.

Class - Node:

- Constructor Node(int data, string hashTag): This is used to initialize the node with required properties so that it can be inserted into Fibonacci heap.

Class – FibHeap:

- **insert(Node *node):**
  Return type: void

  This function is called by main function when a new hashtag is read from input file and its corresponding node has to be inserted into Fibonacci Heap. It takes a pointer to the newly created node as an argument. It inserts the new node to the left of the max node in the root list and if the value of new node is greater than max, the max pointer of Fibonacci heap is changed to newly inserted node.

- **increaseCount(Node* node , int value):**
  Return type : void
  This function is called from main function to increase the frequency(value) of the hashtag which is already present in the hashmap. It takes pointer to the node and value to be increased with, as the arguments. It calls "cut" and "cascadeCut" functions to perform the increase key operation. After performing "cut" and "cascadeCut", if the value of the node is greater than the max node value, the max pointer is changed to point to the current node.

- **cut(FibHeap* fHeap, Node* node, Node* parent) :**
  Return type : void

  This function is called by “increaseCount” function to perform increase key operation. After increasing the value of data field of the corresponding node, the cut operation is called to remove the node from the child list(if the node is a child of other node) and insert into the root list. The function inserts node to the right of the max node.

- **cascadeCut(FibHeap* fHeap, Node* parent):**
  Return type : void

  This function is called by “increaseCount” function and itself(recursively). Cascading cut recurses its way up in the tree until it finds a node with “childCut” value as false or till it reaches the root. If it reaches the root, the childcut value of the root node need not to be changed and is maintained to be false, else, child cut value is changed to “true” once the cascade cut terminates.
- **removeMax():**
  Return type : It returns pointer of the deleted node.

  This function is called several times by the main function based on the query value from the input file. Ex: query = 5, the function is called 5 times continuously. This function is used to remove the max node from the Fibonacci heap. The function adds the subtrees of the max node to the root list and removes the max node.This operation is followed by pairwise combine which combines the equal degree trees from the root list and forms new Fibonacci heap from it with updated max node.
- **pairWiseCombine(FibHeap* fHeap):**
  Return type : void

  It traverses the root list of current FibHeap and restructures it by combining trees of same degree. It does so by maintaining a table where every entry points to the node in the root list and the position of the pointer in the table is equal to the degree of the node. The output of running this function leads to the heap with all the trees having distinct degree. It internally calls “combine” function to combine two trees with same degree into one.

- **combine(Node* t1, Node* t0):**
  Return type : void

  This is called by “pairWiseCombine” function, the pointer to the node t1 is always the node with smaller value and the pointer to the node t0 is always the node with the greater value. This function removes the tree with root node t1 from the root list and makes it a subtree of t0.

Flow Diagram: