Heaps - Recitation 4

Correctness, Loop Invariant And Heaps.

November 10, 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) $(\sim O\left(n^2\right))$.

Heaps.

Heaps are structures that enable computing the maximum efficiency in addition to supporting adding and removing elements.

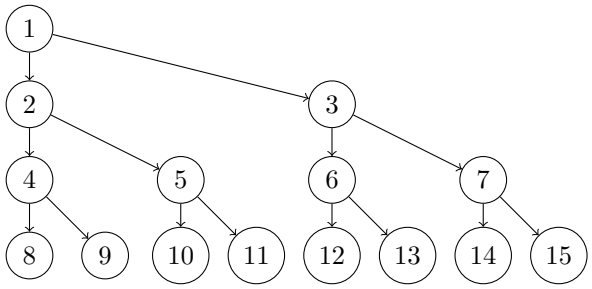
We have seen in the Lecture that no Algorithm can compute the max function with less than $n-1$ comparisons. So our solution above is indeed the best we could expect for. The same is true for the search problem. Yet, we saw that if we are interested in storing the numbers, then, by keeping them according to sorted order, we could compute each query in logarithmic time via binary search. That raises 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 $H_{i>n} = -\infty$.

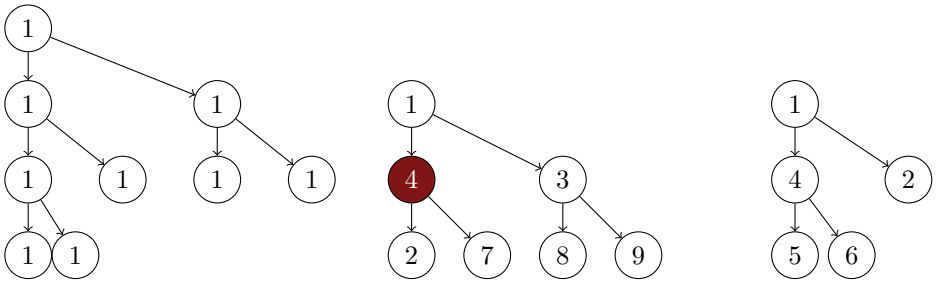
$\Leftrightarrow$

That defintion equivalance to the following requrisve defination: Consider a binary tree, that we associcate a number for each node. Then, we will say that this binary tree is an heap if the root's value is lower than the values of its sons and also each of the subtrees defined by its childrens is also a heap.



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 ← H_1
2 H_1 ← ∞
3 Heapify-down(1)
4 return ret
```

Heapify-down.

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

```
1  next ← i
2  left ← 2i
3  right → 2i + 1
4  if left < n and H_next ≤ H_left then
5      next ← left
6  if right < n and H_next ≤ H_right then
7      next ← right
8  if i ≠ next then
9      H_i ↔ H_next
10     Heapify-down(next)
```

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

Proof. By Induction on the heap size.

- Base, Consider an heap at size at most 3, and prove for each by consider each case seaprtly. (lefts as exersice).
- Assumption, assume the correctness of the claime for any tree, that satisfies the heap inequalilitys except the root, at size $n' < n$.
- Induction step. Cnsider a tree at size n which and assume w.l.g (why could we?) that the right chiled of the root is the minmum between the triple. Then by the defination of the algortihm, at line (9) the values of the right child and the root are sawped. Given that before the swapping all the elements of the heap, except the root, had satisfied the heap inequalility, we have that after the exchange, all the elements in right subtree, except the root of that subtree (the orignal root's right chiled) still satisfy the inequality. As the size of the right subtree is at most $n - 1$ we could use the assumption and have that after line (10) the right subtree is an heap.

  Now, as the left subtree remaines the same, (the values of the nodes of the left side weren't change) we have that this subtree is also an heap. So it's left to show that the root of the new ttree is smaller than both it's children. Suppose that is not the case, then it's clear that the root of the right subtree (heap) is smaller than the new root. Combine the fact that it origin must be the right subtree we have contrudiction to the fact that the orignal right subtree was an heap (as it's root wasn't the minimum elmenent in that subtee).

Question. How to construct an heap? and how much time it will take?

Build.

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

```
leftHeap ← Build ( x_2..x_{n/2})
rightHeap ← Build ( x_{n/2+1}, ..x_n)
Heapify-down(1, x_1..x_n)
return x_1..x_n
```

Let's compute the running time of the construction algorithm. The algortim make two reqursive calls on input at helf of the orignal size, and then pay $\Theta(\log n)$ time to drop the the first node. Hence by using the Master Theorem we have obtain that:

$$T(n) = \Theta(\log n) + 2T\left(\frac{n}{2}\right)$$
$$T(n) = \Theta(n)$$

Heapify-up.

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

```
parent ← ⌊i/2⌋
if parent > 0 and H_perent ≤ H_i then
    H_i ↔ H_perent
    Heapify-up(perent)
```

Heappush.

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

```
H_n ← v
Heapify-up(n)
n ← n + 1
```

Task: Write a datastructure that support insertion and deltion at $O(\log n)$ time and in addition enable to extract the median in $O(\log n)$ time.

Soultlion. We will define two seprate Heaps, the first will be a maximumm heap and will store the first $\lfloor n/2 \rfloor$ smallest elements, and the second will be a minimum heap and will contain the $\lceil n/2 \rceil$ greatest elements.

Median-Heappush.

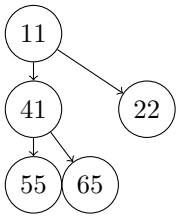
Result: returns the maximum of $a_1 ... a_n \in \mathbb{R}^n$

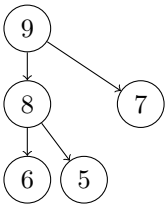
```
median ← extract H_max
if v <median then
    heappush ( H_max, v )
    heappush ( H_min, median )
else
    heappush ( H_min, v )
    heappush ( H_max, median )
```

Median-Heappop.

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

median $\leftarrow$ extract H_{max}
if *size*(H_{min}) - *size*(H_{max}) > 2 **then**
    temp $\leftarrow$ extract $H_{\min}$
    heappush (H_{max}, temp)
return median





1 Appendix. Exercise from last year

Question. Consider the sets $X = \{x_1, x_2..x_n\}$, $Y = \{y_1, y_2..y_n\}$. Assume that each of the values x_i, y_i is unieq. Write an Algorithm which compute the k most small items in $X \oplus Y = \{x_i + y_j : x_i \in X, y_j \in Y\}$ at $O(n + k \log k)$ time.

Solution. Notice that If $a \in X$ is greater than i elements of X and $b \in Y$ greater than j elemnts of Y. Then, $a+b$ greater than $i \cdot j$ elements of $X \oplus Y$. Denote by $X' = \{x'_1..x'_n$ (Y') The elements of X in sorted order. So it's clear that if $x_i + y_j = x'_{i'} + y'_{j'}$ is one of the k smallest elements of $X \oplus Y$ then $i'j' \geq k$. So we are going to create an heap of elements which rescpects the that inequality, and then query that heap.

Heappush.

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

```
H_X ← build (X)
H_Y ← build (Y)
S_X ← extract-k (H_X)
S_Y ← extract-k (H_Y)
H_XY ← Heap ({})
for i ∈ [k] do
    for j ∈ [k/i] do
        Heappush( H_XY, S_{X,i} + S_{Y,j} )
return extract-k ( H_XY )
```