CS170: Discrete Methods in Computer Science
Spring 2025
Recursion and Iteration

Instructor: Shaddin Dughmi[1]

[1] These slides adapt some content from similar slides by Aaron Cote.

Recursion

Something is defined recursively if it is defined in terms of itself.

Fibonacci sequence

- $f_0 = 0$ and $f_1 = 1$. (base cases)
- $f_n = f_{n-1} + f_{n-2}$ for $n > 1$.

Recursion

Something is defined recursively if it is defined in terms of itself.

Fibonacci sequence

- $f_0 = 0$ and $f_1 = 1$. (base cases)
- $f_n = f_{n-1} + f_{n-2}$ for $n > 1$.

Factorials

- $1! = 1$
- $n! = n \cdot (n-1)!$

Recursion

Something is defined recursively if it is defined in terms of itself.

Fibonacci sequence

- $f_0 = 0$ and $f_1 = 1$. (base cases)
- $f_n = f_{n-1} + f_{n-2}$ for $n > 1$.

Factorials

- $1! = 1$
- $n! = n \cdot (n-1)!$

Binary Palindromes

A binary string is a palindrome if it is either

- The empty string, 1, or 0
- $1x1$ or $0x0$ where x is a palindrome

Many sorts of objects can be defined recursively: sequences, functions, algorithms (e.g. mergesort), sets, graphs, . . .

Recursive Algorithms

An algorithm is recursive if it calls itself (you can think of it as being defined in terms of itself)

E.g. Factorial Algorithm

Factorial(n):

- If $n = 1$ return 1
- Else return $n\times$ Factorial($n-1$)

E.g. Binary Search

BinarySearch(a,val,L,R)

- If $L > R$ return "Not Found"
- $m = \frac{L+R}{2}$
- If $a[m] == val$ return m;
- If $a[m] > val$ return Binarysearch(a,val,L,$m-1$)
- If $a[m] < val$ return Binarysearch(a,val,$m+1$,R)

Recursion vs Iteration

- A function is Tail-Recursive if there is one recursive call and its the last thing you do
  - You just return the result of the recursive call, instead of build on it
- Binary search is tail recursive, but factorial and mergesort are not.
- Tail recursive function are just iterative in disguise, but recursive form might be more convenient
- Every iterative function can be made tail recursive
- Some recursive functions (e.g. tail recursive) are easy to turn into iterative. But others are much more challenging (e.g. Mergesort).
  - Recursion really simplifies your life!

Tail Recursive vs Iterative

Iterative Find

Find(a,val):

- For $i = 0$ to $length(a) - 1$
  - If $val = a[i]$ return i;
- Return “not found”;

Tail Recursive vs Iterative

Iterative Find

Find(a,val):

- For $i = 0$ to $length(a) - 1$
  - If $val = a[i]$ return i;
- Return “not found”;

Recursive Find

RecFind(a,val, i):

- If $i > length(a) - 1$ return “not found”;
- Else if $val = a[i]$ return i;
- Else return RecFind(a,val,$i + 1$);

Tail Recursive vs Iterative

Iterative Find

Find(a,val):

- For $i = 0$ to $length(a) - 1$
  - If $val = a[i]$ return i;
- Return "not found";

Recursive Find

RecFind(a,val, i):

- If $i > length(a) - 1$ return "not found";
- Else if $val = a[i]$ return i;
- Else return RecFind(a,val,$i + 1$);

You would call RecFind(a,val,0).

Recursion, Induction, and Loop Invariants

To prove anything about a recursive object, you typically use induction

- We saw using induction to prove correctness and runtime of mergesort
- More generally, you prove what you want for the base case object, then induct using the recursive definition
- Since induction tracks the structure of the definition, we often call it structural induction

Recursion, Induction, and Loop Invariants

To prove anything about a recursive object, you typically use induction

- We saw using induction to prove correctness and runtime of mergesort
- More generally, you prove what you want for the base case object, then induct using the recursive definition
- Since induction tracks the structure of the definition, we often call it structural induction

For tail recursion, the inductive hypothesis is the same as a loop invariant in corresponding iterative implementation!

Loop Invariant for Iteration

A property that is preserved from iteration to iteration, from which what you want follows.