Recursive Definition and Recurrence Relations

- Recursive Definition
  - Sequences
  - Sets
  - Functions
  - Algorithms
- Recurrence Relations
- Solution of Recurrence Relations
  - Proving Closed Form with Induction
  - Substitution Method

IMDAD ULLAH KHAN

Inductive or Recursive Definition

An inductive or recursive definition is just defining things in terms of simpler/smaller version(s) of itself

Explicitly define base case(s) and build upon that

- Recursively Defined Sequences
- Recursively Defined Sets
- Recursively Defined Functions
- Recursive Algorithms

Recursively Defined Sequences

$\{f_n\} = 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$

Fibonacci Numbers

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2} \qquad (n > 1)$$

$$f_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ f_{n-1} + f_{n-2} & n > 1 \end{cases}$$

Recursive Definition: Sequence

$$\{t_n\} = 0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, 120, 136$$

$$t_n = \begin{cases} 0 & \text{if } n = 0 \\ t_{n-1} + n & \text{if } n \geq 1 \end{cases}$$

Triangular Numbers

Recursive Definition: Sequence

Closed form of recurrence relation and almost every statement about recursively defined structures are usually proved using induction

$\{f_n\} = 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$

Fibonacci Numbers

$$\begin{aligned} f_0 &= 0 \\ f_1 &= 1 \\ f_n &= f_{n-1} + f_{n-2} \qquad (n > 1) \end{aligned}$$

$$f_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ f_{n-1} + f_{n-2} & n > 1 \end{cases}$$

Every third number is even

Recursive Definition: Sequence

Theorem

f_{3n} is even

Basis step $n = 0$: $f_{3n} = f_0 = 0$ is even

Inductive Hypothesis: Suppose $f_{3(n-1)}$ is even

Inductive Step: Using IH, show that f_{3n} is even

$$\begin{aligned} f_{3n} &= f_{3n-1} + f_{3n-2} \\ &= f_{3n-2} + f_{3n-3} + f_{3n-2} \\ &= 2f_{3n-2} + f_{3(n-1)} \end{aligned}$$

Hence f_{3n} is even □

Recursive Definition: Sets

The set of natural numbers, $\mathbb{N} = \{0, 1, 2, \dots\}$

Recursive definition of $\mathbb{N}$

1. $0 \in \mathbb{N}$
2. For all x, $[x \in \mathbb{N} \implies (x+1) \in \mathbb{N}]$
3. Nothing is in $\mathbb{N}$ unless it satisfies (1) and (2)

Why is condition (3) necessary?

otherwise $\{0, .7, 1, 1.7, 2, 2.7, 3, 3.7, \dots\}$ could qualify to be called $\mathbb{N}$

Recursively Defined Functions

Recursive definition of factorial function $n! = n(n-1)(n-2)\ldots(3)(2)(1)$

$f(n) = n!$

- $f(0) = 0! = 1$
- $f(n+1) = (n+1)f(n) = (n+1)n! = (n+1)!$

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

Recursively Defined Functions

Definition of exponentiation function

$$a^n = \underbrace{a \times a \times \ldots \times a}_{n \text{ times}}$$

A recursive definition of exponentiation

$$a^n = \begin{cases} a * a^{n-1} & \text{if } n > 1 \\ a & \text{if } n = 1 \\ 1 & \text{if } n = 0 \end{cases}$$

Another recursive definition of exponentiation called repeated squaring

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n > 1 \text{ is even} \\ a \cdot a^{(n-1)/2} \cdot a^{(n-1)/2} & \text{if } n \text{ is odd} \\ 1 & \text{if } n = 0 \end{cases}$$

Recursive Algorithm for Exponentiation

Input: a and $n \geq 0$
Output: a^n

$$a^n = \begin{cases} a * a^{n-1} & \text{if } n > 1 \\ a & \text{if } n = 1 \\ 1 & \text{if } n = 0 \end{cases}$$

Algorithm Computing a^n using the recursive definition

```
function REC-EXP(a,n)
   if n = 0 then
      return 1
   else if n = 1 then
      return a
   else
      return a * REC-EXP(a, n − 1)
```

Recursive Algorithm: Repeated Squaring

Input: a and $n \geq 0$
Output: a^n

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n > 1 \text{ is even} \\ a \cdot a^{(n-1)/2} \cdot a^{(n-1)/2} & \text{if } n \text{ is odd} \\ 1 & \text{if } n = 0 \end{cases}$$

Algorithm Computing a^n using repeated squaring

```
function REP-SQ-EXP(a,n)
    if n = 0 then
        return 1
    else if n > 0 AND n is even then
        z ← REP-SQ-EXP(a, n/2)
        return z * z
    else
        z ← REP-SQ-EXP(a, (n−1)/2)
        return a * z * z
```

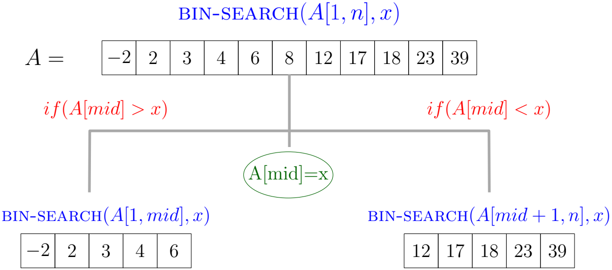
Recursive Algorithms: Searching in Sorted Array

Input: Sorted array A of n numbers and a number x

Output: Index of x in A if $x \in A$ or -1 if $x \notin A$

▷ Notice the input array is sorted

1. Compare $A[mid]$ with x
2. If not equal, eliminate the half where x cannot lie
3. Search x in the remaining half ▷ same problem but smaller



Recursive Algorithms: Binary Search

Input: Sorted array A of n numbers and a number x

Output: Index of x in A if $x \in A$ or -1 if $x \notin A$

Algorithm Binary Search for x in sorted array $A[st, \ldots, end]$

```
function BIN-SEARCH(A,st,end,x)
   if end < st then
      return −1
   else
      mid ← (end + st)/2
      if A[mid] = x then
         return mid                          ▷ If found return index
      else if A[mid] > x then
         return BIN-SEARCH(A, st, mid − 1, x)
      else
         return BIN-SEARCH(A, mid + 1, end, x)
```

Recursion

Defining a structure in terms of itself!

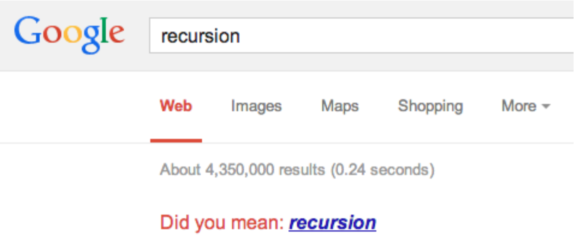


Figure: Google gets the joke!

Recursion

Defining a structure in terms of itself!

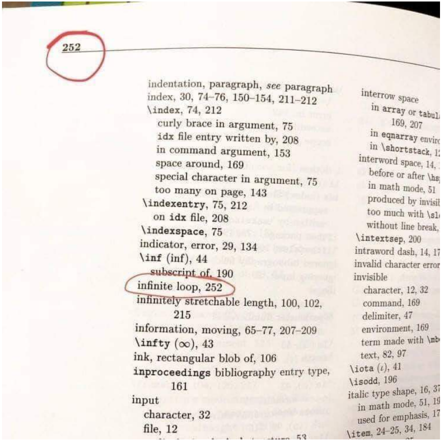


Figure: Learn infinite loops from this book!

Recursion: Base Case

Speaking of Programmers...

Is there a base case for recursive stress?