ADVANCED DATA STRUCTURES

COP 5536

PROGRAMMING PROJECT
FALL 2013

Submitted By:

Ishadutta Yadav

UFID - 5493 1916

Email: iyadav@cise.ufl.edu

INDEX

1) Compilation Steps

This project have been implemented in C++ and compiled in Linux (Ubuntu) operating system with g++ compiler

Following steps need to be followed to compile and execute the program.

a) Copy all the files mentioned in zip folder including the makefile in a separate directory
b) Run "**make clean**" to remove all previous output/executable files
c) Run "**make** "
d) Now to run the program in different mode following command need to be issued
   i) Random Mode - **./mst -r n d**
      Where n= Number of Nodes (0 < n < =5000)
      d= density (0 < d <= 100)
   ii) File Mode -
      - Simple Scheme - **./mst –s <filename>**
      - Fibonacci heap Scheme - **./mst –f <filename>**

Note-

1) STL Bitset is used to check whether there is an Edge between 2 vertices. Maximum array define for Bitset is 5,000 (Five thousand) So this implementation work up to 5,000 node (vertices). By increasing the Bitset size we can increase this limit.

2) If V is total no of vertices, Minimum no of edges needed to make it connected is V-1. So if a density value is input such that this condition violated then it is not possible to get a fully connected graph. Also Prim Algorithm work only for connected graph.

2) Introduction

This project implements Prim Minimum Cost Spanning Tree algorithm using a simple data structure say “Simple scheme” and with Fibonacci .Heap say “f-heap” scheme and measure the relative performance of two Implementation.

Both scheme use the “adjacency list” representation for graph. The Simple scheme uses an array to determine the next edge to include while f-heap scheme uses a Fibonacci heap to do this.

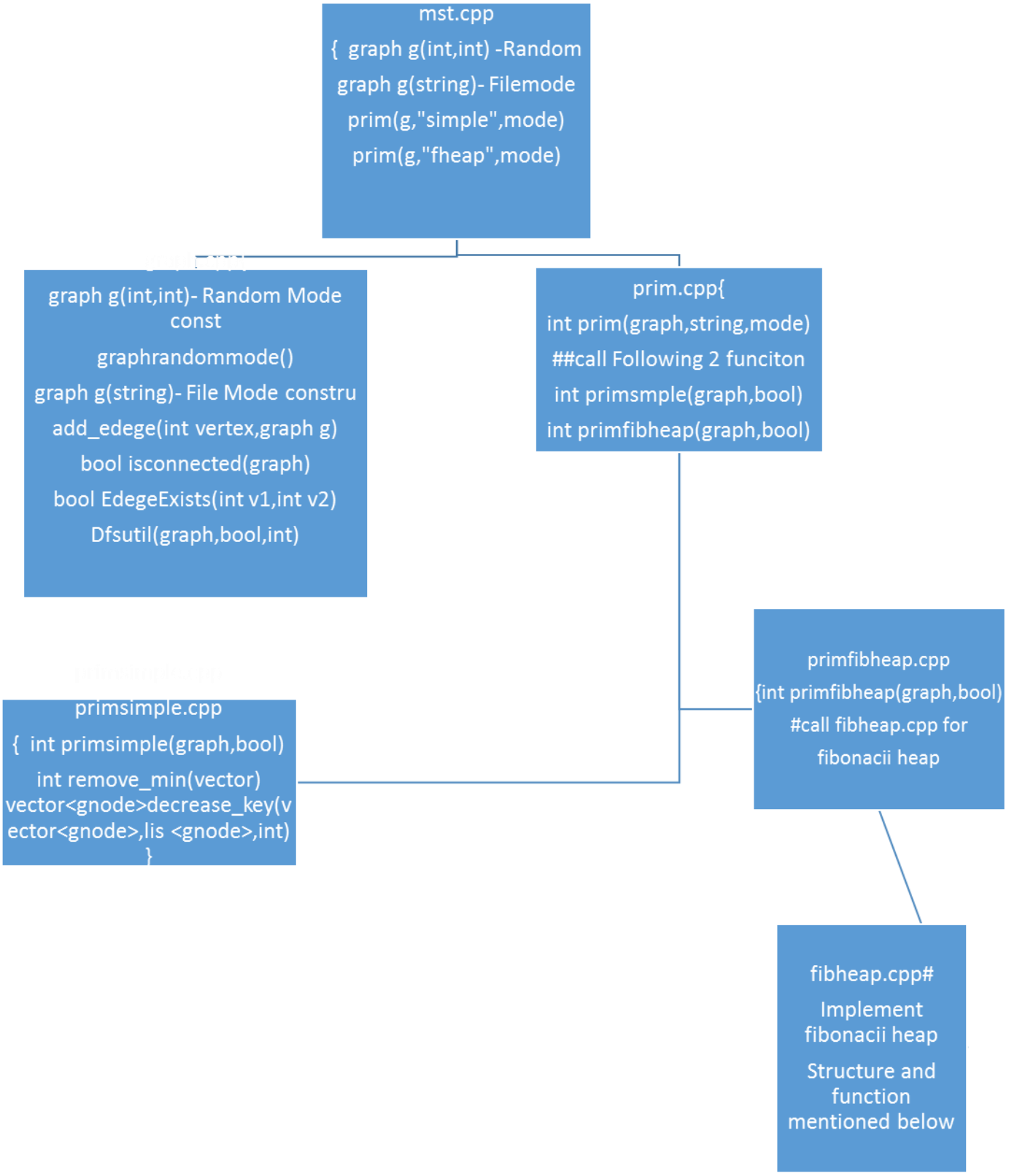
The simple scheme run in O (V^2) where V is number of Vertices. While f-heap scheme runs in O (E+VlogV) time where E is number of edges in the graph.

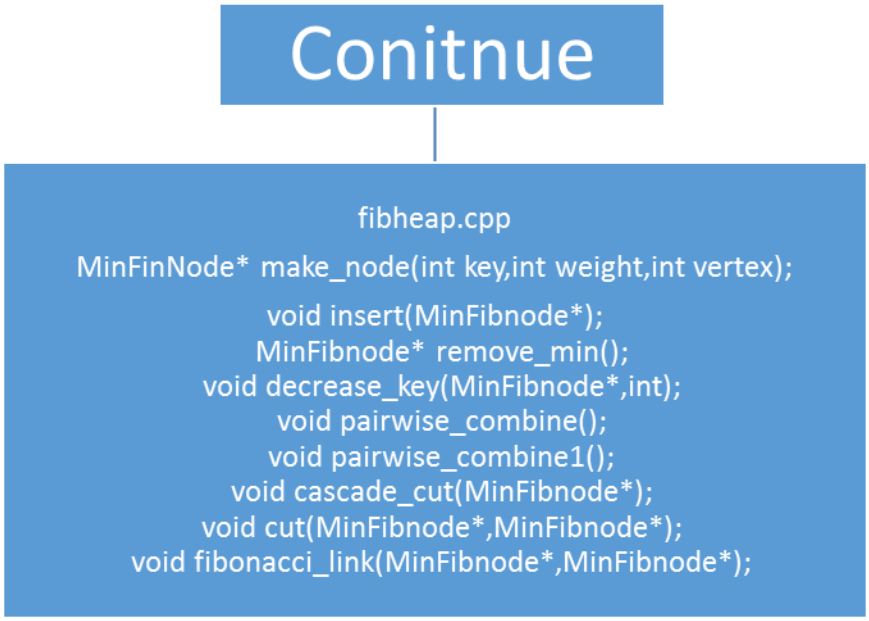
3) Class description, Function and Structure

mst.cpp has main function and it initates the execution of program. First it create a graph object, depending upon argument passed by user graph is created in random mode or user mode .

Then prim scheme is called to get the minimum spanning tree. For random mode Primsimple and prim fibonacci both schem is called but for user mode it call only that prim scheme of which option is provided . i.e it calls simple Scheme when –s is provided and it calls fibonacci heap scheme when –f is provided.

Following diagram shows flow and structure of the graph.


mst.cpp
{ graph g(int,int) -Random
graph g(string)- Filemode
prim(g,"simple",mode)
prim(g,"fheap",mode)
graph g(int,int)- Random Mode const
graphrandommode()
graph g(string)- File Mode constru
add_edege(int vertex,graph g)
bool isconnected(graph)
bool EdegeExists(int v1,int v2)
Dfsutil(graph,bool,int)
prim.cpp{
int prim(graph,string,mode)
##call Following 2 funciton
int primsmple(graph,bool)
int primfibheap(graph,bool)
primsimple.cpp
{ int primsimple(graph,bool)
int remove_min(vector)
vector<gnode>decrease_key(v
ector<gnode>,lis <gnode>,int)
}
primfibheap.cpp
{int primfibheap(graph,bool)
#call fibheap.cpp for
fibonacii heap
fibheap.cpp#
Implement
fibonacii heap
Structure and
function
mentioned below



Detailed Description of File and Function

a) mst.cpp

mst.cpp has main function and it initates the execution of program. First it create a graph object, depending upon argument passed by user graph is created in random mode or user mode .

main(int argc, char *argv)

if Number of argument is 4 (argc ==4 , mst –r n d) it calls graph constructor function of random mode graph g(int,int) else it calls graph constructor of file mode graph(file).

Then it calls appropiate prim function depending upon scheme. For random mode It calls both scheme and for file mode it calls relveant scheme depending upon output.

b) graph.cpp /graph.h

This is the cpp file in which graph structure and all graph function and there implementation are defined. The node structure of graph is as follows.

```
struct gnode
  {
     int vertexid;
     int weight;
     int key;
     int parent;
  };


 class graph
  {
     public:
     int V;   // V represent Number of vertices in the graph
     list<gnode> *adj;  // Adjecency List reprentaion of graph
     vector<gnode> vertex;  // Store all the vertex in a vector
     graph(int,int);
     graph(string);
     void graphrandommode(int);
     bool EdegeExists(int, int);
     void add_edge(int x,gnode node1);
     bool isconnected(graph);
     int  DFSutil(graph,bool visited[],int);
}
```

i) graph(int,int)

Above constructor is used to create the graph in random mode. In this function it initializes the adjacency list, also it creates an Array to store the node off all the vertex.

ii) graph (string)

This constructor is used to create the graph from file which is input by user. User need to give connected graph as an output .Also it is assumed that no two edges are repeated in user input file.

iii) void graphrandommode(int,int);

1st parameter to this function is number of vertices and 2nd parameter is density. It generates two random vertices in which edge need to be created and verify whether an edge already exists .To check whether a edge already exists. It maintain a Bitset and do corresponding bit set if an edge is inserted. After graph creation it also calls isconnected function to verify whether graph is connected or not .If graph is not connected this function calls again.

v) void add_edge(int x,gnode node1)

This function take a vertex and graph node as a parameter and add the node in adjacency list of the vertex, as we are creating a undirected graph so it creates a New node with vertex as x and add it to the adjacency list corresponding to graph Node.

vi) bool isconnected(graph)

This function check whether graph is connected or not connected .It takes a graph Node as a parameter and it call utility function DFSutil. If total Number of node visited by DFSutil is less than total number of vertex in this graph it return our Graph is not connected

vii) int DFSutil (graph, bool [],int)

This function is a Simple DFS implementation and it return number of connected node start from vertex 0. It take a graph node, Boolean array and source node vertex and return number of node visited till now.

3) Prim.cpp/prim.h

This is the driver file which calls prim algorithm with simple scheme and prim algorithm with Fibonacci heap scheme.

i) int prim(graph g,string option,bool flagmode)

Depending upon the option input by user it calls prim algorithms with simple scheme or Fibonacci heap scheme. It takes graph node, option i.e. simple scheme Or Fibonacci scheme, for –r run the both scheme. Flag]mode is true when it run In a file mode otherwise it is set to false.

4) Primsimple.cpp

i) int primsimple(graph g,bool flagmode)

This function take a graph and flagmode as a parameter and return cost of minimum spanning tree with prim algorithm. In case of user mode when flag mode is set to true in standard output it output edges of the MST. It implements a simple Scheme and maintain an array to store the vertices and it travers the whole array in O(n) to find out minimum element.

ii) int remove__min(vector<gnode>)

This function take a vector (dynamic array) as an input and output the vertex whose key value is minimum.

5)) Primfibheap.cpp

i) int primfibheap(graph g,bool flagmode)

This function take a graph and flagmode as a parameter and return cost of minimum spanning tree with prim algorithm. In case of user mode when flag mode is set to true in standard output it output edges of the MST. It implements prim

algorithm with Fibonacci–heap scheme. Remove min has an amortized cost O(logn) and decrease key has an amortized cost O(1),

6) Fibheap.cpp/Fibheap.h

Structure of Fibonacci node for Fibonacci Heap implementation is as follows

```
class MinFibnode
        {
            public:
            int degree;
            int data;
            int fvertexid;
            int weight;
            MinFibnode* child;
            MinFibnode* parent;
            MinFibnode* left;
            MinFibnode* right;
            bool childcut;
         };

       class MinFibheap
       {
            MinFibnode* root;
            int nodecount;

            public:

            MinFibheap();
            void insert(MinFibnode*);
            MinFibnode* remove_min();
            void decrease_key(MinFibnode*,int);
            void pairwise_combine();
            void cascade_cut(MinFibnode*);
            void cut(MinFibnode*,MinFibnode*);
            void fibonacci_link(MinFibnode*,MinFibnode*);
            MinFibnode* make_fibnode(int,int,int);
         };
```

i) void insert(MinFibnode*)

Parameters– Node of Fibonacci node we want to insert
If the heap is empty, create a new min Fibonacci heap node and assign it to be the min element and return.
Else, insert the node in the circular doubly linked list after the min node and reset the min by comparing min with the new element inserted.

ii) MinFibnode* remove_min()

This Function extract Minimum node from Fibonacci Heap. When minimum node is removed from Fibonacci Heap, Pointer remaining in list are updated but pointers in extracted node left unchanged. Also use pairwise_combine procedure for pairwise combining of Min Tree of same degree**.** So after Min Tree Method no two tree of same degree remain in Fibonacci heap.

iii) void decrease_key(MinFibnode*,int)

Parameters- Node whose value we want to decrase,y the new value of the node
Check if new value of the node is smaller than existing value of the node
If yes decrease the node value
Remove subtree if its value is less than the parent and reset its parent's child pointer if required. Otherwise stop.
Add removed subtree at the top level.
Carry out childcut operation from parent of the removed node towards the root.

iv) void pairwise_combine()

Pairwise combining repeatdley merge the node of the same degree until all the Min tree in Fibonacci Heap has a distinct degree**.**It Maintain degree table and merge two tree of the same degree until for reach degree number of tree would be less than one.

v) Void cascade_cut (MinFibnode*)

Parameter – It take 1 MinFibnode and do cascading.
If y is a root list no more cascading required, for y to be in root list it parent would be NULL.
If child cut happens first time no more cut and cascading required
Just mark the node to indicate child cut is true
If y is marked y is just lost the second child, y is cut from here and added to the root list and cascading cut is called to the parent of the node

vi) Void cut(MinFibnode*,MinFibnode*);

Parameter- 2 MinFibnode x and y
This function Remove x from the child list of y and then decrease degree of y

vii) Void fibonacci_link(MinFibnode*,MinFibnode*)

Parameter – 2 MinFibnode x and y This Function take two Fibonacci node and make bigger node is a subtree of the smaller node.

viii) MinFibnode* make_fibnode(int,int,int)

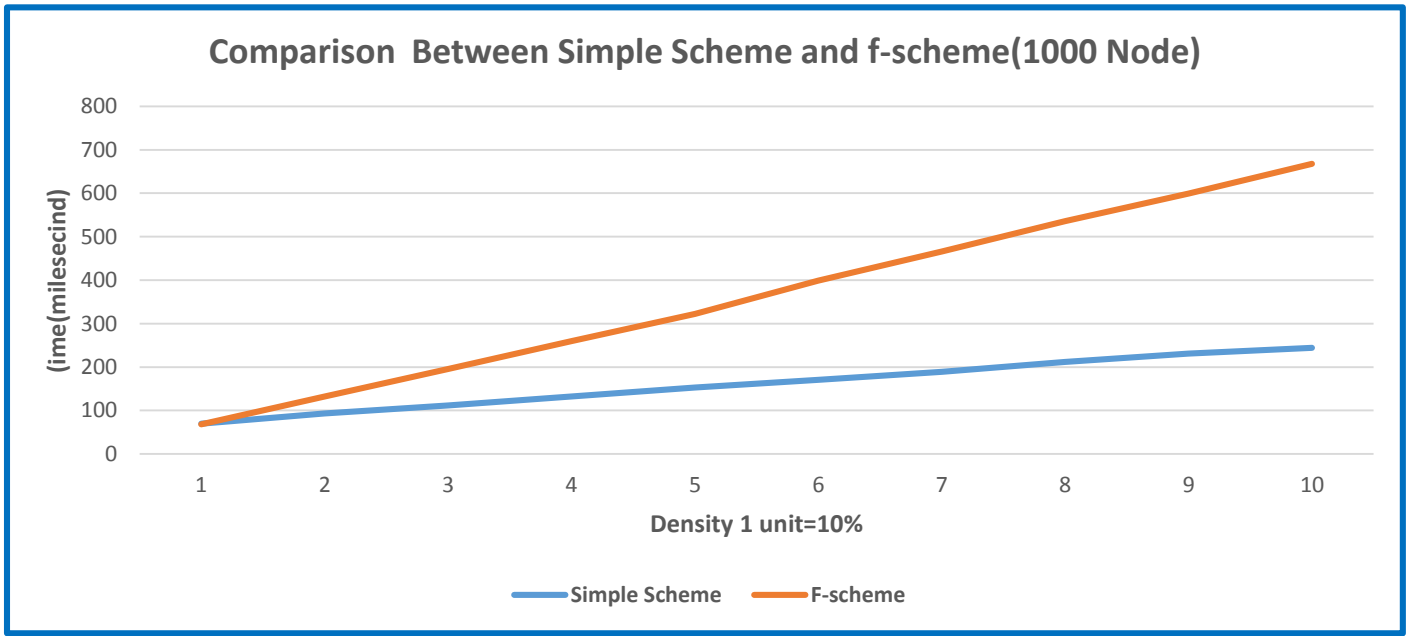
Parameter- This function take key value, weight and vertexid as a parameter.
By those parameter it create a MinFibonacci node and return this Fibonacci node

Result Summary

Prim Algorithm with both scheme in run for 1000 node, 3000 node and 5000 node in random mode and result are as follows.

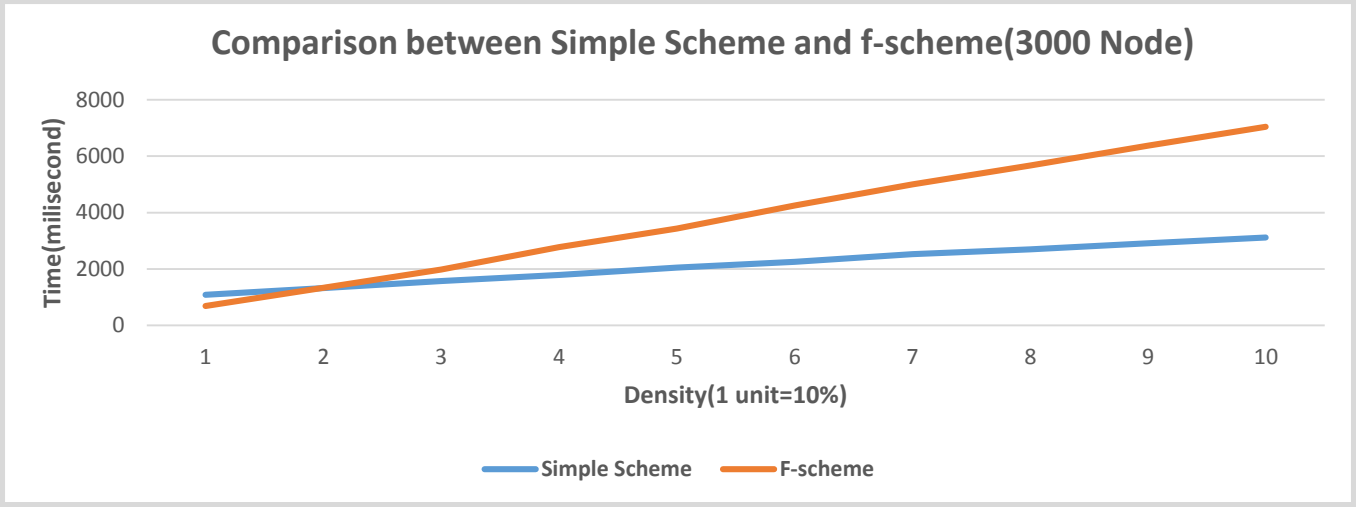
a) 1000 Node

Density ➔➔➔	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
Simple Scheme (Time Mili Second)	70	93	112	132	153	171	189	212	231	244
F-Heap Scheme (Time Mili Second)	68	132	196	260	322	399	466	536	599	668



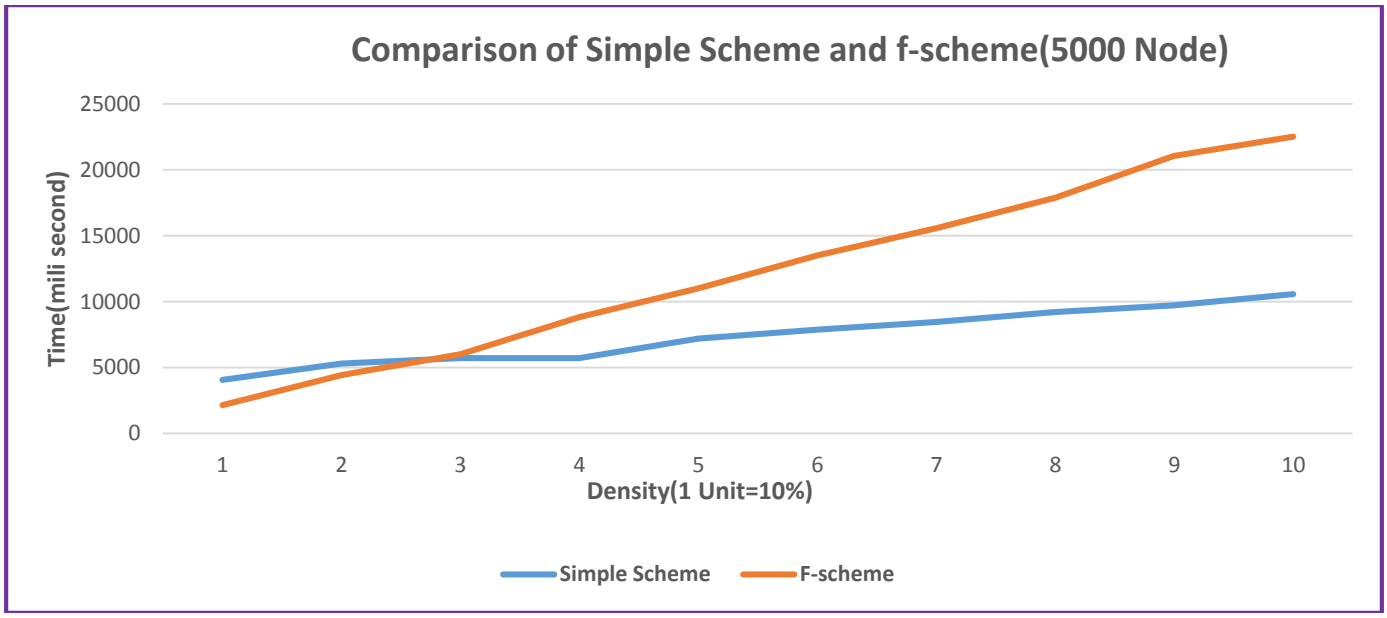
b) 3000 Node

Density ➔➔➔	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
Simple Scheme (Time Mili Second)	1087	1329	1580	1790	2048	2252	2530	2701	2910	3120
F-Heap Scheme (Time Mili Second)	686	1325	1985	2780	3440	4250	5002	5674	6376	7041



c) 5000 Node

Density ➔➔➔	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
Simple Scheme (Time Mili Second)	4080	5302	5724	6523	7202	7877	8456	9223	9732	10588
F-Heap Scheme (Time Mili Second)	2156	4427	6021	8831	11023	13504	15577	17887	21068	22056



5) Conclusion

From above results we can conclude that Fibonacci heap implementation of prim algorithm perform better than Array implementation of prim algorithm of Graph having lower density. As for practical purpose most of the real world graph is very sparse so Fibonacci heap implementation of prim algorithm is quite useful .Also when the number of node increased Fibonacci heap perform more better even in higher density.

6) References

i) Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford (2009) [1990]. *Introduction to Algorithms* (3rd ed.). MIT Press and McGraw-Hill. ISBN 0-262-03384-4.

ii) http://en.wikipedia.org/wiki/Fibonacci_heap

iii) http://stackoverflow.com/questions/504823/has-anyone-actually-implemented-a-fibonacci-heap-efficiently

iv) http://stackoverflow.com/questions/6273833/is-there-a-standard-java-implementation-of-a-fibonacci-heap