REPORT OF SOFTWARE ASSIGNMENT-3

MEMBERS- AYUSH KUMAR-2023CS10600

VIRAAJ NAROLIA-2023CS10552

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 co-ordinates) 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.

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 minimised.

Algorithm of our critical path calculator:

We are using sort of Dynamic Programming to calculate the critical path delay. For each gate we are storing the critical path and critical path delay to go to that gate from all possible primary inputs as its attribute. The input parameter of our cost function is a gate. We first iterate on all the left pins of this gate. If the left pin is connected to some gate g'(say) then we calculate the delay to reach that pin. We do this by adding the delay to reach gate g' (here we are using Recursion) and the wire delay to reach that left pin (using semi perimeter method) and we repeat this for all the left pins and then maximum of it and then add the gate delay of that particular gate. This will be the critical path delay to reach that gate from all possible primary inputs.

After calculating critical path delay for all possible primary outputs, we will take maximum among them.

Time Complexity Analysis:

O(p) is for critical path delay where "p" is total number of pins. In our algorithm for calculating critical path delay we are visiting every gate and storing it for each gate so, till now the complexity becomes O(gates) and then for each gate we are fetching previous delays for each pin which comes in O(1) and hence, complexity for fetching previous delays will be O(pins). So, the overall complexity for wire delay calculator will be O(gates+pins) which is indeed O(pins).

Algorithm for semi perimeter:

For one cluster we are first finding a list of useful pins (means that they have at least one wire associated with them) and then we are applying DFS on first pin to find the group of pins interconnected including first pin and mark those pins as visited and then we are increasing our pointer to second pin. If it is marked as visited it means that it is a part of some previously found group of pins, so we skip this iteration. But if it is not marked as visited then it means that is a part of some new pin group, so we apply DFS on this pin to find all the interconnected pins including this pin too. We continue to do this until the pointer reaches the end of the array of useful pins.

In this way we find an array of pin clusters and for each pin cluster we find minx (minimum x coordinate of any pin), miny (minimum y coordinate of any pin), maxx (maximum x coordinate of any pin) and maxy (maximum y coordinate of any pin. Then the semi perimeter will be (maxx-minx) + (maxy-miny).

Time Complexity Analysis:

We are applying search on List of used pins to form pin clusters. In this we are starting from a pin1 and then going to every pin connected to pin1 and we mark the pin1 as visited and then applying recursion on every connected gate. So, the worst-case scenario will be when all the pins are interconnected (only one cluster) so in this case we will be visiting every pin p times (p=no. of pins) and there are p pins, so complexity will be O(p^2).

Algorithm 1:

Step 1: First, we form the clusters of interconnected gates. By interconnected gates we can explore every gate of that cluster and starting from any gate.

Step2: Then we applied our different algorithms on each such cluster and finally placed all such clusters side by side.

Step3: Sort the rectangles in multiple ways based on number of gates.

Step4: Take a grid of initially 1x1 size. This will represent our bin and we initialise this with False representing that it has not yet filled at that cell.

Step5: Start iterating in grid for y from 0 to length of grid and for each y iterate for X from 0 to length of grid. And for each pair of (x,y) check if you can pace the rectangle with its bottom left corner on (x,y). If you can then place it and break the nested loop. If not then continue iterating. If after nested loops you have not placed your rectangle it means that your grid is not of appropriate size so, increase the dimensions of your grid and then place the rectangle. Continue this process until all rectangles are placed.

Time complexity analysis:

Step1: For cluster formation:

In this we are starting from a gate1 and then going to every gate connected to gate1 and we mark the gate1 as visited and then applying recursion on every connected gate. So, the worst-case scenario will be when all the gates are interconnected (only one cluster) so in this case we will be visiting every gate n times (n=no. of gates) and there are n gates, so complexity will be O(n^2)

O(p^2) for pin cluster formation.

Step2: Sorting Rectangles:

- Time Complexity: O(nlogn)
- Explanation: The algorithm starts by sorting the rectangles. Sorting n rectangles takes O(nlogn) time.

Step3: Placing Rectangles:

- O(n×N^2×width×height)
  where NxN is the final grid size after all placements
  n- for number of rectangles in a cluster
  N^2- for iterating in grid (taking worst case scenario)
  Width, height- for placing a rectangle in the grid. Taking max height and width.
  nt- Total Number of gates
- O(p) is for critical path delay where "p" is total number of pins. In our algorithm for calculating critical path delay we are visiting every gate and storing it for each gate so, till now the complexity becomes O(gates) and then for each gate we are fetching previous delays for each pin which comes in O(1) and hence, complexity for fetching previous delays will be O(pins). So, the overall complexity for wire delay calculator will be O(gates+pins) which is indeed O(pins).

Overall time complexity- O (p^2+n x N^2 x width x height x C+nt^2 +p)

nt^2 is for cluster formation.

C is number of clusters.

p is total number of pins.

Intuition: We are placing gates in a way such that it first checks the right edge of so formed bounding box from bottom to top and then checks the top edge of so formed bounding box from left to right which ensures that interconnected gates are placed as close as possible to minimize the wire length which ultimately reduces the critical path delay. In this algorithm we are using different ways of sorting parameters which give different answers in different testcases.

Analysis:

Even though its complexity is a bit high, it provides better and efficient solution for minimizing the wire length. So, we will use this algorithm when number of gates are less than 700. We are placing the gates that are connected to each other as close as possible which ultimately minimizes the wire length used to connect the gate pins which reduces the critical path delay. This algorithm will take a huge amount of time if we use this for more than 700 gates, but it will provide better result than the second algorithm that we would use when the number of gates is greater than 700.

ALGORITHM 2:

Step 1: First, we form the clusters of interconnected gates. By interconnected gates we mean that we can explore every gate of that cluster.

Step2: Then we applied our different algorithms on each such cluster and finally placed all such clusters side by side.

Step3: Sort the rectangles in order of increasing difference of gate delays and number of right side connections of that gate and take a bin of width 'w'.

Step4: Place all rectangles of width>w/2 one upon another like a stack and let ho be the height of the stack.

Step5: By taking ho as a shelf place further rectangles on this until width of bin is reached.

Step6: Draw a line at width w/2 and let hl be the maximum height in left side (i.e. 0<=width<=w/2) and hr be the maximum height in right side (i.e. w/2<width<=w).

Step7: If hl<=hr. Take new level to be hl and place further rectangles on this level and only in left side. Else If hr<hl. Take new level to be and place further rectangles on this level and only in right side.

Step8: Update your hl and hr and again repeat until all rectangles are placed.

We are minimizing the wire lengths since the gate delays are small.

Time Complexity Analysis:

Step1: For cluster formation:

In this we are starting from a gate1 and then going to every gate connected to gate1 and we mark the gate1 as visited and then applying recursion on every connected gate. So, the worst-case scenario will be when all the gates are interconnected (only one cluster) so in this case we will be visiting every gate n times (n=no. of gates) and there are n gates, so complexity will be O(n^2).

O(p^2) for pin cluster formation.

Step2: Iterating from maximum width of a gate to sum of widths of all gates:

- Time Complexity: O(s) where s is the sum of widths of all gates except the maximum width.
- Explanation: The loop runs for s iterations, so the time complexity would be O(s).

Step3a: Sorting Rectangles:

- Time Complexity: O(nlogn)
- Explanation: Sorting n rectangles takes O(nlogn) time.

Step 3b: Placing Rectangles in Layers:

- Time Complexity: O(n×m)
- Explanation: After sorting, the algorithm places each rectangle into the first layer where it fits. If it doesn't fit in the current layer, a new layer is created. In the worst case, placing each rectangle might require checking every existing layer. If there are m layers, the worst-case time complexity for placement is O(n×m)
- However, typically m (the number of layers) is much smaller than n, making the placement step effectively linear, O(n), in many practical scenarios.

Step 4: Calculating critical path delay

- O(p) is for critical path delay where "p" is total number of pins. In our algorithm for calculating critical path delay we are visiting every gate and storing it for each gate so, till now the complexity becomes O(gates) and then for each gate we are fetching previous delays for each pin which comes in O(1) and hence, complexity for fetching previous delays will be O(pins). So, the overall complexity for wire delay calculator will be O(gates+pins) which is indeed O(pins).

Combining all the steps, the overall time complexity of this algorithm is:

- Total Time Complexity: O(p^2+s*(nlogn+n×m) +p)

Given that the number of layers m is generally much smaller than the number of rectangles n, the sorting step O(nlogn) often dominates, resulting in the overall time complexity:

- Overall Time Complexity: O(p^2+s*nlogn*C + nt^2)

  C is number of clusters formed.

  P Is total number of pins.

  nt^2 is for cluster formation.

  Where nt is the Total number of gates.

Intuition:

The intuition behind using this algorithm for large number of gates is that there will be a greater number of clusters, this algorithm would pack them compactly which will eventually reduce the wire length and ultimately the critical path delay.

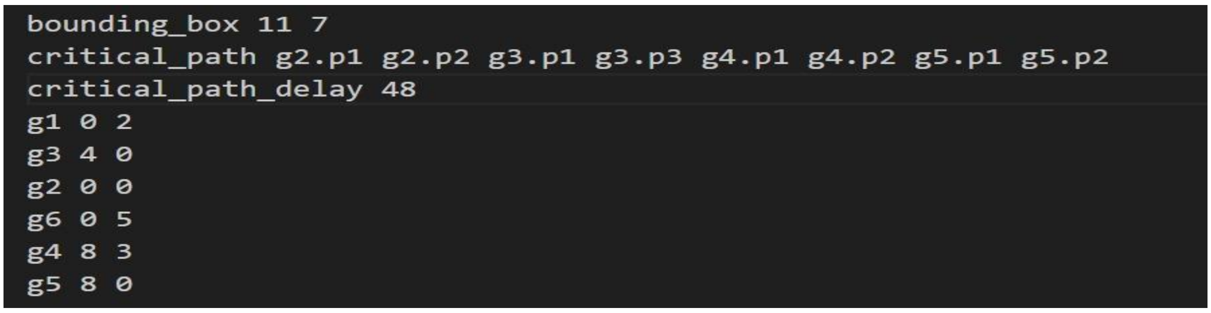
Analysis:

This algorithm gives faster results in a large number of gates, but it is likely to give less efficient results in a smaller number of gates. It is time efficient and would run in less than 5 minutes even when the number of gates is 1000.

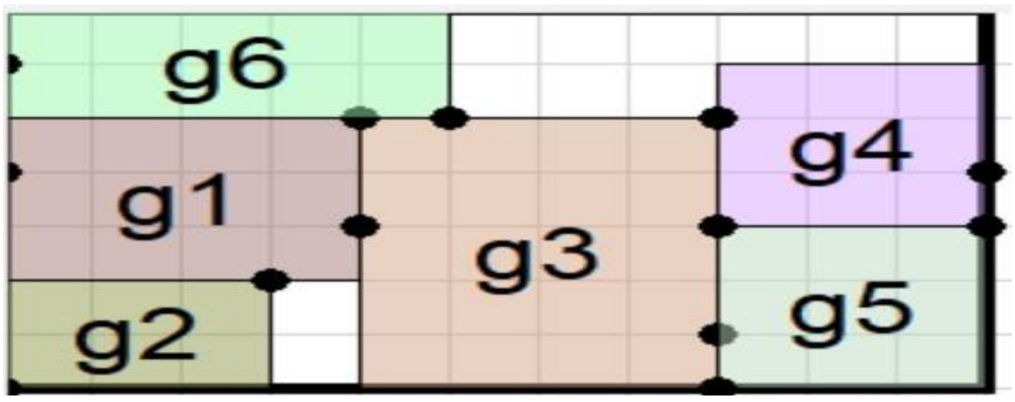
Sample Test Cases:

For sample test Case 1:

Our Critical path delay is 48.

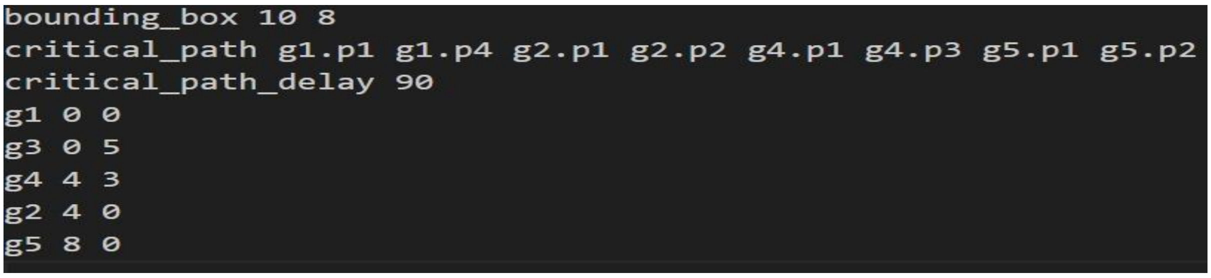


```
bounding_box 11 7
critical_path g2.p1 g2.p2 g3.p1 g3.p3 g4.p1 g4.p2 g5.p1 g5.p2
critical_path_delay 48
g1 0 2
g3 4 0
g2 0 0
g6 0 5
g4 8 3
g5 8 0
```

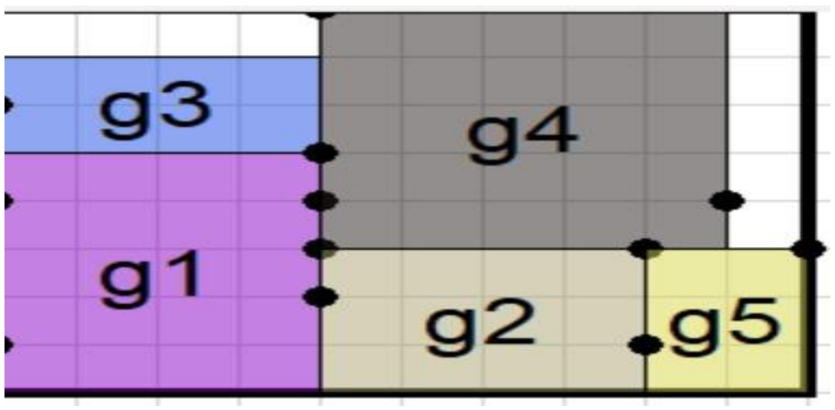


For sample test case2:

Our Critical path delay is 90.



```
bounding_box 10 8
critical_path g1.p1 g1.p4 g2.p1 g2.p2 g4.p1 g4.p3 g5.p1 g5.p2
critical_path_delay 90
g1 0 0
g3 0 5
g4 4 3
g2 4 0
g5 8 0
```



Custom Test Cases

Custom_input_1 has 2 gates connected with a single wire(gate1.delay=5, gate2.delay=3). This test case is important to verify that the critical path delay should be equal to sum of individual gate delays and not depend on the wire delay as the length of wire will come out to be zero for this case. Our output is 8 which is indeed equal to the sum of individual gate delays.

Custom_input_2 has 50 clusters with 2 gates (gate1.delay=gate2.delay=1) each. This testcase is important to verify that our algorithm effectively combines the cluster, and the critical path delay will be $\max_{1<=i<=50}(xi)$, where xi is the critical path delay for ith cluster. Our algorithm outputs 2 as the critical path delay.

Custom_input_3 has 3 gates which are in a loop (gate1 to gate2 and gate2 to gate3 and gate3 to gate1). This testcase is important to verify that our algorithm raises an exception when a loop is present in the gates. The above testcase does not have a primary input.

Custom_input_4 has 4 gates to detect a loop but in this testcase the difference is that the gate which has primary input and primary output, both of which are not present in the loop.

Custom_input_5 has 100 gates with a loop to verify that our algorithm for loop detection also works on large testcases.

Custom_input_6 has 20 gates. The significance of this testcase is that the number of wires is large (for this test case 140) as compared to number of gates so the critical path delay depends largely on the wire delay (for this test case 4). Our algorithm outputs 15247 as the critical path delay.

Custom_input_7 has 10 gates. This testcase has wire delay 3 and gate delays are around 3, that is wire delay and gate delays are comparable. Our algorithm outputs 106 as the critical path delay.

Custom_input_8 has 1000 gates to check the time complexity of our algorithm. This is a large test case for testing the time efficiency of our algorithm and to also analyse the critical path delay in these large test cases. In this case algorithm2 works because algorithm1 would take a lot of time but algorithm2 would give less accurate results than algorithm1.

Custom_input_9 has 1000 gates to check the time complexity of our algorithm. Our algorithm outputs 540902 as the critical path delay.

Conclusion:

Our main algorithm is algorithm1 which is more accurate but takes more time, while algorithm 2 is not that accurate but runs very fast as compared to algorithm 1.

In comparatively small test cases where number of gates are less than 700, algorithm 1 gives better results also it is not that slow that is gives output around 2 to 3 minutes.

In large test cases around 1000 gates, when size of gates and number of connections are large then algorithm 1 gives better results but takes more time. While algorithm 2 gives faster and better results if the number of connections is less.

We are minimizing the wire lengths since the gate delays are small.