MINIMUM COST SPANNING TREE

USING

STANDARD BINARY HEAPS

AND

FREDMAN AND TARZAN FIBONACCI HEAPS

BY-
VAMSI YALAVARTHI
2015MCS2358

BINARY HEAPS:

The Prim's Algorithm for minimum spanning tree is implemented using Binary heaps. The binary heaps code is taken from the GeeksForGeeks website. And the nodes structure is defined by me.The entire code for the Prim's Algorithm is written on my own. The algorithm is developed as per Coremen textbook.

The heap node structure used is

```
struct HeapNode
{
    int v;
    int key;
    int parent;
};
```

i.e vertex number, key and parent vertex of each node.

The graph is represented using Adjacency list in both the implementations.

The standard binary heap operations used are :
insertNode()
extractMin()
Heapify()
decreaseKey().

GRAPH-GENERATION:

The graph generation is done using vectors in c++ and each generated edge is directly placed into the graphFile as per specifications. Necessary preacautions are taken to avoid multiple edges between vertices.

FREDMAN AND TARZAN ALGORITHM FOR MST IMPLEMENTED USING FIBONACCI HEAPS.

The Fibaonacci heap are implemented as per the algorithms mentioned in Coremen.
The fibnode structure used is

```
struct fibnode
{
    struct fibnode *parent;
    int key;
    int degree;
    bool mark;
    bool vmark;
    bool inHeap;
    int label;
    int parent_vertex;
    int dest_tree;
    struct EdgeNode* opedge;
    struct fibnode *left;
    struct fibnode *right;
    struct fibnode *child;

};
```

The standard functions of Fibonacci Heap defined are
insertNewNodeToHeap()
extractMin()
consolidate()
Cut()
Cascading_Cut()
decreaseKey()
heapUnion()
All the above functions are developed as per the algoriths of Coremen book.

Now comming to the implementation of MST using the algorithm developed by Fredman and Tarzan, it is developed as three major functions

1. FredmanMST()
2. TarzanMST()
3. RadixSort()

1. FredmanMST()- This function is used to iteratively call the TarzanMST() and RadixSort() until all the nodes does not come into one single tree. It just has two calling functions called repetetively where first function's output is passed as input to second function and vice-versa.

2.TarzanMST()- This function does the major part of traversing the adjacent nodes of visited vertices and pushing the corresponding key values into Fibonacci heaps.
This function repeats this process untill the heap size increases by pow(2, m/n)
m-edges.
n-nodes.
This capacity increases in each iteration by 2 power the previous max-heap-size.
This ends when the heap size is equal to zero or we get a visted vertex during exractMin().This relabels the vertices according to the tree in which the had occured.

3.RadixSort()- This fuction performs the cleanup process using 2-phase radix sort and generates a tree that is input for the next interatio of TarzanMST(). It first sorts based on destinations and then based on sources.

This is repeated and every time we add an edge to the output graph we add that edge cost to the total_mst_cost.The final total_mst_cost is displayed on console and the final tree is stored in another graph.

ANALYSIS:

TestCases:	Fredman-Fibonacci	Prims-Binary
inputGraph-1.txt	7 sec	70ms
inputGraph-2.txt	1min 2 sec	170ms
inputGraph-3.txt	10ms	1ms
inputGraph-4.txt	10ms	10ms
inputGraph-5.txt	10ms	10ms
inputGraph-6.txt	70 sec	10ms
inputGraph-7.txt	1 sec	50ms
inputGraph-8.txt	19 sec	120ms

The above observations are for graphs with a max of 1,00,000 nodes and 5,00,000 edges and this observations states that Fibonacci takes faster time for much larger nodes of size more than 1 crore . Till then binary beats Fibonacci in performance.