ADS Project Report
FALL 2018

Name: **Noopur Rajesh Kumar Kalawatia**
UFID: **1980 – 9834**
Email: noopur.rajeshkum@ufl.edu.

1. **Introduction:**

The goal of this project is to implement a system to find the n most popular words from the input file provided. Basic idea for the implementation is to use a max priority structure to find out the most popular words.

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

The project is implemented in Java. Fibonacci Heap is implemented in java without utilizing any external libraries and the corresponding address of all the nodes is stored in a Hash Map. For the Hash Map I have used an in-built structure. Max Fibonacci heap is required because it has better theoretical bounds for increase key operation.

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.

Fibonacci Max Heap	
Amortised Complexity	
Space	O(1)
Search	O(1)
Insert	O(log n)
Delete	O(1)
Find Max	O(1)
Delete Max	O(log n)
Increase Key	O(1)
Merge	O(1)

2. **Code Structure:**

The entire program consists of three classes:

1. **fibonacciHeap.java:** This class implements the entire logic of the maximum Fibonacci heap. The various operations supported by the class is inserting the node, increasing the value of a particular node, removing the maximum node, cutting the node, cascade cut.
   The details of the class are as follows

Class name	fibonacciHeap.java	
Description	This class is responsible for the implementation of the Fibonacci heap.	
Class members	MaxNode	The node containing the greatest value of the key
	NumberOfNodes	The total number of nodes present in the heap at any point of time.
Methods	public void insertNode(Node node) public void cut(Node node, Node node_parent) public void cascadingCut(Node node) public void IncreaseKeyValue(Node node, int value) public void makeChildNode(Node node1,Node node2) public void degreeWiseMergeNodes() public Node removeMaximumNode() public Node display()	

The various function prototypes of the following are:

public void insertNode(Node node)		
Description	The function is responsible for inserting the node in the Fibonacci heap	
Parameters	node	The node to be inserted in the Fibonacci heap
Return type	void	

public void cut(Node node, Node node_parent)		
Description	The function is responsible for performing the operation cut on the nodes. Cut is predominantly performed when the value of the key becomes greater than that of the parent.	
Parameters	node	The child node to be cut
	node_parent	The parent node from which the child node is to be removed.
Return type	void	

public void cascadingCut(Node node)		
Description	The function is responsible for performing the cascade cut operation on the nodes of the Fibonacci heap. The cascade cut operation is to keep tract of the childCut values of the nodes.	
Parameters	node	The child node to be cut

Return type	void

public void IncreaseKeyValue(Node node, int value)		
Description	This function is responsible for performing the increase key value of the node function. After the value of the key is increased, we decide whether a cut followed with cascade cut is required.	
Parameters	node	The node whose key is to be modified
	value	The value by which the key is to be increased.
Return type	void	

public void makeChildNode(Node node1,Node node2)		
Description	This function is responsible for forming the child nodes when two nodes are given. Provided two nodes, the parent of the node is decided on the basis of the key of the node. The node with the greater key becomes the parent.	
Parameters	node1	One of the nodes for the forming the child.
	node2	The other node used for child creation
Return type	void	

public void degreeWiseMergeNodes()	
Description	This function is responsible for degree wise merge operation. The degree wise merge operation takes place after the maximum node of the heap is removed.
Parameters	Null
Return type	Void

public Node removeMaximumNode()		
Description	This function is responsible for removing the maximum node from the Fibonacci heap. The criteria for maximum node being the greatest value of the variable – “key” of the node.	
Parameters	void	
Return type	Node	The greatest node present in the Fibonacci heap

public Node display()	
Description	This function is responsible for displaying the contents of the fibonacci heap
Parameters	void
Return type	void

2. **Node.java:** Consists of the definition of the node.

Class name	Node.java	
Description	This class contains the structure of the node required for the implementation of the Fibonacci heap.	
Class members	left_Sibling	Pointer to the left sibling of the node under consideration
	right_Sibling	Pointer to the right sibling of the node under consideration
	parent	Pointer to the parent of the node under consideration
	child	Pointer to the child of the node under consideration
	degree	The total number of children of the node under consideration.
	childCut	True/false for the node depending if the node under consideration has lost a child after it became the child of the present node.
	Data	The keyword from the input file
	key	The value of the frequency of the Data.
Methods	Constructor for the node class.	

Node(String Data, int key)		
Description	Constructor for the node class.	
Parameters	Data	The string which contains the value of the text to be inserted
	node2	The value associated with the Data field.
Return type	This pointer.	

3. **keywordcounter.java:** The main program for the calculation of the tags.

Class name	**keywordcounter.java**
Description	This class contains the main program for the calculation of the words.
Class members	Null
Functions	Main method.

3. **Instructions to run the program:**

The project has been compiled and tested on thunder.cise.ufl.edu and java compiler on local machine.

To execute the program, follow the instructions below:

1. Using,

   ssh username@thunder.cise.ufl.edu

   remotely access the server and upload the zip file.
2. For running the keywordCounter program,
   a. Extract the contents of the zip file
   b. Type 'make' without the quotes.
   c. Type 'java keywordcounter <path to the input file>.
3. To observe the results of the program, open "output_file.txt".

4. New Sample test cases:

- sample1.txt
- sample2.txt
- sampleMillion.txt