PROJECT REPORT

COP 5536 – Spring 2020

Implementation of a Hashtag counter using Max Fibonacci Heap

Sagar Jayanth Shetty
UFID: 4351-7929
sjayanthshetty@ufl.edu

Table of Contents

Introduction

This project implements a system to find the n most popular hashtags that appear on social media such as Facebook or Twitter. For the scope of this project we assume that the hashtags will be given from an input file. The basic idea is to use a Max Fibonacci Heap to keep track of the frequencies of the hashtags for this implementation. The hashtags itself are stored in a hash table which has the hashtag as its key and the pointer to the value as its value. This project uses unordered_map of the STL library for the hashtable.

The Max Fibonacci Heap is a binomial heap with mainly used for implementing priority queues. It is a collection of trees with max-heap property. The main advantage of Fibonacci heaps is that its amortized time complexity beats binary and binomial heaps. The amortized time complexity for all functions except delete-min is O(1) and the amortized complexity for delete-min is O(log N). The Max Fibonacci Heap is recommended for this implementation because the increase key function will be called frequently, and this data structure can execute it in O(1).

Execution

This project has been implemented in C++. It is compatible with g++ compilers and has been verified on thunder.cise.ufl.edu server. A makefile document has been provided which creates an executable file in the same folder with the name "hashtagcounter". This executable takes either input file name or input file name followed by output file name as arguments. If an output file is not specified it writes the result of queries to the console.

Following are the steps to execute the make file:

- 'make' / 'make all'
  Compiles the necessary code and creates executable.
- $hashtagcounter <input_file_name> [output_file_name]
  Returns output either in output file or in the console
- 'make' clean
  Deletes all executables in the directory.

If the makefile is not compatible in your environment, use the following steps to create the executable:

- g++ main.cpp -o hashtagcounter

The input file can have hashtags appear one per line and start with the '#' sign. The hashtag is followed by an integer which specifies that the hashtag needs to be incremented by that number. Queries will appear as a number(n) without the '#' sign. This returns the top 'n' hashtags in the heap. An input line with the word "stop" causes the program to terminate.

Following are my assumptions :

- No uppercase letters in the input stream.
- No spaces in the hashtags. (only one word per one hashtag)
- One query has only one integer.
- Query integer, n<=20. i.e. you need to find at most 20 popular hashtags.
- Input file may consist of more than 1 million hashtags.
- Ties may be arbitrarily broken.

Structure of the program

The program contains two header files and one main file.

- **Main.cpp**
  This file contains the functions to extract the input, parse the strings, perform respective Max Fibonacci Heap functions on it and store / return the output. It takes MaxFibHeap.h as header file.

- **MaxFibHeap.h**
  This file contains the structure of the Max Fibonacci Heap. It also contains all the Max Fibonacci Heap functions as well as intermediate functions required for the main functions. It takes Node.h as header file.

- **Node.h**
  This file contains the structure of a Node and functions to fetch and update its attributes.

MaxFibHeap.h Structure

This file contains the MaxFibHeap class. It has the following attributes:

- **Node* headNodePtr**
  This is a pointer to the head node of the top level doubly linked list.
- **Node* maxNodePtr**
  This is a pointer to the node with the maximum frequency.
- **int nodeIdCounter**
  This is an increasing counter used to set nodeId while initializing a new node.

The following functions implement the operations that a Max Fibonacci Heap should have for it to be used as a priority queue.

- **MaxFibHeap.insertNewNode(int frequency)**
  This function is called when we encounter the first instance of a hashtag. It creates a new node, initializes it with the entered frequency and inserts it into the top level doubly linked list using the "MaxFibHeap.insertIntoTopLevel" function. It returns the newly created node.

- **MaxFibHeap. increaseFrequency(Node* node, int frequency)**
  This function is the 'increaseKey' function of the Max Fibonacci Heap. It updates the given node's frequency removes it from its tree and inserts it into the top level doubly linked list if the change in frequency affects the max heap property of the tree. It uses "MaxFibHeap.removeNode" to remove the node from its tree and perform cascading cut. Then it uses "MaxFibHeap.insertIntoTopLevel" to reinsert it.

- **MaxFibHeap. deleteNode(Node* node)**
  This function is used to arbitrarily delete the given node from the Max Fibonacci Heap. It uses "MaxFibHeap.removeNode" to remove the node from its tree and perform cascading cut. Then it uses "MaxFibHeap.insertChildrenToTopLevel" to insert its subtrees to the Max Fibonacci Heap.

- **MaxFibHeap. removeMax()**
  This function is used to extract the maximum node from the Max Fibonacci Heap. It then removes the max node from its doubly linked list using

"MaxFibHeap.removeFromDoublyLinkedList" function and changes head pointer to either its sibling or its child. It then uses "MaxFibHeap.insertChildrenToTopLevel" to insert its subtrees to the Max Fibonacci Heap. It then calls "MaxFibHeap.pairwiseCombine()" to complete the sequence.

- **MaxFibHeap. pairwiseCombine()**
  This function fulfils the pairwise combine property of the Max Fibonacci Heap which melds trees with same degrees in order to maintain the structure of the data structure. It creates a table which keeps the degree as the key and a pointer to the node as the value. It melds trees which have the same degrees using "MaxFibHeap.combine" method. It also updates the max node after the remove Max function and updates the childCut flag of every node that gets reinserted in order to enable cascading cut.

Following are the intermediate functions used to facilitate the above operations:

- **MaxFibHeap.insertIntoTopLevel(Node* node)**
  This function updates the parent pointer of a given node and inserts it into the top level double linked list.

- **MaxFibHeap.removeNode(Node* node)**
  This function removes the node from its tree along with its subtrees. It uses "MaxFibHeap.removeFromDoublyLinkedList" to perform the same. Based on the childCut flag of its parent it then proceeds to either update its childCut flag or perform cascading cut on its parent.

- **MaxFibHeap.insertChildrenToTopLevel (Node* node)**
  This function inserts all subtrees of the given node to the top level doubly linked list. It uses "MaxFibHeap.removeFromDoublyLinkedList" and "MaxFibHeap.insertIntoTopLevel" to perform the same.

- **MaxFibHeap.removeFromDoublyLinkedList (Node* node)**
  This function changes pointers of the given node and its sibling nodes to remove the given node from its doubly linked list.

- **MaxFibHeap.combine(unordered_map<int, Node*>* table, Node* node)**
  This function checks the table if a node with the same degree exists and then melds the tree recursively checking the table for its new degree.

- **MaxFibHeap. insertChild(Node* parent, Node* child)**
  This function inserts the child to the parent node and updates pointers and degree accordingly.

- **MaxFibHeap**. **printAll(string str)**
  This function prints all contents of the Max Fibonacci Heap with the given string at the top and total count of the nodes at the end. It uses "MaxFibHeap.printChildren" to print children of every node.

- **MaxFibHeap. printChildren(Node* node, int* totalCount)**
  This function prints children of the given node while keeping track of the total count of nodes.

Node.h Structure

This file contains the Node class. It has the following attributes:

- **Node* parent**
  This holds a pointer to its parent.
- **Node* child**
  This holds a pointer to the child.
- **Node* left**
  This holds a pointer to the left sibling.
- **Node* right**
  This holds a pointer to the right sibling.
- **int degree**
  This number indicates the number of subtrees for this node.
- **bool childCut**
  This field is used by cascading cut to ensure that a node cannot lose a child more than once. It is true when the node has lost a child. It is false when the node becomes a child of another node.
- **int frequency**
  This node holds the key, which in this case is the frequency.
- **int nodeId**
  This field is used for printing operations that are not used in this project.

Main.cpp structure

This file contains the code to receive the input, format it and process it accordingly. Following are its functions:

- **Main. main(int argc, char** argv)**
  Creates a new hashtable and initializes a Max Fibonacci Heap. Takes the file names as input and performs "processString" operation for every line. Appends output string and outputs either as a file or in the console.
- **Main.processString(string str, unordered_map<string, Node*>* table, MaxFibHeap* maxFibHeap)**
  This function parses the given line/string using the "Main.extractFreqandTrim" and calls the respective operation on the given Max Fibonacci heap. For a new hashtag it calls the "MaxFibHeap.insertNewNode" function and inserts the returned node into the table along with the hashtag. For an existing node it calls the "MaxFibHeap.increaseFrequency" to increment its frequency. For a query(n) it calls the "MaxFibHeap. removeMax()" n times and stores these nodes in an array. It then finds their respective hashtags, reinserts the nodes using "MaxFibHeap.insertIntoTopLevel" and returns the result.

- **Main.extractFreqandTrim(string str)**
  This function parses the given string to extract the frequency from a string of the type "#hashtag 45". It also trims the string.

- **Main. string trim(string str)**
  This function removes any leading or trailing whitespaces in the string.