Programming Project Report

HashTag Counter using Fibonacci Heap

Advanced Data Structures

Aditya Kant
adikant9@ufl.edu
UFID - 2549-8901

Project implementation done in JAVA.
java version "1.8.0_65"
Java(TM) SE Runtime Environment (build 1.8.0_65-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.65-b01, mixed mode)

Compiler: javac "1.8.0_65"

Program Execution –

- Project submission contains 3 java files and *makefile* which creates an executable with name *hashtagcounter*.

- Command line for program –
    - $ make - the will compile the project
    - $ java hashtagcounter file-name

Program Requirements -

- No uppercase letters in the input stream
- No spaces in the hashtags. (only one word per one hashtag)
- For two hashtags to be same, whole word should match. i.e. #playing and #playing_games are two different hashtags
- One query has only one integer.
- If there are more than one hashtag with similar frequencies,it is printed in any order.
- While running the hashtagcounter program only the ‘input-file’ name should be given as its arguments **without** the complete path of the file.
- Output of the program is written to a file ‘output_file.txt’ in the same directory as the input file.

Program Structure -

Project structure consists of 3 files –
- Node.java
- FibonacciHeap.java
- hashtagcounter.java

Detailed Structure of each file as follows -

Node.java

Class for the nodes in the Fibonacci Heap.

Defines following Properties of a Node –

int key	Frequency of the hashtag corresponding to the Node
String tagName	HashTag corresponding to the Node
int degree	Number of children of the Node
Node parent	Reference to the Parent of the Node
Node child	Reference to the Child of the Node
Node prev	Reference to previous sibling Node
Node next	Reference to next sibling Node
Boolean isMarked	Flag for child cut

Method Detail

- **public int getKey()**
  Procedure to get key of the node.

- **public String getTag()**
  Procedure to get tag corresponding to the node.

- **protected int compare(Node a)**
  This method compares this.key to a.key and returns 1 if this.key is greater than a.key else it will return -1.

FibonacciHeap.java

Class for the fibonacci heap which is an actual implementation of max fibonacci heap which supports various operations specified in the project description.

Function Prototypes for the max fibonacci heap -

Method Detail

- **public Node insert(int key, String tag)**
  This procedure creates a node and inserts node x into Fibonacci heap H
  @param key

@param tag
@return Node

- **public void increaseKey(Node node, int newKey)**
  This procedure increases node's key to newKey
  @param node
  @param newKey

- **private void cut(Node node, Node parent)**
  The CUT procedure cuts the link between node and its parent, adding node to the rootlist.
  @param node
  @param parent

- **private void cascadingCut(Node node)**
  If node is marked, it has just lost its second child; node is cut, and CASCADING-CUT calls itself recursively node's parent. The CASCADING-CUT procedure recurses its way up the tree until either a root or an unmarked node is found
  @param node
- **public Node extractMax()**
  extractMax works by first making a root out of each of the maximum node's children and removing the maximum node from the root list. It then consolidates the root list by linking roots of equal degree until at most one root remains of each degree.
  @return Node

- **private void consolidate()**
  Consolidating the root list consists of repeatedly executing the following steps until every root in the root list has a distinct degree value
  - ❏ Find two roots x and y in the root list with the same degree, where key[x] >= key[y].
  - ❏ Link y to x: remove y from the root list, and make y a child of x. This operation is performed by the **private void linkHeaps(Node nodeA, Node nodeB)** procedure. The field degree[x] is incremented, and the mark on y, if any, is cleared.

- **private void removeNodeFromSiblings(Node nodeA)**
  This procedures removes the nodeA from the linkedList of siblings
  @param nodeA

- **private void linkHeaps(Node nodeA, Node nodeB)**
  Links **nodeA** to **nodeB**: remove **nodeA** from the root list, and make **nodeA** a child of **nodeB**.

- **public Node merge(Node a, Node b)**
  Merges two lists and returns the max node
  @param a
  @param b
  @return Node

References –

- Introduction to Algorithms, 3rd Edition Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.