PROJECT REPORT

Advance Data Structures **(COP 5536)**

AMOGH RAO

amoghrao@ufl.edu

UF ID : 13118639

Project Description:

This project is a java implementation of hashtag counter using the Fibonacci Heap data structure.

List of Functions:

- Insert(FibHeapNode node)
- combine(FibHeapNode node1,FibHeapNode node2)
- pairwiseCombine()
- cut(FibHeapNode parent,FibHeapNode child)
- cascadeCut(FibHeapNode node)
- increaseKey(FibHeapNode node, int k)
- findMax(FibHeapNode node)
- removeMax()

The code has 3 classes:

1. **FibHeapNode**: Stores the structure of node in the heap.
   Structure:
   FibHeapNode left,right,parent,child => Stores pointers to left-right siblings, child and parent nodes
   String key => Stores the hashtag (i.e. the key)
   int val => Stores the frequency (i.e. the value)
   int degree => Stores the degree of the node
   boolean childcut => Stores if any of its child have been removed in the past (e.g. due to increaseKey)

```
class FibHeapNode{
    FibHeapNode left,right,parent,child;
    String key;
    int val;
    int degree=0;
    boolean childcut = false;
    public FibHeapNode(int val,String key)
    {
        this.right = this;
        this.left = this;
        this.val = val;
        this.key = key;
        this.child = null;
    }

}
```

2. **FibonacciHeap**: Stores the Fibonacci Heap with nodes of type FibHeapNode and performs all the manipulations on the heap with its functions described below.

   Function description:

   i. Insert:

      Prototype -> **public void insert (FibHeapNode node)**

      - Insertion always happens at the root list
      - The Max pointer of the Fibonacci heap points to the max node in the root list.
      - Initially the max pointer is set to null
      - Upon first insertion max pointer is set to the node
      - If max is not null (i.e. if a node exists in the root list), then the newly inserted node is placed next to the max node and the right and left sibling pointers are updated in order to complete the circular doubly linked list.
      - Then the value of the new node is compared to the value of the max node. If it is greater than the max node, then the max pointer is updated to the new node.

   ii. Cut:

      Prototype -> **public void cut (FibHeapNode parent, FibHeapNode child)**

      - It is triggered by the IncreaseKey function when new value is greater than its parent's value.
      - If the child node is the only child of the parent node, then set parent's child pointer as null else update child pointer of the parent and the pointers of the child list to complete the circular list.
      - Decrease the parent's degree by 1
      - Insert the detached child to the root list (update the parent of child to null and childcut to false)

   iii. Cascade Cut:

      Prototype -> **public void cascadeCut (FibHeapNode node)**

      - It is used to trace back from the parent of the cut node until root checking for childcut value being true and recursively call cut and cascadeCut when needed.
      - If the node's childcut is false then set it to true.

   iv. Increase Key:

      Prototype -> **public void increaseKey (FibHeapNode node, int k)**

      - Increase the value of the node by k
      - Check if parent != null and if value of the node is greater than value of parent.
      - If true then call cascadeCut() and cut() else make no changes
      - Check if value of the increased node is greater than the max value. If true then update the max pointer.

   v. Find Max:

      Prototype -> **public FibHeapNode findMax (FibHeapNode node)**

      - Iterates over the root list to find the maximum value node and updates the max pointer of the heap.

vi. Combine:

Prototype -> **public FibHeapNode combine (FibHeapNode node1, FibHeapNode node2)**

- It combines 2 nodes of the same degree passed by the pairwiseCombine() function
- It compares the values of nodes and makes the mode of smaller value the child of the larger node.
- It then updates the pointers of the smaller node removed from its original list by completing the circular list.
- The smaller node's parent pointer is set to larger node and its right and left sibling pointers are updated such that the node is added in the child circular list of the larger node.
- The larger nodes degree is incremented by 1.

vii. Pairwise Combine:

Prototype -> **public void pairwiseCombine ()**

- The purpose of this function is combine nodes of same degree in the root list such that no 2 nodes in the root list have the same degree.
- The root list is first broken down from circular list to a doubly linked list then iterated from left to right
- A hashtable is maintained that keeps storing the node pointer with the node's degree as key of the entry
- If there is an entry in the hashtable for the degree of the node of the current pointer then combine the node with the node entry in the hashtable using the combine() function and remove the entry for that degree from the table
- Then as the combined node's degree increased by 1 now recursively check for entry the new degree and then either insert or combine() depending on if the hashtable contains an entry or not
- After all the combines are performed, convert the linked list back to circular linked list by updating the end pointers of the list.

viii. Remove Max:

Prototype -> **public FibHeapNode removeMax()**

- Remove the max node by temporarily changing the max pointer to max.right (if it is not self)
- Update the necessary pointers to complete the circular list
- Check if the removed node has any children. If it does then insert all of its children to the root list.
- After all the insertion is completed findMax() function sets the max pointer to the maximum node value in the current root list.
- Then perform pairwiseCombine() on the root list.

3. **Hashtagcounter**: It stores the main function which reads the hashtags from a file and performs operations on the Fibonacci heap.

   Steps performed:

   - The main function takes in filename as an argument in the command line.
   - Reads the file line by line until it encounters a “stop”, when it stops reading the file and program terminates.
   - If not “stop”, it checks if the first character of line is ‘#’ to determine if it is a hashtag entry.
   - If it is ‘#’, it parses the line to fetch the hashtag (i.e. key) and the occurrence (i.e. val).

     e.g. #chloroform 8 which indicates hashtag=chloroform and occurrence=8
   - A hashtable is maintained that stores the hashtag as the key and its node’s pointer in the heap as a value.

     Hashtable<String, FibHeapNode> database = new Hashtable<String, FibHeapNode>();
   - The code first checks if there is already an entry for ‘chloroform’ in the hashtable. If there isn’t then it means it is a new entry, so it creates a new node and inserts it into the Fibonacci heap and updates an entry for the same in the hashtable.
   - If there is an existing entry then it calls increaseKey(node,k) function of the heap by passing the node pointer fetched from the hashtable and the value k (it increases by).
   - If the first character is not ‘#’ and if the line is not “stop”, then it has to be a number **n** which indicates that top **n** frequent hashtags need to be printed.
   - This is performed by calling the removeMax() function of the heap, **n** number of times and temporarily storing it in a Arraylist (for reinsertion)
   - After **n** removeMax() operations the hashtags are written to an output file (output_file.txt) are reinserted into the heap with new nodes. These new node pointers are updated in the hashtable for the respective hashtag.