Advanced Data Structures (COP 5536)

SPRING 2020

Programming Project Report

SUMANTH CHOWDARY

LAVU

UFID: 5529-6647

sumanthchow.lavu@ufl.edu

PROJECT DESCRIPTION

The goal of this project is to implement a system to find the n most popular hashtags that appear on social media sites such as Facebook or Twitter. For the scope of this project hashtags are taken from 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 structures for implementation.

1. Max Fibonacci heap: used to keep track of the frequencies of hashtags.
2. HashMap: The key for the HashMap is hashtag, and the value is the pointer to the corresponding node in the Fibonacci heap.

The project is written in JAVA. I have implemented Max Fibonacci Heap in Java and stored the addresses of all the nodes in a HashMap (Built in Data Structure). Max Fibonacci Heap is required because it has better theoretical bounds for increase key operation.

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 might not work)

COMPILER

Javac

COMPILING & RUNNING INSTRUCTIONS

This project has been compiled and tested on thunder.cise.ufl.edu and on local java compiler. To execute the program, one must access the server using SSH connection.

Following are the detailed steps for accessing the server and executing the program –

1. Remotely access thunder.cise.ufl.edu using username@thunder.cise.ufl.edu and key in your password.
2. Extract the contents of the hashtagcounter.zip file.
3. Type ‘make’ without typing quotes.
4. Type ‘java hashtagcounter <input_file_path> <output_file_path.txt>’. input_file_path is path to the input file while output_file_path is where we want the output of the program to be written to.

STRUCTURE OF THE PROGRAM AND FUNCTION DESCRIPTIONS

The program has a hashtagcounter file inside which there exists these following 3 classes.

1) hashtagcounter - The main class that reads the input and writes the output.
2) Node - This class is used to instantiate and object of node in memory.
3) maxFibonacciHeap – This class is used for the methods and functions of the Fibonacci Heap class.

The basic workflow of the program is as follows –

1. hashtagcounter class takes the input from a given input file.
2. According to the format mentioned in the input description file, it determines whether to insert nodes into heap or extract the 'N' elements on the top of the heap.
3. If the top 'N' elements are fetched, then it writes them to a file according to the parameters given while the execution of the program.
4. Whenever it encounters a "stop" string, it halts immediately.

The detailed view of class functions is given below –

hashtagcounter.class –

hashtagcounter has following class variables –

BufferedReader br – for reading the input line by line from the given input file.

Printwriter out – for writing the output to either the given output file or a standard output.

map – a hashmap that maps each hashtag to the corresponding node in the Fibonacci heap.

heap – object of maxFibonacciHeap which is used to access its class variables and methods.

params[] – an array used for storing current input lines as tokens. From the format of given input file, if the length of params array is more than 2 then it is assumed

that first token is hashtag and the second one is frequency of that hashtag. Otherwise, we will check if it is a count or look for “stop”. If it is count, then we perform PopMax operation “cnt” times. Otherwise, we simply halt the program.

to_add – an array list that keeps track of the nodes that were removed from the heap. Once the “cnt” tags are removed, they are written to a respective output file, and inserted one by one from toad list to the Fibonacci Heap.

start, end, totalTime – Variables used for checking the running time of the program.

Node.class -

Node class has following class variables –

degree – Number of children a node has in its immediate next level. Degree holds Integer value.

hashTag – contains the hashtag that the node object currently holds.

mark – if the mark is false, it means no child has been removed from it at that point.

key – frequency of the hashtag.

left – a pointer to the next node in the circular list.

right – a pointer to the previous node in the circular list.

parent – a pointer to the parent node.

child – a pointer to the child node.

Node class has following functions -

Function	Return Type	Parameters
Node(String hashTag, int key)	-	hashTag, key

Function	Description
Node(String hashTag, int key)	Constructor that initializes a node with given hashtag and key.

maxFibonacciHeap.class –

maxFibonacciHeap has two variables –

maxNode – a pointer to the max node in the maxFibonacciHeap.

numNodes – size of the current heap

maxFibonacciHeap has following functions –

Function	Return Type	Parameters
insertNode(Node node)	Node	Node node
setKeyValue(Node a, int val)	-	Node a, int val
popMax()	Node	-
cut(Node a, Node b)	-	Node a, Node b
cascadeCut(Node b)	-	Node b
mergeByDegree()	-	-
makeChild(Node b, Node a)	void	Node b, Node a

Function	Description
insertNode(Node node)	insertNode inserts a new node into the heap. Current implementation of this function add the element next to the maxNode
setKeyValue(Node a, int val)	setKeyValue increases the frequency of a hashtag by certain value. Cut and cascadeCut functions follows setKeyValue. They check if the node value is greater than its parent node. If so, they cut the element and add it to the root list.
popMax()	popMax function removes the maxNode from the list and fills the gap by adjusting the next and the previous pointers to it. mergeByDegree follows popMax.

cut(Node a, Node b)	Cut function cuts the node from its parent and adds it to the root list. Also, it sets parent node mark to True as a child from the parent node is removed.
cascadeCut(Node b)	CascadeCut cuts the node and continuously checks all the nodes on its path to the root or until it encounters a node with mark = false.
mergeByDegree()	mergeByDegree merges all the trees in the root list by their degree starting from 1 until it reaches maxSize of the heap.
makeChild(Node b, Node a)	makeChild simply adds one node as a child of the other.

PROGRAM EXECUTION

```
thunder:20% make
javac hashtagcounter.java
thunder:21% ls
 hashtagcounter.class  'hashtagcounter$maxFibonacciHeap.class'   makefile
 hashtagcounter.java   'hashtagcounter$Node.class'               sampleInput.txt
thunder:22% java hashtagcounter sampleInput.txt
Total time in Milli Seconds: 24
cholelithotomy,chlorococcum,chloramine,chon,chivarras
chloramine,chivarras,chloroprene,chloral,chlorococcum,cholelithotomy,chlorothiazide
chloramine,chirurgy,chivarras,chloroprene,chisel,chocolate,chloral,chloroquine,chlorococcum
choke,chokidar
choke,chishona,chloroquine,chokidar,chloroprene,chloramphenicol,chirurgy,chlorothiazide,choleraic,chok
ra,chloramine,chivarras,chon,chlorophyll,choir,chirurgery,cholecystectomy,chlorura
chlorococcum,chishona,choke,chirurgery,cholelithotomy,chitterings,chloroprene,chokra,chisel,choleraic,
cholecystectomy,chirurgy,chloramphenicol,chlorophyceae,chloroquine
chishona,cholelithotomy,chlorococcum,choke,choleraic,chloramphenicol,chivarras
choke,chlorura,chisel,cholelithotomy,chishona,choleraic,chlorophyll,chivarras
choke,chisel,cholelithotomy
chlorura,cholecystectomy,chlorophyll,choleraic,cholelithotomy,chisel,choke
chlorophyll,chlorura,choleraic,cholelithotomy,cholecystectomy,choke,chlorella,chlorococcum,chisel
thunder:23%
```

RESULTS

This code was tested on the provided sampleInput.txt file and found that the program executed in 24 ms. The output with the most popular hashtags was generated and stored in the specified file. Output might be different from the actual output because ties were broken arbitrarily. Also, it entirely depends on the implementation of the maxFibonacciHeap.

CONCLUSION

With the proper implementation of maxFibonacciHeap and successful execution of the program on the given sampleInput file, the objective of the assignment has been met. Furthermore, we can improve the efficiency by adjusting the degreeTable size and Faster IO, but that is beyond the scope of the project.