Advances Data Structures (COP 5536)

Fall 2016

Programming Project Report

Maulik Lalani

UFID 13044420

mlalani@ufl.edu

# PROJECT DESCRIPTION

Project is required to implement a system to find the n most popular hashtags appeared on social media such as Facebook or Twitter. Basic idea for the implementation is to use a max priority structure to find out the most popular hashtags. Project is required to use two data structures for the implementation: Max Fibonacci heap to keep track of the frequencies of hashtags and Hash table to store hashtag and pointer to the corresponding node. The input file will have large number of hashtags and may need to perform increase key functionality many times.

**COMPILING INSTRUCTIONS**

* The project has been compiled and tested on thunder.cise.ufl.edu, on local terminal and on Eclipse Kepler.
* Java Version on my local machine: **1.8.0\_111**
* I have included two input files **millionRecords.txt** which contains around 1.3 million records and **sample\_input1.txt.**
* General command to execute the program,
  + java hashtagcounter file\_name
* Commands to execute the program with the given input file
  + Java hashtagcounter millionRecords.txt
  + Java hashtagcounter sample\_input1.txt

**PROGRAM STRUCTURE AND DESCRIPTION**

There are three classes used to implement the project. Following is the short description of each method used in these classes.

1. **Fibonacci\_Heap\_Node.java**

This class contains the structure of the Fibonacci heap and one constructor to initialize the node. Single node of the Fibonacci heap contains following fields:

**Child, Parent, Right\_sibling, Left\_sibling, data, Degree and ChildCut.**

**Function Methods:**

/\*\*

\* This constructor initializes node fields and assigns data.

\* **@author** Maulik\_Lalani

\* **@param** data

\*/

**a) public Fibonacci\_Heap\_Node(int data )**

This constructor initializes all the fields of the node and initializes data field with the passed parameter.

1. **Hashtagcounter.java**

This class contains the main method and it initializes the Hash table used to store the hashtags.

**Function methods:**

**a)public static void main(String args[]);**

This is the main method and code begins from this method. This method initializes buffer reader and writer to read and write data to the output method. This method also calls either insertHashtag() or executeQuery() based on the input.

/\*\*

\* This method calls the insert methods of Heap\_Methods class based on new \*insert or update.

\* **@author** Maulik\_Lalani

\* **@param** hashtag, frequency

\* **@param** frequency

\*/

**b) public static void insertHashtag(String hashtag, int frequency);**

This method creates the new node with the given frequency if hashtag is not present and calls insertNewNode() method to insert into the heap. If hashtag is present than it calls increaseKey() method which updates the frequency.

/\*\*

\* This method fetches the query results and remove nodes from the heap and then reinserts all the nodes.

\* **@author** Maulik\_Lalani

\* **@param** query

\*/

**c) public** **static** **void** executeQueries(**int** query);

This method takes query as an argument and removes maximum nodes and calls writeToOutputFile() which writes the output data and then reinserts all the nodes using insertHashtag().

/\*\*

\* This method writes the one query result to the output file.

\* **@author** Maulik\_Lalani

\* **@param** queryResult

\*/

**d)public static void writeToOutputFile (LinkedHashMap<String,Fibonacci\_Heap\_Node> queryResult);**

This method writes the data to output file.

**3)Heap\_Methods.java**

This class contains all the methods that perform increase, remove and update operations.

**Function methods:**

/\*\*

\* This function inserts new root node to the Fibonacci heap

\* **@param** newNode

\* **@author** Maulik\_Lalani

\* **@return** Fibonacci\_Heap\_Node

\*/

**a) public Fibonacci\_Heap\_Node insertNewNode(Fibonacci\_Heap\_Node newNode);**

This method insert the new node as the right sibling of heap pointer and updates the max heap pointer. It also assigns appropriate left and right pointers.

/\*\*

\* Removes the max heap pointer from the fibonacci heap and does pairwise combining of all the root nodes.

\* **@author** Maulik\_Lalani

\* **@return** Fibonacci\_Heap\_Node maxValue

\* **@param**

\*/

**b) public Fibonacci\_Heap\_Node removeMax();**

This method removes the heap pointer node from the heap and creates one array which has all the siblings of heap pointer and children of the heap pointer and passes it to meldTrees() to reconstruct the tree with all the nodes.

/\*\*

\* This method iterates through the list of root nodes and calls pairwiseCombine() method to merge nodes based on the degree

\* **@author** Maulik\_Lalani

\* **@param** rootNodes (Contains all the rootnodes for the pairwise combine)

\* **@return** void

\*/

**c) public void meldTrees(ArrayList<Fibonacci\_Heap\_Node> rootNodes);**

This method creates table which contains node and its degrees. Then it iterates through all the nodes and put it in table. If node degree is already present then the node is taken out of the table and pairwiseCombine() method is called which combines two nodes. Once we get all the unique degree nodes, mergeNode() method is called to merge all the nodes one by one.

/\*\*

\* This method combines two root nodes based on the frequency and make small node the child of the large node.

\* @author Maulik\_Lalani

\* @param largeNode

\* @param smallNode

\* @return Fibonacci\_Heap\_Node

\*/

**d) public Fibonacci\_Heap\_Node pairwiseCombine(Fibonacci\_Heap\_Node largeNode, Fibonacci\_Heap\_Node smallNode);**

This method is called when we have two nodes with the same degree and it merges both the nodes based on the frequency. Smaller node is made the child of the parent.

/\*\* This method assigns appropriate right and left child of the newly created root node

\* **@author** Maulik\_Lalani

\* **@param** newNode

\* **@return** newNode

\*/

**e) public Fibonacci\_Heap\_Node mergeAllNodes(Fibonacci\_Heap\_Node newNode);**

This method is called after creating unique degree root nodes and this method assigns right and left sibling pointers to all the nodes. Finally, it creates max Fibonacci heap.

/\*\*

\* This method updates the frequency of a hashtag, removes it from the tree if it is greater than parent and inserts it at the top.

\* **@author** Maulik\_Lalani

\* **@param** node

\* **@param** frequency

\*/

**f) public void increaseKey(Fibonacci\_Heap\_Node node, int frequency);**

This method is called when hashtag is already present and it updates the frequency of the hashtag. It increases the data and if it is greater than parent, then the node is removed from the tree and reinserted as a root node using insertHashtag(). It updates the parent’s child cut and removes the parent also if child cut is true and thus it traverses to the parent and check the child cut field.

**FLOW CHART FOR THE INSERT AND UPDATE OPERATION**

Main() InsertHashtag() If hashtag is not present InsertNewNode()

If hashtag is present

IncreaseKey() If node frequency greater than parent

**FLOW CHART FOR THE DELETE OPERATION**

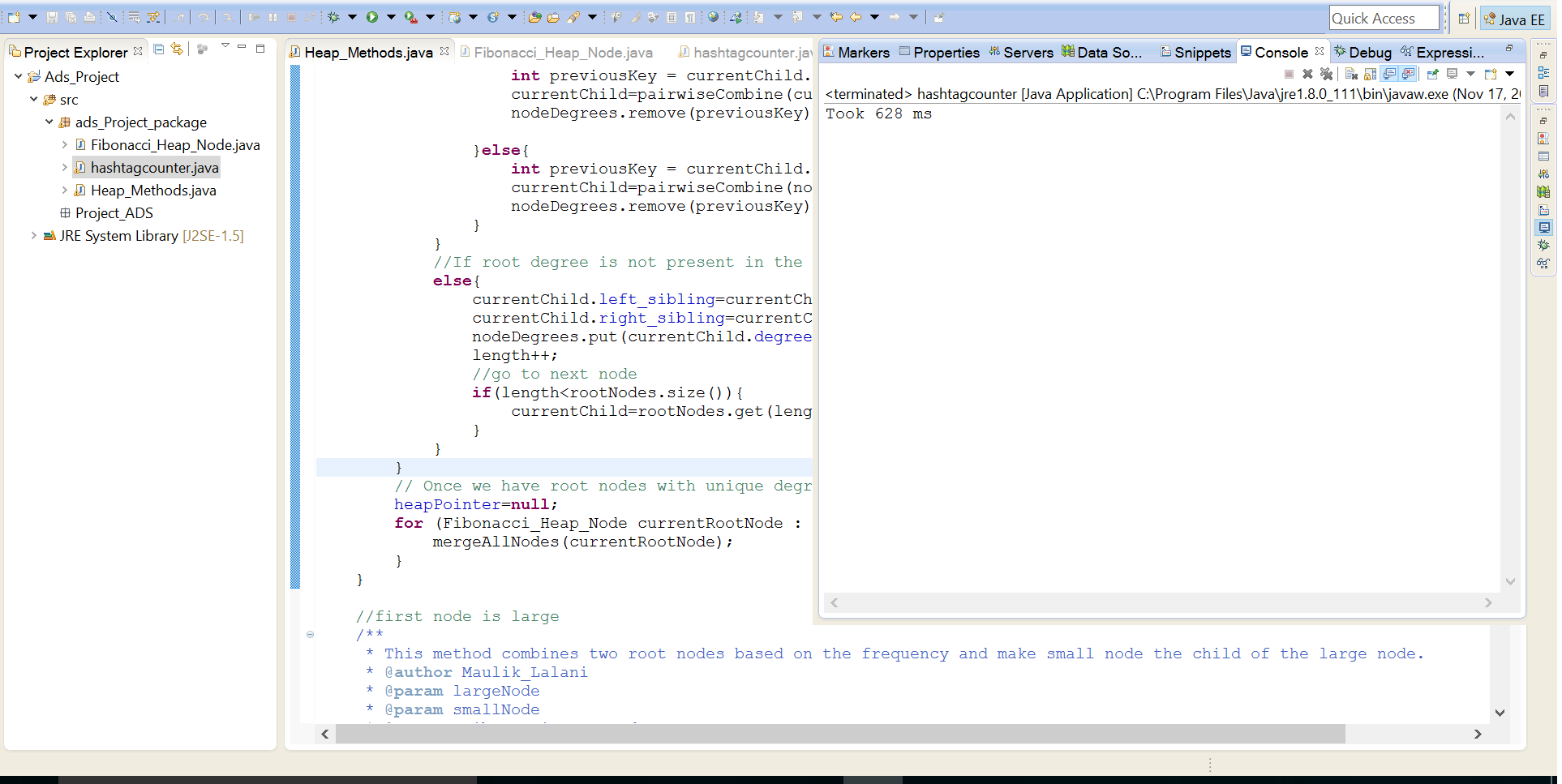
Main() executeQueries() removeMax() Meldtrees()

writeToOutputFile() mergeAllNodes() pairwiseCombine()

insertHashtag()

**RUNNING THE CODE**

* I ran the code on eclipse kepler for the input file millionRecords.txt and it took **628ms** to execute.



* I also ran the code on thunder.cise.ufl.edu server for 1.3 million records and it took **1583ms**.

