COL 215: SOFTWARE ASSIGNMENT 3

REPORT

Neha Kumari: 2023CS50205

Avani Komalkar : 2023CS50325

Problem Statement:

Given:

- A set of rectangular logic gates g1, g2...gn
- Width and height of each gate gi
- The input and output pin locations (x and y coordinates) on the boundary of each gate gi . p1, gi .p2, ..., gi .pm (where gate gi has m pins)
- Gate delay Dg1, Dg2 ...Dgn
- Wire delay per unit length Dwire
- The pin-level connections between the gates

To write a program to assign locations to all gates in a plane so that:

- No two gates are overlapping
- The Critical path of whole circuit is minimized

Approach:

1. We created the Pin class to represent individual pins of a gate in the circuit. Each pin is defined by its unique identifier (pin_id) and its x and y coordinates, which specify its position on the gate.

2. We also made the Gate class to represent a gate in the circuit. This class holds information such as the gate's dimensions, delay, pins, and primary input/output pins.
3. Then, we created a check method: This method checks if the gate is the last in a chain (i.e., it has a primary output) and determines if it has a primary input.

4. We created several functions to handle different operations:
    - **take_input()**: This function reads the input file, processes gate data, and initializes each gate with its properties and pin configurations.

    - **move_ahead()**: This function helps trace the paths between gates based on their pin connections. This is a recursive function that moves through the circuit and finds all paths from all primary input to all primary output.

    - horizontal_placement(): This function calculates the placement of gates in a horizontal layout.

    - **shift_from_gate()**: This function shifts the gate position during placement based on the calculated shift value.

    - verticle_up_placement(): This function places gates in an upward vertical layout. Coordinates **are adjusted vertically. The** height of each gate is added to the previous gate's y-coordinate.

    - verticle_down_placement(): This is the opposite of verticle_up_placement, where gates are placed downward. It subtracts height as it calculates the coordinates.

    - divide_in_two(): It divides a list of gate IDs into two separate lists: one for upper gates and another for lower gates. It alternates between the two lists while dividing.

    - normalise_coordinate(): This function ensures that all gate coordinates are positive, normalizing them by shifting the entire coordinate system if necessary.

    - answer_calculation_of_path(): This function calculates maximum critical path and maximum critical path delay by considering gate delays and wire delays.

    - **print_answer()**: This function prints the final answer, including the gate coordinates and the critical path.

5. Now we created a loop to check each gate using the check () method. Then, we created an empty list to store all the possible paths. For each gate that has a primary input, we traced its path using the move_ahead() function, which moves through the circuit and records the connections between gates and pins.

6. We then sorted all the paths based on the number of gates in each path, starting with the path that contained the most gates and stored it as basline.

7. We created connections between gates using the make_connections() function, which analyzes the paths and creates a dictionary of connections.This allows us to map out which gates are connected and where the connections occur.

8. Based on the list of gate IDs, the layout can be determined using horizonal_placement, verticle_up_placement, or verticle_down_placement, depending on the preferred arrangement. These functions calculate the coordinates of each gate.

9. We created boxes using the BoxxedBox class to organize the gates based on their connections. The gates are divided into two lists, lower_list and upper_list, using the divide_in_two() function. Each list is sorted to ensure proper placement.

10. After calculating gate positions, normalise_coordinate ensures all coordinates are non-negative by shifting them appropriately.

11. Using the coordinates, the function answer_calculation_of_path computes the delay of each path, including both gate delays and wire delays. The path with the largest delay is identified as the critical path.

Time Complexity Analysis:

Taking g as gates and p as pins.

Time complexity of the functionalities implemented:

Take input = O(g+p)	[L6 - L14]
Move ahead = O(g^2)	[L15 - L22]
Sort = O (g^2 log(g))	[L23- L27]
Connections = O(g^2)	[L28]
Made dictionary and baseline = O(g)	[L30 - L35]
Divide in two = O(g^2)	[L41]
Sort = O (g^2 log(g))	[L43 - L42]
Packing boxes = O(g)	[L46 - L57]
Horizontal placement = O(g)	[L58]
Shifting = O(g^2)	[L60 - L72]
Placing packed boxes = O(g)	[L72 - L80]
Creating answer dictionary = O(g)	[L81 - L83]
Normalise coordinates = O(g)	[L85]
Calculation of critical path = O(g^2)	
Print answer = O(g)	

Overall Time complexity = O (g^2 log(g) + p)

Testcases and Result:

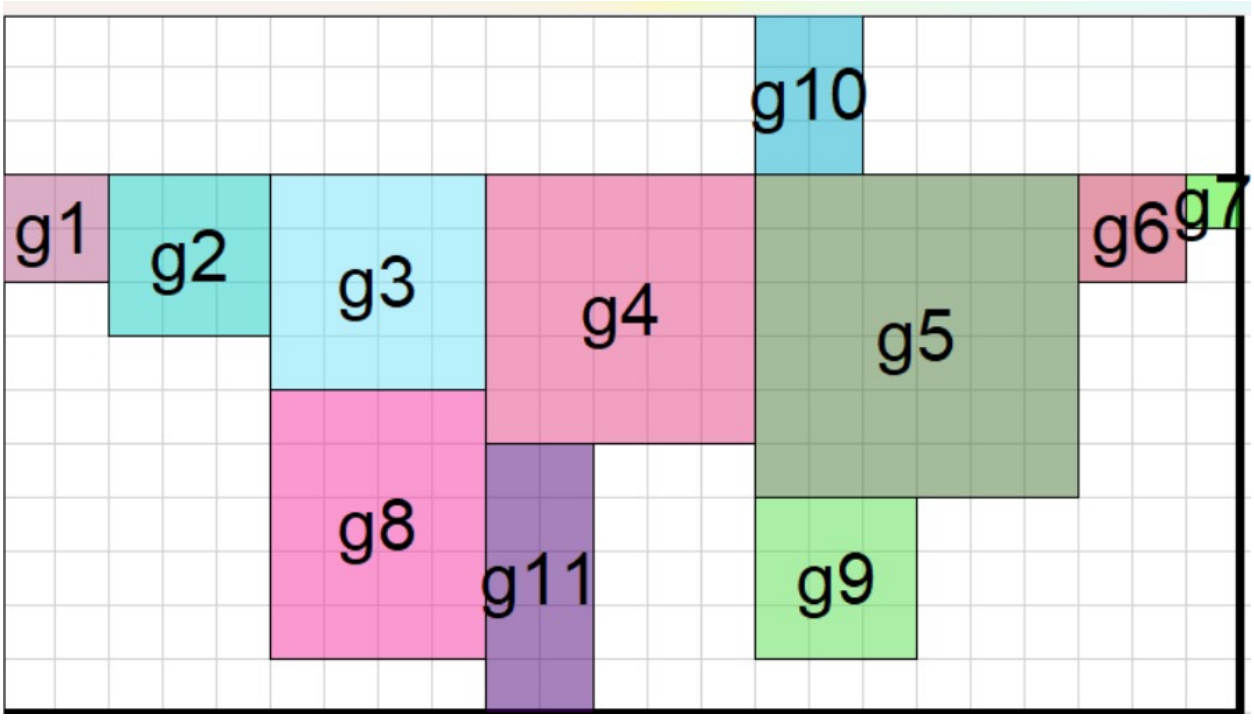
***Testcase* 1:**

```
g1 2 2 1
pins g1 0 0 0 1 2 2
g2 3 3 2
pins g2 0 3 3 0 3 3
g3 4 4 1
pins g3 0 1 0 4 4 3 4 2
g4 5 5 1
pins g4 0 3 0 1 0 4 5 0 5 4
g5 6 6 3
pins g5 0 6 0 1 0 3 6 4
g6 2 2 1
pins g6 0 0 2 1
g7 1 1 1
pins g7 0 0 1 0
g8 4 5 8
pins g8 0 0 4 4
g9 3 3 15
pins g9 0 3 3 1
g10 2 3 2
pins g10 0 2 0 1
g11 2 5 3
pins g11 0 1 2 1
wire_delay 2
wire g1.p3 g2.p1
wire g2.p2 g3.p1
wire g3.p3 g4.p2
wire g4.p3 g5.p2
wire g5.p4 g6.p1
wire g6.p2 g7.p1
wire g3.p2 g8.p1
wire g4.p1 g11.p1
wire g9.p2 g5.p3
wire g10.p1 g5.p1
```

Input

```
bounding_box 23 13
critical_path g9.p1 g9.p2 g5.p3 g5.p4 g6.p1 g6.p2 g7.p1 g7.p2
critical_path_delay 36
g1 0 8
g2 2 7
g3 5 6
g4 9 5
g5 14 4
g6 20 8
g7 22 9
g8 5 1
g9 14 1
g10 14 10
g11 9 0
```

OUTPUT



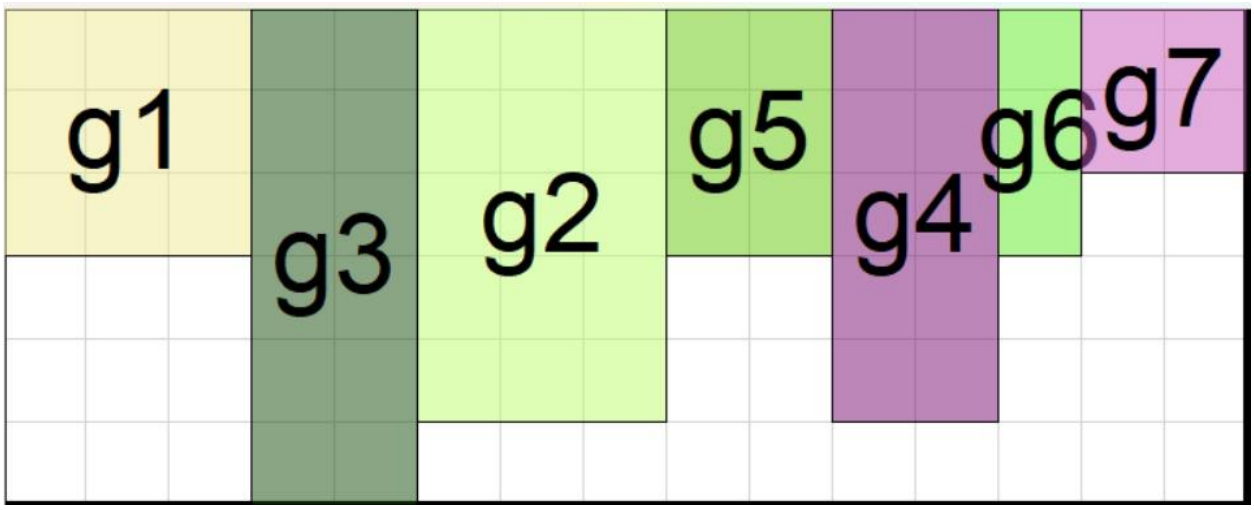
Testcase 2:

```
g1 3 3 1
pins g1 0 1 0 4 3 3 3 1 3 0
g2 3 5 3
pins g2 0 3 3 3
g3 2 6 4
pins g3 0 1 0 4 0 5 2 4 2 5
g4 2 5 1
pins g4 0 3 0 2 0 4 2 0 2 5
g5 2 3 3
pins g5 0 2 0 1 0 3 2 0
g6 1 3 1
pins g6 0 3 1 2
g7 2 2 1
pins g7 0 1 2 1
wire_delay 2
wire g1.p4 g3.p2
wire g3.p4 g2.p1
wire g2.p2 g5.p2
wire g5.p4 g4.p2
wire g4.p5 g6.p1
wire g6.p2 g7.p1
```

Input

```
bounding_box 15 6
critical_path g1.p1 g1.p4 g3.p2 g3.p4 g2.p1 g2.p2 g5.p2 g5.p4 g4.p2 g4.p
5 g6.p1 g6.p2 g7.p1 g7.p2
critical_path_delay 14
g1 0 3
g2 5 1
g3 3 0
g4 10 1
g5 8 3
g6 12 3
g7 13 4
```

Output



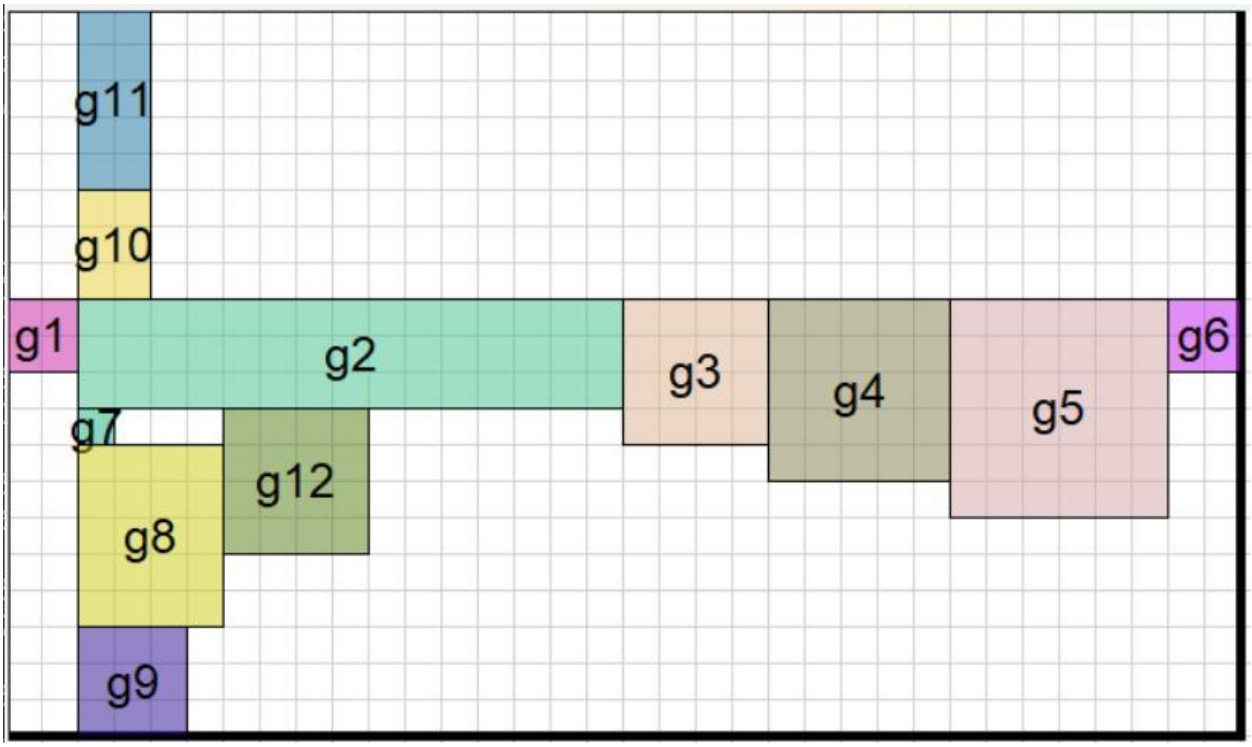
Testcase 3:

```
g1 2 2 1
pins g1 0 0 0 1 2 2
g2 15 3 2
pins g2 0 3 15 0 15 2 15 3
g3 4 4 1
pins g3 0 1 0 4 4 3 4 2
g4 5 5 1
pins g4 0 3 0 1 0 4 5 0 5 4
g5 6 6 3
pins g5 0 6 0 1 0 3 6 4
g6 2 2 1
pins g6 0 0 2 1
g7 1 1 1
pins g7 0 0 1 0 1 1
g8 4 5 8
pins g8 0 0 4 4
g9 3 3 15
pins g9 0 3 3 1
g10 2 3 2
pins g10 0 2 0 1
g11 2 5 3
pins g11 0 1 2 1
g12 4 4 3
pins g12 0 1 4 1
wire_delay 2
wire g1.p3 g2.p1
wire g2.p2 g3.p1
wire g3.p3 g4.p2
wire g4.p3 g5.p2
wire g5.p4 g6.p1
wire g2.p3 g7.p1
wire g7.p2 g8.p1
wire g8.p2 g9.p1
wire g7.p3 g10.p1
wire g10.p2 g11.p1
wire g2.p4 g12.p1
```

Input

```
bounding_box 34 20
critical_path g1.p1 g1.p3 g2.p1 g2.p3 g7.p1 g7.p2 g8.p1 g8.p2 g9.p1 g9.p
2
critical_path_delay 91
g1 0 10
g2 2 9
g3 17 8
g4 21 7
g5 26 6
g6 32 10
g7 2 8
g8 2 3
g9 2 0
g10 2 12
g11 2 15
g12 6 5
```

Output



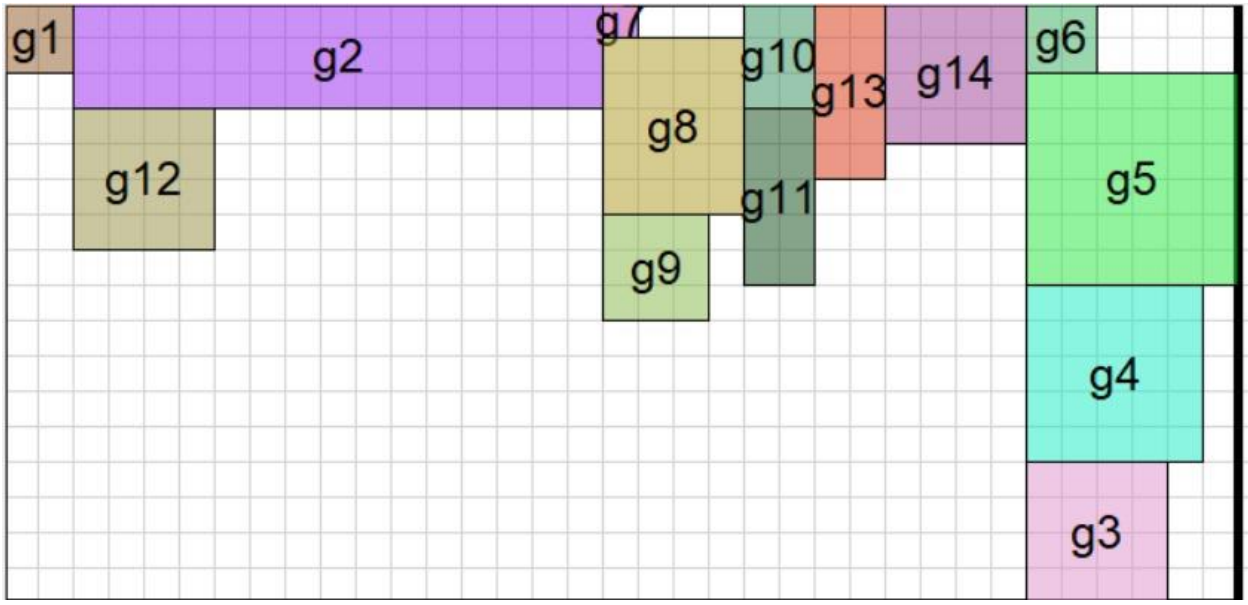
Testcase 4:

```
g1 2 2 1
pins g1 0 0 0 1 2 2
g2 15 3 2
pins g2 0 3 15 0 15 2 15 3
g3 4 4 1
pins g3 0 1 0 4 4 3 4 2
g4 5 5 1
pins g4 0 3 0 1 0 4 5 0 5 4
g5 6 6 3
pins g5 0 6 0 1 0 3 6 4
g6 2 2 1
pins g6 0 0 0 2 2 1
g7 1 1 1
pins g7 0 0 1 0 1 1
g8 4 5 8
pins g8 0 0 4 4
g9 3 3 15
pins g9 0 3 3 1
g10 2 3 2
pins g10 0 2 0 1 2 1
g11 2 5 3
pins g11 0 1 2 1
g12 4 4 3
pins g12 0 1 4 1
g13 2 5 3
pins g13 0 1 2 1
g14 4 4 3
pins g14 0 1 4 1
wire_delay 2
wire g1.p3 g2.p1
wire g2.p2 g3.p1
wire g3.p3 g4.p2
wire g4.p3 g5.p2
wire g5.p4 g6.p1
wire g2.p3 g7.p1
wire g7.p2 g8.p1
wire g8.p2 g9.p1
wire g7.p3 g10.p1
wire g10.p2 g11.p1
wire g2.p4 g12.p1
wire g10.p3 g13.p1
wire g13.p2 g14.p1
wire g14.p2 g6.p2
```

Input

```
bounding_box 35 17
critical_path g1.p1 g1.p3 g2.p1 g2.p2 g3.p1 g3.p3 g4.p2 g4.p3 g5.p2 g5.p
4 g6.p1 g6.p3
critical_path_delay 91
g1 0 15
g2 2 14
g3 29 0
g4 29 4
g5 29 9
g6 29 15
g7 17 16
g8 17 11
g9 17 8
g10 21 14
g11 21 9
g12 2 10
g13 23 12
g14 25 13
```

Output



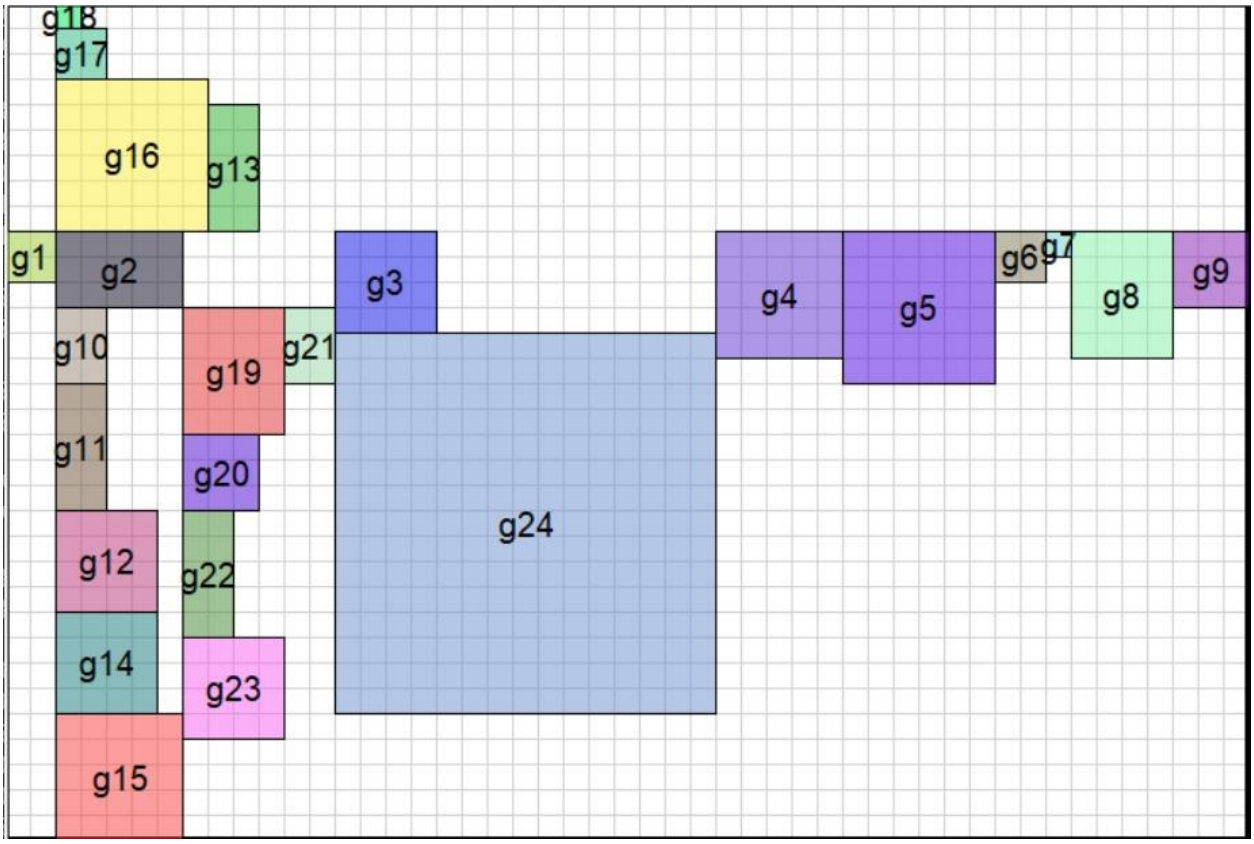
Testcase 5:

```
g1 2 2 1
pins g1 0 0 0 1 2 2
g2 5 3 2
pins g2 0 3 5 0 5 2 5 3 0 0
g3 4 4 1
pins g3 0 1 0 4 4 3 4 2 0 0
g4 5 5 1
pins g4 0 3 0 1 0 4 5 0 5 4
g5 6 6 3
pins g5 0 6 0 1 0 3 6 4
g6 2 2 1
pins g6 0 0 0 2 2 1
g7 1 1 1
pins g7 0 0 1 0 1 1
g8 4 5 8
pins g8 0 0 4 4
g9 3 3 15
pins g9 0 3 3 1
g10 2 3 2
pins g10 0 2 0 1 2 1
g11 2 5 3
pins g11 0 1 2 1
g12 4 4 3
pins g12 0 1 4 1 0 0
g13 2 5 3
pins g13 0 1 2 1
g14 4 4 3
pins g14 0 1 4 1
g15 5 5 1
pins g15 0 3 0 1 0 4 5 0 5 4
g16 6 6 3
pins g16 0 6 0 1 0 3 6 4
g17 2 2 1
pins g17 0 0 0 2 2 1
g18 1 1 1
pins g18 0 0 1 0 1 1
g19 4 5 8
pins g19 0 0 4 4
g20 3 3 15
pins g20 0 3 3 1 0 0
g21 2 3 2
pins g21 0 2 0 1 2 1
g22 2 5 3
pins g22 0 1 2 1
g23 4 4 3
pins g23 0 1 4 1
g24 15 15 3
pins g24 0 1 2 1
wire_delay 2
wire g1.p3 g2.p1
wire g2.p2 g3.p1
wire g3.p3 g4.p2
wire g4.p3 g5.p2
wire g5.p4 g6.p1
wire g6.p3 g7.p1
wire g7.p2 g8.p1
wire g8.p2 g9.p1
wire g2.p3 g10.p1
wire g10.p2 g11.p1
wire g11.p2 g12.p1
wire g12.p2 g13.p1
wire g12.p3 g14.p1
wire g14.p2 g15.p1
wire g2.p4 g16.p1
wire g16.p2 g17.p1
wire g17.p2 g18.p1
wire g2.p5 g19.p1
wire g19.p2 g20.p1
wire g20.p2 g21.p1
wire g20.p3 g22.p1
wire g22.p2 g23.p1
wire g3.p3 g24.p1
```

Input

```
bounding_box 49 33
critical_path g1.p1 g1.p3 g2.p1 g2.p2 g3.p1 g3.p3 g4.p2 g4.p3 g5.p2 g5.p4
6.p1 g6.p3 g7.p1 g7.p2 g8.p1 g8.p2 g9.p1 g9.p2
critical_path_delay 101
g1 0 22
g2 2 21
g3 13 20
g4 28 19
g5 33 18
g6 39 22
g7 41 23
g8 42 19
g9 46 21
g10 2 18
g11 2 13
g12 2 9
g13 8 24
g14 2 5
g15 2 0
g16 2 24
g17 2 30
g18 2 32
g19 7 16
g20 7 13
g21 11 18
g22 7 8
g23 7 4
g24 13 5
```

Output


g18
g17
g16
g13
g1
g2
g3
g4
g5
g6
g7
g8
g9
g10
g19
g21
g11
g20
g24
g12
g22
g14
g23
g15

Testcase 6:

```
g1 2 2 1
pins g1 0 0 0 1 2 2
g2 5 3 2
pins g2 0 3 5 0 5 2 5 3 0 0
g3 4 4 1
pins g3 0 1 0 4 4 3 4 2 0 0
g4 5 5 1
pins g4 0 3 0 1 0 4 5 0 5 4
g5 6 6 3
pins g5 0 6 0 1 0 3 6 4
g6 2 2 1
pins g6 0 0 0 2 2 1
g7 1 1 1
pins g7 0 0 1 0 1 1
g8 4 5 8
pins g8 0 0 4 4
g9 3 3 15
pins g9 0 3 3 1
g10 2 3 2
pins g10 0 2 0 1 2 1
g11 2 5 3
pins g11 0 1 2 1
g12 4 4 3
pins g12 0 1 4 1 0 0
g13 2 5 3
pins g13 0 1 2 1
g14 4 4 3
pins g14 0 1 4 1
g15 5 5 1
pins g15 0 3 0 1 0 4 5 0 5 4
g16 6 6 3
pins g16 0 6 0 1 0 3 6 4
g17 2 2 1
pins g17 0 0 0 2 2 1
g18 1 1 1
pins g18 0 0 1 0 1 1
g19 4 5 8
pins g19 0 0 4 4
g20 3 3 15
pins g20 0 3 3 1 0 0
g21 2 3 2
pins g21 0 2 0 1 2 1
g22 2 5 3
pins g22 0 1 2 1
g23 4 4 3
pins g23 0 1 4 1
g24 15 15 3
pins g24 0 1 2 1
wire_delay 2
wire g1.p3 g2.p1
wire g2.p2 g3.p1
wire g3.p3 g4.p2
wire g4.p3 g5.p2
wire g5.p4 g6.p1
wire g6.p3 g7.p1
wire g7.p2 g8.p1
wire g8.p2 g9.p1
wire g2.p3 g10.p1
wire g10.p2 g11.p1
wire g11.p2 g12.p1
wire g12.p2 g13.p1
wire g12.p3 g14.p1
wire g14.p2 g15.p1
wire g2.p4 g16.p1
wire g16.p2 g17.p1
wire g17.p2 g18.p1
wire g2.p5 g19.p1
wire g19.p2 g20.p1
wire g20.p2 g21.p1
wire g20.p3 g22.p1
wire g22.p2 g23.p1
wire g3.p3 g24.p1
wire g24.p2 g3.p4
```

Input

```
Loop found, plz clarify doubt
```

Output