Advances Data Structures (COP 5536)
Fall 2016
Programming Project Report

Suraj Kota
suraj.k@ufl.edu

Contents

I. Introduction

In this project, we have implemented a system to find the n most popular hashtags that appeared on social media such as Facebook or Twitter. Hashtags are given from an input file.

Basic idea for the implementation is to use a max fibonacci heap to find out the most popular hashtags because it has better theoretical bounds for increase key operation. This section states the functions that support the required operations for our hashtag counter using max Fibonacci heap.

a. Insert a hashtag to the top level root list

   void insert_node(node item_to_insert, bool old)*

b. Remove Maximum frequency hashtag

   node removemax()*

   void pairwise_combine() - *supporting function for removemax*

   void combine_two_trees(node c,node* p)* -*supporting function for pairwise combine*

c. Increase the frequency of a hashtag

   int increase_frequency(node changenode)*

   void cut(node cutnode, node* parentofcut)* -*supporting functions for*

   void cascade_cut(node affected)* *increase frequency*

II. Structure of Program

The project has only one file hashtagcounter.cpp. All the operations stated in previous section are performed on a fibonacci heap node which is represented in the program with the class *node* and has the following attributes (same as a standard max fibonacci heap node except here we store hashtag also),

int frequency;

string hashtag;

unsigned long int degree;

bool childcut;

node left;*

node right;*

node child;*

node parent;*

Other program variables include,

- hashmap named *locationhash* which contains hastags as its keys and value as a pointer to the node where are hashtag resides. This is used to avoid the time to search for the key in the heap in case of the increase key operation.
- *root,* which is the heap pointer and points to the max node in the heap.
- *No_nodes,*which keeps track of the number of nodes in the fibonacci heap

The following flowchart presents the flow of the program.

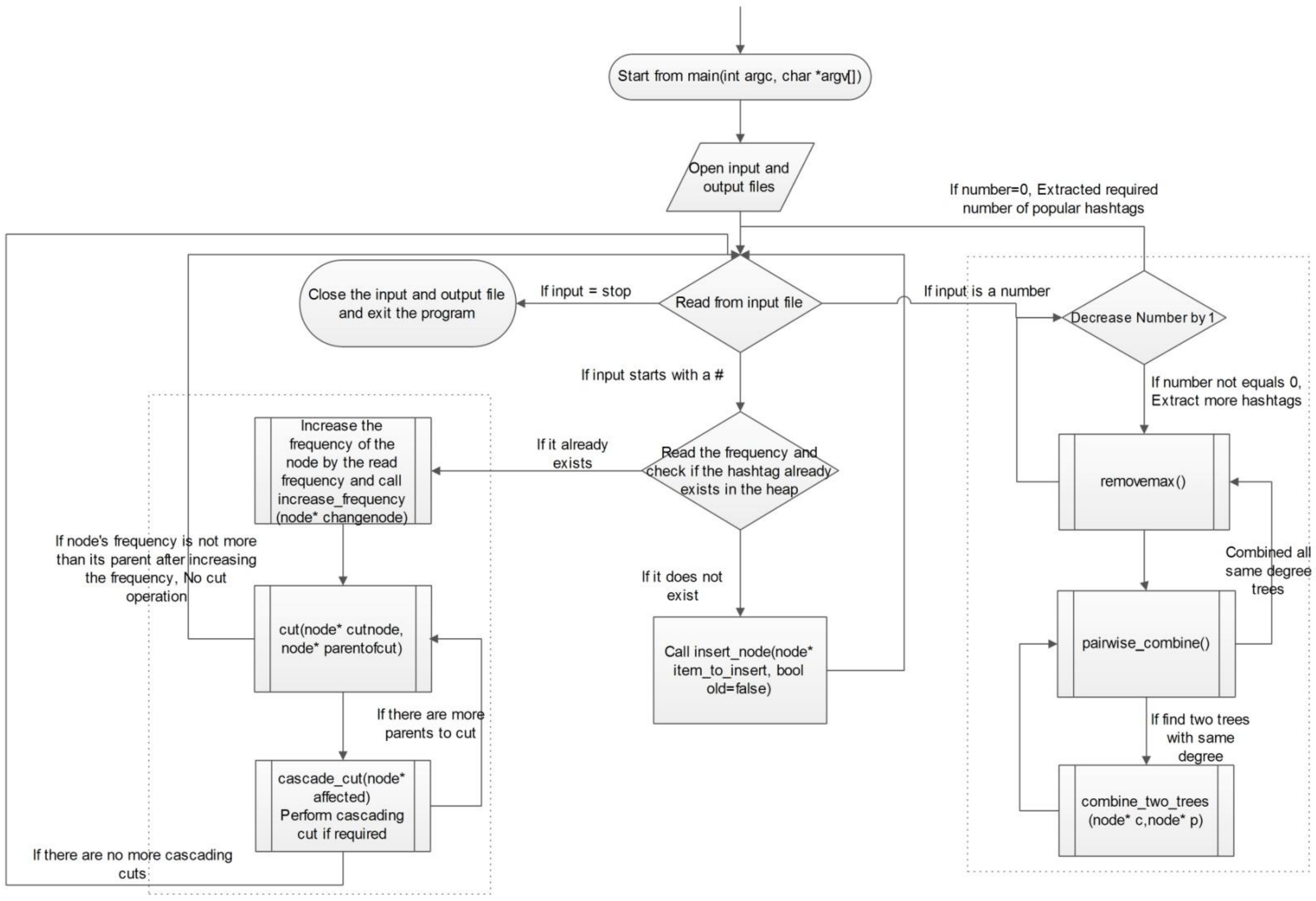

Start from main(int argc, char *argv[])
Open input and output files
If number=0, Extracted required number of popular hashtags
Close the input and output file and exit the program
If input = stop
Read from input file
If input is a number
Decrease Number by 1
If input starts with a #
If number not equals 0, Extract more hashtags
Increase the frequency of the node by the read frequency and call increase_frequency (node* changenode)
If it already exists
Read the frequency and check if the hashtag already exists in the heap
removemax()
If node's frequency is not more than its parent after increasing the frequency, No cut operation
Combined all same degree trees
If it does not exist
cut(node* cutnode, node* parentofcut)
Call insert_node(node* item_to_insert, bool old=false)
pairwise_combine()
If there are more parents to cut
If find two trees with same degree
cascade_cut(node* affected) Perform cascading cut if required
combine_two_trees (node* c,node* p)
If there are no more cascading cuts

Figure 1: Flowchart showing the flow of program

Page intentionally left blank. Please move to the next page

III. Function Prototype and Description

This section describes the working of the functions stated in the previous section.

a. ***int main(int argc, char *argv[])***

Return Type: int

Parameters: argc-number of command line arguments, argv[] pointer to the arguments passed to the executable

The main function is where the execution starts. First if checks if the source file with hashtags is passed. If so, it associates the input filename with standard input and opens the file for reading. Similarly, it associates the output_file.txt with standard output and opens the file for writing the output.

Then, it reads strings from the file till it reads stop,

if the input string starts with 's' or 'S' which corresponds to STOP or stop, it closes the input and output files and returns 0 to operating system. Else,

if the input was a hashtag, it starts with '#', it will have a corresponding frequency to be scanned. The program checks whether this hashtag already exists in the heap using the locationhash. If so, it increases its frequency by the amount scanned and calls *increase_frequency()* function and the program continues to read next input. else, it inserts the hashtag as a new node and the program continues to read next input.

if the input was not a hastag, it will be count of number of maximum frequency hashtags to output. The string is converted to integer using *atoi(*char[])* function and corresponding to the count, that number of calls to *removemax()* are made and removed nodes are stored in a vector to re-insert them afterwards. At the same time, the maximum frequency hashtags are written to output file. Once this is done the removed nodes are re-inserted to the fibonacci heap and the program continues to read next input.

b. ***void insert_node(node* item_to_insert, bool old)***

Return Type: void

Parameters: item_to_insert -The node to insert (the node maybe a single node or a root of sub tree) and a boolean variable indicating whether the node to be inserted a new node or an old node.

Complexity: O(1) actual

If there is no node in the heap, the item being inserted is made as *root*(HEAP POINTER-points to maximum node in the heap),

else,
the node is inserted to the left the root and corresponding pointers are modified in the sibling list and is made the *root* of the the heap if its frequency is more than the old *root.*

The boolean value ***old*** helps keep track of number of nodes in the heap. There are number of operations that take help of insert operation as their subroutine, for example, when we remove the max element, we want to re-insert its sub trees to the top level list or when we do cut or cascading cut we want to insert the nodes or subtree rooted at the cut node to the top level list. While these the examples of nodes being re-inserted i.e. old nodes, the case where a new hashtag is inserted the boolean variable old is false and the count to number of nodes is increased.

c. ***node* removemax()***

Return Type: node with the maximum frequency hashtag

Parameters: None

Complexity: O(log n) amortized

In max fibonacci heap, the *root* of the heap is the maximum node. Before this node is removed, its subtrees are re-inserted to the top level list if any using *void insert_node(node* item_to_insert, bool old=false)* function. If the node being removed is the only node left in the heap, the root is set to null. Else, this node is removed from its sibling list and pairwise combining is done with help of *pairwise_combine()* function which is explained below.

d. ***void pairwise_combine()***

Return Type: void

Parameters: None

Complexity: O(log n) amortized

This functions traverses through the top level list to see if there are any two trees with same degree. To do this, it makes use of a table which stores a previously found tree with degree *d* at table[d]. If two max trees with the same degree are found, it combines the two trees by making the tree with smaller root frequency as the child of the tree with larger root frequency and stores it in table[d+1] if it is empty or iteratively keeps on combining till it finds trees with same degree. To do the combining it takes the help of *combine_two_trees(node* c, node* p)* function as a subroutine. After these operations, there are no two trees with same degree in top level list. Now, the table is traversed to collect all the trees with different degree and combined to form the top level list as well updating the new heap pointer (*root*).

e. ***void combine_two_trees(node* c, node* p)***

Return Type: void

Parameters: Two trees c and p with same degree

Complexity: O(1) actual

Tree *c* becomes the child and tree *p*, becomes the parent of tree *c*. For this, first the tree *c* is removed from the top level list. Then, its *childcut* is made false and *p* is set as its parent. If tree *p* did not have any children earlier, tree *c* is made its child directly, else, tree *c* is inserted into the circular list of p's children and p's degree is increased.

f. ***void increase_frequency(node* changenode)***

Return Type: void

Parameters: the node whose frequency has increased

Complexity: O(1) amortized

After increasing the frequency of the hashtag, this function is called to check if its frequency has become more than its parent. If so, the program calls *cut(node* cutnode, node* parentofcut)* subroutine to cut the subtree rooted at this node and then calls *cascade_cut(node* affected)* to check the *childcut* values of the parent and its ancestors and cut them if it were true if it is not a top level list node. It then checks if the frequency of the changed node has gone higher than the current heap pointer and updates the heap pointer is necessary.

g. ***void cut(node* cutnode, node* parentofcut)***

Return Type: void

Parameters: cutnode-the node to be cut, parentofcut-the parent of the node that is being cut

Complexity: O(1) actual

This function removes a node from the child list of its parent, reduces the parent's degree and inserts the tree rooted at *cutnode* to top level list. If the *cutnode* was the only child of its parent, the parents child pointer is set to null. If this node was being pointed by the parent as its child, it changes that pointer to the next node it the child list.

h. ***void cascade_cut(node* affected)***

Return Type: void

Parameters: affected-the affected node from the cut

Complexity: O(log n); (O(1) per call i.e. without recursion)

This function checks the childcut values and cuts the node if it was true before the child was cut. If the childcut value is false for the affected node, it makes it true indicating that it has lost a child. The function recursively calls itself and keeps cutting the node until it finds a node with childcut false or it has reached the top level list.

IV. Running the Program

Program takes the input file name as an argument. Following is an example of a program that read from a file named file_name.

```
$ ./hashtagcounter file_name
```

<u>Input Format</u>

Hashtags appear one per each line in the input file and starts with # sign. After the hashtag an integer will appear and that is the count of the hashtag (There is a space between hashtag and the integer). You need to increment the hashtag by that count. Queries will also be appeared in the input file and once a query appears you should append the output to the output file. Query will be an integer number (n) without # sign in the beginning. You should write the top most n hashtags to the output file. Once it reads the stop (without hashtag) program should end. Following is an example of an input file.

```
#saturday 5
#sunday 3
#saturday 10
#monday 2
#reading 4
#playing_games 2
#saturday 6
#sunday 8
#friday 2
#tuesday 2
#saturday 12
#sunday 11
#monday 6
3
#saturday 12
#monday 2
#stop 3
#playing 4
#reading 15
#drawing 3
#friday 12
```

#school 6
#class 5
5
stop

Output Format

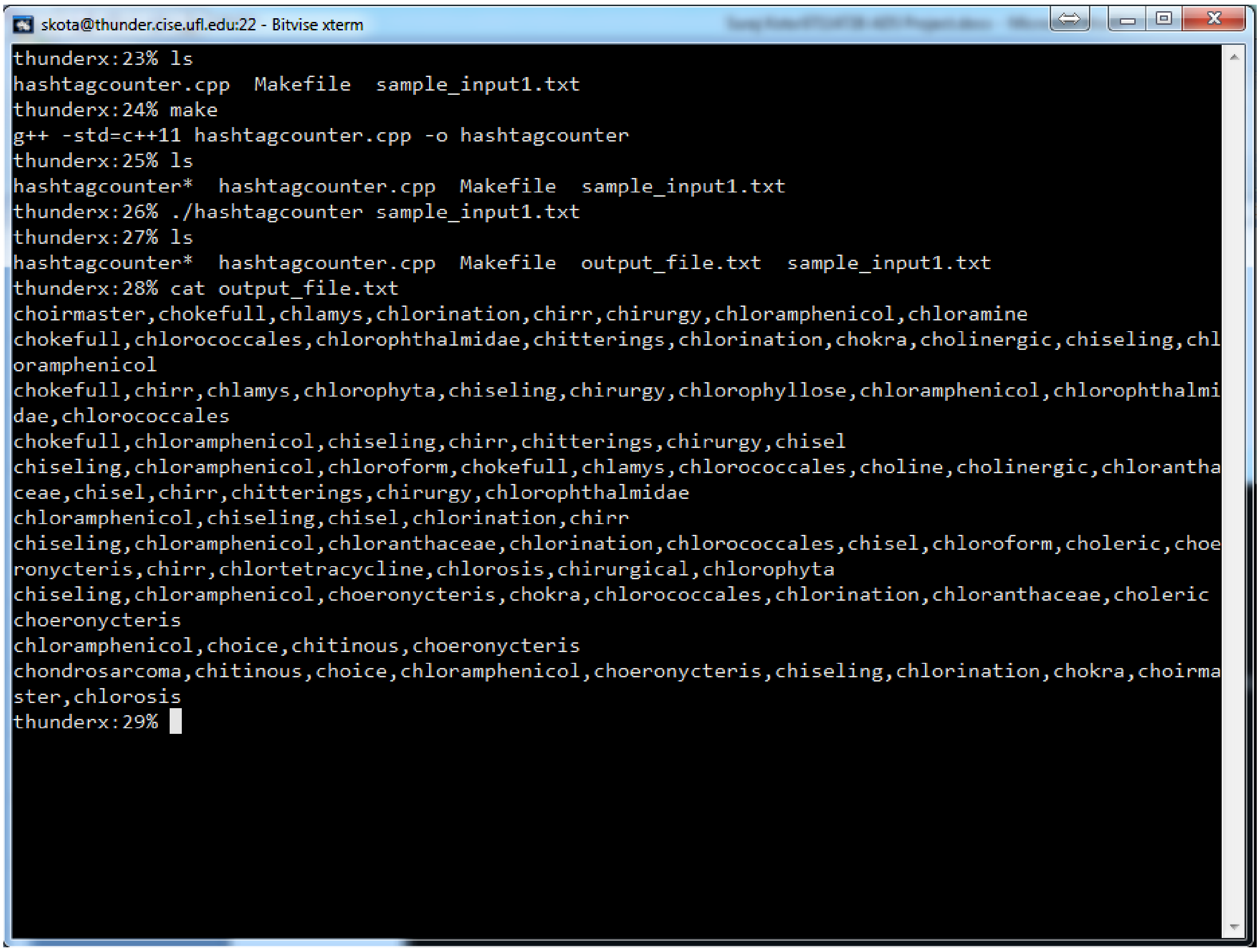
Once a query is appeared you need to write down the most popular hashtags of appeared number in to the output file in descending order. Output for a query should be comma separated list without any new lines. Once the output for a query is finished you should put a new line to write the output for another query. You should produce all the outputs in the output file named “output_file.txt”.

Following is the output file for the above input file.

saturday,sunday,monday

saturday,sunday,reading,friday,monday

Following is the screenshot of how to run the program:



```
thunderx:23% ls
hashtagcounter.cpp  Makefile  sample_input1.txt
thunderx:24% make
g++ -std=c++11 hashtagcounter.cpp -o hashtagcounter
thunderx:25% ls
hashtagcounter*  hashtagcounter.cpp  Makefile  sample_input1.txt
thunderx:26% ./hashtagcounter sample_input1.txt
thunderx:27% ls
hashtagcounter*  hashtagcounter.cpp  Makefile  output_file.txt  sample_input1.txt
thunderx:28% cat output_file.txt
choirmaster,chokefull,chlamys,chlorination,chirr,chirurgy,chloramphenicol,chloramine
chokefull,chlorococcales,chlorophthalmidae,chitterings,chlorination,chokra,cholinergic,chiseling,chl
oramphenicol
chokefull,chirr,chlamys,chlorophyta,chiseling,chirurgy,chlorophyllose,chloramphenicol,chlorophthalmi
dae,chlorococcales
chokefull,chloramphenicol,chiseling,chirr,chitterings,chirurgy,chisel
chiseling,chloramphenicol,chloroform,chokefull,chlamys,chlorococcales,choline,cholinergic,chlorantha
ceae,chisel,chirr,chitterings,chirurgy,chlorophthalmidae
chloramphenicol,chiseling,chisel,chlorination,chirr
chiseling,chloramphenicol,chloranthaceae,chlorination,chlorococcales,chisel,chloroform,choleric,choe
ronycteris,chirr,chlortetracycline,chlorosis,chirurgical,chlorophyta
chiseling,chloramphenicol,choeronycteris,chokra,chlorococcales,chlorination,chloranthaceae,choleric
choeronycteris
chloramphenicol,choice,chitinous,choeronycteris
chondrosarcoma,chitinous,choice,chloramphenicol,choeronycteris,chiseling,chlorination,chokra,choirma
ster,chlorosis
thunderx:29%
```

Figure 2: Screenshot showing steps of how to run the program

Assumptions

1. No uppercase letters in the input stream
2. No spaces in the hashtags. (only one word per one hashtag)

3. For two hashtags to be same, whole word should match. i.e. #playing and #playing_games are two different hashtags.
4. One query has only one integer.
5. If there are more than one hashtag with similar frequencies, you can print them in any order.
6. Input file may consist of more than 1 million hashtags.
In Fibonacci heaps removeMax() will remove the hashtag from the heap. So you need to re-insert them after printing in order make sure they are still in the heap.

V. Conclusion

We have successfully implemented the max fibonacci heap to work as a max priority queue and retrive the most n popular hashtags.