Our goal is to develop a *Simulated Annealing* algorithm to automatically synthesize stochastic computing circuits that operate on probabilities. Such circuits receive input encoded as real numbers between 0 and 1, perform logic operations such as INVERT, AND, XOR on them and output probability values. Our Simulated Annealing engine is going to randomly generate circuits and evaluate them to find a circuit that best approximates a pre-defined function F_TARGET(X). The cost of a circuit (measure of its goodness used in the annealing engine) is essentially the sum of square of the differences between the F_TARGET value and the output of the circuit for different values of input X. In this problem, you can assume that the Simulated Annealing engine (the one that uses the Bolztman function) and the random initialization of the circuit are written already. You are only responsible for the *perturb* function, and the *cost evaluation* function.

The circuits that we use have the following properties:

Probability values: Probability values are represented using fixed-point real numbers on busses that connect nodes (also called blocks). For the purposes of solving this problem, you can ignore how exactly the values are represented in hardware.

Building blocks: a block (node) is either:

- An input X ($0 \le X \le 1$).
- A constant CONST (value $\in\{0.0, 0.1, 0.2, ..., 0.9, 1\}$).
- An inverter INV (if input is α, output=$1-\alpha$).
- An XOR (if inputs are α, β, then output= $\alpha.(1-\beta) + \beta.(1-\alpha)$).

Circuit topology: the circuit is represented by a complete balanced binary tree of depth MAX_DEPTH. A binary tree is a tree in which each node has exactly two children. A complete binary tree is perfectly balanced, i.e., all paths from the root node to leaf nodes are of equal depth. Two examples are shown below for MAX_DEPTH=3. Note that **(a)** the leaf nodes can only be constant or X. **(b)** the tree in Fig. 13.2 is a complete balanced binary tree, but some internal nodes such as INV and CONST 0.6 make their sub-trees irrelevant (nodes 4, 5, 6).

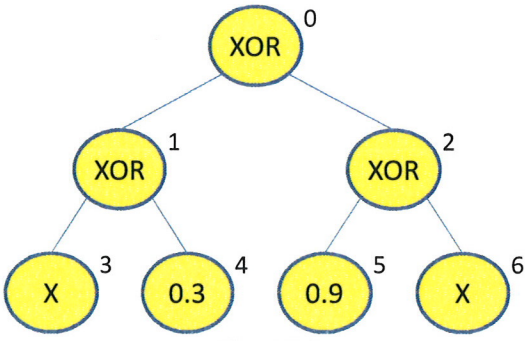


Fig. 13.1

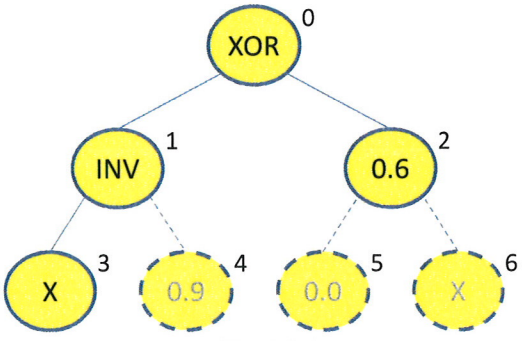


Fig. 13.2

Data structure: a complete balanced binary tree (also called a heap) can be stored in a linear array. The root node is stored at index 0, and if the nodes are stored in a breadth-first order (left sibling first), then

the left and the right children of node *i* are stored at indices *(2.i+1)* and *(2.i+2)* respectively. The small numbers next to the nodes in the above trees show their indices. Because there is such a structured relationship between a node and its two children, there is no need to explicitly store the predecessor/successor edges in a data structure.

The goal of the annealing process is to come up with a circuit of depth MAX_DEPTH that best approximates a function F_TARGET(x). We only have 11 sample points of F_TARGET for x=0.0, 0.1, 0.2, ..., 0.9, 1.0. As an example, Fig. 13.3 below shows the 11 samples and a good approximation of the function shown as the dotted curve. The dotted curve is generated using the circuit XOR (INV(X) , XOR(X,0.3)) shown in Fig. 13.4.

The value output by the circuit for X=0.1 is calculated as follows:

- INV(X)=1-0.1=0.9 (output of node 1 in Fig. 13.4)
- XOR(X,0.3)=0.1 × (1-0.3) + 0.3 × (1-0.1) = 0.1×0.7 + 0.3×0.9 = 0.34 (output of node 2 in Fig. 13.4)
- Circuit output = 0.9×(1-0.34) + 0.34×(1-0.9) = 0.9×0.66 + 0.34×0.1 = 0.628.

The value of F_TARGET(0.1)=0.59, which means at X=0.1, the square error between the circuit output and F_TARGET = $(0.628-0.59)^2$. The overall cost of this circuit is the sum of such square errors for points X=0, 0.1, 0.2, ... , 1.0.

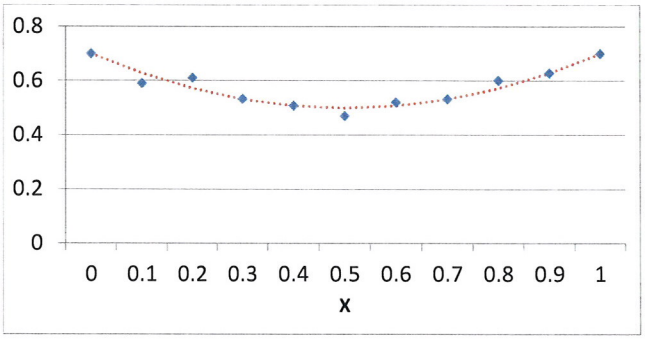
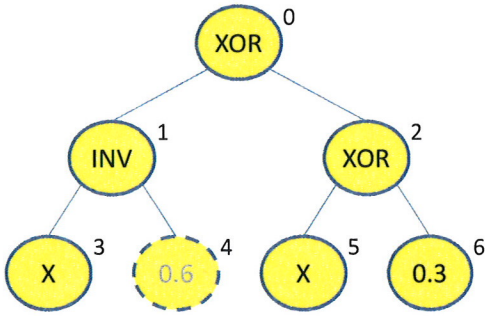


Fig. 13.3



XOR (INV(X) , XOR(X,0.3))

Fig. 13.4

Assume that the tree initialization function and the simulated annealing engine are written already. You are supposed to write the *perturbTree* and the *costOfTree* functions in the C language (or other high-level languages).

A. (0.1 points) The tree is represented as an array of size CIRCUIT_SIZE of the *node* type. Define CIRCUIT_SIZE as a function of MAX_DEPTH.

B. (0.2 point) A crucial calculation used throughout the algorithm is how to determine if a node belongs to the last row of the tree. In the examples of Fig. 13.1 and 13.2 where MAX_DEPTH=3, nodes whose indices are greater than or equal to 3 belong to the last row. You have to come up with a general formula that can determine the index of the first node in the last row for any value of MAX_DEPTH. In addition to the equation, show the value for MAX_DEPTH=4 and

MAX_DEPTH=5.

C. (0.2 points) What data structure do you use to represent a *node*? (remember that for constants, you have to also store the constant value). Also define the circuit as an array of these nodes, and remember, no explicit storage of connections between nodes is needed.

D. (1.5 point) Write the *perturbTree* function. *perturbTree* randomly picks a node in the tree and changes it. The following restrictions apply:
   - If the node is constant, then there is a 50% chance that the node type does not change, but a new constant value is chosen.
   - If the node belongs to the last row of the tree, it can only switch between CONST and X.

E. (2 points) Write the *costOfTree* function. The function should return the mean square error of the function represented by the tree and F_TARGET for the 11 sample points. Your function should not evaluate nodes that are irrelevant (e.g., nodes 4, 5, and 6 in Fig. 13.2).