Hashtag Counter Report

A hashtag counter made using c++

Student:

Prerit Pathak (UFID : 8805 7930)

preritpathak@ufl.edu

Teacher:

Dr. Sartaj Sahni

Course:

COP 5536 – Advanced Data Structures

University of Florida

Objective & Implementation

Objective

We are required to implement 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, hashtags will be given from an input file. Basic idea for the implementation is to use a max priority structure to find out the most popular hashtags.

Implementation

1. Max Fibonacci heap: It is used to keep track of the frequencies of hashtags.

2. Hash table: The key for the hash table is the hashtag, and the value is the pointer to the corresponding node in the Fibonacci heap.

Program Structure

Files

1. hashtagcounter.h
2. hashtagCounter.cpp
3. fibheap.cpp
4. Makefile

hashtagcounter.h

1. struct NodeStructure: This structure stores all the attributes of a node in the fibonacci heap. The attributes are:
    a. degree: Integer storing the degree of the current node
    b. hHashtag: This stores the hashtag
    c. left: This is a pointer to the left sibling of current node
    d. right: This is a pointer to the right sibling of current node
    e. parent: This is a pointer to the parent of current node
    f. child: This is a pointer to the child of current node (if it exists)
    g. childCut: This is a boolean variable which tells us weather any node has lost a child or not
    h. element: This is the integer value stored in the node (frequency)

2. class hashtagcounter: This is a class declared in hashtagcounter.h and used in hashtagCounter.cpp & fibheap.cpp.
    a. Private data member:
        i. hMax: this is a pointer to the maximum element in the heap
    b. Public Methods:
        i. hashtagcounter(): Constructor to initialise the hashtagcounter class
        ii. NodeStructure* InitializeHeap(int element, string hHashTag): Constructor to initialize the heap
        iii. void InsertNode(NodeStructure *newNode): Method to insert a new node in the heap
        iv. void IncreaseKey(NodeStructure *node, int newVal): Method to increase the key of an existing node
        v. NodeStructure *RemoveMax(): Removes the max of the Fibonacci heap
        vi. void addChildrenToRoot(NodeStructure *removedChild): Add the children of max node (if any) to the root node
        vii. void Meld(NodeStructure *pairNode1): This function recursively melds two nodes with the same degree until no such nodes exist
        viii. NodeStructure *CombineThePairs(NodeStructure *pairNode1, NodeStructure *pairNode2): This node combines two nodes in such a way that one node becomes the parent of the other
        ix. NodeStructure *InsertIntoRootList(NodeStructure *oldNode, NodeStructure *newNode): This method merges the new node with the list of nodes already present in the heap
        x. void NodeCut(NodeStructure *childNode, NodeStructure *parentNode): If the key of the child node is greater than its parent, this node is cut from its parent
        xi. void NodeCascade(NodeStructure *parentNode): This inserts the separated child (after the childcut operation) into the root list

hashtagCounter.cpp

I. hashtagcounter(): Constructor to set the max pointer to NULL.
II. InitializeHeap(int elem, string hHashTag): This function sets the initial left and right sibling of the new node to itself and assignes the hashtag. It returs the node.
III. Insert(): This method will insert a new node into the heap. This node will be added to the right of the current max node. If the new node has a key value that's greater than the current max, the max pointer will be updated to the point to the newly created node.
IV. IncreaseKey(): Increase the value of a node in the heap to the given amount "new value". If the new value is greater than its parent, we perform a cut & childCut operations.
V. RemoveMax(): Removes the max node from the heap. If there's only one node at the root, we set the hMax(max pointer) to NULL & add its children nodes to the root list. If there are more than one node in the root, we join the root node siblings and then perfrom the same operations as above.
VI. Meld(hMax): This function will combine nodes with same degree until no two nodes in the root have same degree.
VII. addChildrenToRoot(NodeStructure *removedChild): This method adds the children of a removed node to root node list of the heap.
VIII. Meld(NodeStructure *pairNode1): This method recursively combines nodes until no two nodes have same degree
IX. CombineThePairs(NodeStructure *pairNode1, NodeStructure *pairNode2): This method combines two nodes such that, one of the nodes becomes child of another node
X. InsertIntoRootList(NodeStructure *oldNode, NodeStructure *newNode): Merge new node with the List of nodes already present in the heap.
    a. If both nodes are NULL -> return NULL
    b. If the new node is NULL -> return oldNode
    c. If the old node is NULL -> return newNode
    d. If none of the two is NULL -> Insert both nodes into the root list and return a pointer to the larger node

XI. NodeCut(NodeStructure *childNode, NodeStructure *parentNode):
    a. Set the parent of the removed node to NULL
    b. If the child has siblings, set the right sibling as the new child of parent
    c. Else: set the child to NULL

XII. NodeCascade(NodeStructure *parentNode): This method inserts the separated child into the root node list. It stores grandparent of the current node. If the parent has not lost a child, we set its childCut value to True. Otherwise, we perform cut and casadeCut.

fibheap.cpp

I. int main(int argc, char *argv[]): This is the main function which takes the arguments argc (counter) and char array argv which stores the arguments.
    a. If no inputfile is provided -> prints “Specify the input file containing the hashtags”.
    b. Parse the input file. If it contains a “#”, store the hashtag value in the hashtable as the key and it’s value will be a node pointer to the Fibonacci heap node which contains it’s corresponding frequency.
    c. If it contains a valid query, perform a remove max that many times and then re-insert the delete nodes into the max Fibonacci heap. Otherwise, print an error message for an invalid query.
    d. If output file is provided -> we print the output to the output file, otherwise we print the output to the console.

Program Structure