Project Report on Min Fibonacci Heap & Min Leftist Tree

Submitted By:
Garg, Vineet
5128-8191
vineetg07@ufl.edu

Table of Contents

Development

The code of Fibonacci Heap & Leftist Tree is developed in *C programming language* using *Microsoft Visual Studio 2010* on *Windows 7* platform. The code is also compiled & tested using *gcc compiler* on *thunder.cise.ufl.edu* server. (**Note – gcc compiler gives some warning due to use of log variable, which is a in-built function*).

How to Compile

Code can be compiled on Windows or *nix environment by using *makefile*. *makefile* can be found under root directory *ADS_Project*. MakeFile has following targets:

- *all* – This target compile & link *FibonacciHeap.c*, *LeftistTree.c* & *heap.c* files. This target depends upon *clean* target.
- *clean* – This target cleans all the object files & executable *heap* file.
- *run* – This target runs the program in random mode.

Structure of Program

- The program is divided mainly into three files:
  - ***FibonacciHeap.c*** - This file contains all Fibonacci Heap functions
  - ***LeftistTree.c*** – This file contains all Leftist Tree functions.
  - ***heap.c*** – This file contains all functions to handle various project requirements.

- Apart from above listed files the program contains three header files:
  - ***common.h*** – This header file contains standard libraries & declarations of functions used to handling various project requirements.
  - ***FibonacciHeap.h*** – This file contains standard libraries, structure definition of Fibonacci Heap node & functions declarations to handle various Min Fibonacci Heap operations.
  - ***LeftistTree.h*** – This file contains standard libraries, structure definition of Leftist Tree node & function declarations to handle various Min Leftist Tree operations.

- The program can run in **RANDOM Mode** or **USERINPUT Mode**. In Random mode program performs 5000 random operations on various input sizes. In User Input mode program take input from a file and after performing listed operations it outputs the resultant tree in *result_bino.txt* or *result_left.txt*.
  *(*Note – DecreaseKey & Remove operation for Fibonacci Heap can only be performed in User Input Mode)*

- The program also simulates **logging**. Logging can be used in **DEBUG** mode for debugging the program or logging can be used in **INFO** mode to only print the relevant information.
  *(*Note – The submitted program runs in INFO mode).*

Structure of Functions

FibonacciHeap.c functions

insert_MinFHeap

- **Prototype** - *struct f_node *insert_MinFHeap(struct f_node*, int)*
- This function performs the ***insert operation*** in Min Fibonacci Heap. It takes the heap & data as input. If the heap is empty it simply adds the element. If heap is not empty it melds the new element with the heap using *meld_MinFHeap()* function.

print_MinFHeap

- **Prototype** - *int print_MinFHeap(struct f_node*)*
- This function ***prints*** the Min Fibonacci Heap to result_bino.txt file. It takes the heap as input. It uses a queue *q_element* to maintain all the nodes on the same level.

meld_MinFHeap

- **Prototype** - *struct f_node *meld_MinFHeap(struct f_node*, struct f_node*)*
- This function performs ***Meld operation*** of Fibonacci Heap. It takes two heaps and meld them by simply inserting the second heap into circular list of first heap.

deleteMin_MinFHeap

- **Prototype** - *struct f_node *deleteMin_MinFHeap(struct f_node*)*
- This function performs ***Delete Min operation*** of Fibonacci Heap. It takes the heap as input. It deletes the the min element and then pair wise combine the leftover heap and children of min element using *pairwiseCombine_MinFHeap()* function.

pairwiseCombine_MinFHeap

- **Prototype** - *struct f_node *pairwiseCombine_MinFHeap(struct f_node*, struct f_node*)*
- This function performs ***pairwise combine operation*** of Fibonacci Heap. It takes two heap and pairwise combine them. It uses *degreeTable* structure (which is made of an array) to store the trees of various degrees.

cascading

- **Prototype** - *struct f_node *cascading(struct f_node*, struct f_node*)*
- This function performs ***cascading operation*** of Fibonacci Heap. It takes the heap and pointer to subtree from where cascading is to be begun as input. It keeps removing subtrees and melding with heap until it reaches root or a node whose *childCut* value is 0

removeMinFHeap

- **Prototype** - *struct f_node *removeMinFHeap(struct f_node*, int index)*
- This function ***removes any arbitrary node*** from Fibonacci Heap. It takes heap & the index of node.It first checks if the node exists. If it exists it deletes the node and cascade the heap using *cascading()* function.

decreaseKey

- **Prototype** - *struct f_node *decreaseKey(struct f_node*, int index, int key)*
- This function performs ***Decrease Key operation*** of Fibonacci Heap. It takes heap, index of node and the amount by which key of node is to be decreased. It checks if node exists or not. If node exists it decreases its key, pull its subtree and meld it with heap using *meld_MinFHeap()*. It then starts cascading using *cascading()*.

intializeFMinHeap

- **Prototype** - *struct f_node *intializeFMinHeap(struct f_node*, int[], int)*
- This function ***initializes*** the Fibonacci Heap. It takes heap, elements and number of elements as input. It then inserts the elements in Fibonacci Heap.

LeftistTree.c functions

insertMinLeftistTree

- **Prototype** - *struct node *insertMinLeftistTree(struct node*, int)*
- This function performs the **insert opteration** in Min Leftist Tree. If tree is empty is simply insert the node. If heap is not empty it melds the node with the tree using *meldMinLeftistTree();*

meldMinLeftistTree

- **Prototype** - *struct node *meldMinLeftistTree(struct node*, struct node*)*
- This function performs the **meld operation** of Min Leftist Tree. It compares the root of both trees. It inserts the tree with bigger root as right child of tree whose root is smaller. If right child is not empty it recursively melds the tree and right child.

printNodes

- **Prototype** - *int printNodes(struct node*)*
- This function **prints** the Leftist Tree to *result_left.txt* file under project root directory *ADS_Project*. It uses a queue *leftistQueue* to maintain all the nodes to be printed on same level.

ensureShortest

- **Prototype** - *struct node *ensureShortest(struct node*)*
- This function **ensure the shortest value property** of Leftist Tree i.e. it ensures the *s(right child) < s(left child)*. It finds the s-value of right and left child of the node and swaps the children if s-value is not correct.

deleteMinLeftistTree

- **Prototype** - *struct node *deleteMinLeftistTree(struct node*)*
- This function performs **Delete Min operation** of Min Leftist Tree. It deletes the min element from the Leftist Tree and then meld the right & left children using *meldMinLeftistTree()*;

calSValue

- **Prototype –** *void calSValue(struct node*);*
- This function **calculates the s-value** of a node using formula s-*value(node) = s-value(right child) + 1*;

initializeLeftistTree

- **Prototype** - *struct node *initializeLeftistTree(struct node*, int[], int)*
- This function **initializes** Min Leftist Tree by inserting elements.

heap.c functions

main

- **Prototype –** *int main(int, char *[])*
- This function **parses the command line arguments** and finds out the input mode if it is User Input Mode or Random Mode. It then calls appropriate function to handle the mode

userInputMode

- **Prototype -** *void userInputMode(char *, char *)*
- This function **handles the User Input Mode**. It parses the input file and performs the relevant operation on Min Fibonacci Heap & Min Leftist Tree

randomMode

- **Prototype -** *void randomMode()*
- This function **handles the Random Mode**. It generate permutation of various input size using randomPermutation() and then perform 5000 random operation in generated inputs

randomPermutation

- **Prototype -** *void randomPermutation(int[], int)*
- This function generates a **random permutation for input**.

Summary of Result Comparison

Fibonacci Heap

In Min Fibonacci Heap the *amortized cost of insert operation is O(1)* & *amortized cost of Delete Min operation is O(lg n).* So the total time of 5000 random operations depends upon how many insertions were performed and how many delete min were performed. If we assume that equal number of insertions and delete min were performed or number of delete mins is more than insertions, then the time taken for 5000 operations will increase as we increase the input size. If we run the program and measure the time for Fibonacci Heap we see very minor difference between time-taken for various input size. This is because we measured the time in microseconds but the difference occurs on nanosecond level. Therefore the program shows almost the same time for various input size. Sometimes the execution time decreases for bigger input size. This might be because no of inserts are greater then number of delete min operations.

Leftist Tree

In Min Leftist Tree the *amortized cost of insert as well as Delete Min operation is O (lg n).* So as we increase the input size the time taken should increase. If we run the program and measure the time it shows execution time 0ms for each input size. This is because we are measuring time in microseconds but the difference occurs on nanosecond level. So we zero ms execution time for each input size.

Table of Execution Time

Fibonacci Heap Execution Time

Input Size	Execution Time(*in micro seconds*)	Comments
100	45	This is the execution time per operation on input size 100
500	45	This is almost same because difference might have occurred on nano second level or number of delete min operations is almost same as in previous case.
1000	50	This is the expected result.
2000	47	The execution time decreases because number of delete min operations is lesser than in previous case.
3000	48	The execution time increases but not as expected because again number of delete operations is lesser.
4000	47	The execution time decreases because number of delete min operations is lesser than in previous case.
5000	50	The execution time increases but not as expected because number of delete min operations might be almost same in previous case.

Leftist Tree Execution Time

Input Size	Execution Time(*in micro seconds*)	Comments
100	0	The execution time is zero because the time take to process various operations in Leftist Tree is very less and is almost zero in micro seconds.
500	0	
1000	0	
2000	0	
3000	0	
4000	0	
5000	0	