Hashtag Counter

Advanced Data Structures

COP 5536

Programming Project

Fall 2016

Submitted By

Rishabh Gogia

UFID: 8556-9711

Email: rishabhg@ufl.edu

Description & Implementation

Description

We need to implement a system to find out the n most popular hashtags trending on social media. As the hashtag frequencies are ever changing, there is a need to implement a system that can efficiently support the following operations:

1. Incrementing the frequency of a given hashtag
2. Extracting the most popular hashtag

Implementation

The following data structures are used for the implementation:

1. Fibonacci Heap: Implement a max priority queue based on the frequencies of the hashtags.
    a. Increase Key operation runs in O(1) in the amortized sense.
    b. Remove Max operation runs in O(log n) in the amortized sense.

2. Hash Table: A table to keep track of hashtags in the heap, with pointers to the nodes of the Fibonacci Heap.

Program Structure

Files

1. fibHeap.h - Header file containing “class FibHeap” and “structure node” definitions.
2. fibHeap.cpp - Contains the implementation of the methods of the Fibonacci heap class.
3. hashtagcounter.cpp - Contains the main program that maintains the Fibonacci heap, parses input file, calls appropriate functions of the FibHeap class to perform required operations and write any output to a file called “output_file.txt”.
4. makefile - File to compile and link the program using make command. Creates the executable “hashtagcounter”

Definitions: fibHeap.h

1. “struct node”: Declared in “fibHeap.h”. It is a structure to store a node of the Fibonacci Heap. Its data members are:
    - degree - Integer storing current node degree
    - hashtag - String to store the hashtag
    - frequency - Integer storing the frequency of the hashtag
    - child - Pointer to “struct node” to store the address to a child
    - left - Pointer to “struct node” to store the address of the left sibling
    - right - Pointer to “struct node” to store the address of the right sibling
    - parent - Pointer to “struct node” to store the address of the parent
    - childCut - Boolean storing true/false as an indication of whether current node has lost a child since it became child of its current parent

2. “class FibHeap”: Declared in “fibHeap.h”. It is a class representing a max Fibonacci heap. Its data members and member functions are given below. The member functions are implemented in “fibHeap.cpp”:

    - Data Members
        - numNodes - Integer storing the number of nodes currently in the heap.
        - max - Pointer to “struct node” pointing to the current max node in the heap.

    - Member Functions
        - FibHeap() - Constructor to initialize the heap
        - node* getNode(string hashtag, int freq) - Method to create and return a new node that can be inserted in a Fibonacci heap. Initializes the hashtag and its frequency as per arguments and other data members as per default values.
        - void insert(node* newNode) - Method to insert a node into the Fibonacci heap.

- void cut(node *ptr) - Method to remove a node from the Fibonacci heap and add it to top level list. This may be used during increase key operations.
- void cascadingCut(node *ptr) - Method to recursively performing cascading cut if a node has lost a second child.
- void increaseKey(node *ptr,int c) - Method to increment the frequency of node pointed to by ptr by value c.
- node* removeMax() - Method to remove the max node from the Fibonacci heap and return a pointer to it.
- void pairwiseCombine() - Method that performs pairwise combine.
- void move(node *ptr1,node *ptr2) - Method that makes ptr2 a child of ptr1, used during pairwise combine operation.

Member Function Implementation: fibHeap.cpp

1. FibHeap()

   This is a constructor that initializes the Fibonacci heap. It sets the max pointer to NULL and the number of nodes in the heap to 0.

2. node* getNode(string hashtag, int freq)

   A method to create and return a new node for the Fibonacci heap. The method does the following:
   a. Initializes the hashtag and frequency members as per the arguments.
   b. Sets degree to 0, since this is a new node and does not have any children yet.
   c. Sets child pointer to NULL.
   d. Sets parent pointer to NULL.
   e. Sets left and right sibling pointers to itself, which will be set accordingly during insertion into max heap.
   f. Sets childCut value to false

3. void insert(node* newNode)

   This method will insert the node into the Fibonacci Heap. The node will be added to the right of the current max node of the heap. If the newly inserted node has a higher frequency the max pointer for the heap will be updated.

4. void cut(node *ptr)

   This method will remove the node pointed by ptr from its location in the heap and add it to the top-level list of the Fibonacci heap to the right of the current max node. Since ptr now becomes a root node, its parent pointer is set to NULL and childCut value is set to false.

5. void cascadingCut(node *ptr)

   This method recursively examines the childCut values of node pointed to by ptr and if it is true, implying that the node has lost a second child, ptr is moved to the top-level list. If childCut value is false, it means that ptr is losing its first child and we set the childCut value to true.

6. void increaseKey(node *ptr,int c)

   This method increments the frequency stored in node pointed to by ptr by value c. If ptr is not the root node and its new frequency is more than its parents' frequency, then we perform a cut operation followed by a cascading cut operation. Lastly we need to update the max pointer if the new frequency is more than the frequency of the node pointed to by max.

7. node* removeMax()

   This method returns the current max node after its removal from the heap. If the max node has children they are inserted into the top level list. This is then followed by a pairwise combine operation.

8. void pairwiseCombine()

   This method performs the pairwise combine operation. The length of the array used as a table to check if the same degree node is present while traversing the nodes is calculated as $\log_{\phi} n$, where ϕ is the golden ratio, and n is the number of nodes in the heap.

9. void move(node *ptr1,node *ptr2)

   This method makes the node pointed by ptr2 a child of the node pointed by ptr1. This method is used in the pairwise combine operation.

Main Function: hashtagcounter.cpp

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

   This is the main function that take the input filename from the command line, parse it and calls appropriate Fibonacci heap operations. Any output generated will be written to a file called "output_file.txt". It uses a map to store the nodes of the Fibonacci heap. The map is defined as "std::map<string,node*> hashMap". The parsing is done as follows:

   a. Read a line from the file.
   b. Check if line starts with "#", if not then move on, if yes then do the following
      i. Extract hashtag and its frequency
      ii. Check if hashtag exists in map, if yes, then call increase key for the node taken from the map with value frequency
      iii. If hashtag does not exist, get a new node and insert it into the Fibonacci heap
      iv. Continue from Step a.
   c. If line starts with a number
      i. Perform removeMax() operation as many times as the number and store the nodes in an array.
      ii. Once all remove max operations complete, reinsert the nodes into the heap.
      iii. Continue from Step a.
   d. If line starts with the word stop, there is no need to parse the file anymore. Continue with Step e.
   e. Close input and output files and return.

Input/Output File Format

Input File

Each line in the input file defines an operation. A line beginning with a "#" represents an insert/increase key operation based on whether the hashtag has been encountered before. A line beginning with a number asks to extract the max n nodes from the Fibonacci heap. A line beginning with the word "stop", indicates the end of the file. An example input file may contain:

```
#saturday 5
#sunday 3
#saturday 10
1
#saturday 12
2
stop
```

Output File

The output file is called "output_file.txt". Each line in the output file represents a result of an integer query from the input file and displays the top n hashtags as requested separated by commas. An example output file (for previously shown input file) is as follows:

```
saturday
saturday,Sunday
```

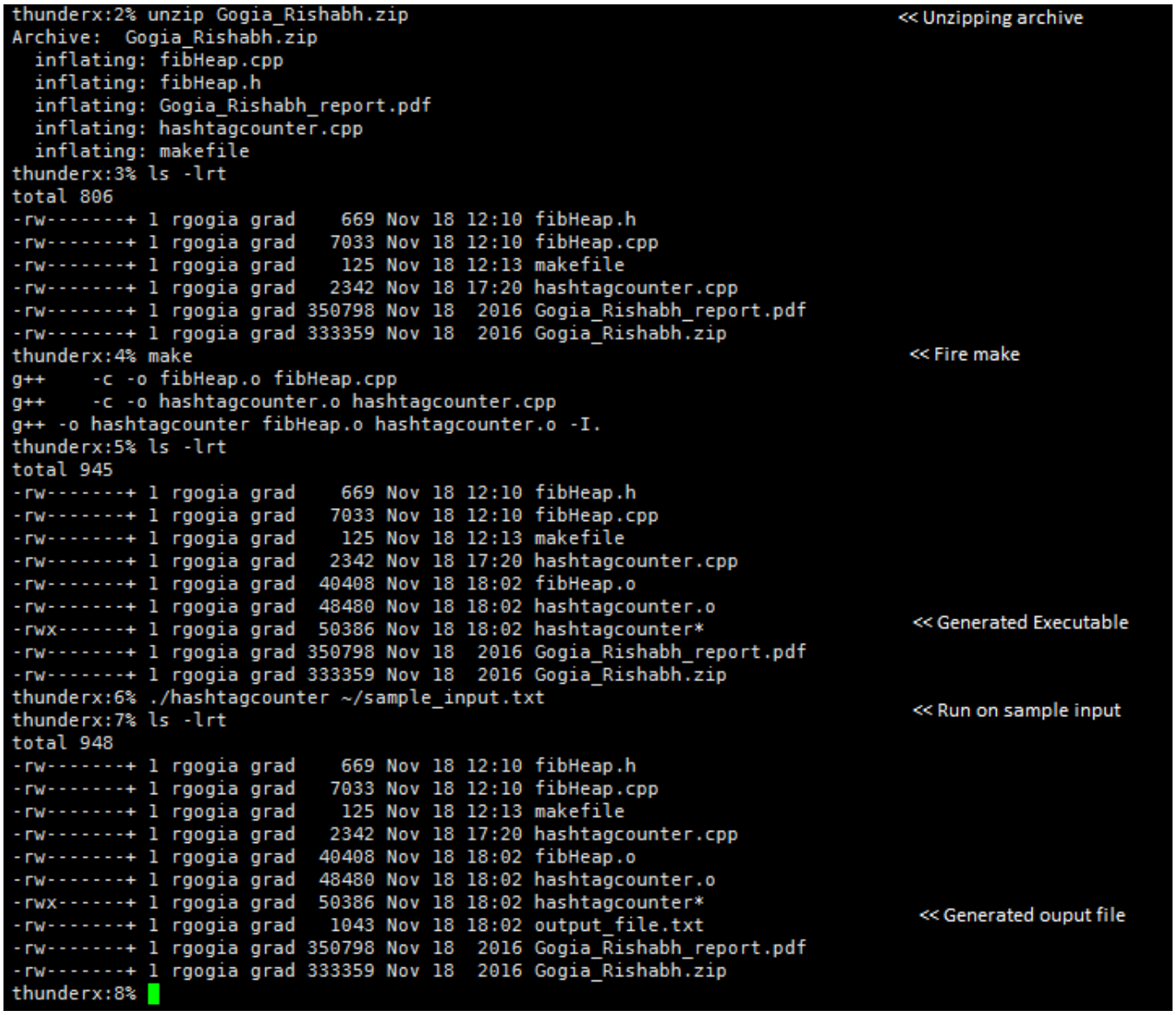
Running the program

Compiling and running the program

The submission consists of a zip file from which the previously mentioned files can be extracted. The following steps can be followed to run the program:

1. Extract the files (using unzip command on Ubuntu)
2. Fire the "make" command to create the binary
3. Run the binary as "./hashtagcounter <inputfile>"
4. The output will be put in the file "output_file.txt"

A screenshot from a sample run on thunder.cise.ufl.edu server (Ubuntu 14.04.5 LTS, gcc version 4.8.4) is shown below:



Conclusion

The implementation of this system required good bounds for both operations viz. incrementing the frequency of a hashtag and extracting the max value. A Fibonacci heap provides a good solution to this by maintaining a max priority queue by having the complexities of increaseKey and removeMax operation in the amortized sense as O(1) and O(log n) respectively. A map is maintained separately for keeping track of nodes in the Fibonacci heap and helps implement the increaseKey operation. Overall the system should perform reasonably well for large number of hashtags.