UF UNIVERSITY of FLORIDA UNIVERSITY OF FLORIDA

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE AND ENGINEERING, COLLEGE OF ENGINEERING

COP 5536 Advanced Data Structures

Project: Using Max Fibonacci Heap and Hash Table to Find The n Most Popular Hashtags in Java

April 4th, 2020

—By—

Name: Zhengyu Li, UFID: 2945-3969, Email: zhengyuli@ufl.edu

—Teaching Assistants—

Pankaj Chand

Shreya Singh

Rohan Wanare

Himani Himani

—Course Instructor—

Dr. Sartaj K Sahni

Introduction

The goal of this project is to implement a hashtag counter system to find the most popular hashtags in Java. The input is given in a text file and the output is either writing results to a text file or printing results in the console depends on the user. The data structures to store hashtags along with their frequencies are max Fibonacci heap and hash table. Fibonacci heap maintains the whole hashtag frequency structure in order to extract the highest frequency and hash table is used for recording current hashtags. The reasons using the Fibonacci heap are the amortized complexity of increasing key operation takes O(1) and there are many increasing key operations need to be done during finding the most popular hashtags. The reasons using the hash table are the average time complexity of search operation takes O(1) and there are many search operations need to be done. After using these data structures, the algorithm runs very efficiently. For instance, processing a million lines input file, the result of eleven times extractions of extracting no more than twenty popular hashtags in each extraction can be shown immediately.

Input and Output

The input is a text format file only containing lower case letters. It contains one type of data which is the hashtag along with its frequency, and two types of commands which are query command and terminate command. Each row in the input file contains only one entity which is either hashtag data or one of two commands.

1. Hashtag data format

Hashtag and its frequency in the input file is as the following format:

#cholangiography 10

There is a '#' sign at the beginning of each hashtag and no spaces between '#' sign and hashtag name. There is also no space in the hashtag name. However, there is a single space between hashtag name and its frequency. It contains only one "int" number as the frequency at the end.

2. Query command format:

Finding n most popular hashtags command is as the following format:

5

This command contains an integer no more than 20. It means to find n most popular hashtags and output the result, where n is 5 in this example.

3. Termination command format:

The termination command is as the following format:

stop

This command only contains a lower case "stop" word and it means to terminate the program.

An example of an input file contains all entities is shown below:

#chlorococcum 2
#chokecherry 5
#chlorothiazide 1
2
#chlorura 2
stop

The input is taken to the program by using command line. The program is named as “hashtagcounter”; therefore, the format of running the program using an input file is as the following (single space between each argument):

$java hashtagcounter <input_file_name>

There are two output formats of this projects. One is output to a text file using the following command. In this case, the program creates a new output file in the same directory using the given file name. Results are outputted to this output file:

$java hashtagcounter <input_file_name> <output_file_name>

Another output format is printing results to console directly. The command is as the following (without indicating output file name):

$java hashtagcounter <input_file_name>

In the results of both output methods, returned hashtag names of each query are in the same row and each word is split by a comma without any spaces. For instance, the result format of above input file example is shown below:

chokecherry,chlorococcum

Implementation

The flowchart of this program is provided below in Fig. 1. In this flowchart, the program structure is shown. Number of command line arguments is checked at the beginning. If there is no input file argument in the command line, there will be a message to remind the user including the input file name in the command line. If there is an input file name but no output file name in the command line, the program will execute the left part in the Fig.1 which is when the argument number is 1. In this case, the program will print results in the console directly. Otherwise, the program will create an output file naming the second command line argument, and write results to the output file. Therefore, there is at least one command line argument exists for providing the input file name, and maybe another command line argument for providing output file name. In this step, some errors like wrong input file name, no input file name will be handled. Besides this, Java Scanner is used to read file because read by line is needed in this project and Scanner is simple to implement. Buffered Writer is used to write the result into the output file if needed.

Then Fig. 2 shows the Fibonacci node structure. In this project, the program consists of 3 Java files which are “hashtagcounter.java”, “MaxFibonacciHeap.java”

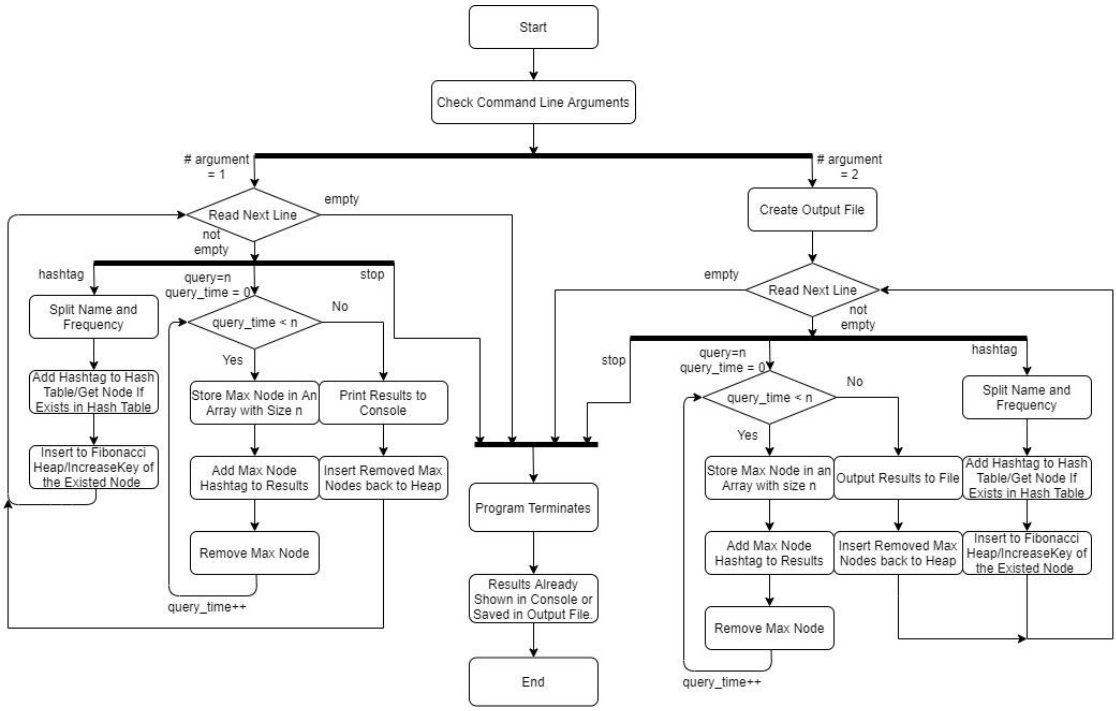

Fig. 1 flowchart/program structure

and “NodeStructure.java”. “hashtagcounter.java” is the main file controlling the whole program and each block in the flowchart. “NodeStructure.java” is a Java class establishing the Fibonacci node structure. It only contains a constructor initializing each parameter shown in Fig. 2. There are four pointers which are left, right, parent and child in each node structure. Left and right pointer point to its sibling if any, or itself if no sibling. Parent and child point to its parent and child if any, or null if no parent or no child. Key is a “String” variable containing hashtag name; value is an Integer variable indicating frequency of the hashtag. Mark is “Boolean” variable and only useful when cascading cut is needed. Degree is an integer variable maintaining the size of the heap table and only useful when pairwise combine after removing max is needed.

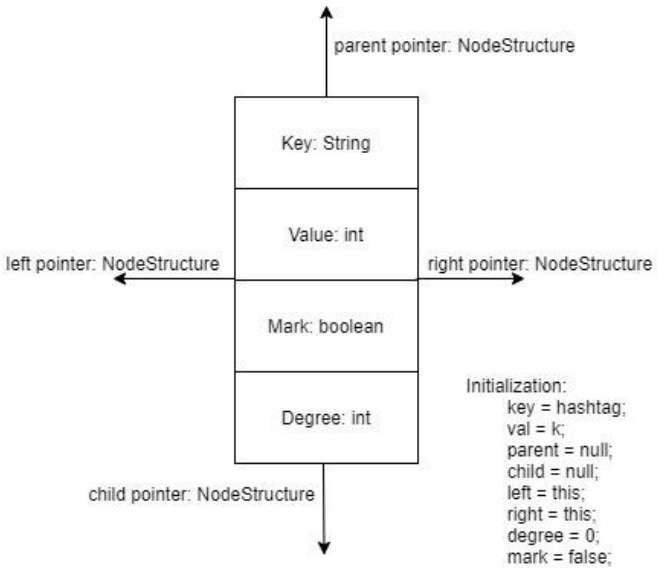

Fig. 2 Fibonacci node structure

Fibonacci Heap Operations		
Insert	increaseKey(removeParentSibling, insert, cascadingCut)	deleteMax(removeParentSibling, insert, pairwiseCombine)
cascadingCut(removeParentSibling, insert)	removeParentSibling	pairwiseCombine

Fig. 3 max Fibonacci heap operations

"MaxFibonacciHeap.java" is a Java class contains all max Fibonacci heap operations needed in this project shown in Fig. 3. In this heap class, two pointers which are "maxNode" and "currentNode" are created to track the max node and the current node in the Fibonacci heap. Variables "totalNode" and "maxDegree" are created for tracking the node number and max degree in the heap in order to perform "deleteMax". Besides pointers and variables, three main heap operations "Insert", "increaseKey", and "deleteMax" are performed in this class. Some other operations such as "cascadingCut", "pairwiseCombine", and "removeParentSibling" operations are also performed.

Fibonacci heap function prototypes and explanations:

1. public void insert(NodeStructure node)
   The input is a node structure and the output is void. The Fibonacci heap is well maintained after calling this function. The Node to be inserted is assumed a single node without parents or sibling, and it is inserted to the left of the current node. Therefore, "removeParentSibling" needed to be called if the node currently has parent or siblings.
2. public void removeParentSibling(NodeStructure node)
   The input is a node structure and output is void. The parent and current siblings of the inputted node is removed after calling this function. The node is removed from its former parent and siblings lists as well.
3. public void cascadingCut(NodeStructure node)
   The input is a node structure and the output is void. The parent of this node is inserted to the heap if the mark value was True, and mark value is set to True if mark value was False after calling this function. And if the parent of this node is True, after inserting the parent to heap, keep calling "cascadingCut" function on its parent until the parent mark is False or parent is null (root node).
4. public void increaseKey(NodeStructure node, int k)
   The input is a node structure and an increased value k. The output is void. The key of this node is increased by k after calling this function. And calling "cascadingCut" function if the value of the node is greater than its parent's.
5. private void pairwiseCombine(NodeStructure[] table)
   The input is a node structure table. Output is a pairwise combined heap. Traverse all root nodes in one direction, combine smaller node to bigger node with the same degree after calling this function. The strategy is to use another table to record degree and if the two nodes have the same degree, merge node with smaller

value to node with bigger value. If latter node is smaller, swap two nodes to avoid traverse repeated nodes.

6. public void deleteMax()
   Both input and output are void since there are only one max node in a heap at a time. Delete max node and do a pairwise combine operation. When deleting max node with child, insert all children to heap first, then delete it. Max degree is used here for maintaining the size of the node structure table used in pairwise combination operation.

After introducing input and output format, command line argument, node structure and Fibonacci heap structure, each step in the main program and the data flow is explained below. First of all, the program reads the input file line by line until "stop" command or the end of the file. Before reading the file, initialization must be done. An empty max Fibonacci heap and an empty hash table need to be created. The key of the hash table is hashtag name, and the value of hash table is node structure.

1. Hashtag
   The hashtag data is recognized by comparing the first character. If the first character is '#', then the following hashtag name and its frequency will be split and wrote into a "String" array. Array[0] is the hashtag name without '#' sign and array[1] is the frequency integer. Then if the hashtag is already in the hash table, the frequency of this node structure needs to be increased. Thus, the Fibonacci heap might be re-structured if the key of this node is greater than its parent node. Therefore, a Fibonacci heap increaseKey operation performed here to add the new frequency number to the old node and maintain the heap structure. Otherwise, the hashtag will be added to hash table. A new node structure is created for this hashtag with an initialized value which is the frequency. Once the node is added to hash table, a Fibonacci heap insert operation is perform to add this node into the heap.
2. Query:
   A node structure type array with the size of query size is created in order to store and re-insert the removed max nodes back to the heap since the goal is to find the max node instead of deleting max node. For each query, store the max node into array, then add the hashtag name of the max node to the result variable with a comma unless it is the last one. At the end, call "deleteMax" function to delete this node in order to find the next Max node. Keep doing these operations until finishing all queries. Finally, insert all removed nodes in the array to the heap.
3. Terminating:
   The program is terminated either a line contains a single word "stop" or file reader reaches the end of the file. For the first case, string comparation is performed. If the input is exactly equal to the word "stop" without any other sign and numbers, the program terminates. For another case, check whether the scanner has the next line or not. If not, the program terminates. Otherwise, keep running.

Experiment

Many experiments had been done in both local PC and “thunder.cise.ufl.edu” server including printing frequencies of each hashtag along with hashtags for checking correctness, creating custom samples to test special cases and creating random samples to test if the program can handle millions of data and so on. Only the final results using the latest given “sampleInput.txt” file containing about a million records running on the server was shown here. Tie breaker in this program was in alphabetical order. A Java function StringA.compareToIgnoreCase(StringB)<0 was used to break tie. In this case StringA was bigger than StringB. For letter a.compareToIgnoreCase(b), the result was less than 0 which is -1. Therefore, in this project, a > b >..> z. For example, assume “chivarras” and “chon” had the same frequency. “chivarras” was bigger than “chon” since “ch” of these two words was the same but letter ’i’ was bigger than ‘o’ in alphabet so that chivarras was bigger.

First of all, went to the project directory:

```
thunder:10% ls
hashtagcounter.java  Makefile  MaxFibonacciHeap.java  NodeStructure.java
thunder:11%
```

There were three Java files and one “Makefile”. “make” command was ran in order to compile these Java files:

```
thunder:11% make
javac -g MaxFibonacciHeap.java
javac -g hashtagcounter.java
thunder:12% ls
hashtagcounter.class  MaxFibonacciHeap.class  NodeStructure.java
hashtagcounter.java   MaxFibonacciHeap.java
Makefile              NodeStructure.class
thunder:13%
```

“$java hashtagcounter sampleInput.txt” was ran in order to print the result on the console.

```
thunder:19% java hashtagcounter sampleInput.txt
cholelithotomy,chloramine,chlorococcum,chivarras,chon
chivarras,chloramine,chloroprene,chloral,chlorococcum,cholelithotomy,chlorothiaz
ide
chloramine,chirurgy,chivarras,chloroprene,chisel,chocolate,chloral,chloroquine,c
hlorococcum
choke,chokidar
choke,chishona,chloroquine,chloramphenicol,chloroprene,chokidar,chirurgy,chlorot
hiazide,chokra,choleraic,chloramine,chivarras,chlorophyll,chon,chirurgery,choir,
chlorura,cholecystectomy
chlorococcum,chishona,choke,chirurgery,cholelithotomy,chitterings,chloroprene,ch
okra,chisel,cholecystectomy,choleraic,chirurgy,chloramphenicol,chivarras,chlorop
hyceae
chishona,cholelithotomy,chlorococcum,choke,choleraic,chloramphenicol,chivarras
choke,chlorura,chisel,cholelithotomy,chishona,chlorophyll,choleraic,chivarras
chisel,choke,chlorura
chlorura,chlorophyll,cholecystectomy,choleraic,cholelithotomy,chisel,choke
chlorophyll,chlorura,choleraic,cholelithotomy,cholecystectomy,chlorella,choke,ch
lorococcum,chisel
thunder:20%
```

“$java hashtagcounter sampleInput.txt output.txt” was ran in order to print the result in the output file. One more file which is “output.txt” created in the folder.

```
thunder:20% java hashtagcounter sampleInput.txt output.txt
thunder:21% ls
hashtagcounter.class  MaxFibonacciHeap.class  NodeStructure.java
hashtagcounter.java   MaxFibonacciHeap.java   output.txt
Makefile              NodeStructure.class     sampleInput.txt
thunder:22% vi output.txt
```

Using “vi” command to open the output file.

```
cholelithotomy,chloramine,chlorococcum,chivarras,chon
chivarras,chloramine,chloroprene,chloral,chlorococcum,cholelithotomy,chlorothiazide
chloramine,chirurgy,chivarras,chloroprene,chisel,chocolate,chloral,chloroquine,chlorococcum
choke,chokidar
choke,chishona,chloroquine,chloramphenicol,chloroprene,chokidar,chirurgy,chlorothiazide,chokra,choleraic
,chloramine,chivarras,chlorophyll,chon,chirurgery,choir,chlorura,cholecystectomy
chlorococcum,chishona,choke,chirurgery,cholelithotomy,chitterings,chloroprene,chokra,chisel,cholecystect
omy,choleraic,chirurgy,chloramphenicol,chivarras,chlorophyceae
chishona,cholelithotomy,chlorococcum,choke,choleraic,chloramphenicol,chivarras
choke,chlorura,chisel,cholelithotomy,chishona,chlorophyll,choleraic,chivarras
chisel,choke,chlorura
chlorura,chlorophyll,cholecystectomy,choleraic,cholelithotomy,chisel,choke
chlorophyll,chlorura,choleraic,cholelithotomy,cholecystectomy,chlorella,choke,chlorococcum,chisel
```

After checking each hashtag with the standard output file, the result was the same as the given “sampleOutput.file”. Results were computed very fast without any waiting time.

Conclusion:

In this project, all operations of max Fibonacci heap were implemented. Besides the heap operations, a node structure class and a main program were implemented correctly as well. Using Fibonacci heap along with hash table is very efficiently. It is able to give results immediately when running on million samples. Therefore, finding n most popular hashtags in the real world can be very fast using this implementation.