Heaps - Recitation 4

Correctness, Loop Invariants And Heaps.

November 5, 2022

Abstract

Apart of quantify the resource requirement of our algorithms we are also intersted to prove 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-trival) data structre in course, and prove that it allow us to compute the maximum efficintly.

1 Correctness And Loop Invariant.

In this course, any algorithm is defined relative to a task/problem/function, And it will be said correct if for any input it compute diserable output. For example, suppose that our task is to extract the maximum element from a given array. So the input space are all the arrays of numbers, and proving that a given algorithm is correct, requires from us to prove that for an aribtrary array the algorithm's out put is the the maximal number. 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 $x \in D_1$ the following equality holds $f(x) = \mathcal{A}(x)$. Sometimes when it's obvious what is the goal function f, we will abbreviate and say that $\mathcal{A}$ is correct.

Examples of functions f might be: file saving, summing numbers, or posting a message in the forum.

Usually it will be convinant to divide the algortims into subsections and then characteristic, and prove correctness for each of them seapratly. One main technique is usning the notation of Loop Invariants. Loop Invariant is a property that characteristic a loop segment code and satisfy 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. (optional) Termination. Exiting from the loop carrying information.

Task: Maximum finding.

Example. Before dealing with the hard problem, let us face the naive algorithm to find the maximum of a given array.

Maximum finding.

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

1
2 let $b \leftarrow a_1$
3
4 **for** $i \in [2, n]$ **do**
5 $\quad b \leftarrow \max(b, a_i)$
6 return b

What is the Loop Invariant here? *"at the i-th iteration, $b = \max\{a_1 ... a_{i-1}\}$"*. The proof is almost identical to the naive case.

Claim. Consider the while loop. The property: *"for every $j' < j \leq n+1 \Rightarrow a_{j'} \leq a_i$"* is a loop invariant that is associated with it.

Proof: first, the initialization condition holds, as the at the first iteration $j = 1$ and therefore the property is trivial. Assume by induction, that for every $m < j$ the property is correct, and consider the j-th iteration. If back again to line (5), then it means that $(j-1) < n$ and $a_{j-1} \leq a_i$. Combining the above with the induction assumption yields that $a_i \geq a_{j-1}, a_{j-2}, ... a_1$.

Correctness Proof. Split into cases, First if the algorithm return result at line (9), then due to the loop invariant, combining the fact that $j = n+1$, it holds that for every $j' \leq n < j \Rightarrow a_i \geq a_{j'}$ i.e a_i is the maximum of $a_1, ...a_n$. The second case, in which the algorithm returns Δ at line number (10) contradicts the fact that n is finite, and left as an exercise. the running time is $O(n^2)$ and the space consumption is $O(n)$.

blabla bla bla

Loop Invariant.

Task: Element finding. ddd

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

1
2 **for** $i \in [n]$ **do**
3
4 $\quad$ **if** $a_i = x$ **then**
5 $\quad\quad$ return i, a_i
6 return Δ

Algorithm 1: naive maximum alg.

Loop Invariant In The Cleverer Alg. Consider now the linear time algorithm:

Heaps.