EC 704 - VLSI Design Automation Assignment 2

-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 Bellman-Ford / Dijkstra's Algorithm, find the shortest path from any given node to the rest of the nodes.

Solution:

Approach, divided in 3 steps:

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.
Alternatively, in case our netlist contains variable names instead of names starting with G, we can code for recognizing gates like NOR, DFF, XOR, AND etc and then calculate the LHS and RHS functions.

STEP 2 : Next step is to use that graph to form the adjacency list/matrix. The adjacency Matrix gives us the information about the nodes that are in the neighbourhood of a particular node.

This can be accomplished by the use of the networkx and the numpy library. This is accomplished using a single command that is shown in the code.

STEP 3 : Now, we need to apply the shortest path algorithms - which can be either Bell Ford algorithm or the Djikstra Algorithm.
This can be done by coding in accordance with the algorithms discussed in the class.

Code, implemented in Python:

```
## STEP 1 - Plotting a graph from the Netlist
def read_gates(l,index):
    t = ""
    t = t + l[index]
    for j in range(1,3):
        if(l[index+j] >= '0' and l[index+j] <= '9'):
            y = l[index + j]
            t = t + y
    return(t)

lhs = []
rhs = []

file1 = open('s27.bench', 'r')
Lines = file1.readlines()

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

            for i in range(1, len(line)):
                if (line[i] == "G"):
                    gate_rhs = read_gates(line,i)
                    lhs.append((gate_rhs,gate_lhs))


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

lhs.sort()
print(lhs)
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)

arr = np.zeros([shruti+1,shruti+1])

for elem in lhs:
   first = int(elem[0][1:])
   second = int(elem[1][1:])
   arr[first][second] = 1

print(arr)

#STEP 3 : Use of BellMan - Ford Algorithm
#Let src be the node from where we need to calculate the shortest distance to all other points.

#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(weight)

def BellmanFord(src):
        # Initializing distance from source vertex to all vertices as INFINITE and distance of
source vertex as 0
        dist = [inf]*vert
        dist[src] = 0
```

```
for t in lhs:
        u = int(t[0][1:])
        v = int(t[1][1:])
        for i in range(vert):
                if dist[u] != float("inf") and dist[u] + 1 < dist[v]:
                        dist[v] = dist[u] + 1


print('Distance from source vertex',src)
print('Vertex \t Distance from source')
for i in range(len(dist)):
        print(i,'\t',dist[i])
```

BellmanFord(0)
#You can put any index in the input (should be in the range of the number of vertices) and make that the source in order to get the list of distance from the selected node to the several other vertices.

Screenshots of the results obtained:

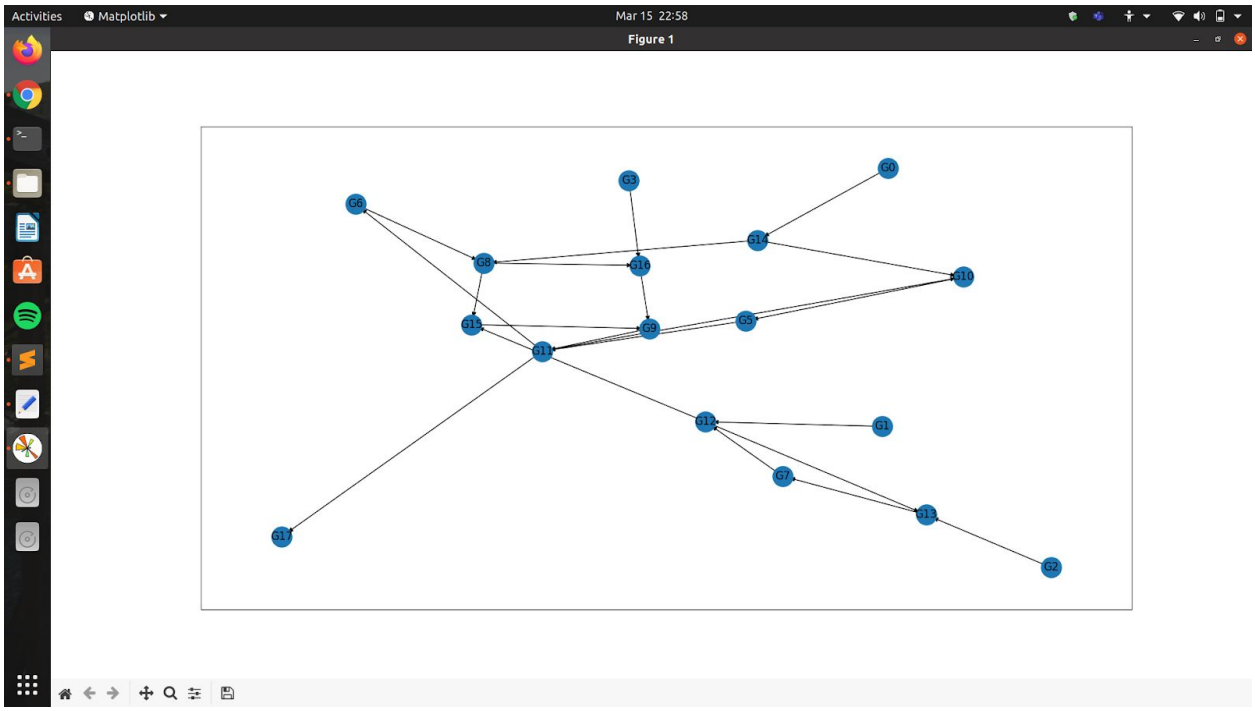

Fig1. Displays the graph obtained using the netlist file : s27.bench

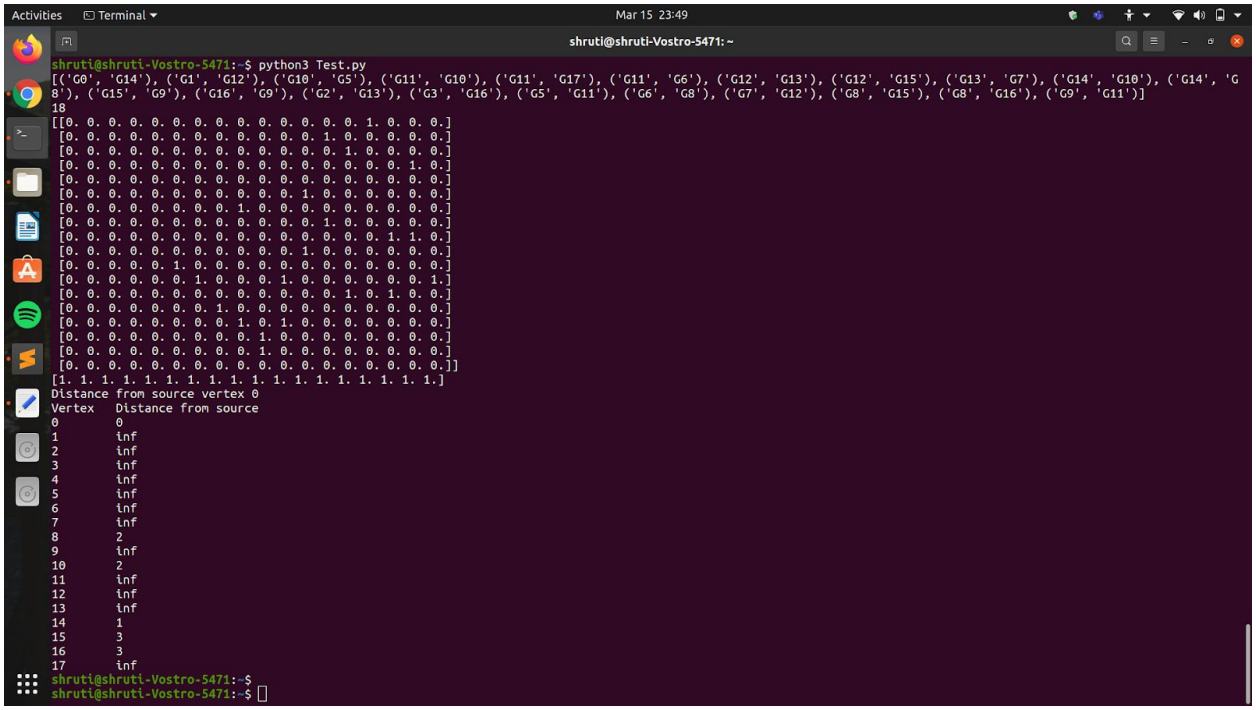

Fig2. Displays :

a) **LHS list that contains the list of all directed edges**

b) **18 - i.e. the number of vertices in the case of s27.bench file**

c) **Adjacency matrix for the s27.bench file**

d) **Matrix of 1s as we don't have any info on the weights, so we give them all value 1**

e) **Table corresponding to the distance of each vertex from the G0th vertex (can be calculated for other vertices by substituting the src variable)**