Advanced Data Structures
Fall 2018
Project Report

Debdeep Basu

UFID: 4301-3324

debdeep.basu@ufl.edu

1. Project Description

Implement a keyword counter for a search engine to count the most popular keywords. The data is read from a text file – containing the keywords and their frequencies of occurrence. A Max Fibonacci Heap is used as a max priority queue. From time to time the heap is queried to return the top 'n' searched keywords.

Two data-structures are used –

1. HashMap: The key is the keyword and the value is the node in the Max Heap.
2. Max FibonacciHeap: This datastructure serves as a max priority queue.

Technology used: Java 1.8

2. Installation

Unzip the project, cd into the project directory and run the make command –

cd basu_debdeep
make
java KeywordCounter test.txt

Running this would generate the output file in the current directory as output_file.txt

3. Structure of the Program

1. The driver class (KeywordCounter) has the main method which accepts a command line argument containing the file url.
2. It reads each line, invoking the Fibonacci heap's insertOrUpdate API or getMostUsed API.
3. The insertOrUpdate checks if the keyword is already present in the heap or not. If it is, then increaseKey is performed on that node in the heap. If the keyword is not present then it is inserted into the heap.
4. getMostUsed retrieves n number of most used keywords by calling deleteMax n times and then returning the top keywords back to the invoking method. The deleted nodes are then inserted back into the heap.
5. When a "stop" is encountered the program exits.

4. Documentation

There are 3 classes – FibonacciHeap, Node, KeywordCounter

4.1 Fibonacci Heap

Description: Max FibonacciHeap implementation in Java. Supports insert O(1), the deleteMax O(logn), meld O(1) and increaseKey O(1) methods in amortized time complexity.

Fields:

Type	Field	Description
Node	head	The head pointer in the top level circular doubly linked list.
HashMap<String, Node>	map	The key is the keyword and the value is the node in the Max Heap.
Node	maxNode	Pointer to the node having the minimum key

Methods:

1. `public void insertOrUpdate(String str, int frequency)`

Description	This method checks if the HashMap contains the keyword. If it is already present, it calls increaseKey to increase the key of the node in the heap. Else it, creates a node and inserts it into the top level doubly linked list.	
Parameters	str	Keyword of the inserted entry
	frequency	number of times the keyword has been entered
Return value	void	

2. public ArrayList<String> getMostUsed(int num)

Description	This method queries the heap to get the most used keywords by iteratively calling deleteMax 'num' numbers of times. After retrieving all of the most used entries, it reinserts it into the heap, because we don't want to delete them.	
Parameters	num	number of most used keywords to be retrieved from the heap
Return value	A list of keywords for the query	

3. void deleteMax()

Description	Deletes the max from the heap and reconstructs the heap by pairwise-merging. The maxNode points to the next largest node in the heap.
Parameters	void
Return value	void

4. Node meld(Node node1, Node node2)

Description	Deletes the max from the heap and reconstructs the heap by pairwise-merging. The maxNode points to the next largest node in the heap.	
Parameters	node1	Node to be melded with node2
	node2	Node to be melded with node1
Returns	The melded node	

5. void remove(Node node)

Description	Remove a node from the top level circular list by re-adjusting the pointers.	
Parameters	node	Remove this node from the top-level doubly linked circular list
Returns	void	

6. Node insert(String str, int frequency)

Description	Insert a new keyword in the heap in the top level doubly linked circular list.	
Parameters	str	Keyword of the to be inserted entry
	frequency	frequency of the to be inserted entry
Returns	The inserted node	

7. Node insert(Node node)

Description	Insert the node into the top level doubly linked list. Re-adjust maxNode and head pointers if necessary.	
Parameters	node	Insert this node in the top level doubly linked circular list
Returns	the inserted node	

8. Node increaseKey(Node node, int val)

Description	Increase the key of this node, cascading cut may follow if it's value is greater than that of it's parent.	
Parameters	node	The node whose key is to be increased.
	val	The value by which the key should be increased by
Returns	void	

9. void cascadingCut(Node node)

Description	Perform cascading cuts on this node and keep going up recursively until the parent is null.	
Parameters	node	The node on which cascading cut is performed.
Returns	void	

10. void cutFromParent(Node node, Node parent)

Description	Detach node from parent.	
Parameters	node	The node which is going to get cut from it's parent.
	parent	The parent which will lose a child.
Returns	void	

4.2 Node

The Fibonacci heap is made up of Nodes which have next, prev, child and parent pointers and data and the degree.

Fields

Type	Name	Description
Node	next	The next pointer points to the sibling node on the right.
Node	prev	The prev pointer points to the sibling node on the left.
Node	child	The child pointer points to it's child.
Node	parent	The parent pointer points to it's parent.
boolean	childCut	Childcut is false by default. It changes to true when the node has lost a child.
int	val	The data part in the Fibonacci heap.
int	degree	The number of children.

Methods

1. public void addChild(Node node)

Description	Add a child by re-adjusting pointers. If the child is null, set it to this child. Other wise add it to the circular doubly linked list.	
Parameters	node	The node which is added
Returns	void	

4.3 KeywordCounter

This is the driver module of the program.

Methods:

public static void main(String[] args)

Description	This has the main method which is responsible for accepting a fileurl as a command line argument and calling the Fibonacci heap's API's.	
Parameters	args	Command Line Argument expecting a fileurl
Returns	void	