Keyword Counter Implementation

using

Fibonacci Heap ad Hash Map

Advanced Data Structures Project
Fall 2018

Priyam Saikia
(UFID 9414-5292)
priyam.saikia@ufl.edu

Problem Statement: The aim of this project is to implement a java program to find the n most popular keywords used in DuckDuckGo's search engine.

Implementation (in Java): We will use an input file to feed the keywords along with their frequency numbers to the program. We use a Max Fibonacci heap to maintain a priority queue that has the maximum value at the root. Fibonacci heaps have better amortized complexity. We implement Hash Map in java to keep track of the various keywords and their corresponding frequency. This program can handle atleast 1 million keywords. Details of implementation can be seen the prototypes mentioned later on the project.
I have included 3 test files in the zip folder in which the program is working.

Data Structures used:
1. Max Fibonacci heap: Keeps track of the keywords as Nodes. maxNode contains the keyword with maximum frequency.
2. Hash Map: Keeps track of the keywords and their corresponding nodes in the Max Fibonacci Heap using key-value pair with key being the keyword and value as the pointer to the corresponding node.

Tested on: Local Machine **and** Thunder.cise.ufl.edu

New Test Cases: sample2.txt and sample3.txt are two new test cases. sample1.txt is the one that professor provided us along with the problem statement.

Instructions to run:
1. Login to thunder.cise.ufl.edu
2. Extract files from Zip folder
3. Change directory to corresponding extracted folder
4. Type 'make' without the quotes and enter. The java files should compile.
5. Type 'java keywordcounter <test_file_name.txt>' and enter
6. Output can be accessed from the newly generated "output_file.txt" file in the same folder as input file.

FILES in Zip Folder:

Java Files:
keywordcounter.java -> Primary .java file, acts as the entry-point.
Most_Frequent_Search.java -> Populates the hash-map, calls Fibonacci Heap functions and returns top *n* keywords.
Max_Fib_Heap.java -> Max Fibonacci Heap Functions Implementation.

Other Files:
makefile -> Compiles the java files Readme.md -> Instructions to run
Report.pdf -> Report for the project
sample1.txt -> Sample Test File 1
sample2.txt -> Sample Test File 2
sample_large.txt -> Sample Test File 3
output_file1.txt -> Output to Sample Test File 1
output_file2.txt -> Output to Sample Test File 2
output_large.txt -> Output to Sample Test File 3

CODE STRUCTURE DETAILS:

Class: keywordcounter.java		
void main(String[] args)		
Description	This is the entry point to the complete implementation. It takes a file name as input and runs the program to find the n most used keywords	
Parameters	String[] args	Holds the input filename/path. Takes only the first argument
Return Value	void	

Class: Most_Frequent_Search.java		
void main(String in_file)		
Description	This function implements the HashMap. It keeps on adding value from input file line by line using a Buffered Reader as long as the line starts with $ and is followed by a numeric value. If it fines a line with just one single numeric value with no $ preceding it then it finds the top n used keywords and writes it to an output file "output_file.txt". It also calls functions from Fibonacci Heap Class.	
Parameters	String in_file	Input file/path
Return Value	void	

String find_first_n_keywords(Max_Fib_Heap max_fh, int n)		
Description	Function to find the top n frequent words by calling Fib Heap. Calls the removeMax function n number of times to get the maximum node for each iteration. Re-inserts each of the removed nodes to maintain the database.	
Parameters	max_fh n	Fibonacci heap object Top n number of most used keywords to be returned
Return Value	String value that consists of first n most used keywords separated by commas	

boolean isNumeric(String str)		
Description	Function to validate of a given string is numeric - consists of nonly numbers	
Parameters	str	Input string to check if it is number
Return Value	Boolean True or False	

Class: Max_Fib_Heap java		
Public class Node		
Description	Sub Class Node for Max Fib Heap Data structure	
Parameters	String Keyword int Freq int Degree Boolean ChildCut Node Right Node Left Node Parent Node Child Node (String keyword, int freq)	Contains the keyword Contains the frequency of the keyword Contains the degree (number of children) for the keyword Contains mark to be used in Cascade Cut Pointer to Right Sibling Pointer to Left Sibling Pointer to Parent Pointer to one child Constructor to populate the above values
Return Value	--	

void insert (Node node)		
Description	Function to insert a node into the Max Fib Heap. Inserted next to the maxNode on its right.	
Parameters	node	Pointer to the node to be inserted
Return Value	void	

void increase_key (Node node, int k)		
Description	Function to increase the frequency of a node to value *k.* If value increases beyond parent, cut and cascade cut operations are performed.	
Parameters	node k	Pointer of the Node whose value is to be increased The amount to which value is increased to
Return Value	void	

void update_max()		
Description	Traverses through the top-level list and update the maxNode.	
Parameters	-	-
Return Value	void	

void cut (Node node)		
Description	Function to perform cut operation on a Node and remove it from a heap and move to top level-list.	
Parameters	node	Pointer of the node that needs to be removed from a heap
Return Value	void	

void cascade_cut (Node node)		
Description	Function to perform cascade cut operation on a parent Node to check if it has a ChildCut value of True. Does Cut() and CascadeCut() operations if required. If ChildCut value is true – Perform Cut and Cascade Cut. If it is false, set true.	
Parameters	node	Pointer to the parent node that needs to be checked recursively for true or false ChildCut value
Return Value	void	

Node remove_max()		
Description	Function to remove max value of the heap. Does Meld() and Update_Max() Operations as a result. We remove the max node and move its children to the top level list and then perform Meld() operation	
Parameters	-	-
Return Value	Pointer to the removed node is returned without any of its children	

void meld ()		
Description	Function to perform Meld operation via degree wise merging using degree Table after removeMax operation. We keep performing merge of heaps as long as no two heaps have the same degree.Then maxNode is updated	
Parameters	-	-
Return Value	void	

Node merge_heaps (Node node1, Node node2)		
Description	Merge two given heaps of same degree based on which one has higher frequency value	
Parameters	node1 node2	Pointer to first heap's header node Pointer to second heap's header node
Return Value	Pointer to the node with higher frequency value – parent node to other one	

void move_children(Node node)		
Description	Function to remove all children from a Heap and move to top-level list	
Parameters	node	Pointer to parent node

Return Value	void

void delete (Node node)		
Description	Function to perform arbitrary delete. The children, if any, are moved to the top level list.	
Parameters	node	Pointer to node to be deleted
Return Value	void	

Node return_max ()		
Description	Function to find and return the max Node of the Max Priority Queue	
Parameters	-	--
Return Value	Pointer to the maximum value of the Priority Queue	