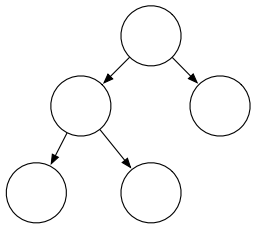
COMP 250: Introduction to Computer Science
Assignment 4

Posted Monday March 24, 2014
Due Thursday April 3, 2014

Please submit the homework through myCourses before midnight on the day it is due.

1. [15 points] **Trees**

   (a) [5 points] Suppose that you have a binary tree such that any internal node has exactly two children. An example of such a tree is depicted in Figure 1.

   Show by induction that a tree of this type with n leaves has exactly $2n - 2$ edges.

   Solution:

   Base case: the height of the tree is 0, i.e. we have 1 leaf. The tree has 1 node ($n = 1$) and $2 * 1 - 2 = 0$ edges

   Induction step: The tree has height > 0, so there is a root node, which has exactly 2 subtrees. Let n_L and n_R be the number of nodes in the left and right subtrees, and e_L, e_R the number of edges in each subtree. The number of nodes in the whole tree is $n = 1 + n_L + n_R$. The number of edges in the tree is $e = 2 + e_L + e_R$. By the induction hypothesis, we have:

   $$\begin{aligned} e_L &= 2n_L - 2 \\ e_R &= 2n_R - 2 \end{aligned}$$

   Hence,

   $$e = 2 + e_L + e_R = 2 + 2n_L - 2 + 2n_R - 2 = 2(1 + n_L + N_r) - 2 = 2n - 2$$

   which concludes the proof. Note that this is a typical structure for induction proofs on trees.

   (b) [5 points] k**-ary trees**

   Suppose that you have a tree such that any internal node has exactly k children, with $k \geq 2$. What is the maximum number of nodes that such a tree can have, if its depth is d? Prove your answer by induction on d.

   Solution: The maximum number of nodes is produced when each internal node has the maximum number of children, k. In this case, at depth 0 we have 1 node (the root), at depth

1 we have k nodes, at depth 2 we have $k * k = k^2$ nodes (each of the k nodes on the previous level has itself k children), at depth 3 we have $k^2 * k = k^3$ nodes etc. Hence, the total number is:

$$1 + k + k^2 + \cdots + k^d = \frac{k^{d+1} - 1}{k - 1}$$

(this is a geometric series).

We now prove by induction that the above holds.

Base case: $d = 0$, so we have 1 node, and $1 = \frac{k^1 - 1}{k-1}$

Induction step: A tree of depth d consists of a root node and k subtrees, each of them of depth $d - 1$. By the induction hypothesis, each of these trees has a number of nodes $n_i = \frac{k^{d+1}-1}{k-1}, i = 1 \ldots k$. The total number of nodes in the red is:

$$n = 1 + \sum_{i=1} k n_i = 1 + \sum_{i=1} k \frac{k^{d+1} - 1}{k - 1} = 1 + k \frac{k^{d+1} - 1}{k - 1} = \frac{k - 1 + k^{d+2} - k}{k-} = \frac{k^{d+2} - 1}{k - 1}$$

which concludes the proof.

(c) [5 points] In such a tree of maximum number of nodes, what fraction of the nodes are leaves? Prove your answer by induction on d.

Solution:

2. [10 points + 5 points extra credit] **More trees**

Suppose that you are given a general tree. Let a and b be nodes in this tree. We denote by $d(a, b)$ the number of edges on the unique path between these two nodes. Let $D(a)$ be the total number of edges on the paths connecting a to the other nodes in the tree: $D(a) = \sum_b d(a, b)$.

(a) [5 points] Show that for any tree and any node, $D(a) \leq \frac{n(n-1)}{2}$

Solution: We will do this by induction on the number of nodes in the tree. Suppose we have 1 node, so it has no paths, and for it, $D(a) = 0 = \frac{1(1-1}{2}$t.

Now consider a tree of n nodes and pick an arbitrary node a to be the root (which we can do in a general tree). Suppose the root has k children $a_1, \ldots a_k$, and these children are roots of subtrees of sizes $n_i, i = 1 \ldots k$. Consider a path from the root a to an arbitrary node in subtree i, b. We have:

$$d(a, b) = d(a, a_i) + d(a_i, b) = 1 + d(a_i, b)$$

as this path has to go through a_i. Hence, the sum of all the paths to nodes in subtree i is:

$$\begin{aligned} \sum_{b \in \text{subtree } i} d(a, b) &= \sum_{b \in \text{subtree } i} (1 + d(a, b_i)) \\ &= n_i + \sum_{b \in \text{subtree } i} d(a, b_i) = n_i + D(a_i) \leq n_i + \frac{n_i(n_i - 1)}{2} \end{aligned}$$

where in the last step we applied the induction hypothesis on subtree i.

Hence, over the whole tree, we have:

$$\begin{aligned} D(a) &= \sum_b d(a,b) = \sum_i \sum_{b \in \text{subtree } i} d(a,b) \\ &\leq \sum_i \left(n_i + \frac{n_i(n_i-1)}{2} \right) = \sum_i \frac{n_i^2 - n_i + 2n_i}{2} = \sum_i \frac{n_i^2 + n_i}{2} = \frac{1}{2} \left(\sum_i n_i^2 + \sum_i n_i \right) \end{aligned}$$

Note that $n = 1 + \sum_i n_i$. Hence, working backward from what we want on the right-hand side, we have:

$$\begin{aligned} \frac{n(n-1)}{2} &= \frac{n^2 - n}{2} = \frac{(1 + \sum_i n_i)^2 - (1 + \sum_i n_i)}{2} \\ &= \frac{1}{2} \left(1 + (\sum_i n_i)^2 + 2 \sum_i n_i - 1 - \sum_i n_i \right) = \frac{1}{2} \left((\sum_i n_i^2)^2 + \sum_i n_i \right) \end{aligned}$$

which is the same as above. This concludes the proof.

(b) [5 points] Show that for any n there exists a tree for which equality is achieved in the bound above (describe what this tree looks like).

Solution: If you have a linear tree (aka a list) the equality holds. You can prove this by induction exactly as above, except if you start with equality instead of inequality, this will be preserved throughout.

Note that there is a

(c) [5 points] **Extra credit:** Show that in any tree there exists a node such that, if we remove this node and the edges adjacent to it, we will obtain trees which have *at most* $n/2$ nodes (the removed node is not counted anymore).

3. [15 points] **Heaps**

Add to the Heap class provided to you a method for finding the *maximum* element from the heap in the most efficient manner possible, without changing the fact that each node is smaller than its children. Your method should be called:
public Object getMax() throws EmptyHeapException

What is the $O()$ of your method?

4. [10 points] **List Algorithms**

For the single linked list implementation, write a removeBefore(Object o) and removeAfter(Object o) method. In both cases, you should remove from the list the element before or after the object passed in as a parameter. If o is not present in the list, or if there is no element before or after it, you should throw a NoSuchElementException.

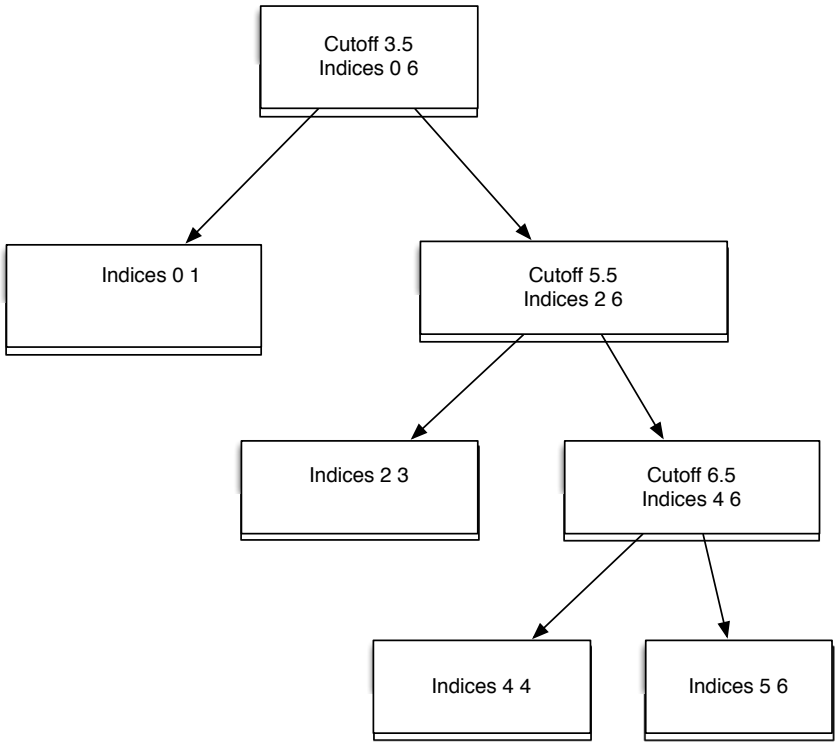


Figure 1: Example decision tree

5. [50 points] **Decision trees**

   Decision trees are used often in machine learning in order to solve problems in which one needs to learn a complicated function, such as how to diagnose if a patient has a particular disease or not, based on seeing examples (e.g. people who have some symptoms, but have or do not have the disease). In this question, we will develop a Java package for a simplified version of this problem.

   Class Instance.java, provided to you, contains a description of an instance. It contains a double value (e.g. the result of a blood test) and a boolean value, which indicates the class (e.g. does the patient have the disease or not). We will build the tree from a sorted array of instances (sorted by the value of the double variable).

   Class DTNode.java is a node of a decision tree. All nodes contain a pair of indices in the array, delimiting the instances that have reached the node. Internal nodes also contain a double value, which indicates the input value based on which instances are split at this node. For example, consider the array of instances:

   $$(1,t),(3,t),(4,f),(5,f),(6,t),(7,f),(8,f)$$

   A corresponding tree that classifies all instances correctly is given in Figure 1.

   The decision tree construction algorithm is greedy, and is based on a measure of "purity" of the instances at a node. If we had a .set of instances with the same label, this would be perfect, and

they would not need to be split further. If the instances are mixed, we can consider "cutting" the array in between each pair of instances where the sign flips. E.g. in the example above, we could cut at 3.5, 5.5 and 6.5. Which of these is the better cutoff is measured by considering the average entropy of the resulting nodes. Entropy is a measure of how "random" a distribution is. In our case, we measure the probability of a class, p, as the fraction of instances of that class among all instances. The entropy in our case is:

$$-p_t \log_2 p_t - p_f \log_2 p_f$$

Note that if all examples are of one class the entropy is 0, while if the number of examples in each class is equal, entropy is maximal, and equal to 1. We will consider all possible splits, and pick the one that generates the lowest average entropy in the resulting node partitions. For example, in the array above, the split that cuts the array at 3.5 gives entropy of 0 in the left partition, and $-\frac{1}{5}\log_2\frac{1}{5} - \frac{4}{5}\log_2\frac{4}{5}$on the other side, so the average entropy is:

$$\frac{2}{7} * 0 + \frac{5}{7}\left(-\frac{1}{5}\log_2\frac{1}{5} - \frac{4}{5}\log_2\frac{4}{5}\right)$$

(a) [20 points] Write a constructor for the Decision Tree class, which takes in a sorted array of instances, and constructs recursively a tree that splits the data until all leaves are pure. For simplicity, assume that all instances have a different value of the double attribute. you are allowed to modify any of the classes provided in any way you want.

(b) [10 points] Sometimes we like to avoid leaves with too few instances, because they might be noise. Write a method which takes as a parameter a minimum number of instances at a leaf, l. Traverse the tree, looking at each leaf. If it has fewer than l instances, the leaf and all its sibling sub-trees should be deleted and their parent should become a leaf. An example of pruning the leaf with just one instance from the tree in Figure 1 is given in Figure 2.

(c) [10 points] Write a method called classify, which takes a double value, and returns a boolean, which is the class associated by the tree to it. This accomplished by traversing the tree, following the left and right branches as appropriate, until a leaf is reached. At the leaf, the answer is the class of the majority of the instances. For example, in the tree in Figure 2, if we pass in the value 2, we would take the left branch and classify the result as true. Value 6.5 would take the right branch, then right again, and be classified as false.

(d) [10 points] Write a printing function, which prints the tree in an indented format. At each internal node you should print the cutoff value, and at each leaf, the class associated with it and the fraction of instances at this leaf that have the class. The information for a node should be indented by a number of spaces equal to its depth in the tree.

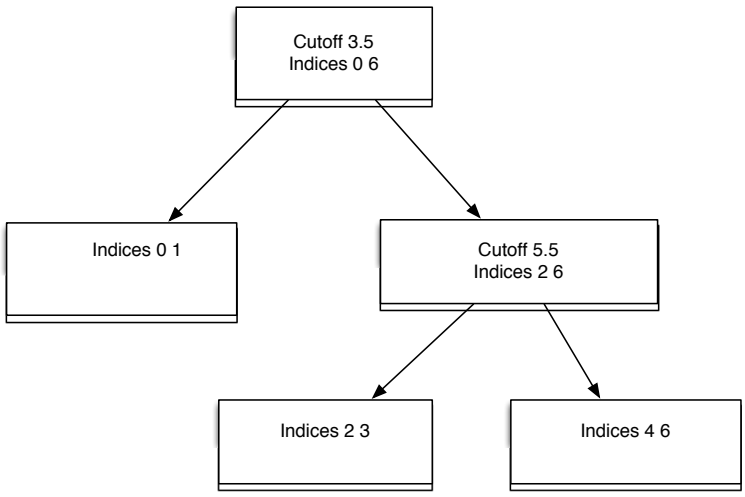


Figure 2: Example pruned decision tree