FOCS Notes
2/4/2020

Lecture 7 Recursion: Recursive Functions, Sets, Recurrrences

- Recursive functions
  - Analysis using induction
  - Recurrences
  - Recursive programs/algorithms
- Recursive sets
  - natural numbers
  - The Finite Binary Strings Σ^*
- Recursive structures
  - Rooted binary trees (RBT)

Contents

1 Examples of Recursion

1.1 Factorial

$$f(n) = n! = \begin{cases} 1 & n = 0 \\ n * f(n-1) = n * (n-1)! & n \geq 1 \end{cases}$$

1.2 More examples

$$f(n) = \begin{cases} 0 & n \leq 0 \\ f(n-1) + 2n - 1 & n > 0 \end{cases}$$

0	1	2	3	4
0	1	4	9	16

claim: $f(n) = \begin{cases} 0 & n < 0 \\ n^2 & n \geq 0 \end{cases}$

Proof. $f(n) = n^2, \forall n \geq 0$
By induction:

Base case $f(0) = 0^2$ is true
Induction By direct proof. Assume $f(n) = n^2 \;\therefore$

$$\begin{aligned} f(n+1) &= f(n) + 2(n+1) - 1 \\ &= n^2 + 2n + 1 \\ &= (n+1)^2 \end{aligned}$$

□

$$f(n) = \begin{cases} 0 & n \leq 0 \\ f(n-1) + 2n - 1 & n > 0 \end{cases}$$

Induction	Recurstion
$P(0)$ is true; $P(n) \to P(n+1)$ conclude $P(n)$ is true, $\forall n \in \mathbb{N}_0$	$f(0) = 0;\; f(n+1) = f(n) + 2n - 1$ conclude that we can compute $f(n)\ \forall n \geq 0$
$P(0) \to P(1) \to P(2) \ldots$	$f(0) \to f(1) \to \ldots$

1.3 Fibonacci Sequence

$$f(n) = \begin{cases} 1 & n = 1 \\ 1 & n = 2 \\ f(n-1) + f(n-2) & n > 2 \end{cases}$$

$F_0 = 0$
$F_1 = 1$
$F_n = F_{n-1} + Fn - 2,\; n \geq 2$

Everything relies on all points before it, so this is similar to strong induction.

Claim $f(n) \leq 2^n$

Proof. By strong induction
Base cases $f(1) = f(2) = 1 \leq 2^1$ true
Inductive Step Assume $F(k) \leq 2^k$ for all $k \leq n$. Then we want to show this implies $f(n+1) \leq 2^{n+1}$.
Direct proof. $f(n+1) = f(n-1) + f(n-2) \leq^? 2^{n+1} \leq 2^n + 2^{n-1} \leq 2^n + 2^n = 2 * 2^n = 2^{n+1}$ true by strong inductive hypothesis. □

Claim $f(n) \geq (\frac{3}{2})^n$ when $n \geq 11$
Ex prove this claim via strong induction.

2 Checklist for Analyzing Recursion

1) Tinker. Plug in a few values for n and see:
   - does f have a well-defined set of implications and base cases
   - can you form a hypothesis about $f(n)$?
2) Prove your conjecture about f using a form of induction.
   - the particular type of induction will depend on how exactly recursion is used in the definition of f

```
out=Big(n);
    if (n==0) out=1;
    else out = 2 * Big(n-1);
Big(n)=2^n
```

Let $T_n = f(n)$ be the runtime for Big(n)

$$T_n = \begin{cases} 2 & n = 0 \\ 3 + T_{n-1} & n > 0 \end{cases}$$

The operations for $n > 0$ are: evaluate if (n==0), evaluate Big(n-1) in T_{n-1} time, multiply by 2, and assign to output
$T_0 \to T_1 \to \dots$

0	1	2	3	4
2	5	8	11	14

$T_n = 3n + 2$
Ex: prove this

3 Recursively Defined Sets

$\mathbb{N} = \{1, 2, 3, \dots\}$
Recursive definition of $\mathbb{N}$

1. $1 \in \mathbb{N}$
2. $x \in \mathbb{N} \to x + 1 \in \mathbb{N}$
3. Minimality: these are the only elements of $\mathbb{N}$ $\longleftarrow$ very important for recursive definitions of sets; usually implicit

Finite Binary String Σ^*
Let ϵ be the empty string

1. $\epsilon \in \Sigma^*$
2. if $x \in \Sigma^* \to x \cdot 0 \in \Sigma^*$ and $x \cdot 1 \in \Sigma^*$. $\cdot$ means concatenation

Question: $x \in \Sigma^* \to 0 \cdot x \in \Sigma^*$? $1 \cdot x \in \Sigma^*$?

4 Recursive Structures

4.1 Trees

Rooted Binary Trees (RBT)

1. the empty binary tree (ϵ) is a RBT
2. If T_1 and T_2 are disjoint RBTs with roots r_1 and r_2,then linking r_1 and r_2 to a new root r gives a new RBT with root r

- Can we recognize (i.e. prove or disprove) RBTs?
- Is there more than one way to construct a RBT?
- Properties/definitions of trees:
  - A tree is a connected graph with n vertices and $n - 1$ edges
  - A tree is a connected graph with no cycles
  - A tree is a graph in which any two nodes are connected by exactly one path
- Are RBTs trees? That is, do they have these properties?