CS 209
Data Structures and Mathematical Foundations

03 / 22 / 2024

Instructor: Michael Eckmann

Today's Topics

- Questions/Comments?
- More Recursion
- Change making algorithm and memoization applied

Recursion

- 1. have at least one base case that is not recursive
- 2. recursive case(s) must progress towards the base case
- 3. trust that your recursive call does what it says it will do (without having to unravel all the recursion in your head.)
- 4. try not to do redundant work. That is, in different recursive calls, don't recalculate the same info.

Recursion

- Let's write find_max iteratively and then again recursively

Recursion

- Change making algorithms.
  - Problem: have some amount of money for which you need to make change in the fewest coins possible.
  - You have unlimited numbers of coins C_1 ... C_N each with different values.
- example: make change for .63 and the coins you have are C_1 =.01, C_2 =.05, C_3 =.10, and C_4 =.25 only.
- We always assume we have .01 coin to guarantee a way to make change for any amounts.
- ideas?

Recursion

- Change making algorithms.
  - Problem: have some amount of money for which you need to make change in the fewest coins possible.
  - You have unlimited numbers of coins C_1 ... C_N each with different values.
- example: make change for .63 and the coins you have are C_1 =.01, C_2 =.05, C_3 =.10, and C_4 =.25 only.
- The algorithm that works for these denominations is a greedy algorithm (that is, one that makes an optimal choice at each step to achieve the optimal solution to the whole problem.) Let's write it in Python.

Recursion

- What if : make change for .63 and the coins you have are C_1 =.01, C_2 =.05, C_3 =.10, C_4 =.21 and C_5 =.25 only.
- A 21 cent piece comes into the picture.

Recursion

- What if : make change for .63 and the coins you have are C_1 =.01, C_2 =.05, C_3 =.10, C_4 =.21 and C_5 =.25 only.
- A 21 cent piece comes into the picture.
- The greedy algorithm doesn't work in this case because the minimum is 3 coins all of C_4 =.21 whereas the greedy algorithm would yield 2 .25's, 1 .10 and 3 .01's for a total of 6 coins.

Recursion

- So, we want to create a way to solve the minimum # of coins problem with n arbitrary coin denominations.
- A recursive strategy is:
  - BASE CASE: If the change K, we're trying to make is exactly equal to any coin denomination, then we only need 1 coin.
  - RECURSIVE STEP: Otherwise, we can say the fewest coins is the minimum of
    - 1 + fewestcoins(K – C1)
    - 1 + fewestcoins(K – C2)
    - .
    - .
    - or
    - 1 + fewestcoins(K – Cn)

Recursion

- split the total into parts and solve those parts recursively.
  - e.g.
  - fewcoins = 1 + fewestcoins(63-1=62)
  - Or
  - Fewcoins = 1 + fewestcoins(63-5=58)
  - Or
  - Fewcoins = 1 + fewestcoins(63-10=53)
  - Or
  - Fewcoins = 1 + fewestcoins(63-21=42)
  - Or
  - Fewcoins = 1 + fewestcoins(63-25=38)

Recursion

- split the total into parts and solve those parts recursively.
  - The base case of the recursion is when the change we are making is equal to one of the coins – hence 1 coin.
  - Otherwise recurse.
  - Why is this bad?

Recursion

- split the total into parts and solve those parts recursively.
  - The base case of the recursion is when the change we are making is equal to one of the coins – hence 1 coin.
  - Otherwise recurse.
  - Why is this bad? It makes many redundant calls.
  - Let's see (let's try to make change for some amounts with a Python implementation of this.)

Recursion

- The major problem with that change making algorithm is that it makes so many recursive calls and it duplicates work already done.
- But just like we did for fibonacci, we can use memoization.
- Instead of making recursive calls to figure out something that we already figured out we compute it once and save the value in a table for lookup when we need it later.