COP 5536 Spring 2020 - Programming Project
HashTagCounter

Name: **Balaji Balasubramani** | UFID: **9876 3981** | Email id: **bbalasubramani@ufl.edu**

Problem Statement:

We are required to implement a system to find the n most frequent hashtags that appear on Facebook or Twitter like social media.

Implementation Idea:

The idea is to use a max priority structure like Fibonacci Heap to determine the most frequent hashtags. If there are millions of hashtags, the number of calls to the increase key functionality will be very high. Since, Fibonacci heap has an amortized complexity of O(1) for the increase key operation, it is best to implement it.

Data structures used for the implementation:

1. **Max Fibonacci Heap**: Hashtag objects are stored ordered by the number of times it appeared in the input file. If there are multiple hashtags with the same number of times, we have chosen to keep the first appeared hashtag first.
2. **Hashmap**: A hashmap table to keep track of the hashtags that already appeared and to increase its value if it appears again later in the input.

Functionalities/Operations Required:

1. **Insert**: When a new hashtag key is read from the input file, create a new node with those data and insert it into the fibonacci heap.
2. **Increase Key**: When a hashtag key that is already existing in the hashmap table is read from the input file, increase its value by the given value do the cascade cut operation if needed.
3. **Cascade Cut**: While increasing the hashtag key value, if an increased value is greater than its parent node value(if any), then remove the increased node from its parent and insert it into the heap at the root level.
4. **Pairwise Combine**: While writing the top N popular hashtags into the output file, remove the max pointer value from the heap. Insert all its child nodes into the heap at root level and do pairwise combine operation to merge subtrees of the same degree.
5. **Top N hashtags**: Remove max pointers from the fibonacci heap for N times one by one and write the removed hashtags into the output file in the descending order.

Execution of program:

1. The project is implemented in **c++** language. Compiler used: **g++**.
2. Extract **balasubramani_balaji.zip**.
3. Open command prompt and enter command **make** to execute the makefile and build the binary.
4. Execute the binary with the input file by running the command:
   ./hashtagcounter <<input file name>> <<output file name>>.
5. Output is written into the **output file name** in the same directory.

Project Structure:

- hashtag.cpp
- Makefile

File Description: Makefile

This is an executable file that is used to build the binary. The binary is later used to run the program by passing the input file name as a command line argument.

File Description: hashtag.cpp

The file contains two classes named **TagNode** and **HashTags** and the driver function **main().**

Class TagNode: defines the structure of the each node created when a new hashtag is read from the input and a constructor to it.

Data Members:

- Public int **degree** - stores the current degree of the node.
- Public int **data** - stores the frequency of the hashtag
- Public bool **childCut** - Indicates whether a child has already been cut from the node.
- Public string **tagName** - stores the hashtag key associated with the node.
- Public TagNode* **leftNode** - pointer to the left sibling of the node.
- Public TagNode* **rightNode** - pointer to the right sibling of the node.
- Public TagNode* **child** - pointer to the child node.
- Public TagNode* **parent** - pointer to the parent node.

Member Functions:

- **TagNode()** - Default constructor.
- **TagNode(int val, string key)** - Overloaded constructor to initialize the node with the given hashtag key and value.

```
//  Below class defines the node structure and initiates the node with the default values when it is created.
class TagNode {
    public:
        //Node structure
        int degree, data;                       //Degree of the node and the Data stored in the node
        bool childCut;                          //boolean to check if a child has been already cut from this node
        string tagName;                         //The tag name associated to the node
        TagNode* leftNode;                      //pointer to the left node
        TagNode* rightNode;                     //pointer to the right node
        TagNode* child;                         //pointer to the child node
        TagNode* parent;                        //pointer to the parent node

        TagNode() {
        }
        //constructor to initiate the default values when a node is created
        TagNode(int val, string key) {
            childCut = false;
            data = val;
            tagName = key;
            degree = 0;
            leftNode = NULL;
            rightNode = NULL;
            parent = NULL;
            child = NULL;
        }
};
```

Class HashTags: It is a class representing the Max Fibonacci heap. It defines the fibonacci heap functions and handles its implementation.

Data Members:

- TagNode* **maxPointer** - pointer to the maximum value in the heap.

Member Functions:

- **HashTags()**

  Constructor that Initializes the heap and assigns the maxPointer to NULL.

- **void createNode(int value, string key, map<string, TagNode*>& hashmap)**

  It initializes the new node with the value and key and calls the insertNode to insert it into the heap at the root level. It also inserts the <key, value> pair into the hashmap table for future reference.

- **void insertNode(TagNode** start, TagNode* node)**

  This method inserts the node into the doubly linked list at the respective level. The start pointer always points to the maximum value node at that level. If the node has higher frequency than the start pointer node, then the start pointer is updated to the node. Otherwise, it is inserted in the appropriate position after the start node.

- **void increaseKey(TagNode* node, int val)**

  This function increases the data of the already existing hashtag node by the value val. If the node is not the maxPointer and its parent has lesser data than the node data, then it calls the cascadeCut function to perform the cascade cut operation. Otherwise, it rearranges the node to proper position at its level and finally updates the maxPointer if the node data is greater than the maxPointer data.

- **void cascadeCut(TagNode* node)**

  Cascade cut operation is performed on the node. It first removes the node from the child doubly linked list of its parent and updates the degree. If the node is the only child to its parent, then the child pointer is marked as NULL. Then it re-inserts the node into the heap at the root level. This repeats until there is no parent or the childCut of parent is set to false.

- **void higherFrequencyTags(int n)**

  This method extracts the n most popular hashtags. It first extracts the maxPointer and inserts all its child trees into the heap at the root level. Then it calls the pairWiseCombine() to merge all the subtrees of the same degree. Finally, after writing all the n popular hashtags into the output file, it re-inserts those nodes into the heap at the root level to process the next set of instructions.

- **void pairWiseCombine()**

  This function uses a degreeTable to keep track of the visited subtrees in the heap and pair-wise combines all the subtrees of the same degree. If two subtrees of the same degree are found, it calls the makeChild() to insert the subtree with smaller data as a child of the other and updates the degree and pointer of both the nodes. Finally, it re-constructs the heap with all the subtrees in the degreeTable.

- **void makeChild(TagNode* parent, TagNode* child)**

  This is a helper function for the pairwise combine operation. It takes two node pointers parent, child and make the child node as a child to the parent node. It updates the degree and the child of the parent accordingly and also updates the parent of the child.

```
class HashTags {
    public:
        TagNode* maxPointer;                    //Pointer to the maximum value in the heap

        HashTags() {
            maxPointer = NULL;                  //when a new object is created, maxPointer is assigned to NULL because the heap is empty.
        }

        /*…
        */
        void insertNode(TagNode** start, TagNode* node) {…
        }

        /*…
        */
        void createNode(int value, string key, map<string, TagNode*>& hashmap) {…
        }

        /*…
        */
        void increaseKey(TagNode* node, int val) {…
        }

        /*…
        */
        void higherFrequencyTags(int n) {…
        }

        /*…
        */
        void pairWiseCombine() {…
        }

        /*…
        */
        void makeChild(TagNode** parent, TagNode** child) {        …
        }

        /*…
        */
        void cascadeCut(TagNode* node) {…
        }
};
```

int main(int argc, char* argv[]): This is the main driver program. It takes input file name from the command prompt and reads the input file line by line and also opens a file to write the output. It uses a map<string, TagNode*> hashmap to store the hashtag keys and its associated nodes in the heap. Based on the input data, it calls the different fibonacci heap operations as below.

Driver program workflow:

It reads the input file line by line until the end of the file.

If the input line starts with “#”, then it checks if the hashtag key in the input line is already present in the hashmap. If it's present, it calls the increaseKey method to increase its data. Otherwise, it calls the createNode method to create a new node with the key and the value.

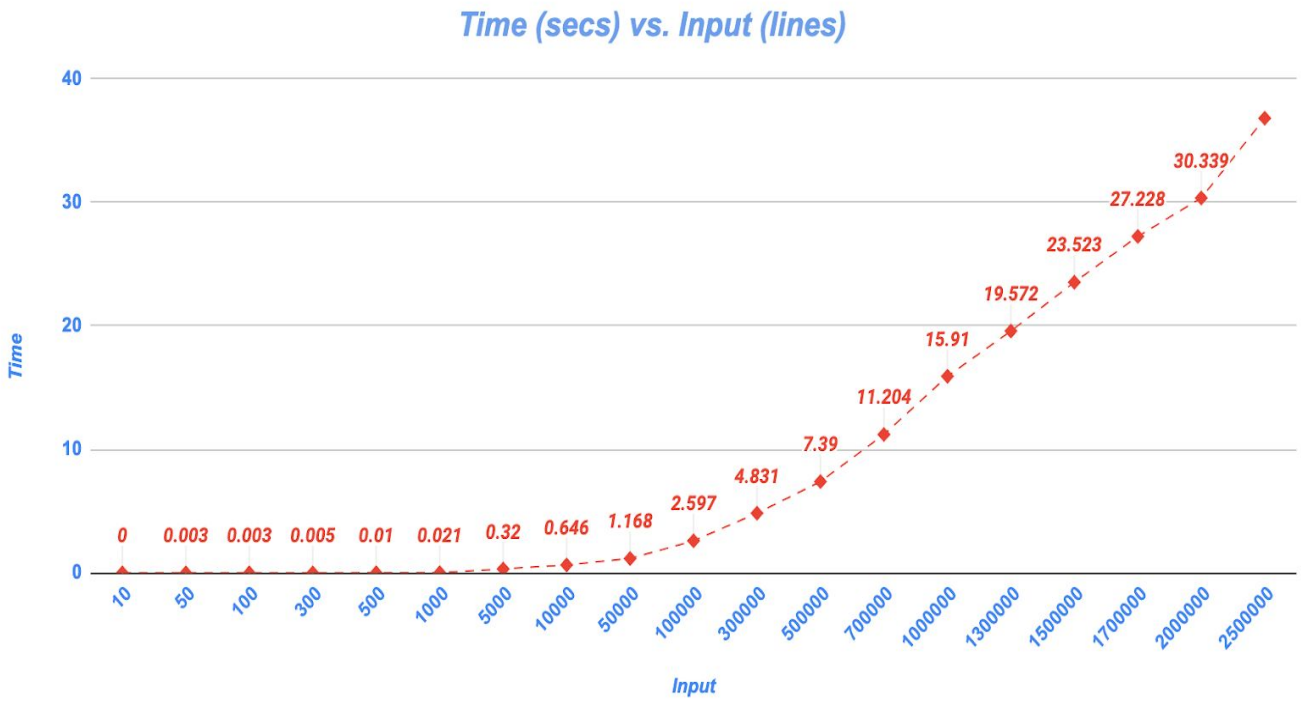
Else if the input line starts with an integer, then it calls the higherFrequencyTags method to extract and write the top n most popular hashtags into the output file.

Else if the input line matches the string “stop”, it stops processing the input file and closes all the opened files and returns.

Performance:

The run time of the program was analysed on various text files with different numbers of lines as input each time using the bash command **time.**

Command: time ./hashtagcounter <<input_file>> <<output_file>>



Conclusion:

The project to print most popular hashtags has been successfully implemented using Fibonacci heap and a hashmap. Increasing hashtag key value and extracting the maximum value for a large number of inputs hugely impacts the performance. But, this implementation yields better performance for large inputs as fibonacci heap provides a better solution by maintaining a max priority structure and having the complexities for those operations as O(1) and O(logn) respectively.