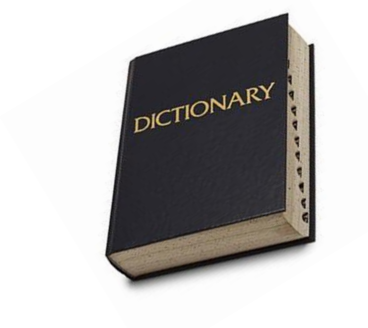
CSCI 2200 FOUNDATIONS OF COMPUTER SCIENCE

David Goldschmidt
goldsd3@rpi.edu
Fall 2022

RECURSION

...to understand recursion, you must first understand recursion...

Recursion is a broadly applicable technique that involves a self-reference...



look-up(w_0): find the definition of word w_0 in the dictionary;
if the definition has unknown words w_1, w_2, ..., w_n,
call look-up(w_1), look-up(w_2), ..., look-up(w_n)

Will the recursion here ever stop?

The recursive look-up() function works if there are known words to which everything else reduces—i.e., one or more base cases

WELL-DEFINED RECURSIVE FUNCTIONS

A *well-defined* recursive function…

…must have the necessary base case or base cases

…and in computing $f(n)$, at each iteration, must move *strictly closer* to the base case(s)

A well-defined recursive function for the *n*th Fibonacci number…

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

RECURSION AND INDUCTION

Induction and recursion share a similar structure…

Induction

Base case: $P(0)$ is **T**

Induction step: show $P(n) \rightarrow P(n+1)$

$\therefore$ $P(n)$ is **T** for all $n \geq 0$

$\boxed{P(0)} \rightarrow P(1) \rightarrow P(2) \rightarrow P(3) \rightarrow \ldots$

Recursion

Base case: $f(0) = 0$

Recursive function: $f(n) = f(n-1) + 2n - 1$

$\therefore$ we can compute $f(n)$ for all $n \geq 0$

$\boxed{f(0)} \rightarrow f(1) \rightarrow f(2) \rightarrow f(3) \rightarrow \ldots$

We can conclude $P(n+1)$ if $P(n)$ is **T**…

We can compute $f(n+1)$ if $f(n)$ is known…

UNFOLDING THE RECURSION

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

1. Obtain $f(n)$ from $f(n-1)$
2. Below this, obtain $f(n-1)$ from $f(n-2)$
3. Repeat the pattern down to the base case…
4. Equate the sum of all of the LHS terms with the sum of all of the RHS terms…
   (we also might use the product instead of the sum)
5. Every LHS term except for $f(n)$ cancels with a corresponding RHS term—and $f(0) = 0$

We have a conjecture that needs a proof…

Notice we have ($\frac{1}{2} \times n$) combined terms here!

$$\begin{aligned}
f(n) &= \cancel{f(n-1)} + 2n - 1 \\
\cancel{f(n-1)} &= \cancel{f(n-2)} + 2n - 3 \\
\cancel{f(n-2)} &= \cancel{f(n-3)} + 2n - 5 \\
&\vdots \qquad \vdots \\
\cancel{f(3)} &= \cancel{f(2)} + 5 \\
\cancel{f(2)} &= \cancel{f(1)} + 3 \\
+ \quad \cancel{f(1)} &= f(0) + 1 \\
\hline
f(n) &= 1 + 3 + \ldots + 2n - 1
\end{aligned}$$

Gauss's trick… $f(n) = \frac{1}{2} \times n \times 2n = n^2$

PROVING OUR CONJECTURE

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

Can we prove our claim $P(n)$ that $f(n) = n^2$ (thereby removing the recursion)?

Proof. We prove by induction that $P(n)$ is **T** for $n \ge 0$.

1. **[Base case]** $P(0)$ claims $f(0) = 0^2 = 0$, which is **T** from the recursive definition.
2. **[Induction step]** We prove $P(n) \to P(n+1)$ for all $n \ge 0$ via a direct proof.

   Assume $f(n) = n^2$; we must prove that $f(n+1) = (n+1)^2$.

   LHS: $f(n+1) = \underbrace{f(n)}_{\text{induction hypothesis}} + 2(n+1) - 1$ ← from the recursive definition of $f(n)$

   $= n^2 + 2n + 1$

   $= (n+1)^2$, as was to be shown.
3. By induction, $P(n)$ is **T** for all $n \ge 0$. ■

UNFOLDING THE RECURSION – TOUGHER EXAMPLE

Given $f(n)$ below, first determine if $f(n)$ is well-defined…

$$f(n) = \begin{cases} 1 & n = 1 \\ f(n/2) + 1 & n > 1\text{, even} \\ f(n + 1) & n > 1\text{, odd} \end{cases}$$

…if well-defined, can we rewrite $f(n)$ without the recursion?

Tinker by writing out values of n and $f(n)$ until you see a pattern…

UNFOLDING THE RECURSION – TOUGHER EXAMPLE

$$f(n) = \begin{cases} 1 & n = 1 \\ f(n/2) + 1 & n > 1\text{, even} \\ f(n + 1) & n > 1\text{, odd} \end{cases}$$

n	1	2	3	4	5	6	7	8	9	10	11	12	13	…
$f(n)$	1	2	3	3	4	4	4	4	5	5	5	5	5	…

etc. etc. etc.

Arrows show how $f(n)$ is obtained, e.g., $f(2) \leftarrow f(1)$ and $f(3) \leftarrow f(4) \leftarrow f(2) \leftarrow f(1)$

Is there a path to every n from base case $n = 1$, making $f(n)$ well-defined?

Yes! And what pattern emerges…? How can we rewrite $f(n)$ here…?

UNFOLDING THE RECURSION — TOUGHER EXAMPLE

$$f(n) = \begin{cases} 1 & n = 1 \\ f(n/2) + 1 & n > 1, \text{even} \\ f(n+1) & n > 1, \text{odd} \end{cases}$$

n	1	2	3	4	5	6	7	8	9	10	11	12	13	…
$f(n)$	1	2	3	3	4	4	4	4	5	5	5	5	5	…

The pattern is logarithmic—we propose that $f(n) = 1 + \lceil \log_2 n \rceil$

Tinker with a few values to confirm that it appears to work…

…try $f(1) = 1 + \lceil \log_2 1 \rceil = 1 + 0 = 1$

…and $f(5) = 1 + \lceil \log_2 5 \rceil = 1 + 3 = 4$

We have a conjecture that needs a proof…

PROVING OUR CONJECTURE

$$f(n) = \begin{cases} 1 & n = 1 \\ f(n/2) + 1 & n > 1, \text{even} \\ f(n+1) & n > 1, \text{odd} \end{cases}$$

Can we prove our claim $P(n)$ that $f(n) = 1 + \lceil \log_2 n \rceil$ for all $n \geq 1$?

Proof. We prove by strong induction that $P(n)$ is **T** for $n \geq 1$.

1. **[Base case]** $P(1)$ claims $f(1) = 1 + \lceil \log_2 1 \rceil = 1 + 0 = 1$, which is **T**.
2. **[Induction step]** We prove $P(1) \wedge P(2) \wedge \ldots \wedge P(n) \rightarrow P(n+1)$ for $n \geq 1$.

   Assume $f(k) = 1 + \lceil \log_2 k \rceil$ for $1 \leq k \leq n$.

   We must prove $f(n+1) = 1 + \lceil \log_2 (n+1) \rceil$.

   Case 1. When $n + 1$ is even…

   Case 2. When $n + 1$ is odd…

Complete this (somewhat difficult) proof…

RECURRENCES

A *recurrence* is a recursive function defined on $\mathbb{N}$

We simplify the notation for $f(n)$ by defining a recurrence as A_n or F_n or T_n or etc.

$$A_1 = 0; \quad A_n = A_{n-1} + 2n - 1 \text{ (for } n > 1)$$

Defining Fibonacci as a recurrence is…

$$F_1 = 1; \quad F_2 = 1; \quad F_n = F_{n-1} + F_{n-2} \text{ (for } n > 2)$$

We can use recurrences to describe the runtime of recursive programs…

RECURRENCES AND RUNTIMES — EXAMPLE 1

```
int f( int n )
{
  if ( n == 0 ) return 1;
  else return 2 * f( n – 1 );
}
```

What does this function produce?

Assuming integers are unbounded, function $f(n)$ calculates 2^n for $n \geq 0$

Proof. We prove by induction that $f(n) = 2^n$ for $n \geq 0$.

1. **[Base case]** $f(0) = 2^0 = 1$, which is **T**.
2. **[Induction step]** Assume $f(n) = 2^n$ for $n \geq 0$.

   We must prove $f(n + 1) = 2^{n+1}$.

   LHS: $f(n + 1) = 2 \times f(n) = 2 \times 2^n = 2^{n+1}$, as was to be shown.
3. By induction, $P(n)$ is **T** for all $n \geq 0$. ■

What is the runtime of this function…?

Write this as a recurrence…

RECURRENCES AND RUNTIMES — EXAMPLE 1

```
int f( int n )
{
  if ( n == 0 ) return 1;
  else return 2 * f( n – 1 );
}
```

When $n = 0$, we have 2 operations…

…a *test* (if) and a *set* (return)

When $n = 1$, we have a test, a multiplication, a set, and the $f(0)$ case…

…a total of 5 operations

For general n with $n \geq 2$, the runtime of $f(n)$ is a test, a multiplication, and a set, plus the $f(n - 1)$ case—we can write the runtime recurrence as…

$$T_0 = 2; \quad T_n = T_{n\text{-}1} + 3 \text{ (for } n > 0)$$

Rewrite T_n without recursion…

…then prove this by induction…

RECURRENCES AND RUNTIMES – EXAMPLE 2

Assuming *is_even*() always takes 2 operations, write a recurrence for runtime T_n

```
int f( int n )
{
  if ( n == 0 ) return 1;
  elif ( is_even( n ) ) return f( n/2 ) * f( n/2 );
  else return 2 * f( n – 1 );
}
```

Next, rewrite T_n without recursion…

…then prove this by induction…

RECURRENCES AND RUNTIMES – EXAMPLE 2

```
int f( int n )
{
  if ( n == 0 ) return 1;
  elif ( is_even( n ) ) return f( n/2 ) * f( n/2 );
  else return 2 * f( n – 1 );
}
```

(assume *is_even*() always takes 2 operations)

For $n = 0$, we have 2 operations, so $T_0 = 2$

For $n = 1$, we have $T_1 = 9$, i.e., 2 tests, *is_even*(), a multiplication, a subtraction, a set, $f(0)$

For $n = 2$, we have 2 tests, *is_even*(), a multiplication, two divisions, a set, and two $f(1)$ calls

For $n = 3$, we have …

Finish rewriting T_n without recursion…

…see Problem 7.44

RECURSIVE SETS

Write a recursive definition for set S that contains the powers of 3, i.e., $S = \{ 3^0, 3^1, 3^2, 3^3, \ldots \}$

Recursion is so powerful, we can also use it to precisely construct sets…

Recursive definition of the natural numbers $\mathbb{N}$.

1. $1 \in \mathbb{N}$. **[basis]**
2. $x \in \mathbb{N} \rightarrow x + 1 \in \mathbb{N}$. **[constructor]**
3. Nothing else is in $\mathbb{N}$. **[minimality]**

$\mathbb{N} = \{ 1, 2, 3, \ldots \}$

Can you write a recursive definition for the set of integers $\mathbb{Z}$?

From rule 3, *minimality* means that $\mathbb{N}$ is the smallest set that satisfies rules 1 and 2

(minimality is implied, so we can omit rule 3…)

RECURSIVE SETS – FINITE BINARY STRINGS

Write a recursive definition for set S containing all binary string palindromes, e.g., 010, 11011, 000, …

Let ε be the *empty string* (similar to $\varnothing$ for sets)

Recursive definition of Σ^* (finite binary strings).

1. $\varepsilon \in \Sigma^*$. **[basis]**
2. $x \in \Sigma^* \rightarrow x \bullet 0 \in \Sigma^*$ and $x \bullet 1 \in \Sigma^*$. **[constructor]**

(note that • indicates concatenation)

$$\varepsilon \rightarrow 0, 1 \rightarrow 00, 01, 10, 11 \rightarrow 000, 001, 010, 011, 100, 101, 110, 111 \rightarrow \ldots$$

$$\Sigma^* = \{ \varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \ldots \}$$

WHAT NEXT…?

Be patient as Exam 1 is graded—we will have grades and review solutions **this week**

Problem Set 4 is due at recitations on October 12

- Covers recursion and proofs with recursive objects (Chapter 7)

Homework 3 will be posted Wednesday afternoon, due by 11:59PM on October 20

Practice! Tinker! Practice! Tinker! Practice! Tinker! Practice! Tinker! Practice!