EC 704 - VLSI Design Automation Assignment 3

-Shruti Masand, 181EC245

Problem Statement:

1. Use the netlist of the ISCAS85 Benchmark circuits given in the course website. Or use similar benchmark netlists.

2. Using Prim's / Kruskal's Algorithm, find the Minimum Spanning tree of the graph.

Solution:

Approach:

STEP 1 : Firstly, we need to convert the given netlist into a graph. To do that, I have tried to read the Netlist File line by line (using the readline() function) and tried to interpret the meaning of each line. As we have to read the gates, we take all the equations that start with "G".
Then, we can simply read LHS and RHS separately. The LHS part of that particular line will tell us about the output and the RHS part will tell us about the inputs. Hence, we create 2 lists by the name of LHS and RHS that will be used to store the values of the outputs and the inputs respectively.

These (input, output) pairs will then be appended to a list that is given as input to the Networkx library functions.
The commands written using this library called Networkx, very conveniently plot these functions into graphs with gates as nodes and directed connections shown as Edges.
For an even enriching visual aid, we can change the size and color of these edges and vertices as well. The root code for the given library ws thoroughly understood and is also given as one of the links in the references.

STEP 2 : Now, we need to apply the Minimum Spanning Tree Algorithms - which can be either Prim's or Kruskal Algorithm.

This can be done by coding in accordance with the algorithms discussed in the class.

- We need to sort the list of the edges in accordance with their weights. Since, through the netlists given, we don't really have weights, we consider each edge to be of Weight 1.

- Next, we pick each edge one by one (we pick the one with minimum weight first) and keep adding to the MST list by checking if it doesn't end up forming a cycle.

- If our added edges include all the vertices and don't even form any cycle in between, we have our Minimum Spanning Tree ready !!

Code, implemented in Python:

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    def add_edge(self, u, v, w):
        self.graph.append([u, v, w])

    # Search function

    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])

    def apply_union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
            rank[xroot] += 1

    #  Applying Kruskal algorithm
    def kruskal_algo(self):
        result = []
        i, e = 0, 0
        self.graph = sorted(self.graph, key=lambda item: item[2])
        parent = []
        rank = []
        for node in range(self.V):
            parent.append(node)
            rank.append(0)
```

```
        while e < self.V - 1:
            u, v, w = self.graph[i]
            i = i + 1
            x = self.find(parent, u)
            y = self.find(parent, v)
            if x != y:
                e = e + 1
                result.append([u, v, w])
                self.apply_union(parent, rank, x, y)
        for u, v, weight in result:
            print("%d - %d: %d" % (u, v, weight))


## STEP 1 - Plotting a graph from the Netlist

def read_gates(l,index):
    t = ''
    t = t + l[index]
    j = index + 1
    while ((l[j] != ' ') and (l[j] != ',') and (l[j] != ')')):
        t = t + l[j]
        j = j +1
    return(t)

lhs = []
rhs = []
exray = []
vega = []

abc = 0

dict = {}

file1 = open('s27bench', 'r')
Lines = file1.readlines()

for line in Lines:
    if(len(line)>0):
        if(line[0] == 'G'):
            gate_lhs = read_gates(line, 0)
            if(dict.get(gate_lhs)):
                gate_lhs_index = dict[gate_lhs]

            else:
```

```
            gate_lhs_index = abc
            dict[gate_lhs] = abc
            abc = abc+1


        for i in range(1, len(line)):
            if (line[i] == "G"):
                gate_rhs = read_gates(line,i)
                if(dict.get(gate_rhs)):
                    gate_rhs_index = dict[gate_rhs]

                else:
                    gate_rhs_index = abc
                    dict[gate_rhs] = abc
                    abc = abc+1

                lhs.append((gate_rhs_index,gate_lhs_index))

#To get a list of all Vertices:
                exray.append(gate_rhs)
                exray.append(gate_lhs)

lhs.sort()
#print(lhs)

import networkx as nx
import matplotlib.pyplot as plt
from math import *

G = nx.DiGraph()

G.add_edges_from( lhs )

pos = nx.spring_layout(G)

nx.draw_networkx_nodes(G, pos, node_size = 500)
nx.draw_networkx_edges(G, pos, edgelist = G.edges(), edge_color = 'black')
nx.draw_networkx_labels(G, pos)
plt.show()

## STEP 2 - Creating an Adjacency Matrix

import numpy as np
A = nx.to_numpy_matrix(G)
```

```
shruti = np.shape(A[0])[1]
#print(shruti + 1)

#STEP 3 : Use of Kruskal Algorithm

#list with all the vertices
exray = list(set(exray))
#print(exray)

#number of edges = length of LHS
edges = len (lhs)

#number of vertices = the variable shruti + 1
vert = shruti + 1

#since no weights are given in these graphs,
#we assume that all edges have the weight 1
weight = np.ones([vert])
print(len(exray))
for j in range(len(exray)):
    #vega.append((exray[j], weight[j]))
    print(exray[j], '\t' ,weight[j] )


g = Graph(abc)
for elem in lhs:
    g.add_edge(elem[0],elem[1],1)

g.kruskal_algo()
```

Screenshots of the results obtained:

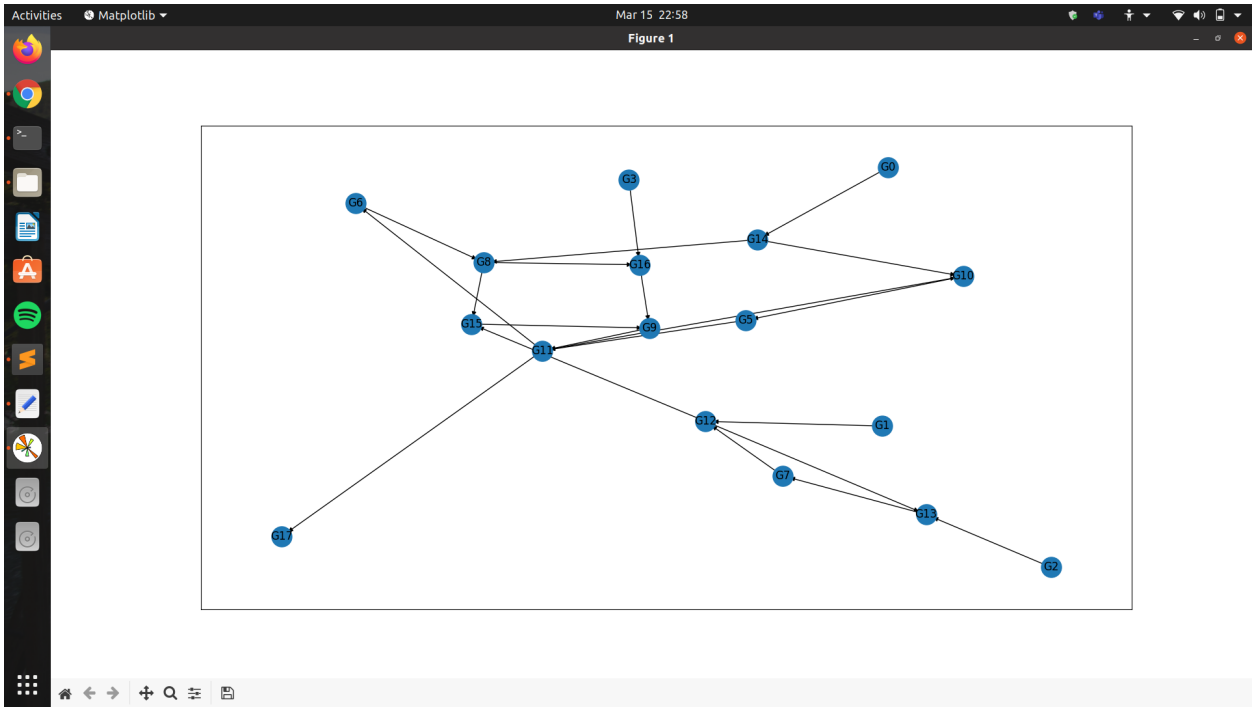

Fig1. Displays the graph obtained using the netlist file : s27.bench

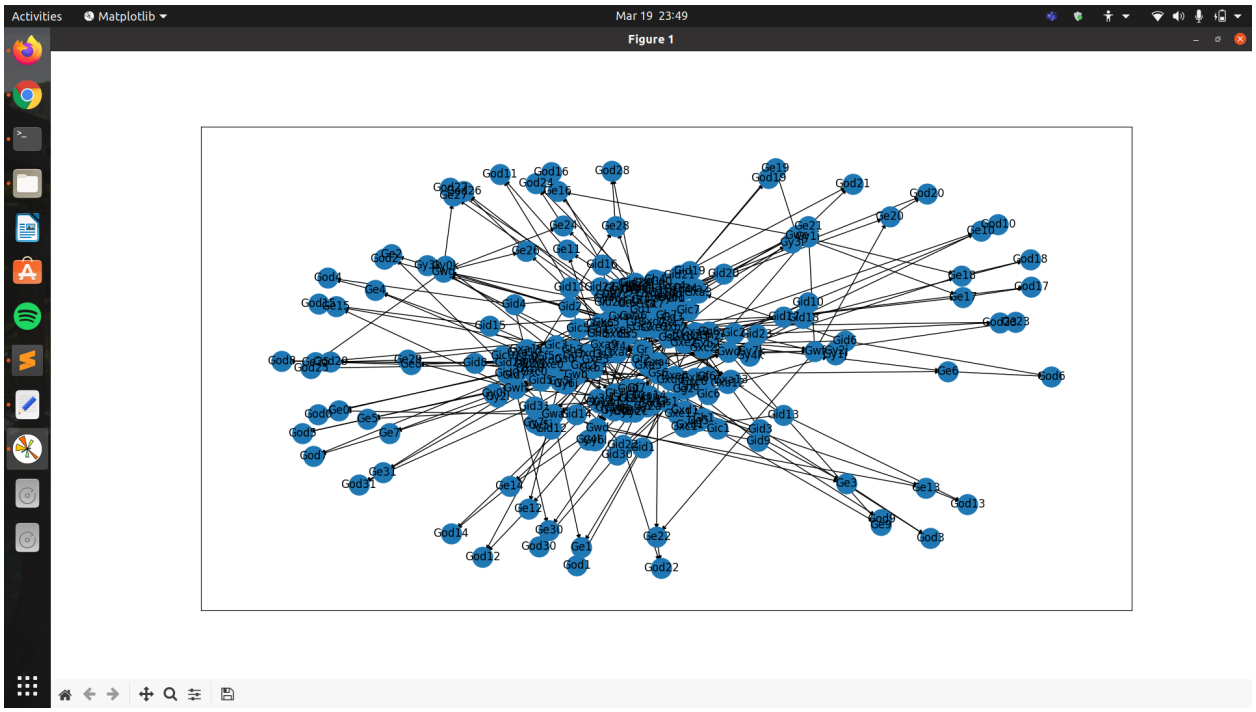

Fig2. Similarly, the graph can also be obtained for something as complex as the c499.bench file, as given in the figure

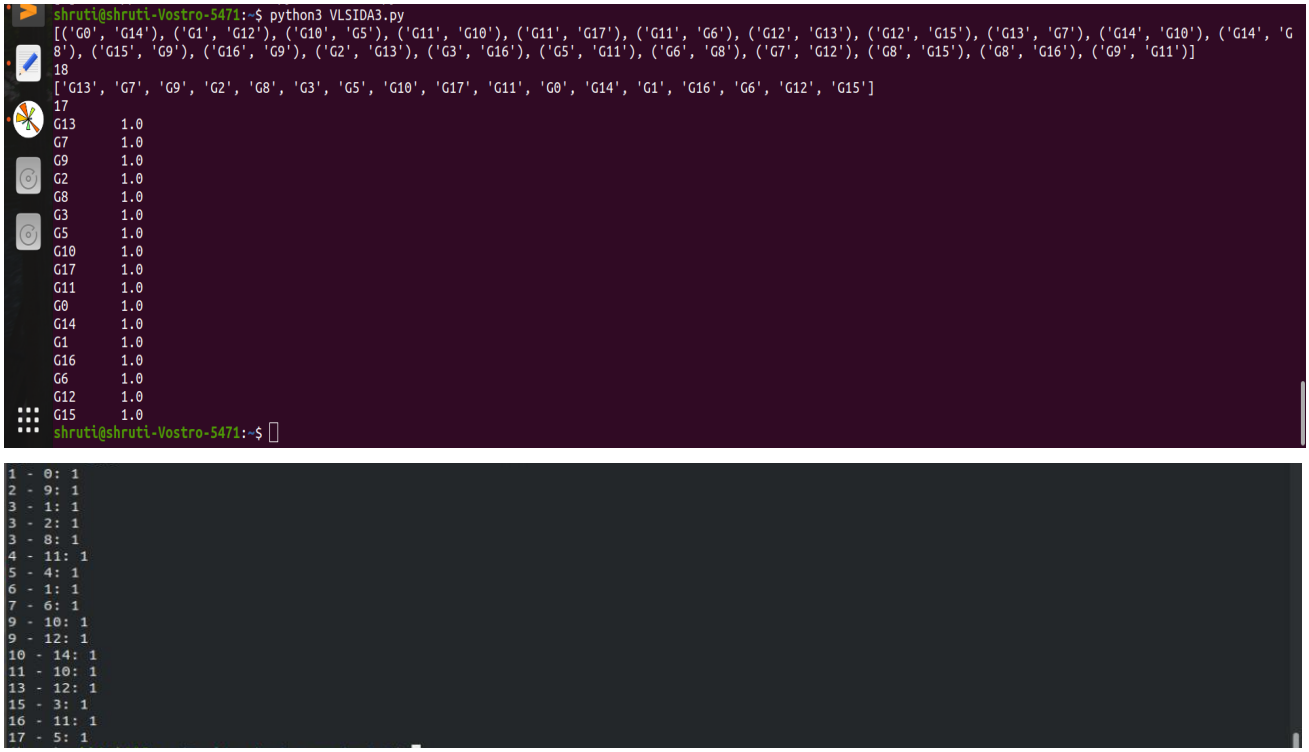

Fig3. The figure displays:

a) List of the edges corresponding to the vertices

b) Number of vertices

c) List of vertices

d) List of vertex with its weight

e) Kruskal Table