

Synthesis of trees of gates – Fanin Tree Embedding

A study and implementation of construction of Fanin Trees to minimize on the cost construction and keeping output delay within the provided limit

Technique Used – Dynamic Programing

By,

Pavan Rao Chickbellavangala Rahavendra - A20354970

Anand Narasimhamurthy – A20354712

Synthesis of trees of gates – Fanin Tree Embedding: A study and implementation of construction of Fanin Trees to minimize on the cost construction and keeping output delay within the provided limit

Introduction:

In modern VLSI design, on chip variation and low power design schemes greatly affect the operation and reliability of circuits. For instance, the variations on the power supply voltages (power/ground rails) lead to delay variations on the tree branches and logic paths, which degrade the chip performance.

Problem Definition : Synthesis of Trees of Gates (Please refer `RunDpWithMemoization.java` and `RunDpWithoutMemoization.java`)

Suppose there are N input signals/bits b_0, b_1, \dots, b_{n-1} and you need to build a tree of 2-input gates to compute the XOR of the bits (or, for the purposes of this problem, an AND, OR, or any other associative operator).

Signal i arrives at its input pin at time t_i . The arrival time at the output of a gate g with inputs x and y is determined by

$$t_g = \max(t_x, t_y) + d_g$$

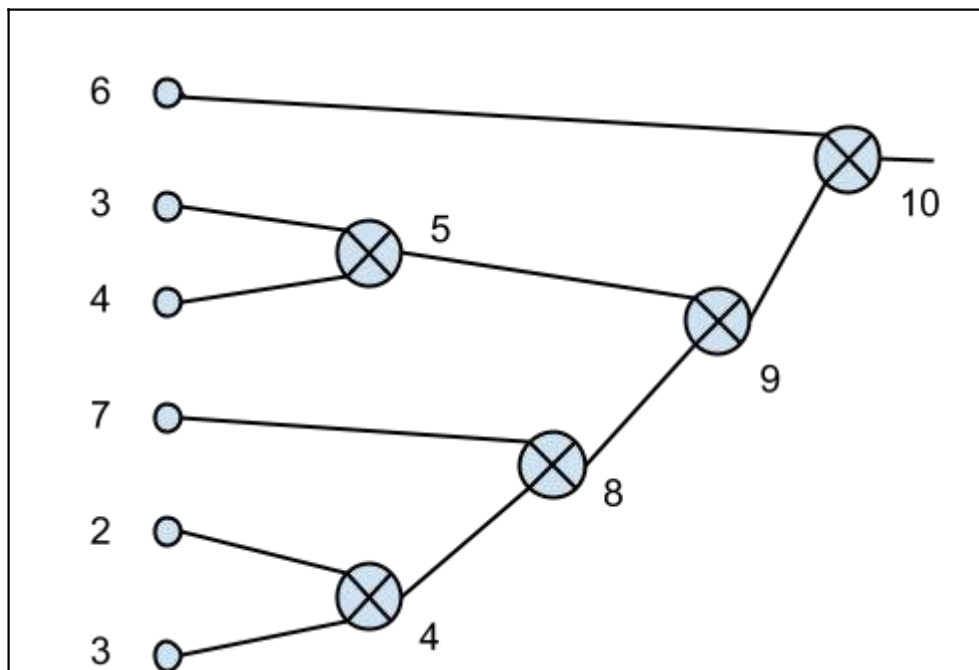
where,

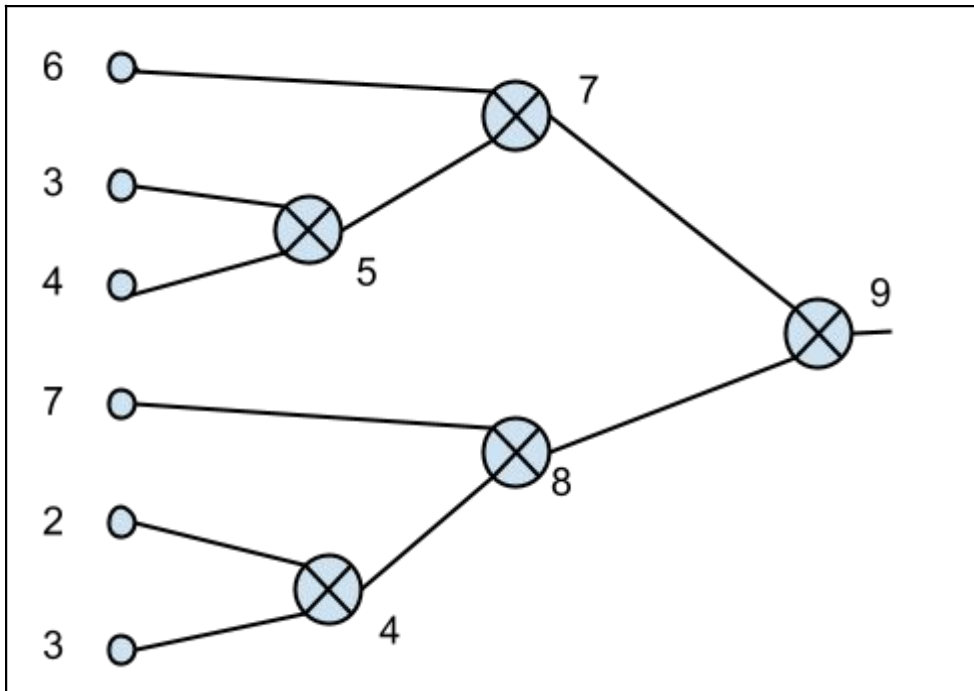
t_x and t_y are the arrival times of inputs,

x and y respectively and

d_g is the delay through the gate.

The figures below give two trees for the given input. The input pins are at the left of the diagrams and are labeled with their arrival times. The two configurations



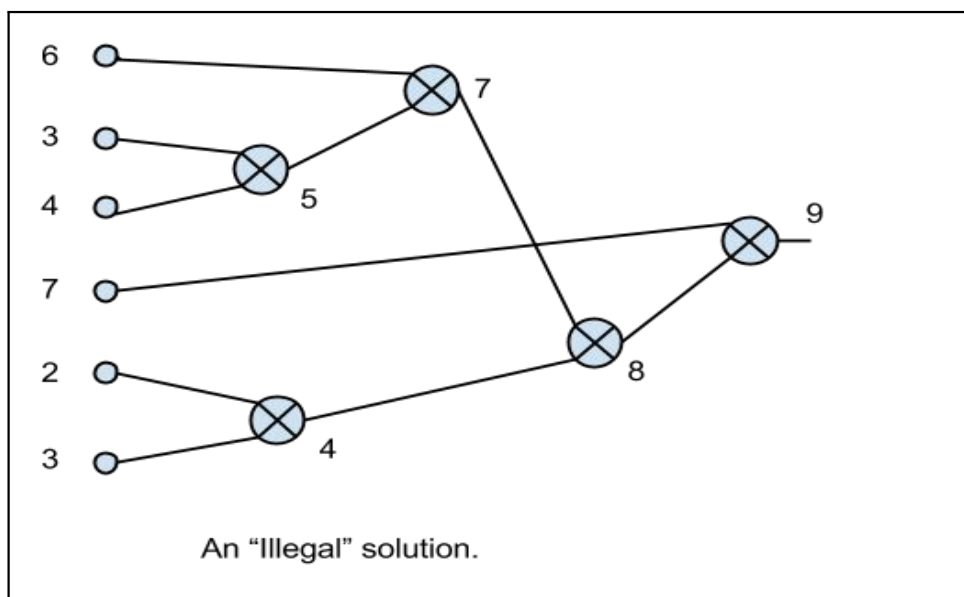


Legal Trees

Because of the given ordering of the inputs, we will not allow just any tree structure; the trees must be “consistent” with the input pin ordering:

The leaves (inputs) of every subtree must be a consecutive subsequence of length two or more from the given pin ordering.

The tree in the diagram below violates this rule.



Description of Algorithm :

Step 1) Read each line of the file which contains the time at which the input arrives.

Step 2) Input delay dg.

Step 3) For all values of k recursively call:

 If $i < j$ then

 Min delay = $\text{Min}\{\text{Max}_{i < k < j}(\text{OptimalSolution}[i, k], \text{OptimalSolution}[k+1, j]) + dg\}$

 Calculate Postfix Notation

Step 4) Return Min cost and postfix Notation at root of the tree.

Recursion:

OptimalSolution[i,j] =

| 0

i=j

| $\text{Min} (\text{Max}_{i < k < j} (\text{OptimalSolution}[i, k], \text{OptimalSolution}[k+1, j]) + dg)$

i<j

Explanation:

The algorithm uses recursion to achieve the desired result. At the leaf nodes of the tree all the arrival time of the input signals are present Optimal Solution is a function that get recursively called for all the partition index (K) value which ranges from i+1 to j-1 and returns 0 when it converges to a single node which signifies that it has hit the leaf node and can no longer be partitioned.

Complexity

When we encounter the sub problem for the first time, we compute its solution and store in the Memoization table. After that whenever we encounter the sub problem again, we simply look up in the table and return the solution. So, we are computing each of table entry once.

Hence Complexity is : $O(n^2)$.

Solution Extraction: A description of how we construct the optimal topology

1) Optimal Substructure Property:

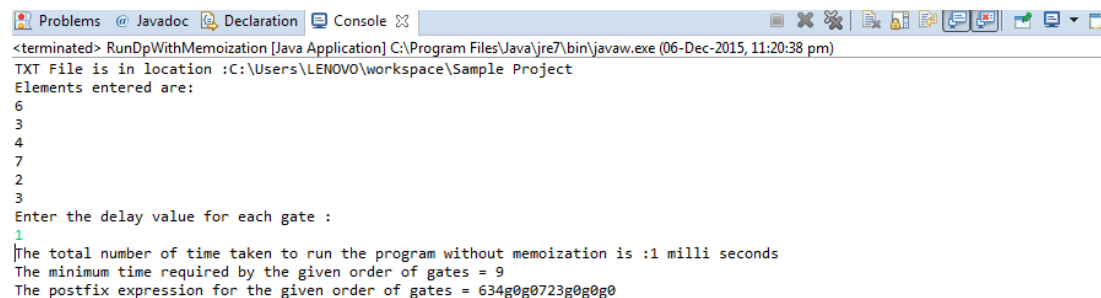
The solutions iterates through each of the possible combination of gates that can made with the given inputs and retrieves the optimal solution recursively. For n arrival times, the given combinations can be $n-1$ ways. For instance, if the given arrival times are 4 in number (ABCD), then there are 3 ways to form gates: A(BCD), (AB)CD and (ABC)D. So we divide the problem into sub problems of smaller size. Therefore, the problem has optimal substructure property and can be solved using recursion.

2) Memoization

Since a problem is divided to sub problems and the same sub problem needs to be solved multiple number of times, memorization can be used in order to avoid redundant computations. Every time a sub problem is solved for the first time, it can be stored in a table (in this case a 2D matrix). The next time same sub problem need not be resolved as the value is available in the table

Implementation Details:

Experimental data considered:



```
Problems Javadoc Declaration Console
<terminated> RunDpWithMemoization [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (06-Dec-2015, 11:20:38 pm)
TXT File is in location :C:\Users\LENOVO\workspace\Sample Project
Elements entered are:
6
3
4
7
2
3
Enter the delay value for each gate :
1
The total number of time taken to run the program without memoization is :1 milli seconds
The minimum time required by the given order of gates = 9
The postfix expression for the given order of gates = 634g0g0723g0g0g0
```

Fig1: Shows the time taken to run the program with the memoization for 6 number of leaf nodes, postfix expression and the minimum time required.

```
Problems @ Javadoc Declaration Console
<terminated> RunDpWithoutMemoization [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (06-Dec-2015, 11:22:01 pm)
TXT File is in location :C:\Users\LENOVO\workspace\Sample Project
Elements entered are:
6
3
4
7
2
3
Enter the delay value for each gate :
1
The total number of time taken to run the program without memoization is :3 milli seconds
The minimum time required by the given order of gates = 9
The postfix expression for the given order of gates = 634g0g0723g0g0g0
```

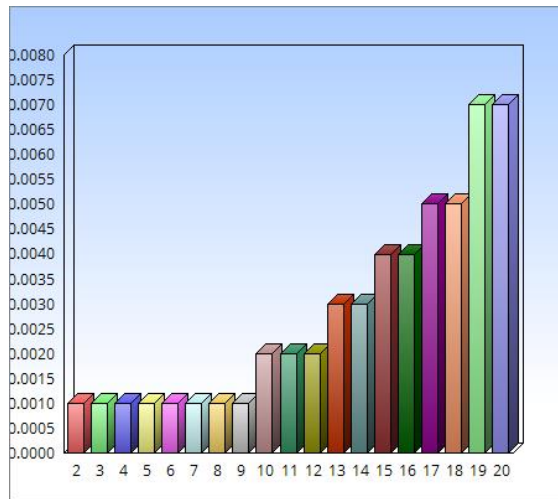
Fig2: Shows the time taken to run the program without the memoization for 6 number of leaf nodes, postfix expression and the minimum time required.

```
Problems @ Javadoc Declaration Console
<terminated> RunDpWithMemoization [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (06-Dec-2015, 11:24:46 pm)
TXT File is in location :C:\Users\LENOVO\workspace\Sample Project
Elements entered are:
2
6
3
4
2
7
5
6
2
1
3
5
6
8
3
Enter the delay value for each gate :
1
The total number of time taken to run the program without memoization is :3 milli seconds
The minimum time required by the given order of gates = 11
The postfix expression for the given order of gates = 26342g0g0g07g056g021g03g05g06g0g0g083g0g0g0
```

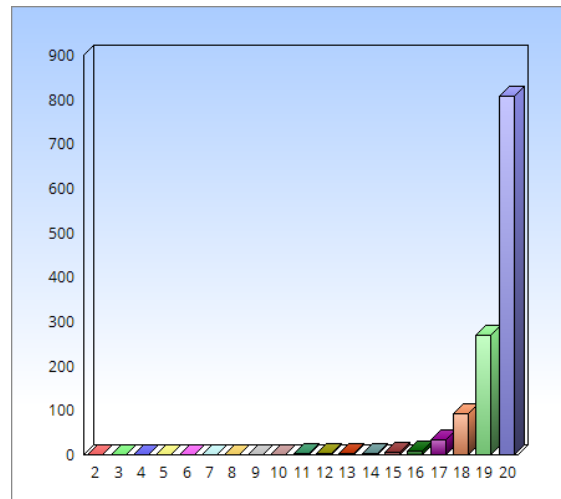
Fig3: Shows the time taken to run the program with the memoization for 15 number of leaf nodes, postfix expression and the minimum time required.

```
Problems @ Javadoc Declaration Console
<terminated> RunDpWithoutMemoization [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (06-Dec-2015, 11:26:06 pm)
TXT File is in location :C:\Users\LENOVO\workspace\Sample Project
Elements entered are:
6
3
4
7
2
3
Enter the delay value for each gate :
1
The total number of time taken to run the program without memoization is :3 milli seconds
The minimum time required by the given order of gates = 9
The postfix expression for the given order of gates = 634g0g0723g0g0g0
```

Fig4: Shows the time taken to run the program without the memoization for 15 number of leaf nodes, postfix expression and the minimum time required.



With Memoization



Without Memoization

Chart 1: Time Complexity(in milliseconds) Vs Number of leaf nodes

Problem Statement 2

In a more realistic approach there are multiple forms of gates that can be used in a given circuit. The Problem is to find an optimal plan to place the gates such that the output delay at the end of the final gate in the tree is below the tolerance value provided by the user and the cost of such tree construction is minimized.

Now there are 2 files which are given as inputs

InputOrder.TXT (File 1) – Contains the list of all the input arrival times and the total number of inputs is provided in the first line of the file.

GateInput.TXT (File 2) – Contains a library of gates which have different tradeoffs between gate delay and cost.

First line of the file which denotes the number of gates that can be considered following with the gate name, cost and their respective delays.

(User Input) – The user also inputs a value which is used as the tolerance value for the output delay beyond which the solution to the problem is not considered.

Objective: Construct a legal tree of gates which minimizes the total cost of the tree subject to the constraint that the arrival time at the output is no greater than *tolerance level* .

Description of Algorithm:

The algorithm uses recursion and Memoization to achieve the desired result. At the leaf nodes of the tree all the arrival time of the input signals are present.

Optimal Solution is a function that get recursively called for all the partition index (K) values which ranges from $i+1$ to $j-1$ and returns 0 when it converges to a single node which signifies that it has hit the leaf node and can no longer be partitioned.

At each level of the function call the result returned is entered into the memoization table and whenever there is an access to the same key values the table is referred for the output rather than making the function call.

OptimalSolution[i , j] =

| 0 i = j

| Min ((0 < x < numberOfGates (Max i<k<j (OptimalSolution[i , k], OptimalSolution[k+1 , j]) + d(x))) i < j

Where, D(x) is the delay function which retrieves the delay of a gate G(x).

Algorithm

Step 1) Read all the input files and creates variable for gates and input pins

Step 2) Accept the delay tolerance value from the user

Step 3) For all values of k recursively call:

If i < j then

Min delay = Min((0 < x < numberOfGates (Max i<k<j(OptimalSolution[i,k], OptimalSolution[k+1,j]) + d(i)))

Calculate Postfix Notation

Step 4) Return Min cost and postfix Notation at root of the tree.

Complexity:

When we encounter the sub problem for the first time, we compute its solution and store in the Memoization table. After that whenever we encounter the sub problem again, we simply look up in the table and return the solution. So, we are computing each of table entry once. The size of the table has to be “n X n X m”.

So, we consider the number of input arrival pins as “n” and the number of gates in the library of gates as “m”.

Thus, the complexity of the algorithm is given by : O(n² m) .

Solution Extraction: A description of how we construct the optimal topology.

1) Optimal Substructure Property:

The solutions iterates through each of the possible combination of gates that can made with the given inputs and retrieves the optimal solution recursively. For n arrival times, the given combinations can be n-1 ways. For instance, if the given arrival times are 4 in number (ABCD), then there are 3 ways to form gates: A(BCD), (AB)CD and (ABC)D. So we divide the problem into sub problems of smaller size. Therefore, the problem has optimal substructure property and can be solved using recursion.

2) Memoization

Since a problem is divided to sub problems and the same sub problem needs to be solved multiple number of times, memorization can be used in order to avoid redundant computations. Every time a sub problem is solved for the first time, it can be stored in a table (in this case a 3D matrix). The next time same sub problem need not be resolved as the value is available in the table. The third dimension is required when compared to the previous problem solution is to take into consideration the cost and delay factor of different gates.

Implementation Details:

Experimental data considered:

```
Console  Tasks
<terminated> Solution [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (Dec 6, 2015, 10:58:48 PM)
TXT File is in location :C:\Users\Pavan\JavaWorkspace\Dynamic Programing
Input Pin Arrival Times:
7
2
6
3
Following gates are considered to build optimal plan plan
Gate 1 with a delay of 1 and a cost of 10
Gate 2 with a delay of 2 and a cost of 5
Enter the tolerance value of the output :
13
The minimum time required by the given order of inputs and gates = 11
The minimum cost required for building the solution = 15
The postfix expression for the given order of inputs with its respective gates = 72g263g2g2
```

Fig 1: This shows the output for the above inputs and tolerance value provided by the user. For the given tolerance value a valid tree can be constructed and the topology of the tree is given by the postfix expression

```
Console  Tasks
<terminated> Solution [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (Dec 6, 2015, 11:02:15 PM)
TXT File is in location :C:\Users\Pavan\JavaWorkspace\Dynamic Programing
Input Pin Arrival Times:
7
2
6
3
Following gates are considered to build optimal plan plan
Gate 1 with a delay of 1 and a cost of 10
Gate 2 with a delay of 2 and a cost of 5
Enter the tolerance value of the output :
1
There is no possible solution for the given tolerance value
```

Fig 2: This shows the output for the above inputs and tolerance value provided by the user. For the given tolerance value no valid tree can be constructed and an appropriate message is displayed for the same.

References:

[1] Milos Hrkic, John Lillis, Giancarlo Beraudo title- “*An approach to Placement – Coupled Logic Replication*”
Topic – “*Fannin Tree*”

[2] M. Hrkić and J. Lillis, “*S-Tree: A technique for buffered routing tree synthesis*,” in Proc. 39th DAC, New Orleans, LA, Jun. 2002, pp. 578–583.

[3] A. Marquardt, V. Betz, and J. Rose, “*Timing-driven placement for FPGAs*,” in Proc. Int. Symp. Field Program. Gate Arrays, Monterey, CA, Feb. 2000, pp. 203–213.