Implementation of Max-Fibonacci Heap to track frequency of tweets

Sharma, Siddharth (UFID: 53569805), Email: siddharthsharma@ufl.edu

I. INTRODUCTION

This programming assignment deals with the implementation of Max-Fibonacci Heap and use it to keep a track of the frequency of tweets entered using a input file. The implementation has been divided into two parts. One part deals with the design of the max-Fibonacci heap and its operations that are performed on the heap. While the other part describes the application part of the assignment i.e. to read from the file and calling the required functions from the heap setup. Modular approach of designing is used to keep the program readable and reusable.

II. BACKGROUND

Fibonacci heap is a type of data structure that is a collection of heap-ordered trees, used for priority queuing based application. One of the reason of using such a data structure is attributed to its better amortized cost than many other data structure schemes. Developed by Michael L. Fredman and Robert E. Tarjan in 1984, it provides constant $O(1)$ amortized cost for functions such as inserting new nodes, find max/min node and melding. While other functions such as remove min/max and increase/decrease key can be achieved with a amortized cost of $O(log_2 n)$. Use of Fibonacci heap for priority queues improves the asymptotic running time of important algorithm, such as Dijkstra's algorithm for computing the shortest part between two nodes in a graph, compared to the same algorithm using other slower priority queue data structures.

III. STRUCTURE

The structure of Max-Fibonacci heap is one with a collection of tress satisfying the properties of maximum-heap. This implies that the key of the child always less than or equal to that of the parent. This results in the maximum key always placed at the root of the tree. Although, the heap has relaxed constraints when compared with binomial heap but, need some order is maintained to achieve desired time of operation. To achieve this, the node has the following fields:

```
struct node
{
        //pointers//
        node* parent;
        node* child;
        node* right;
        node* left;
        //Degree of node//
        int degree;
        //Mark
        char marked;
        // key
        int key;
};
```

As a result of a relaxed structure, some operations can take a long time while others are done very quickly

IV. IMPLEMENTATION

Max-Fibonacci Heap Design:

The implementation of the has been divided in terms of the operation the max-fibonacci heap has to perform, namely:

1. EstLINK: To establish parent child relationship
2. insertNODE: This is used to insert a node into the circular link list of the max-fibonacci tree.
3. createNODE: Used to create a new node.
4. increaseKEY: This function is called to increase the key of a certain node.
5. removeMAX: This function remove the MAX node from the root circular linked list and moves the MAX pointer to the next max node.
6. cutNODE: This function simply break a given node from its parent.
7. childCUTnode: This will do a cascade cut on the nodes it the marked field of the node has been set.
8. meldHEAP: This function is used to meld the heap structure.

There are other helper functions that were used to initialize the heap and update the pointer to the maximum node. These functions are:

1. InitHEAP: This function initializes the heap
2. retHEADnode: This function is used to send the updated pointer to the application after every removeMAX operation.
3. displayTREE: This is a testing function prepared to check and display the root circular list of the heap structure.

Hash Tag Counter:

The hash tag counter is the other part of the application. This part of the application utilizes the functions described above in a sequence of operations as listed in the input file. In order to read the text file and extrapolate the values, 2 helper functions were created. They are as follows:

1. value_extractor: This function takes string value as the input and returns an integer of the same. This function is used to supply the key required for the heap.
2. key_extractor: This function takes string as an input and return a sting as well. This function is used to resize the supplied string, truncating the supplied hashtag string and removing the supplied key from it.

In order to make search for a node in the heap to be $O(1)$ time, the implementation of Fibonacci heap was so designed to use unordered Hash Map library of C++. The key goal was to incorporate the address of the newly created node, before insertion, into the hash table as the value. Consequently, the key of the hash table will be the hash tag itself. Therefore this will lead to the following structure of the hash map:

$$std :: unordered\_map < std :: string\ key, node *> hm;$$

Where, hm = hash map identifier, node * is the structure type of the node classification.

Now, when a new value is to be added into the heap, first the heap is checked for existing entries using the hash tag supplied in the input file. If an entry is found, the associated address of the node in the heap is obtained. Now we can perform any operation on the given node. If a new node is to be added, first the node is created. The associated hash tag for the key is used to make an entry into the hash map with the value as the address of the newly created node.

V. FUNCTIONAL DESCRIPTION

InitHEAP: To start the heap, the root node of the structure is initialized in this function. This function returns the pointer to the location which has been initialized as the root of the heap.

EstLINK: This function takes in 3 arguments both of the node * . The first argument takes in the pointer to the root of the heap. The other 2 nodes are pointers to nodes which will share parent-child relationship after the function exits. The 2nd argument is the pointer of the node which has to become the child of the node pointed by the 3rd argument.

insertNODE: This function takes in 2 node * as arguments and returns the pointer to the root node. The first argument of the function is root node itself while the other argument is the pointer to the node that is to be added to the circular link list. The function compares whether the key of the new-to-be-inserted node is greater than the key at the root. If the key of existing root is smaller than that of the new node, root pointer is changed to point the new node and the rest of the circular list is shifted right once. Else, the node will be added to the end of the list.

createNODE: This function takes in 1 argument i.e. the key to be added to the node. It creates a node that can be used by the insertNODE function to add the newly created node into the heap structure.

increaseKEY: 4 arguments are passed into the function. 1st argument is the pointer to the root node. 2nd argument is the pointer to the address of the node who's key has to be increased. 3rd argument is the old key value and the 4th argument is new key value. After comparing the supplied keys, cutNODE and childCUTnode are called which cut the node from its present location and place it into the root circular list.

removeMAX: This function takes in 1 node * argument and returns the node with the maximum key value. The value thus returned is stored in the application part of the project into a node * array which is used written into 2 files "output.txt" and "finalO.txt". The output.txt file be used as a temporary file that consists of all the removeMAX operations executed in sequence and is deleted once they are reinserted back to the heap. While the "finalO.txt" will contain all the removeMAX till "STOP" is seen in the input file. By keeping a temporary file ("output.txt"), I have eliminated collision that will occur it we use the key of the heap to search the hash map and also eliminating the need of additional field of value in the node structure.

cutNODE: This function takes in 3 arguments of the type node *. The 1st argument is the pointer to the MAX node. 2nd argument is the node that is cut from the heap. 3rd argument is the parent of the node that is to be cut.

childCUTnode: To implement Cascade Cut functionality for amortized cost, childCUTnode is called. This function takes in 2 arguments of the type node *. The 1st argument is the root of the heap and 2nd argument is node which was earlier the parent of the node cut from the heap structure. This function checks for the marked field of the node structure. If it is not set, childCUTnode sets it to TRUE. Else, it will recursively call cutNODE and childCUTnode till it reaches the max root node.

meldHEAP: This function takes in node * type of single argument which is the max root node of the heap structure. In order to meld the nodes available in the circular list at the root, an array of dimension equal to the nodes ("nNodeINlist") in the heap is created. This helps to keep the track of the degree of nodes. While loop design is used to traverse all the nodes in the root circular list and check for their respective degrees. If two nodes have the same degree, then EstLink is called upon. Depending on their key and/or degree (if keys are equal), child-parent relationship is established. Once the entire array of nodes has been traversed, the new nodes formed during the pass are connected to each other. Thus, forming a new root circular list.

VI. FUTURE SCOPE

Although most of the functions of the Max-Fibonacci Heap have been implemented successfully, but the design has been tailored for the application of tracking the tweets. Hence, as per the problem statement, some functions such as delete node and union of two heaps were omitted from being implemented. They can be implemented to make the implementation of the heap complete and make it independent of the application.