csci 210: Data Structures

Recursion
Summary

- **Topics**
  - recursion overview
  - simple examples
  - Sierpinski gasket
  - Hanoi towers
  - Blob check

- **READING:**
  - GT textbook chapter 3.5
Recursion

- In general, a method of defining a function in terms of its own definition
  - \( f(n) = f(n-1) + f(n-2) \)
  - \( f(0) = f(1) = 1; \)
- In programming, recursion is a call to the same method from a method
- Why write a method that calls itself?
  - a method to solve problems by solving easier instance of the same problem
- Recursive function calls can result in an infinite loop of calls
  - recursion needs a base-case in order to stop
  - \( f(0) = f(1) = 1; \)

- Recursion (repetitive structure) can be found in nature
  - shape of cells, leaves

- Recursion is a good problem solving approach
- Recursive algorithms
  - elegant
  - simple to understand and prove correct
  - easy to implement
Problem solving technique: Divide-and-Conquer
- break into smaller problems
- solve sub-problems recursively
- assemble solutions

recursive-algorithm(input) {
  //base-case
  if (isSmallEnough(input))
    compute the solution and return it
  else
    break input into simpler instances input1, input 2,...
    solution1 = recursive-algorithm(input1)
    solution2 = recursive-algorithm(input2)
    ...
    figure out solution to this problem from solution1, solution2,...
  return solution
}
Problem: write a function that computes the sum of numbers from 1 to n

```c
int sum (int n)
```

1. use a loop
2. recursively
Problem: write a function that computes the sum of numbers from 1 to n

```cpp
int sum (int n)
```

1. use a loop
2. recursively

```cpp
int sum (int n) {
    int s = 0;
    for (int i=0; i<n; i++)
        s += i;
    return s;
}
```

```cpp
int sum (int n) {
    int s;
    if (n == 0) return 0;
    //else
    s = n + sum(n-1);
    return s;
}
```

How does it work?
def sum(n):
    if n == 0:
        return 0
    return n + sum(n-1)

print(sum(10))
Recursion

• **How it works**
  - Recursion is no different than a function call
  - The system keeps track of the sequence of method calls that have been started but not finished yet (active calls)
    - order matters

• **Recursion pitfalls**
  - miss base-case
    - infinite recursion, stack overflow
  - no convergence
    - solve recursively a problem that is not simpler than the original one
Perspective

• Recursion leads to
  • compact
  • simple
  • easy-to-understand
  • easy-to-prove-correct
• solutions

• Recursion emphasizes thinking about a problem at a high level of abstraction

• Recursion has an overhead (keep track of all active frames). Modern compilers can often optimize the code and eliminate recursion.

• First rule of code optimization:
  • Don’t optimize it..yet.

• Unless you write super-duper optimized code, recursion is good

• Mastering recursion is essential to understanding computation.
Class–work

- Sierpinski gasket

- Fill in the code to create this pattern
Towers of Hanoi

Consider the following puzzle

- There are 3 pegs (posts) a, b, c
- n disks of different sizes
- each disk has a hole in the middle so that it can fit on any peg
- at the beginning of the game, all n disks are on peg a, arranged such that the largest is on the bottom, and on top sit the progressively smaller disks, forming a tower
- Goal: find a set of moves to bring all disks on peg c in the same order, that is, largest on bottom, smallest on top
  - constraints
    - the only allowed type of move is to grab one disk from the top of one peg and drop it on another peg
    - a larger disk can never lie above a smaller disk, at any time

PS: the legend says that the world will end when a group of monks, somewhere in a temple, will finish this task with 64 golden disks on 3 diamond pegs. Not known when they started.
Find the set of moves for $n=3$
Solving the problem for any $n$

- Think recursively
- Problem: move $n$ disks from A to C using B
- Can you express the problem in terms of a smaller problem?
- Subproblem:
  - move $n-1$ disks from A to C using B
Solving the problem for any $n$

- Think recursively
- Problem: move $n$ disks from A to C using B
- Can you express the problem in terms of a smaller problem?
- Subproblem:
  - move $n-1$ disks from A to C using B

Recursive formulation of Towers of Hanoi

- move $n$ disks from A to C using B
  - move top $n-1$ disks from A to B
  - move bottom disks from A to C
  - move $n-1$ disks from B to C using A

Correctness
- How would you go about proving that this is correct?
Hanoi-skeleton.java

• Look over the skeleton of the Java program to solve the Towers of Hanoi
• It’s supposed to ask you for n and then display the set of moves
  • no graphics

• finn in the gaps in the method
  public void move(sourcePeg, storagePeg, destinationPeg)
Correctness

• Proving recursive solutions correct is related to mathematical induction, a technique of proving that some statement is true for any n
  • induction is known from ancient times (the Greeks)

• Induction proof:
  • Base case: prove that the statement is true for some small value of n, usually n=1
  • The induction step: assume that the statement is true for all integers \( \leq n-1 \). Then prove that this implies that it is true for n.

• Exercise: try proving by induction that \( 1 + 2 + 3 + \ldots + n = n(n+1)/2 \)

• A recursive solution is similar to an inductive proof; just that instead of “inducting” from values smaller than n to n, we “reduce” from n to values smaller than n (think n = input size)
  • the base case is crucial: mathematically, induction does not hold without it; when programming, the lack of a base-case causes an infinite recursion loop

• proof sketch for Towers of Hanoi:
  • It works correctly for moving one disk (base case). Assume it works correctly for moving n-1 disks. Then we need to argue that it works correctly for moving n disks.
Analysis

• How close is the end of the world?

• Let’s estimate running time

• the running time of recursive algorithms is estimated using recurrent functions

• let $T(n)$ be the time it takes to compute the sequence of moves to move $n$ disks from one peg to another

• then based on the algorithm we have
  • $T(n) = 2T(n-1) + 1$, for any $n > 1$
  • $T(1) = 1$ (the base case)

• I can be shown by induction that $T(n) = 2^n - 1$
  • the running time is exponential in $n$

• Exercise:
  • 1GHz processor, $n = 64 \Rightarrow 2^{64} \times 10^{-9} = \ldots$ a log time; hundreds of years