Heaps - Recitation 4

Correctness, Loop Invariant And Heaps.

November 8, 2022

Apart from quantifying the resource requirement of our algorithms, we are also interested in proving that they indeed work. In this Recitation, we will demonstrate how to prove correctness via the notation of loop invariant. In addition, we will present the first (non-trivial) data structure in the course and prove that it allows us to compute the maximum efficiently.

Correctness And Loop Invariant.

In this course, any algorithm is defined relative to a task/problem/function, And it will be said correctly if, for any input, it computes desirable output. For example, suppose that our task is to extract the maximum element from a given array. So the input space is all the arrays of numbers, and proving that a given algorithm is correct requires proving that the algorithm's output is the maximal number for an arbitrary array. Formally:

Correctness.

We will say that an algorithm $\mathcal{A}$ (an ordered set of operations) computes $f : D_1 \to D_2$ if for every $x \in D_1 \Rightarrow f(x) = \mathcal{A}(x)$. Sometimes when it's obvious what is the goal function f, we will abbreviate and say that $\mathcal{A}$ is correct.

Other functions f might be including any computation task: file saving, summing numbers, posting a message in the forum, etc. Let's dive back into the maximum extraction problem and see how correctness should be proven in practice.

Task: Maximum Finding. *Given $n \in \mathbb{N}$ numbers $a_1, a_2, \cdots a_n \in \mathbb{R}$ write an Algorithm that returns their maximum.*

Consider the following suggestion. How would you prove it correct?

Maximum finding.

Result: returns the maximum of $a_1 ... a_n \in \mathbb{R}^n$

```
1 let b ← a_1
2 for i ∈ [2, n] do
3     b ← max(b, a_i)
4 return b
```

Usually, it will be convenient to divide the algorithms into subsections and then characterize and prove their correctness separately. One primary technique is using the notation of Loop Invariant. Loop Invariant is a property that is characteristic of a loop segment code and satisfies the following conditions:

Loop Invariant.

1. Initialization. The property holds (even) before the first iteration of the loop.
2. Conservation. As long as one performs the loop iterations, the property still holds.
3. Termination. Exiting from the loop carrying information.

Let's denote by $b^{(i)}$ the value of b at line number 2 at the ith iteration for $i \geq 2$ and define $b^{(1)}$ to be it's value in the it's initialization. What is the Loop Invariant here? **Claim.** *"at the i-th iteration,* $b^{(i)} = \max\{a_1...a_i\}$*"*.

Proof. Initialization, clearly, $b^{(1)} = a_1 = \max\{a_1\}$. Conservation,by induction, we have the base case from the initialization part, assume the corretness of the clain for any $i' < i$ and consider the ith iteration (ofcourse, assume that $i < n$). Then:

$$b^{(i)} = \max\{b^{(i-1)}, a_i\} = \max\{\max\{a_1, ..a_{i-2}, a_{i-1}\}, a_i\} = \max\{a_1, ..a_i\}$$

And that compleate the Conservation part. Termination, followes by the conservation, at the n iteration, $b^{(i)}$ is seted to $\max\{a_1, a_2..a_n\}$.

Task: Element finding. *Given* $n \in \mathbb{N}$ *numbers* $a_1, a_2, \cdots a_n \in \mathbb{R}$ *and additional number* $x \in \mathbb{R}$ *write an Algorithm that returns* i *s.t* $a_i = x$ *if there exists such* i *and* False *otherwise.*

Element finding.

Result: returns the maximum of $a_1...a_n \in \mathbb{R}^n$

```
1 for i ∈ [n] do
2     if a_i = x then
3         return i, a_i
4 return Δ
```

Correctness Proof. First, let's prove the following loop invariant.

Claim *Suppose that the running of the algorirthm reached the i'th iteration, then* $x \notin \{a_1..a_{i-1}\}$. **Proof.** Initialization, for $i = 1$ the claim is trival, let's use that as the induction base case for proving Conservation. Assume the correctness of the claim for any $i' < i$. And consdier the ith iteration. By the induction assumption we have that $x \notin \{a_1..a_{i-2}\}$, and by the fact that we reached the ith iteration we have that in the $i-1$ iteration, at line (2) the condintional weren't satisfied (otherwise, the function would returned at line (3) namely $x \neq a_{i-1}$. Hence, it follows that $x \notin \{a_1, a_2..a_{i-2}, a_{i-1}\}$.

Seperate to cases. First consdier, the case that given the input $a_1..a_n$ the algorithm reutrn Δ. In this case we have by the termaionation property that $x \notin \{a_1..a_n\}$.Now, Suppose that the algorithm return the pair (i, x) then it's mean that the condintional at line (2) were satisfied at the ith iteration. So, indeed $a_i = x$ and the algorithm returns the expected output.

Task: The Superpharm Problem. *You are requested to maintain a pharmacy line. In each turn, you get one of the following queries, either a new customer enters the shop, or the pharmacist requests the next person to stand in front. In addition, different customers have different priorities, So you are asked to guarantee that in each turn, the person with the height priority will be at the front.*

Before we consider a sophisticated solution, What is the running time for the naive solution?(maintaining the line as a linear array) $(\sim O\left(n^2\right))$.