Lecture 35: Analysis and Sample Exam
Problem
10:00 AM, Nov 25, 2019

Contents

1 Introduction 1

2 Analysis for log 1

3 Sample Exam Problem 2

1 Introduction

Racket and Reason have their own advantages. Reason has its own advantages for patter matching and if your program uses pattern matching, you would prefer ReasonML over Racket. Proving a complicated language like C++ is safe is hard but when a particular language has a less complicated syntax, proving it’s safer is easier.

2 Analysis for log

Let’s look at the previous homework.
\[ V(0) = 2 \]
\[ V(n) = 3 + V(\lfloor \frac{n}{2} \rfloor) \]
We want to prove that \( V(n) \in O(n \log_2 n) \)

Using plug and chug, we have
\[ V(0) = 2 \]
\[ V(1) = 3 + 2 \]
\[ V(2) = 3 + 3 + 2 \]
\[ V(3) = 3 + 3 + 2 \]
\[ V(4) = 3 + 3 + 3 + 2 \]
\[ V(5) = 3 + 3 + 3 + 2 \]
\[ V(6) = 3 + 3 + 3 + 2 \]

Look at \( V(2^k) \) because that is where we see changes that happen, i.e in \( V(1), V(2), V(4), \) etc
\[ V(2^k) = 3 + V(2^{k-1}) \]
\[ V(2^k) = 3 + 3 + V(2^{k-2}) \]
\[ V(2^k) = 3 + 3 + \ldots + V(2^0) \]
\[ V(2^k) = 3 + 3 + \ldots + 3 + 3 + V(0) \]
\[ V(2^k) = 3(k + 1) + 2 \]
if \( n = 2^k \), \( \log_2 k \), which indicates that
\[
V(n) \leq 3(\lfloor \log_2 n \rfloor + 1) + 2
\]
\[
V(n) \leq 3(\lfloor \log_2 n \rfloor) + 5
\]

**Claim** \( V(n) \in O(n \rightarrow \log_2 n) \)
\[
3 \log_2 n < \log_2 n
\]
for \( n > M \)
Is there any value for \( M \) which the above equation is satisfied? No.
\[
3 \log_2 n + 5 > 3 \log_2 n > \log_2 n
\]

**Claim** \( V(n) \in O(n \rightarrow n \log_2 n) \)
\[
3 \log_2 n + 5 < 4 \log_2 n \quad \text{for } n > 1000000
\]
5 \( < \log_2 n \) which works!

**Claim** \( \log_2 n + 1 \in \Omega(\log_2 n) \)
Restate: There are numbers \( M, c>0 \) such that for \( n > M \), \( \log_2 n + 1 > c \log_2 n \)
Pick \( c = 1, M = 1 \)

**Claim** \( 2 \log_2 n - 1 \in \Omega(\log_2 n) \)
If picking \( c \) and \( M \) is hard, try graphing it
For \( M \),
\[
\log_2 n - 1 \geq 0
\]
\[
\log_2 n \geq 1
\]
\[
n > 2
\]
Therefore, \( M = 2 \)

Suppose \( n > M \), then \( n > 2 \) needs to show that
\[
2 \log_2 n - 1 \geq \log_2 n
\]
\[
2 \log_2 n - 1 \geq \log_2 n
\]
\[
\log_2 n - 1 > 0
\]
\[
\log_2 n > 1
\]
\[
n > 2
\]
We also have to mentally check if we can replicate the same equations backwards.
\[
n > 2 \quad \text{Assumption}
\]
\[
\log_2 n > 1 \quad \text{proposition of log}
\]
\[
\log_2 n - 1 > 0 \quad \text{algebra}
\]
\[
\log_2 n - 1 + \log_2 n > \log_2 n \quad \text{algebra}
\]
\[
2 \log_2 n - 1 > \log_2 n \quad \text{algebra}
\]
Note: \( V(n) \in O(n \rightarrow \log_2 n) \) is equivalent to \( V(n) \in O(\log_2 n) \)

## 3 Sample Exam Problem

Problem: Backbone similarity have the same "overall appearance", if you ignore the values at the nodes. Write a backboneEquals(tree1, tree2) procedure
Solution:

```ocaml
type tree('a) = |Leaf | Node('a, tree('a), tree('a));
let rec backboneEquals = (t1, t2) →
```

2
switch(t1, t2) {
  | (Leaf, Leaf) ⇒⇒ true
  | (Leaf, _) ⇒⇒ false
  | (_, Leaf) ⇒⇒ false
  | (Node(_, left1, right1), Node(_, left2, right2)) ⇒⇒
    backboneEquals(left1, left2) && backboneEquals(right1, right2)
};

We can also use split tree.

type tree('a) = Leaf | Node('a, tree('a), tree('a))
type stree = SLeaf | SNode(stree, stree)
let rec strip : tree('a) ⇒⇒ stree = fun
  | Leaf ⇒⇒ SLeaf
  | Node(_, left, right) ⇒⇒ SNode(strip(left), strip(right))

let backboneEquals: (tree('a), tree('b)) ⇒⇒ bool = (t1, t2) => strip(t1) == strip(t2);

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS 17 document by filling out the anonymous feedback form: [http://cs.brown.edu/courses/csci0170/feedback](http://cs.brown.edu/courses/csci0170/feedback).