Project: Fibonacci Heap
Course: Advanced Data Structure (COP 5536)
Fall 2016

Submitted by: Karan Goel
UFID :81160185
Email id: karangoel16@ufl.edu

Index

Introduction

A Fibonacci heap is a data structure, consisting of a collection of heap ordered tree, but the amortized cost is relatively better than other priority queue structure such as binomial.

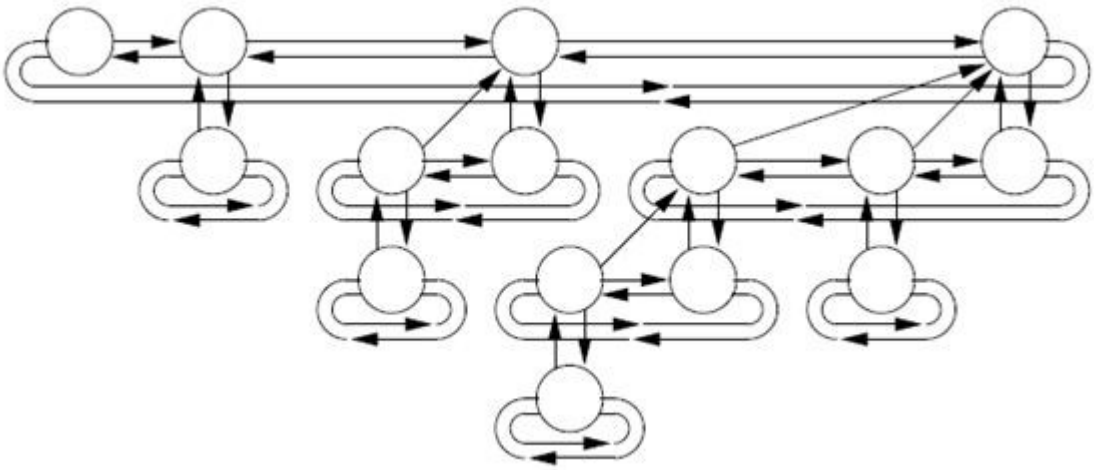

Figure 1 A detailed view of a Fibonacci Heap. Null pointers are omitted for clarity.
Image Reference: http://neil-inthemaking.blogspot.com/p/fibonacci-heap-and-its-applictions.html

Various Complexity of Fibonacci heap

	Actual Complexity	Amortized Complexity
Insert	O(1)	O(1)
Remove min (or max)	O(n)	O(log n)
Meld	O(1)	O(1)
Remove	O(n)	O(log n)
Decrease or (Increase Key)	O(n)	O(1)

Compiling Instruction

The Project has been compiled and tested under the following platforms:

Sr. No.	Environment	Compiler	Pass
1.	Windows	Mingw	Test passed
2.	Linux	gcc	Test passed

To compile the program in gcc compiler:

1. Open a new project

2. Copy the files into the project.

3. Give the file path in the argument of the project and then run the following code.

To compile the following project on thunder(linux environment) follow the following steps:

1. Open putty and enter thunder.cise.ufl.edu as the server name

2. Copy the files on the server using any filetransfer software like Filezilla or Winscp.

3. run using the following command g++ -std=c++11 ads_project.cpp -o hashtagcounter or use command Make all.

Structure of the node

The structure of the node in the Fibonacci heap has following components. The variables in the nodes are:

1. Degree

This variable is of type integer and it helps us to determine the number of nodes in the next level, if it also used in pairwise combine which is again one of the important operation of Fibonacci heap.

2. Child cut
This variable is of type Boolean and it helps us to determine if the child has been previously removed from the node or not, if it is false, that simply means that no child has been removed from the current node, and vice versa.

3. Key
Key stores the string that the node denotes hashtags and this is used to determine the uniqueness of the node and it is later used to output in the file as well.

4. Parent
Parent stores the address of the node that is parent of the current node

5. Link_child
This variable stores the address of the child of the current node, in case if we have no child of the current node then the link_child would be null.

6. Link_list_right and Link_list_left
This variable stores the address of the left and the right node at the same level, if there is only one node at the same level, then left and right pointer will be node itself.

Prototype

Various Functions written in our program are:
1. Add
2. Increase Key
3. RemoveMax
4. Pairwise combine
5. Meld
6. Check

Add

struct node * add (string name,int freq)

Return type: structure node

Parameter: string name, integer frequency

Description: This function inserts a node into the Fibonacci heap, it works initially when head is null, at that instant it creates a new head node, and rest of the time it creates a node meld it with head and checks if the new node has frequency(val) greater than the head, if yes, we move head pointer to this new position.

Increase Key: -

int increasekey(struct node *root,int freq)

Return type: integer

Parameters: struct node * root, integer frequency

Description: This function is used to increase key of the root node and if its value increase from its parent node, we will remove this node from that place and meld it with the head and check if the new element is greater than head, then we will update the head, also this function will perform child cut operation in it.

Remove Max: -

struct node * remove_max()

Return: structure node

Description: This function is used to return the node with the maximum frequency and also create a new Fibonacci heap as well, after removing the node.

Pairwise Vector: -

Void pairwise_vec(struct node *root)
Return : void
Parameter: structure node
Description: This function helps the remove max function , after the element is removed to create a new Fibonacci heap , with the help of pairwise comb function.

Meld: -

bool meld (struct node *root1, struct node *root2)

Return: Boolean

Parameter: structure node root1, structure node root2

Description: This function is written with the purpose to meld two separate Fibonacci heap, and it is needed in various operation like increase key operator after node and subtree is removed and again added in the structure.

Check

vector<struct node *> check (struct node *root)

Return: vector

Parameter: structure node

Description: This function is used to store root nodes of all the max trees, which is pairwise combined.

Pairwise Comb

struct node * pairwise_comb(struct node *root1,struct node *root2)

Return: structure node
Parameter: structure node *root1, structure node *root2

Description: This function is used to combine the two nodes one and two, making one child of other, if the nodes with bigger value has no child then we will simply connect the two , otherwise we will meld the node with smaller value, with the child of bigger node.

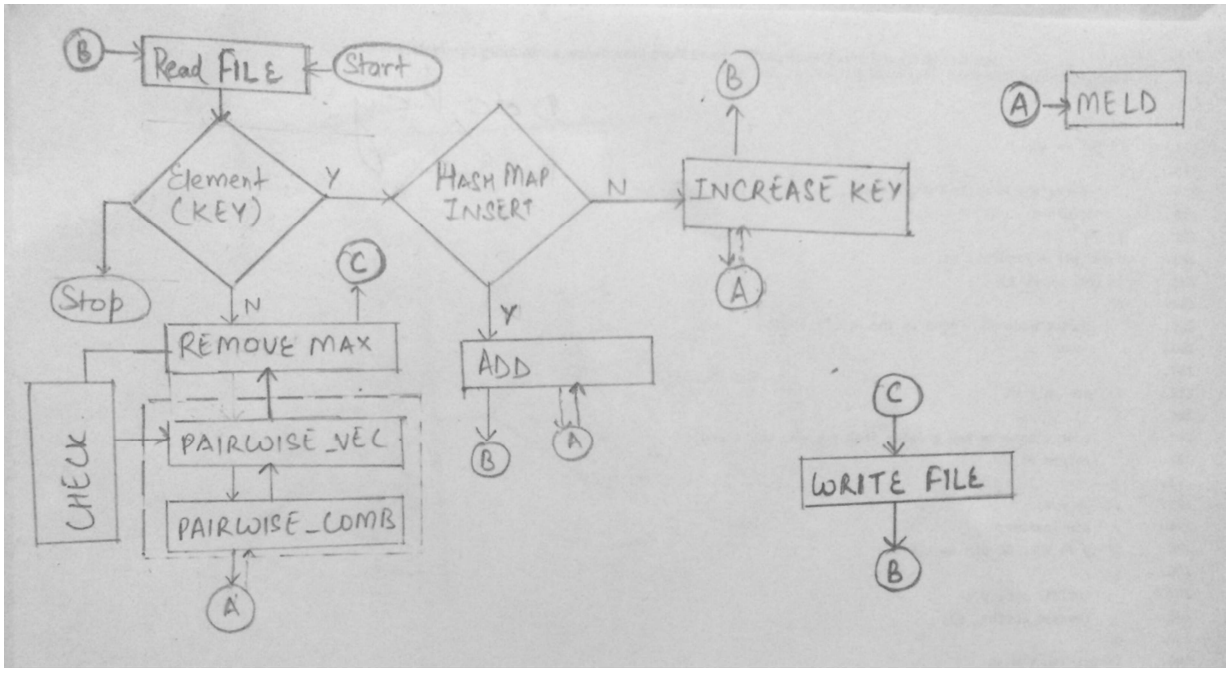

Figure 2: Program flow of Fibonacci heap