CS 624 - HW 2

Max Weiss

10 February 2020

1. Suppose that x is a binary tree. I'll write $c(x)$ for the set of trees rooted at child nodes of x. Every binary tree belongs to the smallest set X such that (i) each singleton tree belongs to X, and (ii) if $y \in X$ for all $y \in c(x)$, then $x \in X$. So, to show that some predicate P holds of all binary trees, it suffices to show

$$\forall x(Sx \to Px) \land \forall x(\forall y(y \in c(x) \to Py) \to Px) \tag{1}$$

where S defines the class of singleton trees.

I'll also write $c^*(x)$ for the smallest set X such that $x \in X$ and $y \in X$ for all $y \in c(x)$. It follows that

$$\forall x(Px \to \forall y(Py \to \forall z(z \in c(y) \to Pz)) \to \forall y(y \in c^*(x) \to Py)). \tag{2}$$

Let's say that a predicate P *perists downward* provided that Py follows from $y \in c(x)$ and Px. For a lemma, let me argue that the property of being a pre-tree is preserved downward. So, suppose that x is a pre-tree, and that y, z are the trees rooted at the left and right children of x. Then there are four possibilities: either y, z are complete of height n, or complete of heights $n, n-1$, or y is complete of height n and z is a pretree of height n, or y is a pretree of height n and z is complete of height $n-1$. In any case, both y and z are pretrees, as required.

Finally, for the main claim to be proven: suppose that P persists downward, that Q is true of all singleton trees, and that H_1, H_2, P, Q are predicates satisfying

$$H_1x \leftrightarrow Px \land \forall y(c^*(y) \to Qy) \tag{3}$$
$$H_2x \leftrightarrow Px \land Qx \land \forall y(y \in c(x) \to H_2y). \tag{4}$$

I'll argue that $H_1x \leftrightarrow H_2x$ for all binary trees x.

$\Leftarrow$. I'll use the induction (1). The claim is clear for singletons. So, assume that H_1x holds. From (3), it follows that H_1x implies Px and also Qx. By (4), it therefore remains to show $\forall y(y \in c(x) \to H_2y)$.

So, assume $y \in c(x)$; I will first argue that H_1y. Since P persists downward, we do have Py. Furthermore, $c^*(y) \subseteq c^*(y)$, which implies that $\forall z(z \in c^*(y) \to Qz)$.

So by (3), $H_1 y$ does hold. By induction hypothesis, we can thus infer $H_2 y$ as required.

$\Rightarrow$. Suppose that $H_2 x$ holds; I will argue for $H_1 x$. From (4), it is immediate that Px holds. So by (3), it remains to establish $\forall y(y \in c^*(x) \to Qy)$. Using the induction (2) on $c^*(x)$ it will be easier to prove $\forall y(y \in c^*(x) \to H_2 y)$. So, first suppose $y = x$; then $H_2 y$ amounts to $H_2 x$, and this we assumed at the outset. On the other hand, given $H_2 y$, the third clause of the definition of H_2 immediately implies $H_2 z$ for any $z \in c(y)$. This completes the proof that $H_2 y$ holds for all $y \in c^*(x)$, as desired.

2 (6.5.8, p166. Suppose we extend the ordering of keys (i.e., the ordering on the natural numbers, if that's what the keys are) so that $\infty > k$ for every key k, where ∞ is some new symbol. Now consider the following algorithm:

```
heap_increase_key(A, i, math.inf)
heap_extract_max(A)
```

Since $\infty > A[i]$, the result of line 1 is that A is a heap whose elements are precisely those of A except with ∞ in place of $A[i]$. Since $\infty > k$ for all keys k of nodes in A, therefore the result of line 3 is that A is a heap whose elements are precisely those of A except without $A[i]$. Therefore, the given algorithm implements HEAP-DELETE.

3 (6.5.9, p166) If A, B are heaps, write $A < B$ if $A[1] < B[1]$. In the below, I assume that H is an array of k heaps with n total elements.

```
Z = MaxHeap()
S = [None]*n
for L in H:
    maxheap_insert(Z, L)
for i in 1 .. n+1:
    L = maxheap_extract_max(Z)
    x = L.pop()
    S[i] = x
if L is not empty:
    maxheap_insert(Z, L)
```

This algorithm breaks into two tasks: first, inserting the k heaps into the meta-heap Z, and second, inserting elements of the heaps into the array S.

The first task requires k insertions into a heap of size at most k; hence it is in $O(k \lg k)$. The second task requires n iterations of (at most) the following: heap-extract the minimum L from Z; pop the last (maximum) x from L and array-insert x into S; and heap-insert L back into Z. Since Z has size at most k, the heap-insertions and extractions are $O(\lg k)$, while the list operations are constant time. Consequently, the second task is $O(n \lg k)$. So, the whole thing is $O(n \lg k)$.

4 (6.1, lecture 3). Let S be an unsorted array of n distinct elements. Consider the following algorithm:

```
build_min_heap(S)
R = []
for _ in 1 .. k+1:
    R.push(minheap_extract_min(S))
```

The first line of the algorithm converts S into a MinHeap. I'll now argue that the loop satisfies this property: after i iterations, S remains a MinHeap and all elements of R are smaller than all elements of S, while R contains i elements and S contains $S - i$ elements. This is clear for $i = 0$, so suppose it holds for i. Since S is now a heap with $n - i$ elements, the effect of calling MinHeapExtractMin is to convert S into a heap with $n - i - 1$ elements and to return the smallest element x of those. So when x is pushed onto R, the result is that R contains $i + 1$ elements, all of which are smaller than all elements of S. This establishes the invariant. It follows that after the prescribed k loop iterations, R contains the k smallest elements of the originally given set, as desired.

As for running time: the call to BUILD-MINHEAP is $O(n)$, while each of the k calls to MINHEAP-EXTRACT-MIN (and push of the result onto a list) costs $\lg n$. So, the overall running time is $O(n + k \lg n)$.

5 (7.3.2, p180).

6 (7.4, p188).

7. Let x be a binary tree, and let d be the depth of x. Let me write $|x|$ for the number of nodes of x. I will use the lemma (1) from question 1 to prove the stronger claim that

$$|x| \leq 2^d - 1. \tag{5}$$

If x is a singleton, then x has one node, and depth 1; hence x has at most $1 = 2^d - 1$ leaves. So, suppose that (5) is satisfied by all trees rooted at children of x. Those trees must have at most depth $d - 1$ (this follows from the definition of depth). Furthermore, there are at most two such trees. Thus,

$$\begin{aligned} \sum_{y \in c(x)} |y| &\leq 2^{d-1} - 1 + 2^{d-1} - 1 \\ &= 2^d - 2. \end{aligned} \tag{6}$$

Meanwhile,

$$|x| \leq 1 + \sum_{y \in c(x)} |y|. \tag{7}$$

The desired result is immediate from (6) and (7).

7 redux. As suggested in the homework, it is also possible to prove the result using "mathematical induction", i.e., to deduce (5) from the induction principle

$$P0 \wedge \forall x(Nx \wedge Px \to Px') \to \forall x(Nx \to Px) \tag{8}$$

where N defines $\mathbb{N}$ and x' represents the successor of x.

7a. The suggested statement is equivalent to the countable conjunction

$$\bigwedge_{d \in \mathbb{N}} \forall x \in \mathcal{B}(d = \mathrm{depth}(x) \to |x| \leq 2^d). \tag{9}$$

where $\mathcal{B}$ is the class of binary trees.

7b. I'll use the induction (8) to prove

$$\forall d \in \mathbb{N}\, \forall x \in \mathcal{B}(d = \mathrm{depth}(x) \to |x| \leq 2^d). \tag{10}$$

Here we must use the property Pd defined by

$$d > 0 \to \forall x \in \mathcal{B}(d = \mathrm{depth}(x) \to |x| \leq 2^d - 1). \tag{11}$$

The case $P0$ is vacuous. So suppose d satisfies (11); we need to show that $d+1$ satisfies this as well. To this end, pick $x \in \mathcal{B}$ such that $d + 1 = \mathrm{depth}(x)$ with $d > 0$. The children of x must have depths at most d. By (8), we may thus assume each child has at most $2^d - 1$ nodes. The sum of the numbers of nodes of the children is therefore at most $2^{d+1} - 2$, while the number of nodes of x is one more than that sum. Hence the number of nodes of x is at most $2^{d+1} - 1$, as desired.

7c. The sketched line of reasoning would prove at most

$$\forall d \in \mathbb{N}\, \exists x \in \mathcal{B}(d = \mathrm{depth}(x) \wedge |x| \leq 2^d - 1)$$

which does not imply (10).