CS401: DP Project

Bhavin Chauhan(676553751)

## Summary

The formulation 3 for the gateTree problem was implemented using java. The code can also be used for formulation 1 on supplying appropriate command line arguments. The code was tested on input size of 10 pins with gate library of 3 gates for formulation 3 and gave the correct output. The algorithm uses top-down recursion with memoization to find an optimal valid gate tree that gives minimum solution time. It then uses this structure to get time for solution and cost for each combination of gates and stores non dominated candidate solutions.

## Algorithm

The algorithm for formulation 3 has two parts: In first part, we find the structure of optimal binary tree having minimum time delay using memoization. The algorithm reads arrival times of pins from a file and stores the values in list *pins*. Let *p0,pi,...,pn-1* be arrival times of pins. The arrival time of output from a gate would be given by: *tg = max(tx, ty) + dg* where tx, ty are arrival times of input and dg is gate delay.

We can divide the entire problem into subproblems. Let opt(i,j) denote the optimal time to get solution for pins pi,....pj , i<j. For i=j, the problem becomes trivial and it will return the arrival time of *pi*. When i=j-1, we take gate delay as *max(pi,pj)+dg.* For all other i,j we split the problem into two subproblems, opt(i,k) and opt(k+1,i) where i≤k<j. We can then take max value from two subproblems and add gate delay to get final arrival time.

opt(i,j) if i=j

if i=j-1

= otherwise

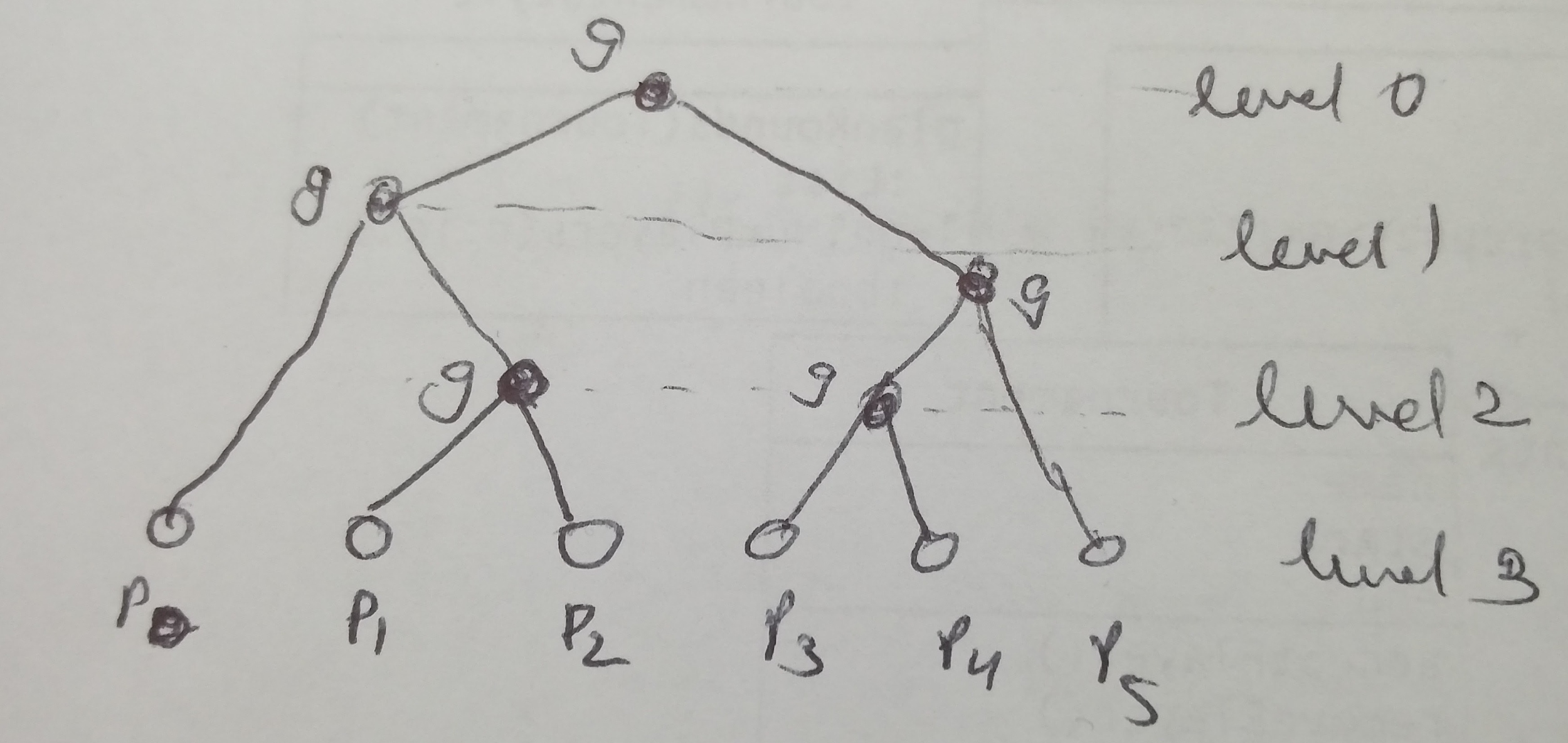
Correctness: For entire sequence to be optimal, the solution time for subproblem should be smallest. Suppose opt(i,k)=t ; if there exist t0< t thencan be greater than t0 which is contradiction. So, the subproblems need to be optimal for entire solution to be optimal.

The algorithm uses a 2 dimensional array *opt[i][j]* to memoize subproblem values. So this will have Θ(n2) solution space. There would Θ(n2) recursive call to the function and each function call will have O(n3) time complexity. So overall complexity of finding optimal gateTree would be O(n3).To keep track of optimal subsolutions, we store the value of split point in a table *c\_opt[i, j]*. Now that we have optimal gate tree, we will have to check all possible gate usage from the gate library provided in second part of algorithm. The gate library file is read and stored in list gates such that gates(2\*i) will give cost of gate and gates(2\*g+1) will give gate delay. To get all the non dominant cost, delay combination we will list out all the combinations of gates at different position and save the combination which gives smallest cost for a given time delay. This problem does not have optimal subproblem property and so conventional DP techniques cannot be used to solve this.

We will first construct a binary gate tree by calling createGateTree() function and then construct different solutions in bottom up fashion. At each level, we save the number of possible solution with different cost and time and postfix sequence of solution. For any two solution (ci,d) and (cj,d), if cj>ci then (cj,d) would not be considered for further levels. At any internal node, we will have two child nodes with different solution size.Once we find all possible nodes at level i, we solve problem for level i-1 using only valid solutions of level i. We keep on repeating this till we reach level 0 i.e root node. Now root node will have all valid combinations of delay and time.

There are O(n) pairs at any level of tree. If there are g gates in gate library, there are O(nd) solutions at any level. And there would be O(n) levels. If there are O(nd) possible solution of one child and, there would be O(nd) solution for parent node. So overall the complexity will be O(n2d).

## Example:

Input:

p0 :6.0

p1 :3.0

p2 :4.0

p3 :7.0

p4 :3.0

p5 :8.0

Gate Cost Delay

g0 5.0 1

g1 4.5 1.5

g2 3.5 3

The optimal tree obtained is as shown in figure.

At level 3: Each node will have just 1 solutions.

At level 2:There are two gate positions. The one between p1 and p2 has 3 distinct solution having <cost,delay>={ <5.0,8.0>, <4.5,5.5>, <3.5,5.0>} And the node between p3 and p4 has 3 distinct solution having <cost,delay>={ <5.0,11.0>, <4.5,8.5>, <3.5,8.0>}.

At level 1: Again there are two gate positions.

The one between p0 and (p1 p2 g) has 3 distinct valid solution with (cost,time):

* p0 p1 p2 g0 g0(10.0,12.0)
* p0 p1 p2 g0 g1(9.5,9.5)
* p0 p1 p2 g0 g2(8.5,9.0)

The node between (p3 p4 g) and p5 has 3 distinct valid solution with (cost,time) :

* p3 p4 g0 p5 g0(10.0,15.0)
* p3 p4 g0 p5 g1(9.5,12.5)
* p3 p4 g0 p5 g2(8.5,12.0)

At level 0:

Possible solutions with (cost,time) are:

p0 p1 p2 g0 g0 p3 p4 g0 p5 g0 g0(25.0,19.0) : Dominated solution : To be Ignored

p0 p1 p2 g0 g1 p3 p4 g0 p5 g0 g0(24.5,19.0) : Dominated solution : To be Ignored

*p0 p1 p2 g0 g2 p3 p4 g0 p5 g0 g0(23.5,19.0) : Non Dominated : Include in final solution*

p0 p1 p2 g0 g0 p3 p4 g0 p5 g0 g1(24.5,16.5) : Dominated solution : To be Ignored

p0 p1 p2 g0 g1 p3 p4 g0 p5 g0 g1(24.0,16.5) : Dominated solution : To be Ignored

*p0 p1 p2 g0 g2 p3 p4 g0 p5 g0 g1(23.0,16.5) : Non Dominated : Include in final solution*

p0 p1 p2 g0 g0 p3 p4 g0 p5 g0 g2(23.5,16.0) : Dominated solution : To be Ignored

p0 p1 p2 g0 g1 p3 p4 g0 p5 g0 g2(23.0,16.0) : Dominated solution : To be Ignored

*p0 p1 p2 g0 g2 p3 p4 g0 p5 g0 g2(22.0,16.0) : Non Dominated : Include in final solution*

## Output of program:

*Formulation 3:*

*Arrival time of solution:16.0*

*Cost:22.0*

*Postfix Sequence:p0 p1 p2 g0 g2 p3 p4 g0 p5 g0 g2*

*Arrival time of solution:16.5*

*Cost:23.0*

*Postfix Sequence:p0 p1 p2 g0 g2 p3 p4 g0 p5 g0 g1*

*Arrival time of solution:19.0*

*Cost:23.5*

*Postfix Sequence:p0 p1 p2 g0 g2 p3 p4 g0 p5 g0 g0*