Heaps - Recitation 4

Correctness, Loop Invariant And Heaps.

November 5, 2022

Apart of quantify the resource requirement of our algorithms we are also interested 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-trivial) data structure in course, and prove that it allow 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 correct if for any input it compute desirable 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 to prove that for an arbitrary 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 $f : D_1 \rightarrow D_2$ 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 Examples of 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 prove in practice.

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

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

> **Maximum finding.**
>
> **Result:** returns the maximum of $a_1...a_n \in \mathbb{R}^n$
> 1 let $b \leftarrow a_1$
> 2 **for** $i \in [2, n]$ **do**
> 3 $\quad\lfloor\ b \leftarrow \max(b, a_i)$
> 4 return b

Usually it will be convineint to divide the algorithms into subsections and then characteristic, and prove correctness for each of them separately. One main technique is using the notation of Loop Invariant. Loop Invariant is a property that characteristic a loop segment code and satisfy the following conditions:

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 $O(n)$.

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
```

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

Task: The Superpharm Problem. *you are requested to maintain a pharmacy line, in each turn you get one of the fallowing queries, either a new customer enter into the shop, either a pharmacist request for the next person stand in front. In addition, different customers have different priorities, So you are asked to gurnte 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 linear array) $(\sim O(n^2))$.

Heaps.

Heaps are structures that in addition for supporting adding and removing elements are also enable to compute the maximum efficiently.

We have seen in the Lecture that no Algorithm can compute the max function with less then $n-1$ comparisons. So our solution above is indeed the best we could expect for. The same is true for the searching problem, and yet we saw that if we are interested in storing the numbers then, by storing them

according to sorted order, we could compute each query in logarithmic time via binary search. That arise the question, is it possible to have a similar result regarding the max problem?.

> **Heap**
>
> Let $n \in \mathbb{N}$ and consider the sequence $H = H_1, H_2 \cdots H_n \in \mathbb{R}\,(*)$. we will say that H is a Heap if for every $i \in [n]$ we have that: $H_i \leq H_{2i}, H_{2i+1}$ when we think of the value at indices greater than n as $H_{i>n} = -\infty$.

Checking vital signs. Are the following sequences are heaps?

1. 1,2,3,4,5,6,7,8,9,10 (Y)

2. 1,1,1,1,1,1,1,1,1,1 (Y)

3. 1,4,3,2,7,8,9,10 (N)

4. 1,4,2,5,6,3 (Y)

How mach is cost (running time) to compute the the min of H? (without change the heap). ($O\,(1)$). Assume that option 4 is our Superpharm Line, let's try to imagine how should we maintain the line. After serving the customer at top, what can be said on $\{H_2, H_3\}$? or $\{H_{i>3}\}$? (the second heighset value is in $\{H_2, H_3\}$.)

Subtask: Extracting Heap's Minimum. *Let H be an Heap at size n, Write algorithm which return H_1, erase it and returns H', an Heap which contain all the remain elements.* **Solution:**

> **Heappop.**
>
> **Result:** returns the maximum of $a_1...a_n \in \mathbb{R}^n$
> 1 ret $\leftarrow H_1$
> 2 $H_1 \leftarrow \infty$
> 3 Heapify-down(1)
> 4 return ret

> **Heapify-down.**
>
> **Result:** returns the maximum of $a_1...a_n \in \mathbb{R}^n$
> 1 next $\leftarrow$ i
> 2 **if** $2i < n$ *and* $H_{next} \leq H_{2i}$ **then**
> 3 $\quad$ next $\leftarrow 2i$
> 4 **if** $2i+1 < n$ *and* $H_{next} \leq H_{2i+1}$ **then**
> 5 $\quad$ next $\leftarrow 2i+1$
> 6 **if** $i \neq next$ **then**
> 7 $\quad$ $H_i \leftrightarrow H_{next}$
> 8 $\quad$ Heapify-down(next)

Claim. Assume that H satisfies the Heap inequality for all the elements except H_j. Namely for any $i \neq j$ we have that $H_i \leq H_{2i}, H_{2i+1}$. Then applying Heapify-down on H at index j returns an heap.

Proof.   Induction on $n - j$. Base case, $n - j < n/2 \Rightarrow j > n/2$, Hence $2j, 2j + 1 \notin [n]$ and we have that the Heap property holds for any i.

Assume the correctness of the claim for any j' satisfy $j' > j$ and consider j. Denote by H' the state of the heap after the swapping at line number 7 and let $i \in [n]$. First Notice that if $j \neq 2i, 2i + 1$ and $i \neq 2j, 2j + 1$ then $H'_i = H_i \leq H_{2i} = H'_{2i}$ (and in similar to $2i + 1$). So it left to consider the case where $i = \lfloor j/2 \rfloor$ and $i = j$.

Insertion.   blabla

Heapify-up.

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

1   parent $\leftarrow \lfloor i/2 \rfloor$
2   **if** $parent > 0$ *and* $H_{perent} \leq H_i$ **then**
3      $H_i \leftrightarrow H_{\text{perent}}$
4   Heapify-up(perent)

Heappush.

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

1   $H_n \leftarrow v$
2   Heapify-up(n)
3   $n \leftarrow n + 1$