COP5536 ADVANCED DATA STRUCTURES
FALL 2016
PROJECT REPORT

Submitted By:

Abishek Sethuraman
abisheksethu@ufl.edu
UFID: 5718-6021

COMPILING INSTRUCTIONS:

This project has been written in Java and can be compiled using jdk1.8.0_111.
The submitted folder also contains:

1. Makefile
2. The .java files:
   HeapNode.java
   Heap.java
   hashtagcounter.java

PROGRAM STRUCTURE:

The project has three classes:

1. HeapNode.java
2. Heap.java
3. hashtagcounter.java

1. HeapNode.java:

This class handles the node structure used in the Fibonacci Heap.
The following are the class variables:

```
int frequency;
String word;
HeapNode parent;
HeapNode leftSibling;
HeapNode rightSibling;
HeapNode child;
int degree;
boolean isTopLevel;
boolean isMaxNode;
boolean childCut;
```

This is the constructor for the class:

```
public HeapNode(String name, int frequency)
```

Parameters: The word s as String type and frequency as int type.
Description: Constructor

2.Heap.java

This class handles implementation of the heap. It works on HeapNode instances i.e. nodes of the heap.

The following are the class variables:

```
public HeapNode heapMax
int noOfNodes
Hashtable<String,HeapNode> ht = new Hashtable<String,HeapNode>()
```

```
int noOfTopLevel
List<HeapNode> ptrList = new ArrayList<HeapNode>()
```

The following are the class methods:

`public Heap(String s,int frequency)`
Parameters: The word s as String type and frequency as int type.
Description: Constructor

`public void insert(String s,int frequency)`
Parameters: Word s in String and frequency in int.
Return Type: void
Description: If the word does not exists then it creates a HeapNode and inserts it into the Fibonacci Heap and adds an entry into the HashTable ht with the word(String) as the key and the HeapNode i.e. the pointer to the node as the value.
If the word already exists it calls increaseKey() with the HeapNode and frequency. It gets the pointer i.e. HeapNode using the HashTable ht. The heapMax is adjusted finally by calling checkWithMax().

`public void increaseKey(HeapNode h, int frequency)`
Parameters: HeapNode h whose frequency is to be increased and frequency in int.
Return Type: void
Description:Increases the frequency of a node.
If it is a top level node it simply increases the frequency.
If it is any other node not at top level then it checks to see if the frequency after the increase is greater than the frequency of its parent. If so, it removes the node (and its subtree)and attaches it to the top level of the heap as the rightsibling of heapmax i.e. the max node. At this point, it also checks if its parent has a childCut value equals to true and calls cascadeCut() to remove the parent and its subtree and place it on the top level of the heap. The check for parent's childcut is done recursively till the corresponding node's parent's childcut is false. This node's childcut is made true if it is not a top level node. The heapMax is adjusted finally by calling checkWithMax() for all top level nodes.

`public HeapNode removeMax()`
Parameters: None
Return Type: HeapNode
Description: Removes the max frequency node from the heap and returns it to the caller.
This function initially checks if the max node has children and assigns the heapmax *temporarily* to one of its children. If the max node does not have children then the heapmax is *temporarily* assigned to its rightsibling. The tree is added to an arraylist *ptrList* as per its degree(only one tree per degree in the arraylist) within a while loop. Within the while loop, the tree(using top level node) is merged using combineBack() if its degree is same as that of a degree which has been already added into the *ptrList* already in one of the iterations loop before this iteration. The while loop ends the moment all trees are merged as much as possible(i.e. While matching degree as much as possible). To be noted that combineBack() is a recursive function. The heapMax is adjusted finally by calling checkWithMax().

`public void checkWithMax(HeapNode h)`
Parameters: HeapNode to be checked with the max node.
Return Type: void
Description: The HeapNode's frequency is compared with the Max frequency node's frequency and updates the heapMax if it is greater accordingly.

`public void cascadeCut(HeapNode h)`
Parameters: HeapNode
Return Type: void
Description: Removes a node from the tree and places it adjacent to the max node if after an increase key the node's frequency is greater than its parent's frequency and further if the childcut is true.

`public void combineBack(HeapNode h)`
Parameters: HeapNode
Return Type: void
Description: Called within removeMax(). This is a recursive function that merges a specific tree i.e. top level node with another that is already in the arraylist if the degree is same. It calls itself if two trees have been newly merged to get a new tree which(is removed from the arraylist) and will be checked again with the trees in the arraylist for further merging if possible and simply added back into the arraylist. The recursion stops when all the trees/top level nodes in the arraylist have unique degrees again. It then returns to the while loop in the removeMax() to go to the next tree that has not been merged or handled yet.

3.hashtagcounter.java

This class contains the main method.
This reads the input file and writes the required data to the 'output_file.txt'. For every query, the corresponding number of maximum nodes are removed and stored in an arraylist. Once they have been written into the output file, the nodes are inserted back into the heap for further computation.