**RISHIK SAHA-2022CS11932| PARTH VERMA-2022CS11936**

# COL215-SOFTWARE_1

# REPORT

## PART-A

## ALGORITHM:

The content from the given input files is extracted and stored in dictionary (for comfortable retrieval) named "data".

Basically, here we create a backward directed graph representing the circuit and use recursion to get the result. Now when we want the delay of a node say 'n':

We call a function calc_delayA(n) which does the following:
If the node is a primary input:

we return 0. (A zero delay)

If not:

We retrieve the connected nodes to 'n' through the dictionary "data" in O (1) and find the maximum delay among them through recursively calling the calc_delayA() function for each of them. Further we add the delay of the corresponding gate and return the sum.

Finally, we call the calc_delayA() for the primary_outputs and get the desired "circuit_delay".
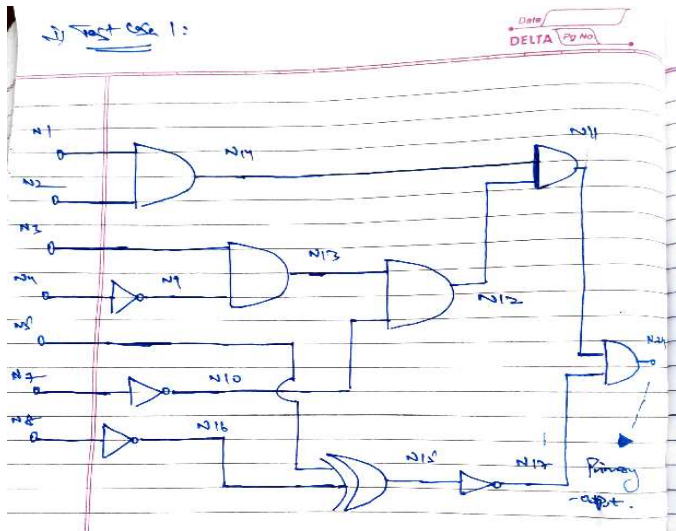
## TIME COMPLEXITY:

The time complexity of the above algorithm will be

O(v+e) where v is the no of vertices and e is the no of edges in the graph.

This is because for each node we recursively call the calc_delayA () function and hence as a result travel all the possible paths of the graph.

# TESTING STRATEGY:

## Test Cases:

## Test Case 1:



```
circuit                    X    gate_delays

File    Edit    View

PRIMARY_INPUTS N1 N2 N3 N4 N5 N7 N8
PRIMARY_OUTPUTS N24
INV N4 N9
AND2 N9 N3 N13
AND2 N2 N1 N14
INV N7 N10
AND2 N13 N10 N12
INV N8 N16
XOR2 N16 N5 N15
INV N15 N17
AND2 N17 N11 N24
AND2 N14 N12 N11
```
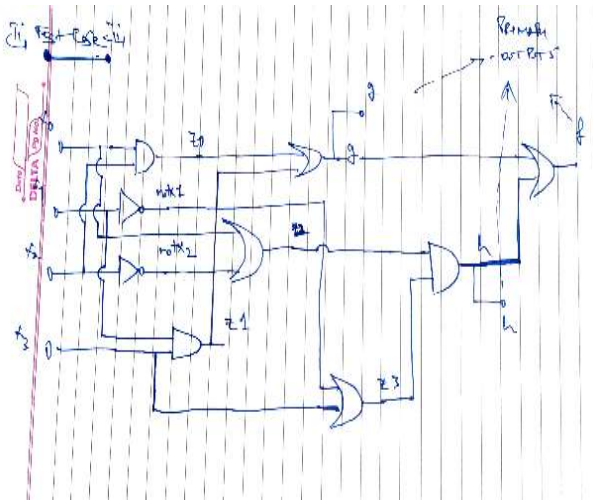
```
gate_delays               X

File    Edit    View

NAND2 3
AND2 4
NOR2 3.5
OR2 4.5
INV 2
XOR2 2.5
```

## Expected output:

```
output_delays            X    +

File    Edit    View

N24 18.0
```

## Test Case2:



```
circuit                          gat

File    Edit    View

PRIMARY_INPUTS x0 x1 x2 x3
PRIMARY_OUTPUTS g f h
INV x2 notx2
INV x1 notx1
AND2 x0 x2 z0
AND2 x1 x3 z1
OR2 x0 notx2 z2
OR2 notx1 x3 z3
OR2 z0 z1 g
AND2 z2 z3 h
OR2 g h f
```

```
gate_delays (1)                    ×

File    Edit    View

NAND2 3
AND2 4
NOR2 3.5
OR2 4.5
INV 2
XOR2 2.5
```
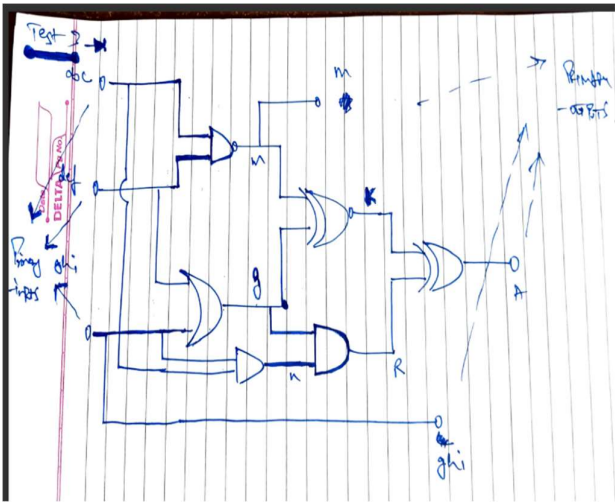
## Expected output:

```
output_delays                     ×

File    Edit    View

g 8.5
f 15.0
h 10.5
```

# Test Case _3:



circuit

File    Edit    View

```
PRIMARY_INPUTS abc def ghi
INTERNAL_SIGNALS g n R k
PRIMARY_OUTPUTS m A ghi
NAND2 abc def m
OR2 def ghi g
AND2 ghi abc n
AND2 g n R
XNOR2 m g k
XOR2 k R A
```
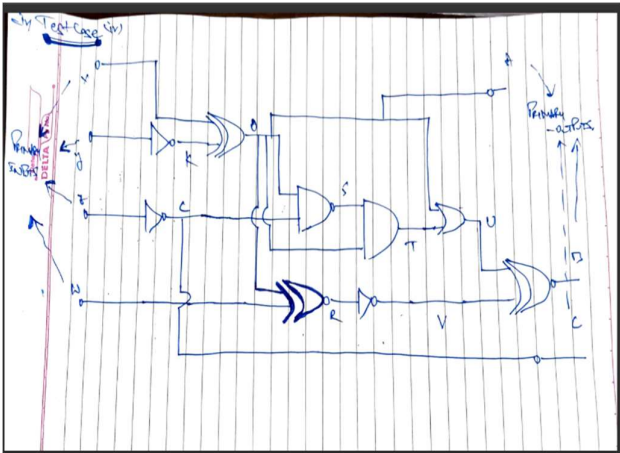
gate_delays

File    Edit    View

```
NAND2 3
AND2 4
NOR2 3.5
OR2 4.5
INV 2
XOR2 2.5
XNOR2 3
```

## Expected output:

output_delays

File    Edit    View

```
m 3.0
A 11.0
ghi 0
```

## Test Case _4:



```
circuit                                    ×

File    Edit    View

PRIMARY_INPUTS x y z w
INTERNAL_SIGNALS A k C S T R V U
PRIMARY_OUTPUTS A B C
INV y k
XOR2 x k A
INV z C
XNOR2 A w R
NAND2 A C S
AND2 A S T
INV R V
OR2 A T U
XNOR2 U V B
```

```
gate_delays                          ×

File    Edit    View

NAND2 3
AND2 4
NOR2 3.5
OR2 4.5
INV 2
XOR2 2.5
XNOR2 3
```

## Expected output:

```
gate_delays

File    Edit    View

A 4.5
B 19.0
C 2.0
```

# Reason for choosing the above Test Cases:

The above testcases were chosen for testing the algorithm in different scenarios.

The test cases ensured that our algorithm manages sufficiently high no of nodes and complex graphs.

The nodes were given variable names (multi-character) to ensure there is no error in reading of input files.

All possible gates including (XOR2, XNOR2) were introduced in the test cases.

The test cases were designed so that a node is connected to multiple nodes (in both forward and backward directed graphs) which was essential in testing the recursive algorithm.

The output signals were taken from all the regions of the circuit (input stage, middle region, ending region) of the graph.