ADS PROJECT-FALL 2013

BHARATH KURUMADDALI

6513-0561

bharath92@ufl.edu

Steps to compile the program:

This program has been compiled and executed on thunder.cise.ufl.edu using the g++ compiler. To compile the code use the following command:

g++ mst.cpp –o mst

After compilation to execute the code use the following commands:

For random mode:

./mst –r n d

Here n and d are integer values.

For file input mode:

./mst –s filename.txt (this is for simple implementation of the input file)

./mst –f filename.txt (this is for Fibonacci implementation of the input file)

Classes and functions used in the program

There are 5 basic classes which have been used in this program. These clases have multiple functions which will be described while class explanation. The classes used are as follows:

- Class for vertex key. This has the vertex and key stored this class has been defined for usage during the Prims implementation using Fibonacci heap.
- Class for Fibonacci heap node. This class has all the components that the node must contain i.e parent,child,left sibling,right sibling,degree and also the Boolean value of childcut. This class is also used for generating the Fibonacci heap of the given graph as the values of the heap are stored using this class.

- Class Fibonacci heap. This class has different functions embedded into it which perform the various operations, most important functions are as follows:

    1. Function for Insert. This function does the basic operation of insert into the Fibonacci heap.
    2. Function to Combine with top level. As evident from the name this function is to combine all elements/trees at top level and make this into a heap.
    3. Remove min. This is the function to remove the minimum element in the heap and change its pointer to the element that has the least value after this operation has been completed.
    4. Pairwise combine. This function after every remove min checks if there are any nodes with the same degree and combines them. This process ensures that the heap at a point of time will not have all n elements scattered around without forming any trees hence improving the amortized complexity of the system.
    5. Decrease Key. The decrease key operation is done in this function.
- Class edgenode. This class has the two vertices which are used in the program i.e the source vertex and the destination vertex and also the cost between them. This nodes are public and can be accessed by all functions.
- Class graph. This class has all the functions that are used for graphs and also the basic functions for simple scheme and the Fibonacci scheme. Functions included in this class are as follows:
    1. Graph function. This function creates the adjacency list of the randomly generated graph/graph from the text as input.
    2. Reallocate. This function is to reallocate the adjacency list.
    3. Generate random graph. In this function using random values between 0 to n-1 an n node graph is generated using the random function.
    4. Graph connected or not. This function uses Breadth first search algorithm to check if each and every node of the graph is connected to the finally formed graph.
    5. Add edge. This function is to add an edge when doing the prims algorithm to the list so as to calculate the minimum cost.
    6. Simple scheme. This function is where prims algorithm is implemented for simple scheme.

7. Fibonacci scheme. This function operates the prims method using Fibonacci scheme to get the minimum spanning tree.
8. Generate graph from file. This takes the filename from the command line and reads the input file and depending on the command does either simple scheme or Fibonacci scheme on this input text.

Main():

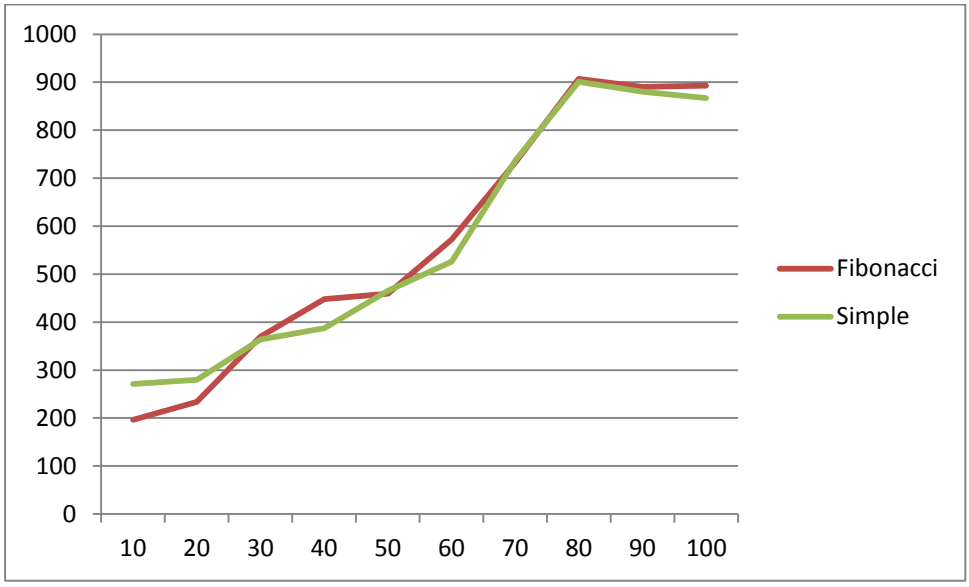
The main function takes in the arguments from the command line and does a check if the command line is valid or not and also main has the function to calculate the time elapsed in computing the minimum cost using both simple scheme and Fibonacci scheme.

Expected performance:

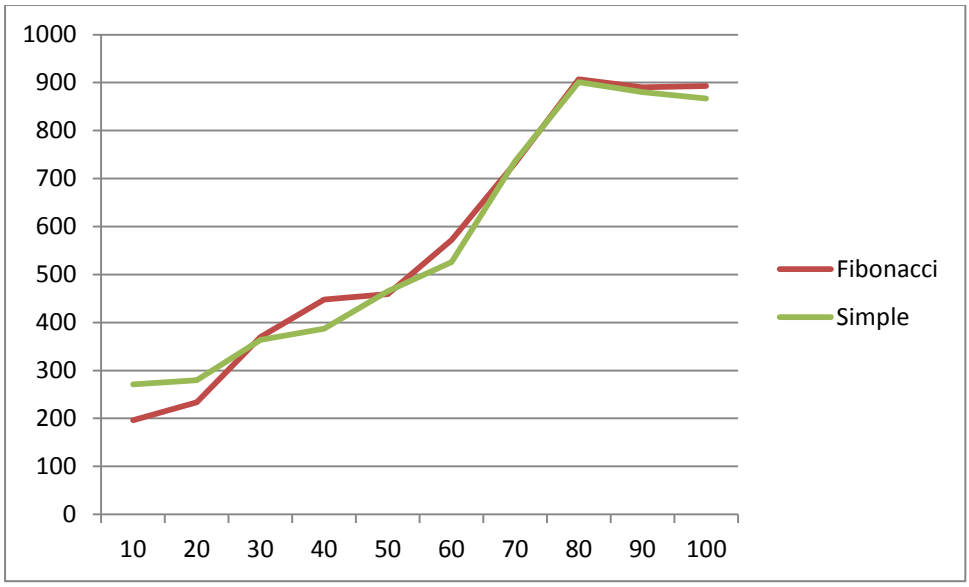
We know that the simple scheme has time complexity of O(n^2), where as Fibonacci heap has O(nlogn+e). so we can expect that at high densities performance of simple will be the same whereas Fibonacci will take more time because the no of edges are in the order of n^2, it also has a nlogn component which is added to this and must take higher time than the simple scheme.

Actual performance

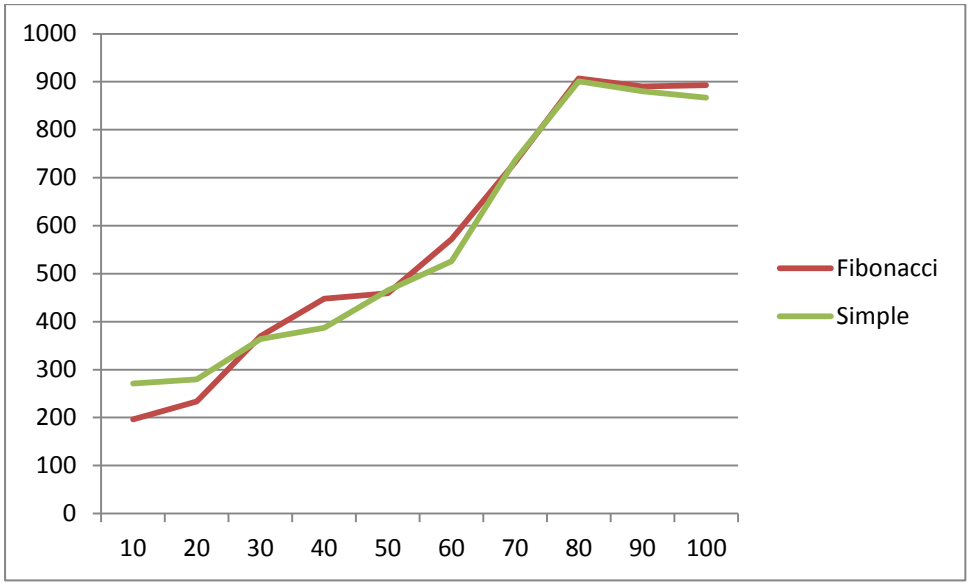
N=1000 nodes



N=3000 nodes



N=5000 nodes:



From the above graph it can inferred that Fibonacci heap is much better when the graphs are sparse ie at lower densities and the performance is almost the same at higher densities ie a dense graph. It can also be inferred that the performance of the algorithms also depend on the number of nodes. When the number of nodes are high the performance is almost the same for both the simple scheme and Fibonacci scheme.