# Change Making Problem

## Definition

* m coins
* value

Np hard

We will suppose w1 > w2 > ... > wn

## Canonical systems and the *greedy* method.

Explanation of the greedy method.

* Greedy method not necessarily optimal
* greedy method finds feasible solution when wn=1, otherwise not guaranteed.
* greedy method finds optimal when system canonical.

When wn = 1, non-canonical systems have counter example within  
 w[n-2] + 1 < x < w1 + w2.

Logic for greedy method

function greedy(coins, val)  
 local s = {}  
 for i = 1,#coins do  
 s[i] = math.floor(val/coins[i])  
 val = val - s[i]\*coins[i]  
 end  
 if val > 0 then  
 -- failed  
 s = nil   
 end   
 return s  
end

## General observations

* remove any coin from an optimal solution is also an optimal solution.

## Recursive approach

Given a value to make n and a number of coins m and a list of coin denominations coins. We can define the optimal number of coins recursively.

C(0,m) = 0  
C(n,m) = infinity, if n < 0 or m ≼ 1  
C(n,m) = min C(n, m-1), C(n-wm, m) + 1

Here we consider the coins in ascending order.

function minChange(coins, n, m) -- start with minChange(coins, val, #coins)  
 if n == 0 then return 0 end  
 if n < 0 or m <= 1 then  
 return infinity  
 end  
 local a = minChangeAux(coins, n, m-1)  
 local b = minChangeAux(coins, n-coins[#coins-m+1],m) + 1  
 return math.min(a,b)  
end

## Dynamic Programming Method

* Is O(val\*m)

function minChangeDynamic(coins, val)  
 local k = #coins  
 local c = {}  
 local s = {}  
 table.sort(coins, function (a,b) return a > b end)  
 c[0] = 0  
 for p = 1,val do  
 local min = val + 1  
 local coin  
 for i = 1,k do  
 if coins[i] <= p then  
 local t = 1 + c[p-coins[i]]  
 if t < min then  
 min = t  
 coin = i  
 end  
 end  
 end  
 c[p] = min  
 s[p] = coin  
 end  
 local sol = {}  
 for i = 1,k do sol[i] = 0 end  
 while val > 0 do  
 local ci = s[val]  
 if ci then  
 sol[ci] = sol[ci] + 1  
 val = val - coins[ci]  
 else  
 -- insoluble  
 return nil  
 end  
 end  
 return sol  
end

* Hugh's defunct "forwards" method?

## lower Bounds

* ceil(v/w1) is a lower bound
* if m > 1, then floor(v/w1) + ceil((v mod w1)/w2)
* if m > 2, then

MIN  
 floor(v/w1) + floor((v mod w1)/w2) + ceil((v mod w1 mod w2)/w3)  
 floor(v/w1) + floor((v mod w1)/w2) - 1 + ceil((v mod w1 mod w2 + w1)/w2)

The first case is the bound when x2 <= floor((v mod w1)/w2), ie when a greedy amount of w2 is taken.

The latter when x2 > floor((v mod w1)/w2), ie when less than greedy w1 is taken, ie x1 <= floor(v/w1)-1.

function lowbound(coins, val)  
 local m = #coins  
 local lb = math.floor(val/coins[1])  
 if m > 1 then  
 local v = val - lb\*coins[1]  
 local q = math.floor(v/coins[2])  
 lb = lb + q  
 if m > 2 then  
 v = v - q\*coins[2]  
 lb = math.min(lb + math.ceil(v/coins[3]),   
 lb - 1 + math.ceil((v + coins[1])/coins[2]))  
 end  
 end  
 return lb  
end

## upper bounds

* when the greedy methods succeeds, count(greedy) is an upper bound.
* when Wn = 1, greedy succeeds.
* when greedy fails, can use ceil(val/Wn).

function limit(coins, val)  
 local lim = 0  
 local q, cj  
 local v = val  
 for j = 1,#coins do  
 cj = coins[j]  
 q = math.floor(v/cj)  
 v = v - q\*cj  
 lim = lim + q  
 end  
 if v > 0 then lim = math.floor(val/coins[#coins]) end  
 return lim + 1  
end

### Tests for Canonical

* number of coins <= 1
* if Wn = 1 and n <= 2 then canonical
* if Wn = 1 and n = 3 then canonical if,  
   r = w1 mod w2  
   r = 0 or w2 - floor(w1/w2) ≼ r

function iscanonical(coins)  
 local res = false -- not or don't know  
 local n = #coins  
 if n == 1 then res = true  
 else  
 if coins[n] == 1 then  
 if n < 3 then res = true   
 elseif n == 3 then  
 local q = math.floor(coins[1]/coins[2])  
 local r = coins[1] - q\*coins[2]  
 res = r == 0 or r >= coins[2] - q  
 end  
 end  
 end  
 return res  
end

* The Regularity Theorem of Magazine-Nemhauser-Trotter, 1975

Opt(S) = G(S) for all S if  
 G(St - Wt) ≼ mt − 1 ∀ t = 2..n

Where mt = ceil(Wt-1/wt) and St = mt\*Wt

function mnt(coins)  
 -- Theorem of Magazine-Nemhauser-Trotter, 1975  
 local can = false  
 -- ensure descending  
 table.sort(coins, function (a,b) return a > b end)   
 local n = #coins  
 if n < 2 then return true end  
 if coins[n] == 1 then  
 can = true  
 if n >= 3 then  
 for t = 2,n do  
 local mt = math.ceil(coins[t-1]/coins[t])  
 local st = mt\*coins[t]  
 if countCoins(greedy(coins, st - coins[t-1])) >= mt then  
 can = false  
 break  
 end  
 end  
 end  
 end  
 return can  
end

Example Output

{100, 50, 20, 10, 5, 2, 1} MNT canonical  
{100, 50, 20, 10, 5, 2, 1} MNT canonical  
{75, 50, 20, 10, 5, 2, 1} MNT non-canonical  
{25, 21, 10, 5, 2, 1} MNT non-canonical  
{23, 11, 7, 5, 3, 2} MNT non-canonical  
{11, 8, 5, 4, 1} MNT non-canonical  
{65536, 32768, 16384, 8192, 4096, 2048, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1} MNT canonical  
{7, 6, 4, 3} MNT non-canonical  
{3, 2, 1} MNT canonical  
{5, 3, 2} MNT non-canonical  
{74, 50, 1} MNT non-canonical  
{74, 50, 1} MNT non-canonical  
{1} MNT canonical  
{11, 8, 5, 4, 1} MNT non-canonical  
{60, 48, 24, 12, 6, 2, 1} MNT non-canonical  
{40, 39, 24, 23, 17, 16} MNT non-canonical  
{51, 50, 15, 13, 2, 1} MNT non-canonical  
{8, 4, 2} MNT non-canonical  
{2, 1, 0} MNT non-canonical  
{0} MNT canonical  
{40, 39, 24, 23, 17, 16, 1} MNT non-canonical  
{26, 19, 9, 3, 1} MNT non-canonical  
{26, 19, 9, 3, 1} MNT non-canonical  
{31, 29, 23, 19, 17, 13, 11, 7, 5, 3} MNT non-canonical  
{33, 17, 9, 5, 3, 2} MNT non-canonical  
{23, 3} MNT non-canonical

## Truncated Recursive Method

Idea is to truncate the search, pruning worst cases against the ongoing best limit.

First version:

function changerec1(coins, s, i, val, lim)  
 while coins[i] > val do  
 i = i + 1  
 if i > #coins then return nil end  
 end  
 local ci = coins[i]  
 if math.ceil(val/ci) + countCoins(s) >= lim then -- lower bound + existing length, can't beat lim  
 return nil  
 end  
 local s2 = copy(s)  
 s2[i] = s2[i] + 1  
 -- if adding the biggest coins makes a solution, it is optimal from here.   
 -- otherwise, if not last coin, try take it or skip it.  
 if val > ci then  
 s2 = changerec1(coins, s2, i, val - ci, lim)  
 if s2 then lim = countCoins(s2) end  
 i = i + 1  
 if i <= #coins then  
 -- try skipping this coin choice for a better solution than `lim'  
 local s3 = changerec1(coins, s, i, val, lim)  
 if s3 then -- solution better than `lim'  
 return s3  
 end  
 end  
 end  
 return s2  
end

Improved version:

* add an impossibility cutoff, which prevents algorithm from stalling on unsolvable problems.
* use second order lower bound (third order makes minimal difference).
* unroll leaf case in recursion.
* code improved to avoid copying partial solutions.
* take constant parameters, coins list, partial solution and limit, out of recursion.

function changerec(i, val)  
 -- version, eliminates recursion for the last coin  
 -- gcoins, s, cutoff and lim are global  
 local nc = #gcoins  
 while gcoins[i] > val do  
 i = i + 1  
 if i > nc then  
 cutoff[val] = -1  
 return false  
 end  
 end  
 local more = i < nc  
 local ci = gcoins[i]  
 local ni = math.floor(val/ci)  
 local lb = ni   
 if more then  
 lb = lb + math.ceil((val - lb\*ci)/gcoins[i+1])  
 ni = 1  
 end  
 local c = countCoins(s)  
 if lb + c >= lim then -- lower bound + existing length, can't beat lim  
 cutoff[val] = -1  
 return false  
 end  
 if val == ni\*ci then  
 -- if adding the biggest coins makes a solution, it is optimal from here.  
 lim = c + ni  
 best = copy(s) -- update best and restore sol  
 best[i] = best[i] + ni  
 return true  
 end  
 local s2 = false  
 -- if not last coin, try take it or skip it.  
 if more then  
 if cutoff[val - ci] >= 0 then -- try taking first coin  
 s[i] = s[i] + 1  
 s2 = changerec(i, val - ci)   
 s[i] = s[i] - 1 -- restore  
 end  
 -- try skipping this coin choice for a better solution than `lim'  
 if changerec(i + 1, val) then s2 = true end  
 end  
 if not s2 then cutoff[val] = -1 end  
 return s2  
end

## Testing

Using the above method after checking with the MNT test, we get the following example output for specific tests.

make value 57 with coins {100, 50, 20, 10, 5, 2, 1} = 50x1,5x1,2x1  
make value 36 with coins {100, 50, 20, 10, 5, 2, 1} = 20x1,10x1,5x1,1x1  
make value 101 with coins {75, 50, 20, 10, 5, 2, 1} (trials=11) = 50x2,1x1  
make value 63 with coins {25, 21, 10, 5, 2, 1} (trials=15) = 21x3  
make value 75 with coins {23, 11, 7, 5, 3, 2} (trials=11) = 23x3,3x2  
make value 29 with coins {11, 8, 5, 4, 1} (trials=13) = 11x1,8x1,5x2  
make value 99999 with coins {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536} = 65536x1,32768x1,1024x1,512x1,128x1,16x1,8x1,4x1,2x1,1x1  
make value 8 with coins {7, 6, 4, 3} (trials=7) = 4x2  
make value 9999 with coins {3, 2, 1} = 3x3333  
make value 9999 with coins {5, 3, 2} (trials=4001) = 5x1999,2x2  
make value 1849 with coins {74, 50, 1} (trials=469) = 74x2,50x34,1x1  
make value 100 with coins {74, 50, 1} (trials=5) = 50x2  
make value 99 with coins {1} = 1x99  
make value 29 with coins {1, 4, 5, 8, 11} (trials=13) = 11x1,8x1,5x2  
make value 96 with coins {1, 2, 6, 12, 24, 48, 60} (trials=7) = 48x2  
make value 100 with coins {16, 17, 23, 24, 39, 40} (trials=106) = 17x4,16x2  
make value 76 with coins {51, 50, 15, 13, 2, 1} (trials=19) = 50x1,13x2  
make value 9 with coins {2, 4, 8} (trials=6) = insoluble  
make value 9 with coins {0, 1, 2} (trials=9) = 2x4,1x1  
make value 2 with coins {0} = insoluble  
make value 9999 with coins {1, 16, 17, 23, 24, 39, 40} (trials=499) = 40x249,39x1  
make value 65 with coins {1, 3, 9, 19, 26} (trials=13) = 26x1,19x2,1x1  
make value 66 with coins {1, 3, 9, 19, 26} (trials=19) = 19x3,9x1  
make value 1000 with coins {3, 5, 7, 11, 13, 17, 19, 23, 29, 31} (trials=127) = 31x32,5x1,3x1  
make value 1000 with coins {2, 3, 5, 9, 17, 33} (trials=65) = 33x30,5x2  
make value 24 with coins {23, 3} (trials=3) = 3x8

Try some random "bulk" tests.

Pick 100 random distinct coins from [1,400] to sum to floor(Sum(wi)/2). Compare against dynamic method.

dynamic method, order 955900  
396x23,371x1,80x1  
truncated recursive method  
make value 9559 with coins {396, 390, 381, 380, 377, 373, 371, 369, 363, 360, 349, 344, 342, 340, 333, 329, 322, 315, 314, 311, 307, 298, 292, 289, 284, 272, 271, 270, 265, 263, 261, 259, 256, 253, 252, 249, 248, 245, 237, 236, 230, 228, 222, 210, 203, 199, 192, 187, 186, 185, 184, 180, 179, 178, 172, 171, 162, 156, 153, 150, 147, 144, 142, 140, 129, 126, 120, 115, 109, 106, 105, 104, 98, 94, 88, 87, 85, 84, 80, 77, 75, 66, 64, 62, 54, 53, 49, 48, 46, 45, 44, 41, 40, 37, 33, 31, 27, 14, 9, 7} (trials=69) = 396x23,371x1,80x1  
  
dynamic method, order 1070000  
399x25,371x1,354x1  
truncated recursive method  
make value 10700 with coins {399, 389, 386, 385, 384, 383, 382, 379, 378, 377, 371, 370, 369, 368, 365, 363, 360, 356, 354, 353, 344, 338, 332, 328, 317, 311, 306, 305, 304, 300, 297, 287, 283, 277, 276, 272, 269, 265, 262, 259, 256, 255, 237, 230, 229, 225, 223, 222, 219, 217, 215, 210, 203, 200, 189, 186, 184, 180, 178, 177, 175, 174, 172, 170, 167, 166, 154, 153, 152, 146, 139, 136, 127, 121, 120, 112, 107, 103, 100, 98, 91, 87, 83, 78, 65, 64, 63, 59, 57, 53, 52, 44, 41, 36, 29, 23, 20, 17, 7, 1} (trials=81) = 399x25,371x1,354x1  
  
dynamic method, order 948600  
400x22,397x1,289x1  
truncated recursive method  
make value 9486 with coins {400, 397, 382, 377, 373, 371, 370, 358, 357, 354, 347, 340, 334, 330, 322, 314, 304, 299, 298, 297, 296, 294, 291, 290, 289, 282, 281, 279, 278, 270, 268, 264, 258, 257, 252, 249, 248, 246, 242, 239, 226, 225, 208, 207, 206, 205, 203, 200, 198, 192, 190, 183, 179, 178, 174, 169, 166, 153, 152, 146, 145, 136, 135, 134, 133, 123, 121, 117, 114, 113, 100, 99, 96, 95, 93, 92, 89, 86, 78, 71, 70, 68, 64, 61, 55, 52, 49, 45, 44, 43, 32, 31, 29, 28, 24, 20, 17, 9, 3, 1} (trials=67) = 400x22,397x1,289x1  
  
dynamic method, order 1070700  
398x25,390x1,367x1  
truncated recursive method  
make value 10707 with coins {398, 397, 393, 390, 385, 384, 376, 374, 367, 365, 363, 353, 350, 347, 345, 330, 329, 328, 327, 326, 314, 309, 307, 303, 301, 293, 290, 289, 286, 278, 276, 275, 274, 273, 272, 267, 261, 260, 259, 253, 249, 243, 240, 239, 238, 235, 234, 233, 230, 229, 224, 220, 210, 208, 207, 201, 197, 196, 195, 190, 189, 183, 177, 176, 175, 168, 164, 163, 158, 145, 144, 139, 134, 132, 123, 122, 120, 117, 113, 109, 108, 106, 101, 95, 94, 93, 80, 76, 68, 62, 59, 46, 41, 36, 33, 27, 23, 13, 10, 7} (trials=83) = 398x25,390x1,367x1  
  
dynamic method, order 1020100  
396x24,379x1,318x1  
truncated recursive method  
make value 10201 with coins {396, 393, 386, 383, 380, 379, 376, 374, 370, 369, 366, 361, 359, 356, 355, 351, 344, 342, 339, 331, 318, 310, 308, 307, 296, 293, 290, 288, 287, 284, 278, 275, 268, 264, 263, 262, 258, 257, 256, 243, 242, 238, 233, 231, 229, 223, 221, 219, 215, 208, 207, 201, 200, 194, 191, 190, 189, 181, 177, 172, 170, 166, 165, 159, 157, 152, 148, 145, 143, 136, 128, 125, 119, 118, 109, 104, 103, 95, 82, 81, 72, 68, 64, 61, 59, 58, 51, 49, 46, 37, 34, 26, 25, 18, 17, 11, 10, 7, 5, 3} (trials=63) = 396x24,379x1,318x1  
  
dynamic method, order 1034100  
400x24,390x1,351x1  
truncated recursive method  
make value 10341 with coins {400, 392, 390, 388, 386, 385, 384, 381, 378, 373, 370, 369, 362, 353, 351, 350, 347, 328, 327, 325, 320, 317, 315, 313, 303, 300, 295, 292, 291, 287, 286, 283, 278, 270, 269, 268, 265, 262, 261, 258, 257, 248, 239, 237, 236, 235, 225, 222, 221, 212, 211, 210, 205, 204, 185, 181, 178, 177, 174, 167, 165, 160, 156, 155, 150, 147, 146, 143, 141, 138, 134, 127, 112, 106, 105, 101, 100, 89, 88, 84, 83, 79, 70, 68, 60, 57, 56, 54, 51, 39, 37, 33, 32, 30, 27, 21, 17, 16, 8, 2} (trials=65) = 400x24,390x1,351x1  
  
dynamic method, order 981200  
399x23,392x1,243x1  
truncated recursive method  
make value 9812 with coins {399, 392, 387, 384, 381, 376, 374, 370, 367, 365, 359, 351, 346, 342, 336, 334, 328, 326, 323, 312, 310, 308, 303, 302, 301, 292, 289, 286, 283, 277, 271, 255, 254, 250, 248, 246, 245, 243, 242, 241, 235, 229, 228, 224, 223, 221, 220, 213, 205, 204, 197, 194, 188, 185, 184, 170, 169, 165, 163, 162, 154, 153, 149, 144, 138, 131, 129, 124, 123, 114, 112, 108, 100, 95, 92, 90, 82, 77, 69, C-c C-cchangemake.lua:420: interrupted!  
stack traceback:  
 changemake.lua:185: in function 'minChange2'  
 changemake.lua:420: in function 'randomtest'  
 changemake.lua:453: in function 'randomtests'  
 stdin:1: in main chunk  
 [C]: ?  
68, 67, 66, 60, 57, 55, 53, 50, 48, 46, 43, 41, 36, 33, 32, 31, 30, 28, 9, 7, 4} (trials=55) = 399x23,392x1,243x1  
  
dynamic method, order 1078100  
396x27,89x1  
truncated recursive method  
make value 10781 with coins {396, 393, 392, 391, 390, 383, 382, 381, 379, 378, 376, 375, 374, 373, 370, 368, 361, 360, 359, 355, 350, 338, 330, 321, 317, 315, 312, 310, 308, 306, 305, 300, 297, 296, 292, 283, 277, 273, 271, 270, 267, 260, 247, 246, 240, 238, 237, 234, 227, 223, 217, 212, 210, 209, 207, 194, 191, 185, 180, 173, 172, 169, 163, 158, 154, 151, 150, 140, 139, 126, 125, 120, 119, 116, 115, 110, 108, 107, 99, 95, 91, 89, 86, 85, 83, 76, 61, 56, 54, 52, 32, 30, 29, 28, 23, 21, 11, 10, 4, 1} (trials=55) = 396x27,89x1  
  
dynamic method, order 1088100  
393x26,391x1,272x1  
truncated recursive method  
make value 10881 with coins {393, 391, 390, 388, 381, 377, 376, 372, 371, 370, 369, 366, 365, 363, 359, 358, 354, 351, 350, 343, 340, 338, 336, 331, 330, 328, 326, 319, 317, 310, 308, 303, 296, 292, 286, 285, 280, 275, 274, 272, 269, 267, 266, 257, 255, 242, 237, 235, 230, 221, 220, 211, 210, 206, 203, 200, 196, 195, 189, 185, 181, 175, 166, 163, 162, 160, 147, 145, 144, 143, 138, 128, 126, 125, 123, 118, 108, 106, 104, 103, 101, 92, 90, 81, 79, 65, 60, 59, 57, 56, 55, 49, 45, 22, 15, 11, 10, 9, 8, 7} (trials=73) = 393x26,391x1,272x1  
  
dynamic method, order 996000  
397x25,35x1  
truncated recursive method  
make value 9960 with coins {397, 394, 393, 391, 390, 387, 383, 378, 375, 370, 367, 365, 358, 356, 355, 354, 350, 342, 341, 335, 328, 310, 306, 302, 295, 290, 288, 287, 285, 283, 274, 263, 262, 254, 252, 246, 243, 240, 239, 237, 236, 232, 229, 222, 212, 211, 209, 208, 207, 205, 198, 196, 193, 185, 184, 180, 179, 177, 165, 159, 158, 150, 148, 144, 143, 140, 139, 131, 124, 121, 118, 117, 112, 107, 97, 94, 90, 82, 80, 79, 72, 63, 58, 55, 52, 49, 43, 38, 36, 35, 32, 31, 30, 27, 26, 17, 15, 13, 2, 1} (trials=51) = 397x25,35x1  
  
dynamic method, order 989500  
398x23,375x1,366x1  
truncated recursive method  
make value 9895 with coins {398, 387, 376, 375, 374, 371, 369, 366, 357, 355, 353, 350, 344, 341, 331, 330, 329, 328, 321, 319, 316, 315, 303, 300, 299, 295, 294, 293, 292, 290, 288, 286, 284, 269, 268, 261, 258, 248, 237, 235, 228, 227, 223, 221, 216, 211, 208, 207, 202, 199, 197, 189, 186, 185, 177, 174, 171, 168, 166, 158, 155, 152, 148, 143, 140, 137, 131, 128, 122, 121, 119, 114, 111, 108, 98, 92, 90, 86, 85, 79, 76, 75, 68, 67, 65, 62, 55, 51, 45, 39, 38, 36, 31, 30, 25, 22, 13, 11, 3, 2} (trials=57) = 398x23,375x1,366x1  
  
dynamic method, order 1043200  
400x25,394x1,38x1  
truncated recursive method  
make value 10432 with coins {400, 398, 395, 394, 387, 385, 378, 373, 370, 365, 362, 360, 357, 354, 349, 345, 344, 341, 340, 335, 332, 331, 328, 325, 322, 319, 314, 313, 312, 304, 299, 298, 286, 276, 274, 272, 268, 264, 255, 250, 248, 247, 241, 239, 237, 232, 230, 229, 224, 216, 205, 202, 195, 188, 186, 184, 174, 171, 168, 162, 155, 146, 137, 136, 135, 134, 133, 130, 127, 126, 125, 123, 119, 109, 107, 106, 103, 101, 100, 96, 95, 92, 88, 78, 76, 67, 66, 54, 53, 47, 45, 43, 38, 31, 29, 25, 15, 12, 8, 2} (trials=78) = 400x25,394x1,38x1

etc.