COP5536: Advanced Data Structures, Fall 2016

Project Report

Sanjana Shashidhar
UFID: 48954285 Email: sanjanarao83@ufl.edu

Compiling Instructions:

The project has been written in java and can be compiled using standard JDK 1.8.0_60, with javac compiler.

Unzip the submitted folder. The folder contains the project files (.java files), the Makefile, and my output file (.txt). The input text files must be placed in the project directory before running the program.

To compile in the Unix environment, run the following command after changing the directory to the project directory:

```
$ make
```

To run the program such that it supports redirected input from a text-file “filename” that contains the initial sorted list, use the following command:

```
$ java hashtagcounter inputfile
```

Program Structure:

The project has three classes, “hashtagcounter.java”, “HeapNode.java” and “FibonacciHeap.java” with “hashtagcounter.java” the Main function.

1. **hashtagcounter.java**

This class contains the main function that reads the input file, initializes the hashtable that contains the hashtag and the node address and parses through the input file to perform relevant tasks.

2. **HeapNode.java**

This class describes the structure of the node to be inserted in the Fibonacci heap.
The class contains the following variables and pointers:
Pointers:

a. Child (HeapNode): Pointer to a child of the node
b. Parent (HeapNode): Pointer to the parent of the node
c. Leftsib (HeapNode): Pointer to the left sibling of the node
d. Rightsib (HeapNode): Pointer to the right sibling of the node

Variables:

a. Nodedata (int): Contains the frequency of the hashtag (Value of the heap node)
b. Degree (int): Stores the number of children of the node
c. Mark (boolean): Sets the child cut value to true or false

We have a constructor in this class that initializes the pointers to the current node and marks the node to false.

3. **FibonacciHeap.java:**

The function prototypes of FibonacciHeap Class are given below:

```
public void heapInsertion(HeapNode node)
```

Parameters: HeapNode node
Return Type: void
Description: A function to insert a node into the Fibonacci heap.

```
public void reinsertHeapNode(HeapNode node)
```

Parameters: HeapNode node
Return Type: void
Description: A function that reinserts removed node back to the heap. It calls the heapInsertion function with the node to be reinserted as the parameter.

```
public HeapNode InsertHeapNode(int data)
```

Parameters: int data
Return Type: HeapNode
Description: A function that creates a new node to insert into the heap. The node will have the data which was passed as a parameter. The function returns the address of the node to the main function which stores that node address in the hash table.

```
public HeapNode removeMax()
```

Parameters: void
Return Type: HeapNode
Description: A function that removes the maximum element from the Heap. Once removed it inserts all the max nodes children in the root list and calls pairwiseCombine() function to consolidate the elements of the heap.

```
protected void pairwiseCombine()
```

Parameters: void
Return Type: void
Description: A function that checks two nodes of same degree and calling the meld() function to make the smaller node as the child of the bigger node. It then sets a new max node for the heap.

```
public void meld(HeapNode childnode, HeapNode parentnode)
```

Parameters: HeapNode childnode, HeapNode parentnode
Return Type: void
Description: A function that takes two nodes of same degree as parameters and makes the smaller node as the child of the bigger node. If the bigger node already has children it makes the smaller node connect to the right of the child of the bigger node.

```
public void IncreaseKey(HeapNode node, int data)
```

Parameters: HeapNode node, int data
Return Type: void
Description: A function that takes the node and the data that needs to be added to the value of that node. On increasing the node value if the value is greater than its parent, the node has to be disconnected from its parent and added to the root list. For this the function calls childcut and cascading cut functions respectively.

```
public void childCut(HeapNode node)
```

Parameters: HeapNode node
Return Type: void

Description: A function that takes the node to be disconnected from its parent and inserts it in the root list. It also marks the parent of the node as false.

`public void cascadingCut(HeapNode node)`

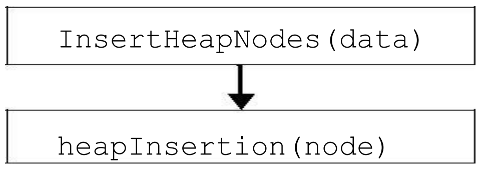
Parameters: HeapNode node

Return Type: void

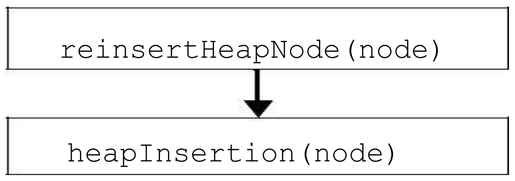
Description: A function that checks the parent of the current node repeatedly calls child cut and cascading cut on the node until it finds a node with childcut value false. If the node encountered is an internal node, it sets it to true.

The various operations carried out by this class are diagrammatically represented below:

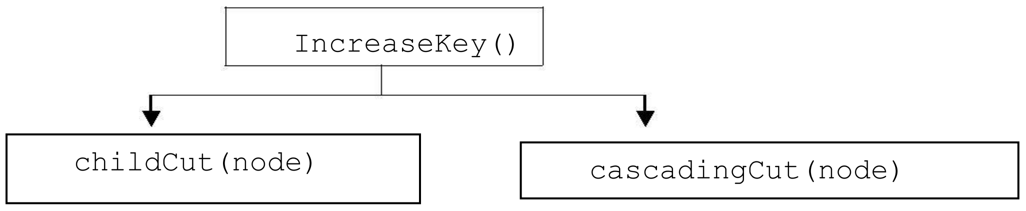
1. Initialize Fibonacci Heap:



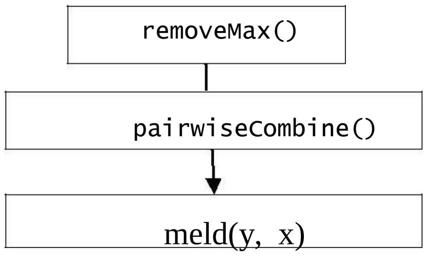
2. Reinsert Fibonacci Heap Node:



3. Increase Hashtag frequency:



4. Remove the max element:



Output:

The output for the input file is included in the project directory as output_file.txt