EC 704 - VLSI Design Automation Assignment 1

-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. Represent the netlist in the graph format with gates as nodes and connections as edges.

3. Obtain the adjacency list/matrix for the netlist/circuit.

4. Traverse the graph in Depth-First Search and display the nodes.

5. Traverse the graph in Breadth-First Search and display 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 : Using this information, we can carry out the Depth First Search (DFS) and Breadth First Search (BFS) by using their algorithms as taught in the class.

The next section contains the code implemented using the above Approach. It also contains the important and significant results obtained at the subsequent steps.

Code, implemented in Python:

```
def DFS(arr,shruti,visited,start):

    # Print current node
    print(start, end = ' ')

    # Set current node as visited
    visited[start] = True

    # For every node of the graph
    for i in range(shruti):

      if (arr[start][i] == 1 and
            (not visited[i])):
          DFS(arr,shruti, visited,i)


def BFS(arr,shruti):


    visited = [False] * shruti
    q = [0]

    # Set source as visited
    visited[0] = True

    while q:
       vis = q[0]
```

```
        # Print current node
        print(vis, end = ' ')
        q.pop(0)

        # For every adjacent vertex to
        # the current vertex
        for i in range(shruti):
            if (arr[vis][i] == 1 and
                (not visited[i])):

                # Push the adjacent node
                # in the queue
                q.append(i)

                # set
                visited[i] = True

        if(len(q)==0):
                d = 0
                for x in range(shruti):
                        if(visited[x] == False):
                                visited[x] = True
                                q.append(x)
                                d=1
                                x= shruti+1



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('s27bench', '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

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)
print("Abhi")
print(A)

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

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

# map[string] = {false}
# count_var=0
# for elem in lhs:
```

```
#         if(dict[elem[0]] == false):
#                 dict[elem[0]]=true;
#                 count_var+=1
#         if(dict[elem[1]] == false):
#                 dict[elem[1]]=true;
#                 count_var+=1

# Arr[count_var][count_var] = {0}

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

print(arr)



print("BFS:")

BFS(arr,shruti+1)

print("DFS:")
visited = [False] * (shruti+1)
for i in range(shruti+1):
        if(visited[i]==False):
                visited[i]=True
                DFS(arr,shruti+1,visited,i)
```

Screenshots of the results obtained:

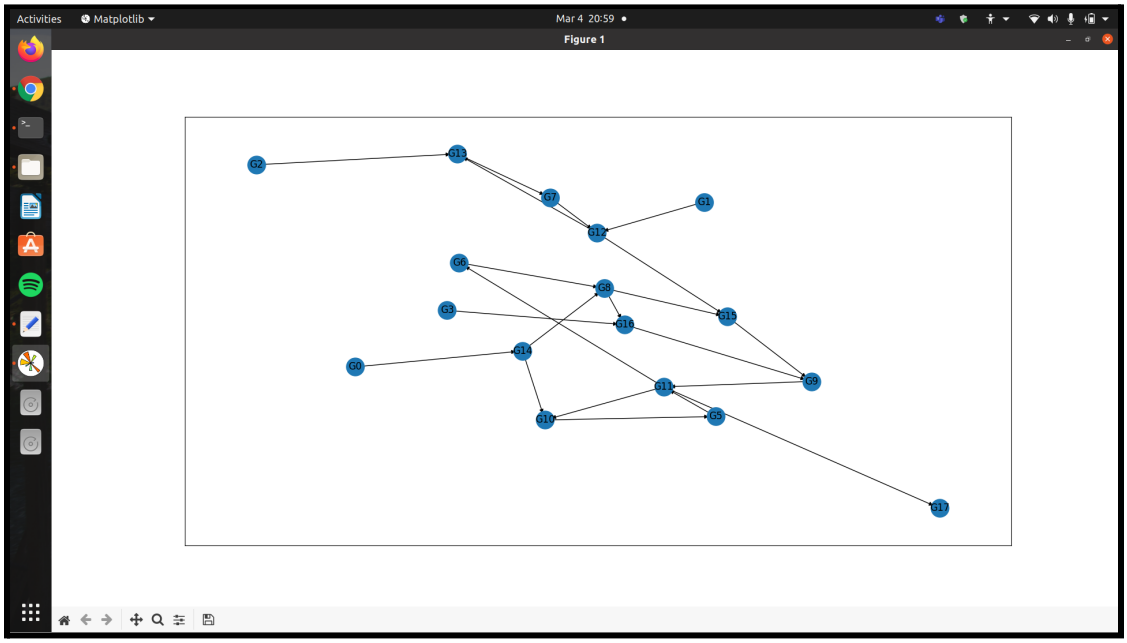

Fig1. Graph produced from the netlist (s27.bench)

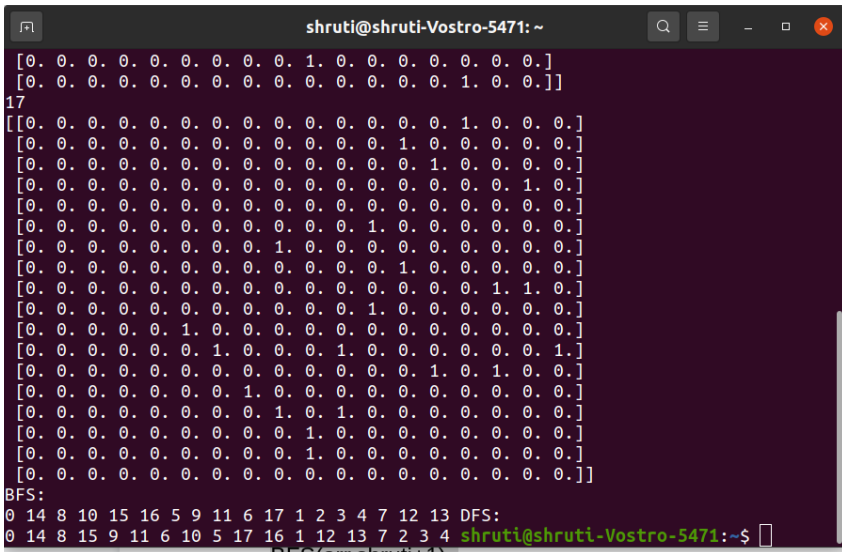

Fig2. Adjacency List produced from the above graph + BFS +DFS