Section 1 Overview

Agenda

1. Review Introduction - More than 3 unexcused absence to section = NC, mini-assignments.
2. Analysis of algorithms (Big O)
   a. Movie_night
   b. sumList
3. Dynamic Programming
   a. Change problem
   b. Rope-cutting problem (optional)
4. Python tips and tricks

Pseudocode

Analysis of Algorithms

Movie Night

movie_night(x,y,z):
    for i in range 0 to x: // O(x)
        for j in range 0 to y: //O(y)
            print “LIGHTS CAMERA ACTION” //O(1)
    return 3*z //O(1)
answer: O(x*y)

Sum List

sumList(list): //of length n
s = 0
for element in list:
    s += element
return s
answer: O(n)
Dynamic Programming

Greedy Algorithm for Min Change

greedy_change(amt, denoms):
//remaining amount to make change for
remainder = amt
//output array of denominations (number of each denomination)
pieces = []
//for each denomination (starting with largest)
for k = 0 to denom.length:
    pieces[k] = remainder/denoms[k]
    remainder = remainder % denoms[k] //practice mod!
return sum(pieces)

Dynamic Programming for Min Change

def get_best_num_coins(amt, denoms):
    best_num_coins = []
    //number of coins needed for subproblem for amount 0
    best_num_coins[0] = 0
    // for each subproblem (each amount to make change for)
    for curr_amt = 1 to amt:
        // O(amt)
        min stores the current best min no. coins to make ‘curr_amt’ amount
        min_num_coins = infinity
        // for each denomination
        for coin_val in denoms: // O(denoms.length)
            if curr_amt >= coin_val:
                // if 1 + subproblem is better than current best
                if best_num_coins[curr_amt - coin_val] + 1 < min_num_coins:
                    // update the min_num_coins tracker
                    min_num_coins = best_num_coins[curr_amt-coin_val]+1
                    // store min no. of coins to make ‘curr_amt’ in the array
                    best_num_coins[curr_amt] = min_num_coins
        return best_num_coins[amt]

Dynamic Programming for Rope Cutting (Optional)

function maxProduct(length):
if(length < 2):
    return 0
if(length == 2):
    return 1
if(length == 3):
    return 2

int[] products = [length + 1]
// products[i] corresponds to cutting a rope of length i+1
// fill in products array with base cases
products[0] = 0
products[1] = 1
products[2] = 2
products[3] = 3

// populate from length 4 till length
for i in range(4, length):
    max = 0
    // use max[f(i)*f(i-j)] for 0<i<length needed
    for j in range(1, i/2):
        p = products[j] * products[i - j]
        if(max < p):  // update the max counter if needed
            max = p
    products[i] = max  // store the max @ len - i in products

return products[length]