ADVANCED DATA STRUCTURES (COP 5536) SPRING 2020

PROGRAMMING PROJECT REPORT

Submitted to PROF. SARTAJ SAHNI

TOPIC - N – POPULAR HASHTAGS

(USING MAX FIBONACCI HEAP AND HASHTABLE)

**SUBMITTED BY -**

**Name - SHRUTI PARIHAR**

**UFID – 92153237**

**Email – shruti.parihar@ufl.edu**

1. INTRODUCTION

In this project we implemented a hashtag tracker to determine the n- most famous hashtags on any social networking site such as Instagram, Twitter or Facebook. This project can prove to be useful because of the very small working complexity of the code.

Basic idea behind the project is to implement a Max-Fibonacci Heap where each node represents a distinct hashtag. The node store the frequency of the hashtag. These hashtags and frequencies can be mapped to each other with the help of Maps in C++. Since the Fibonacci Heap has the amortized complexity of O(1) that will significantly reduce the processing time while computing the n-most famous hashtags from the file. The output is then written to an output file.

1. STRUCTURE OF PROGRAM

My project comprises of only one file ‘main.cpp’. The project contains only one class that is used to describe a node of the heap. Various operations are performed on a user defined Fibonacci heap which is represented in the given program in the form of the structure node. The class *node* consists of various attributes –

* *int occurence*
* *node\* left\_sib*
* *node\* right\_sib*
* *node\* parent*
* *node \*child*
* *string hash\_tag*
* *unsigned long int deg*
* *bool child\_cut*

Other program variables include a nodetype element rootnode and nodes. The rootnode points to the node with the largest frequency.

The other nodes of type nodetype represent the nodes of the Max Fibonacci Heap.

The various functions that support a specific operation are :

* **Insert a hashtag to the root level list**

*void insert\_node\_function(node\* new\_node, bool bool\_var)*

*void link\_node()*

* **Remove Maximum Frequency Hashtag**

*void combine\_two\_trees\_function(node\* c,node\* p)*

*void pairwise\_combine\_function()*

*node\* remove\_max\_function()*

*void reinsert\_node\_function()*

* **Increase the frequency of Hashtag**

*void cut\_node\_function(node\* cut\_node, node\* parent\_node)*

*void increase\_frequency\_function(node\* newnode)*

*void cascade\_cut\_function(node\* affected)*

1. FUNCTION PROTOTYPE AND DESCRIPTION

* **int main(int argc, char \*argv[])**

return type – int

argument – integer argc, character pointer array argv

The execution of the program starts with main. It takes an input file and an output file as parameter. The input file containing hashtags is opened and the file is read line by line till there is a stop command in the input file.

If it encounters a hashtag it increases its frequency if it is present in the heap and creates a new node if there is no node associated with the hashtag.

* **void insert\_node\_function(node\* new\_node, bool bool\_var)**

The function creates a new hashtag whenever it encounters a new one. It creates a new node and then insert it into the Max Fibonacci Heap to the root level.

* **void link\_node()**

Add the node to the root's children list. It removes node from the sibling list it was earlier a part of and then add the given node to root's children list.

* **void combine\_two\_trees\_function(node\* c,node\* p)**

The function performs merging of two sub trees. The one with the larger occurence will become the parent and the other one the child.

* **void reinsert\_node\_function()**

Function to re-insert the deleted heap nodes. The deleted heap nodes are stored in a vector and then after extracting the n popular hashtags, the deleted nodes are inserted back into the heap.

* **void pairwise\_combine\_function()**

It is a function that combines two sub-trees that are on the root level and have the same degree. It recursively combines all the trees until either there is only one tree left or the trees have unequal degrees.

* **node\* remove\_max\_function()**

Remove the node with the maximum occurence from the heap. Its subtress are then re-inerted into the heap at the root level. Then a pairwise combining function is called that combines all the pair of sub-trees formed.

* **void cut\_node\_function(node\* cut\_node, node\* parent\_node)**

Cut a given node from its parent's child list. Reduce the degree of the parent since it is losing a child. Then a pairise combine function is called that merges the subtrees formed.

* **void increase\_frequency\_function(node\* newnode)**

This function increase the occurence of the node by the given value. After increasing the occurence frequency the frequency of the node may get larger than its parent. A cut operation is done if the frequency of the node is greater than the frequency of its parent. After cutting the sub tree, performing cascading cut is done accordingly.

* **void cascade\_cut\_function(node\* affected)**

This function performs cascading cut on the parent if the childcut value of the parent is true. The parent is also detached from its parent until the parent is root or the childcut value of the parent is false