Data Structures and Algorithms: Homework #4

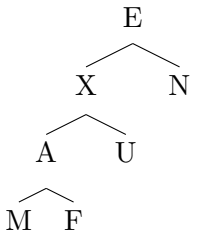
Due on May 15, 2015 at 16:20

Instructors Hsuan-Tien Lin, Roger Jang

Tim Liou (b03902028)

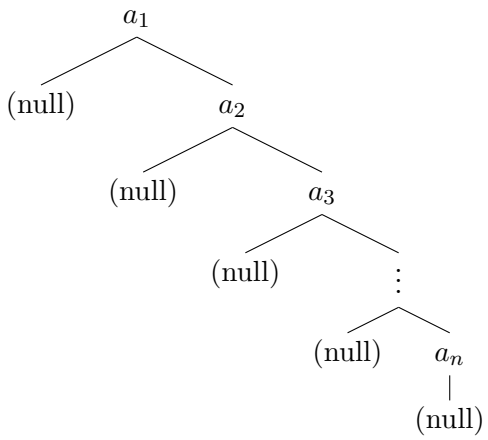
4.1 Tree

(1) Do Exercise R-7.15 of the textbook.



(2) Do Exercise R-7.24 of the textbook.

(a) We can easily find the level of right child is larger than its parent and its sibling. It implies that it will obtain the largest level if all the nodes (except the root) is its parent's right child.



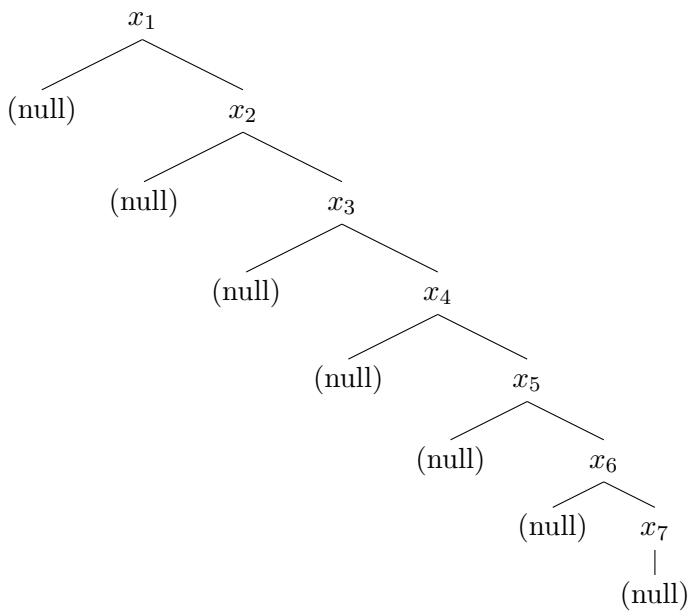
Note that $f(a_n) = 2f(a_{n-1}) + 1$, and we can find

$$\begin{aligned}
&\Rightarrow f(a_n) + 1 = 2(f(a_{n-1}) + 1) \\
&\Rightarrow f(a_n) + 1 = 2^2(f(a_{n-2}) + 1) = \cdots = 2^{n-1}(f(a_1) + 1) \\
&\Rightarrow f(a_n) + 1 = 2^n \\
&\Rightarrow f(a_n) = 2^n - 1
\end{aligned}$$

We find the upper bound of level for a binary tree with n nodes. That is, for every node v of T,

$$f(v) \leq 2^n - 1$$

(b)



There are 7 nodes in this tree. The upper bound of the level is $2^7 - 1 = 127$. We can easily find $f(x_7) = 127$ by the formula in (a). The node x_7 attains the above upper bound on f in this tree.

(3) Do Exercise C-7.3 of the textbook.

$post(v) = pre(v) + desc(v) - depth(v)$

For the root, with post-order, we will visit its descendents first, then visit it, but with pre-order, we will visit it first, then visit its descendents.

(4) Do Exercise C-7.7 of the textbook with pseudo code.

```
1:  function PRINT(root, NumOfIndent)
2:      if root has children then
3:          print out the content in root and '(' with NumOfIndent indent.
4:          for all i ← children do
5:              PRINT(i, NumOfIndent + 1)
6:          end for
7:          print out ')' with NumOfIndent indent
8:      else
9:          print out the content in root with NumOfIndent indent.
10:     end if
11:     return
12: end function
```

Algorithm 1: Print a tree with indented parenthetic representation

Call this function by **Print**$(root, 0)$.

4.2 Decision Tree

(1) Using the property that the v_m values are sorted, describe an $O(M)$ algorithm to calculate the best threshold.

for all $i \leftarrow$ data **do**
  if The value of this feature is bigger than $threshold_j$ **then**
    initial $a_{j+1}\text{Y}b_{j+1}\text{N}$ with the value in $a_j\text{Y}b_j\text{N}$
    $j \leftarrow j + 1$
  else
    Update $a_j\text{Y}b_j\text{N}$.
  end if
end for
for all $i \leftarrow$ thresholds **do**
  calculate the confusion by using $a_i\text{Y}b_i\text{N}$ and $(TotalY - a_i)\text{Y}(TotalN - b_i)\text{N}$.
  check if this confusion is smaller than previous.
end for

Algorithm 2: Calculate the best threshold with an $O(M)$ algorithm

The purpose of first **for loop** is to calcuate and store how many Yes and No below each threshold. We will know the values of TotalY and TotalN because we can get them when we encounter the biggest thresold $v_M + 1$.

(2) Implement the decision tree algorithm.

Code part.

(3) Illustrate (ideally with drawing) the internal data structure you use to represent the decision tree in your memory. Please be as precise as possible.

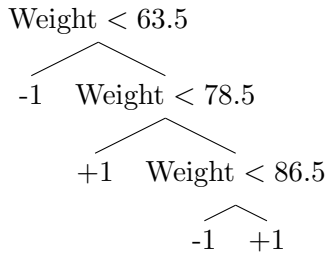
I use this data structure to represent the decision tree.

```
1  struct BranchChoice {
2      int BestFeature;
3      double BestTotalConfusion;
4      double BestThresholdID;
5      double BestThreshold;
6  };
7  struct DTree {
8      BranchChoice Choice;
9      DTree *left;
10     DTree *right;
11 };
```

(4) Teach your decision tree with the following examples to learn a function f with $\epsilon = 0$. Draw

the tree you get.

Left side means Yes and right side means No.



(5) Construct your own data set with at least 2 numerical factors and at least 6 examples. Teach your program to make a decision tree of at least 2 levels with this data set. List the examples as well as draw the tree found. Briefly explain the tree.

====== Pending ======

(6) Implement the random forest algorithm.

Code part.