Advanced Data Structures

(COP 5536)

2018

Fibonacci Heap Implementation
Keyword Counter

Akshay Sehgal
UFID: 1416-7988
akshay.sehgal@ufl.edu

PROJECT DESCRIPTION

The goal of the project is to implement a system that can count the most popular keywords in the search engine. The input file is fed into the system containing the list of keywords and the output file will contain "n" most popular keywords. The value of n will also be provided as a number in the input file.

The project uses the following data structure.
1. Max Fibonacci heap: Use to keep track of the frequencies of the keywords.
2. Hash table(Hash Map in java) : Key for the hash table is keyword and value is pointer to the corresponding node in the Fibonacci heap.

The project is implemented in JAVA language. The HashMap is the built in data structure used. The Fibonacci Heap is implemented without the use of any internal data structure. Fibonacci heap is required because it has better theoretical bounds for increase key operation.

FIBONACCI HEAP

A Fibonacci heap is a data structure for priority queue operations, consisting of a collection of heap-ordered trees. It has a better amortized running time than many other priority queue data structures including the binary heap and binomial heap. Michael L. Fredman and Robert E. Tarjan developed Fibonacci heaps in 1984 and published them in a scientific journal in 1987. They named Fibonacci heaps after the Fibonacci numbers, which are used in their running time analysis.

Fibonacci Max Heap		
Operations	Actual Cost	Amortized Cost
Insert	O(1)	O(1)
Remove Min/Max	O(n)	O(log n)
Meld	O(1)	O(1)
Remove	O(n)	O(log n)
Decrease/Increase Key	O(n)	O(1)

SYSTEM SPECIFICATION

HARDWARE REQUIREMENT

Hard Disk space: > 8 GB

Memory: > 1GB

CPU: x86

OPERATING SYSTEM

LINUX/UNIX/MAC OS

COMPILER

Javac (JAVA Compiler)

How to Run the Program?

1. You can remotely access the server using ssh username@thunder.cise.ufl.edu
2. For running the Keyword Counter application
3. Unzip the file to extract content.
4. Enter "make" without quotes.
5. Enter java "keywordcounter filepath/input_file_name.txt" without quotes.
6. Run "make clean" without quotes to clean .class files.

STRUCTURE OF THE PROGRAM

The program consists of 3 classes.

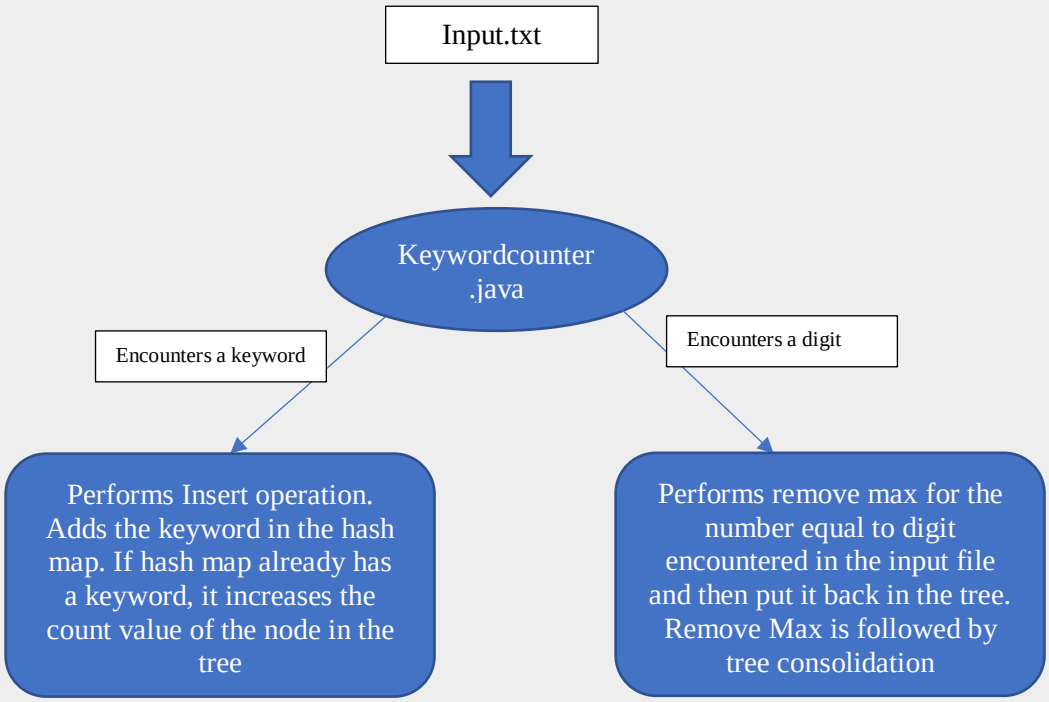
1) keywordcounter - The main class that reads the input file and performs the insert and writes operation based on the values read.

2) treeNode – This class is used to instantiate the object of node in memory which has following properties.

- Objects of treeNode which acts as pointers in the tree - left, right, child, parent
- degree of the node
- childCut value of the node for the operation of cascading cut
- data field holds the value of the keyword
- count field holds the value of number of times the keyword has been searched
- getData() function returns the data field

3) fibonacciHeap – This class is used to instantiate the methods and functions of the Heap class. All the operations pertaining to Fibonacci Heap are defined in this class.

The basic workflow of the program is the keywordcounter.class takes input file and reads it to create a hash map with different values of the keyword, it also performs insert node operations on an instance of the fibonacciHeap.class by creating new nodes using treeNode.class. As soon as it reads a digit in the input file, it performs remove max operation and writes it in the output_file.txt, after which it reinserts the node back into the hash map.



JAVA CLASS DESCRIPTION

The detailed overview of all the class functions is given below.

- **keywordcounter.java**

Variables	Type	Description
1. pathtofile	String	The input filename taken from the user.
2. hash	HashMap<String, treeNode>	The HashMap where tree nodes and keywords are stored.
3. heap	Fibonacci Heap	The instance of the Fibonacci heap.
4. OutputFile	String	Name of the output file.

5. keywordPattern	Pattern	Stores the pattern of the input given, the format of the input. ([#])([a-z_]+)(\\s)(\\d+)
6. digitsPattern	Pattern	Stores the pattern of the digits to be removed from the Fibonacci heap (max values)
7. matcher	Matcher	Matcher object for pattern 'keywordPattern'
8. digitMatcher	Matcher	Matcher object for pattern 'digitsPattern'
9. word	String	Keyword to be stored.
10. count	Integer	Count of the occurrence of particular keyword.
11. increasedCount	Integer	Old count value+ new count value.
12. noOfWriteValues	Integer	Number of tree nodes to be removed (remove max operation)
13. removedNodes	ArrayList<Node>	Stores all the removed nodes
14. startTime	Time(milliseconds)	Start time of program.
15. completionTime	Time(milliseconds)	Completion time of the program.

Function Name	Return Type	Parameters	Description
1. public static HashMap<String, treeNode> getHash()	HashMap	-	This function returns the hash generated.

- **treeNode.java**

Class Variable	Type	Description
Degree	Integer	Signifies the number of nodes that a node can have in its next level.
data	String	Contains the keyword data.
count	Integer	Signifies the count value of the keyword.
left	treeNode (Object)	Points to the left node in the circular list.
right	treeNode (Object)	Points to the right node in the circular list.
parent	treeNode (Object)	Points to the parent node
child	treeNode (Object)	Points to the child node.
childCut	Boolean	A Child Cut (default value false) is used to hold a Boolean expression indicating whether the child of the node has been removed or not.

		If child has been removed it is set to TRUE. It helps to perform cascading cut.

Function Name	Return Type	Parameters	Description
1. getData()	String – data field	-	Returns the data field i.e. the name of the keyword.
2. treeNode(String, Integer)	void	String, Interger	Used to initialize the node with the data of the keyword and its count value. Only when the new treeNode is instantiated.

- **FibonacciHeap.java**

Class Variable	Type	Description
1. maxNode	Node	Points to the maximum node in the heap.

Function Name	Return Type	Parameters	Description
1. public void insert(treeNode node)	Void	treeNode node	Used to insert the node in the heap.
2. public void cut(treeNode node, treeNode nodeParent)	Void	treeNode node, treeNode nodeParent	To perform the operation of cutting the node from its parent and adding it to the root list.
3. public void cascadingCut(treeNode node)	Void	treeNode node	Performs the operation of cascading which propagates up the heap.
4. public void increaseCount(treeNode node, int newCount)	Void	Node treeNode node, int newCount	Increases the value of count fort the keyword already in the heap.
5. public treeNode removeMax()	Node	Null	Remove the max node and calls the consolidateHeap().
6. public void consolidateHeap()	Void	Null	Called after remove max operation is done. It performs degree wise merge of meld.
7. public void concatenateInRootList(treeNode node)	Void	treeNode node	Puts the node back in the root list.
8. Public void HeapLink(treeNode x, treeNode y)	Void	treeNode x, treeNode y	Links the nodes as child and parent during consolidateHeap operation

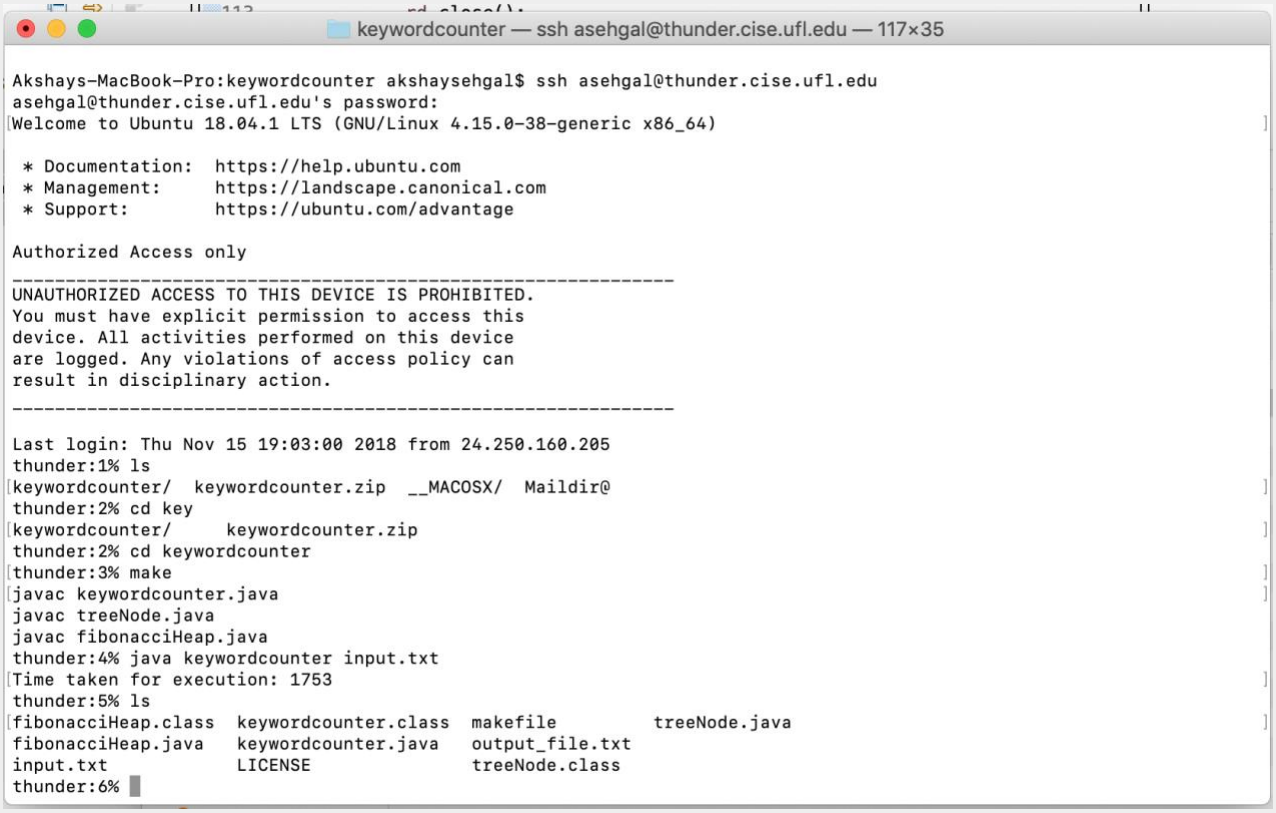
Function Name	Description
1. public void insert(treeNode node)	This function inserts a node into the max Fibonacci heap. The function works in two ways. If the max is null i.e if the maxNode is null, maxNode is assigned to be the root of the Fibonacci heap since it is the only node in the Fibonacci heap right now. However, if there is a maxNode, new tNode is added to the top level of the Fibonacci heap, to the right of the max root in the doubly linked list of that level.
2. public void cut(treeNode node, treeNode nodeParent)	This function performs cut operation on treeNode node. It cuts treeNode node from treeNode nodeParent, then it decreases the degree of treeNode nodeParent.
3. public void cascadingCut(treeNode node)	This function checks the childCut value and makes the necessary changes and performs remove if needed. If the childCut value is false for the parent, it changes it to true. If the childCut is true for the parent, the function will keep going up the tree and removing the nodes until it finds a node whose childCut is false. It calls itself recursively till the node has a parent i.e. ends at root node.
4. public void increaseCount (treeNode node, int newCount)	This function is used to increase the value of the count to newCount in treeNode node. The count value of parent is checked, and if it is less than the parent the node is cut and cascading cut is performed on the parent if the parent childCut is true.
5. public Node removeMax()	This function removes the maximum node, from the Fibonacci heap. And while there are children of max node, put them on root list. After remove max operation it calls consolidateHeap().

6. public void consolidateHeap()	This function performs degree wise merge. It melds the nodes in the root list of the heap to make a consolidated heap with each element in the root list having a unique degree.
7. public void heapLink(treeNode y, treeNode x)	This function is used to make treeNode y as child of treeNode x.

RESULTS

The code was run for 1 million inputs and the output time required was found to be **1753 ms**. It was observed that the time taken for execution varies each time.
This is because there are many nodes with same values when remove max was called.

Screenshot: How To Run The Program



CONCLUSION

The objective is achieved with the implementation of the Fibonacci Heap to hold all the keywords and there count i.e. number of times the keyword has been searched. The in-built data structure Hash map is used to hold the keyword value and the corresponding tree node in the heap. The program is run and tested for all the possible cases and the results are obtained as per the best of the knowledge.

Assumptions:

1. The program stops as soon as it encounters the “stop” as the input string from the input file.
2. The keywords are not case sensitive i.e. the whole word should match with considering the case sensitiveness.
3. If the number of write values is greater than the available nodes in the heap. The program shall print all the available nodes.
4. Rest everything is implemented with reference to the problem statement provided.

REFERENCES:

[1] Fibonacci heap. (n.d.). Retrieved November 18, 2016, from https://en.wikipedia.org/wiki/Fibonacciheap

[2] Introduction to Algorithms:
http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap21.htm

End of Report

For any further details or clarification, please contact Akshay Sehgal.
Email: akshay.sehgal@ufl.edu