Lesson 1: Week 9

Created: 2024-05-27T01:25:50.005439+10:00

Exercise 1: Canonical coin systems

Write a program that prompts the user for an amount, and outputs the minimal number of coins needed to yield that amount, as well as the detail of how many coins of each value are used. The available coins have a face value which is one of \$1, \$2, \$5, \$10, \$20, \$50, and \$100.Insert your code into canonical_coin_systems.py.Here are examples of interactions: $ python3 canonical_coin_systems.py Input the desired amount: 10 1 banknote is needed. The detail is: $10: 1 $ python3 canonical_coin_systems.py Input the desired amount: 739 12 banknotes are needed The detail is: $100: 7 $20: 1 $10: 1 $5: 1 $2: 2 $ python3 canonical_coin_systems.py Input the desired amount: 35642 359 banknotes are needed The detail is: $100: 356 $20: 2 $2: 1

Exercise 2: Unit fractions

Let N and D be two strictly positive integers with N\frac{N}{D} = \frac{1}{d_1}+\frac{1}{d_2}+\dots+\frac{1}{d_k}$.There are actually infinitely many such representations. Indeed, since$1=\frac{1}{2}+\frac{1}{3}+\frac{1}{6}$if $\frac{N}{D} = \frac{1}{d_1}+\frac{1}{d_2}+\dots+\frac{1}{d_k}$ then also$\frac{N}{D} = \frac{1}{d_1}+\frac{1}{d_2}+\dots+\frac{1}{d_{k-1}}+\frac{1}{2d_k}+\frac{1}{3d_k}+\frac{1}{6d_k}$.One particular representation is obtained by a method proposed by Fibonacci, in the form of a greedy algorithm. Suppose that N/D cannot be simplified, that is, N and D have no other common factor but 1. If $N=1$ then we are done, so suppose otherwise. Let d_1 be the smallest integer such that $\frac{N}{D}$ can be written as $\frac{1}{d_1}+f_1$, with f_1 necessarily strictly positive by assumption. Looking for the smallest d_1 is what makes the algorithm greedy. Of course, d_1 is equal to $D\div N + 1$. By the choice of d_1, $\frac{1}{d_1-1}>\frac{N}{D}$, hence $D>N(d_1-1)$, hence $N>Nd_1-D$. Since f_1 is equal to $\frac{N}{D}-\frac{1}{d_1}=\frac{Nd_1-D}{Dd_1}$, it follows that $\frac{N}{D}$ can be written as $\frac{1}{d_1}+\frac{N_1}{D_1}$ with N_11 then the same argument allows one to greedily find $d_2>d_1$ such that for some strictly positive integers N_2 and D_2, $\frac{N}{D}$ can be written as $\frac{1}{d_1}+\frac{1}{d_2}+\frac{N_2}{D_2}$ with N_21 then the same argument allows one to greedily find $d_3>d_2$ such that for some strictly positive integers N_3 and D_3, $\frac{N}{D}$ can be written as $\frac{1}{d_1}+\frac{1}{d_2}+\frac{1}{d_3}+\frac{N_3}{D_3}$ with $N_3The number of summands in the sum of unit fractions given by Fibonacci's method is not always minimal: it is sometimes possible to decompose $\frac{N}{D}$ as sum of unit fractions with fewer summands. For instance, Fibonacci's method yields$\frac{4}{17} = \frac{1}{5} + \frac{1}{29} + \frac{1}{1233} + \frac{1}{3039345}$whereas $\frac{4}{17}$ can be written as a sum of 3 unit fractions, actually in 4 possible ways:$\frac{4}{17} = \frac{1}{5} + \frac{1}{30} + \frac{1}{510}$$\frac{4}{17} = \frac{1}{5} + \frac{1}{34} + \frac{1}{170}$$\frac{4}{17} = \frac{1}{6} + \frac{1}{15} + \frac{1}{510}$$\frac{4}{17} = \frac{1}{6} + \frac{1}{17} + \frac{1}{102}$Complete the program unit_fractions.py so as to have the functionality of the two functions:fibonacci_decomposition(N, D), that takes two strictly positive integers N and D as arguments, and writes N/D as a sum of unit fractions following Fibonacci method, plus an integer in case $N ≥ D$ (in a unique way)shortest_length_decompositions(N, D), that also takes two strictly positive integers N and D as arguments, and writes N/D as a sum of unit fractions with a minimal number of summands, plus an integer in case N ≥ D (in possibly many ways)Here are possible interactions: >>> from unit_fractions import * >>> fibonacci_decomposition(1, 521) 1/521 = 1/521 >>> fibonacci_decomposition(521, 521) 521/521 = 1 >>> fibonacci_decomposition(521, 1050) 521/1050 = 1/3 + 1/7 + 1/50 >>> fibonacci_decomposition(1050, 521) 1050/521 = 2 + 1/66 + 1/4913 + 1/33787684 + 1/2854018941421956 >>> fibonacci_decomposition(6, 7) 6/7 = 1/2 + 1/3 + 1/42 >>> shortest_length_decompositions(6, 7) 6/7 = 1/2 + 1/3 + 1/42 >>> fibonacci_decomposition(8, 11) 8/11 = 1/2 + 1/5 + 1/37 + 1/4070 >>> shortest_length_decompositions(8, 11) 8/11 = 1/2 + 1/5 + 1/37 + 1/4070 8/11 = 1/2 + 1/5 + 1/38 + 1/1045 8/11 = 1/2 + 1/5 + 1/40 + 1/440 8/11 = 1/2 + 1/5 + 1/44 + 1/220 8/11 = 1/2 + 1/5 + 1/45 + 1/198 8/11 = 1/2 + 1/5 + 1/55 + 1/110 8/11 = 1/2 + 1/5 + 1/70 + 1/77 8/11 = 1/2 + 1/6 + 1/17 + 1/561 8/11 = 1/2 + 1/6 + 1/18 + 1/198 8/11 = 1/2 + 1/6 + 1/21 + 1/77 8/11 = 1/2 + 1/6 + 1/22 + 1/66 8/11 = 1/2 + 1/7 + 1/12 + 1/924 8/11 = 1/2 + 1/7 + 1/14 + 1/77 8/11 = 1/2 + 1/8 + 1/10 + 1/440 8/11 = 1/2 + 1/8 + 1/11 + 1/88 8/11 = 1/3 + 1/4 + 1/7 + 1/924 >>> fibonacci_decomposition(4, 17) 4/17 = 1/5 +…

Exercise 3: Diophantine equations

We consider Diophantine equations of the form \$ax+by=c\$ with \$a\$ and \$b\$ both not equal to 0. We will represent such an equation as a string of the form ax+by=c or ax-by=c where a and c are nonzero integer literals (not preceded by + in case they are positive) and where b is a strictly positive integer literal (not preceded by +), possibly with spaces anywhere at the beginning, at the end, and around the +, - and = characters. The equation \$ax+by=c\$ has a solution iff \$c\$ is a multiple of \$\gcd(a,b)\$. In case \$c\$ is indeed a multiple of \$\gcd(a,b)\$, then \$ax+by=c\$ has has infinitely many solutions, namely, all pairs \$(x,y)\$ of the form\$\left(x_0 + \frac{\mathrm{lcm}(a,b)}{a}n,y_0 - \frac{\mathrm{lcm}(a,b)}{b}n\right)\$for arbitrary integers \$n\$, where \$\mathrm{lcm}(a,b)\$ denotes the least common multiplier of \$a\$ and \$b\$, and where \$(x_0,y_0)\$ is a solution to the equation. That particular solution can be derived from the extended Euclidian algorithm, that yields not only \$\gcd(a,b)\$ but also a pair of Bézout coefficients, namely, two integers \$x\$ and \$y\$ with \$ax+by=\gcd(a,b)\$. To normalise the representation of the solutions, we rewrite the equation above as\$\left(x_0 + \frac{\mathrm{lcm}(a,b)}{|a|}n,y_0 - \mathrm{sign}(a)\frac{\mathrm{lcm}(a,b)}{b}n\right)\$where \$\mathrm{sign}(a)\$ is 1 if \$a\$ is positive and -1 if \$a\$ is negative, and we impose that the pair \$(x_0,y_0)\$ is such that \$x_0\$ is nonnegative and minimal.Write a Python program diophantine_equation.py that defines a function diophantine() that prints out whether the equation provided as argument has a solution, and in case it does, prints out the normalised representation of its solutions. The output reproduces the equation nicely formatted, that is, with a single space around the +, - and = characters. As for the representation of the solutions, it is also nicely formatted, omitting \$x_0\$ or \$y_0\$ when they are equal to 0, and omitting 1 as a factor of \$n\$. Press the Run or Mark buttons for possible interactions:>>> diophantine('1x + 1y = 0') 1x + 1y = 0 has as solutions all pairs of the form (n, -n) with n an arbitrary integer. >>> diophantine('-1x + 1y = 0') -1x + 1y = 0 has as solutions all pairs of the form (n, n) with n an arbitrary integer. >>> diophantine('1x - 1y = 0') 1x - 1y = 0 has as solutions all pairs of the form (n, n) with n an arbitrary integer. >>> diophantine('-1x - 1y = 0') -1x - 1y = 0 has as solutions all pairs of the form (n, -n) with n an arbitrary integer. >>> diophantine('1x + 1y = -1') 1x + 1y = -1 has as solutions all pairs of the form (n, -1 - n) with n an arbitrary integer. >>> diophantine('-1x + 1y = 1') -1x + 1y = 1 has as solutions all pairs of the form (n, 1 + n) with n an arbitrary integer. >>> diophantine('4x + 6y = 9') 4x + 6y = 9 has no solution. >>> diophantine('4x + 6y = 10') 4x + 6y = 10 has as solutions all pairs of the form (1 + 3n, 1 - 2n) with n an arbitrary integer. >>> diophantine('71x+83y=2') 71x + 83y = 2 has as solutions all pairs of the form (69 + 83n, -59 - 71n) with n an arbitrary integer. >>> diophantine(' 782 x + 253 y = 92') 782x + 253y = 92 has as solutions all pairs of the form (4 + 11n, -12 - 34n) with n an arbitrary integer. >>> diophantine('-123x -456y = 78') -123x - 456y = 78 has as solutions all pairs of the form (118 + 152n, -32 - 41n) with n an arbitrary integer. >>> diophantine('-321x +654y = -87') -321x + 654y = -87 has as solutions all pairs of the form (149 + 218n, 73 + 107n) with n an arbitrary integer.

Exercise 4: Fibonacci codes

Recall that the Fibonacci sequence \$(F_n)_{n>=0}\$ is defined by the equations: \$F_0=0\$, \$F_1=1\$ and for all \$n>0\$, \$F_n=F_{n+1}+F_{n-2}\$.It can be shown that every strictly positive integer \$N\$ can be uniquely coded as a string \$\sigma\$ of 0's and 1's ending with 1, so of the form \$b_2b_3\ldots b_k\$ with \$k\geq 2\$ and \$b_k=1\$, such that \$N\$ is the sum of all \$F_i\$'s, \$2\leq i\leq k\$, with \$b_i=1\$.For instance, \$11=3+8=F_4+F_6\$, hence 11 is coded by 00101.Moreover:there are no two successive occurrences of 1 in \$\sigma\$;\$F_k\$ is the largest Fibonacci number that fits in \$N\$, and if \$j\$ is the largest integer in \$\{2,\ldots,k-1\}\$ such that \$b_j=1\$ then \$F_j\$ is the largest Fibonacci number that fits in \$N-F_k\$, and if \$i\$ is the largest integer in \$\{2,\ldots,j-1\}\$ such that \$b_i=1\$ then \$F_i\$ is the largest Fibonacci number that fits in \$N-F_k-F_j\$...Also, every string of 0's and 1's ending in 1 and having no two successive occurrences of 1's is a code of a strictly positive integer according to this coding scheme. For instance:There is only one string of 0's and 1's of length 1 ending in 1 and having no two successive occurrences of 1's; it is 1, and it codes 1.There is only one string of 0's and 1's of length 2 ending in 1 and having no two successive occurrences of 1's; it is 01, and it codes 2.The strings of 0's and 1's of length 3 ending in 1 and having no two successive occurrences of 1's are 001 and 101 and they code 3 and 4, respectively.The strings of 0's and 1's of length 4 ending in 1 and having no two successive occurrences of 1's are 0001, 1001 and 0101 and they code 5, 6 and 7, respectively.The strings of 0's and 1's of length 5 ending in 1 and having no two successive occurrences of 1's are 00001, 10001, 01001, 00101 and 10101 and they code 8, 9, 10, 11 and 12, respectively....The Fibonacci code of \$N\$ adds 1 at the end of \$\sigma\$; the resulting string then ends in two 1's, therefore marking the end of the code, and allowing one to let one string code a finite sequence of strictly positive integers. For instance, 00101100111011 codes \$(11,3,4)\$.Implement the two functions in the stub, one that takes one argument \$N\$ mean to to be a strictly positive integer and returns its Fibonacci code, and one that takes one argument \$\sigma\$ meant to be a strict consisting 0's and 1's, returns 0 if \$\sigma\$ cannot be a Fibonacci code, and otherwise returns the integer \$\sigma\$ is the Fibonacci code of.Here are possible interactions:\$ python3 ... >>> from fibonacci_codes import * >>> encode(1) ’11’ >>> encode(2) ’011’ >>> encode(3) ’0011’ >>> encode(4)

’1011’ >>> encode(8) ’000011’ >>> encode(11) ’001011’ >>> encode(12) ’101011’ >>> encode(14) ’1000011’ >>> decode(’1’) 0 >>> decode(’01’) 0 >>> decode(’100011011’) 0 >>> decode(’11’) 1 >>> decode(’011’) 2 >>> decode(’0011’) 3 >>> decode(’1011’) 4 >>> decode(’000011’) 8 >>> decode(’001011’) 11 >>> decode(’1000011’) 14

Exercise 5: Change making

Write a program change_making.py that prompts the user for the face values of coins and their associated quantities as well as for an amount, and if possible, outputs the minimal number of coins needed to match that amount, as well as the detail of how many coins of each type value are used.The face values and associated quantities should be input as a dictionary. You might find the literal_eval() function from the ast module to be useful.A solution is output from smallest face value to largest face value. If a solution is represented as a list of pairs of the form (coin face value, number of coins) ordered from smallest to largest face value, then the solutions themselves are output in lexicographical order (for sequences of pairs). All face values for a given solution are right aligned.Insert your code into change_making.py.Here are examples of interactions:$ python3 change_making.py Input a dictionary whose keys represent coin face values with as value for a given key the number of coins that are available for the corresponding face value: {2: 100, 50: 100} Input the desired amount: 99 There is no solution. $ python3 change_making.py Input a dictionary whose keys represent coin face values with as value for a given key the number of coins that are available for the corresponding face value: {1: 30, 20: 30, 50: 30} Input the desired amount: 60 There is a unique solution: $20: 3 $ python3 change_making.py Input a dictionary whose keys represent coin face values with as value for a given key the number of coins that are available for the corresponding face value: {1: 100, 2: 5, 3: 4, 10: 5, 20: 4, 30: 1} Input the desired amount: 107 There are 2 solutions: $1: 1 $3: 2 $10: 1 $20: 3 $30: 1 $2: 2 $3: 1 $10: 1 $20: 3 $30: 1 $ python3 change_making.py Input a dictionary whose keys represent coins face values with as value for a given key the number of coins that are available for the corresponding face value: {1: 7, 2: 5, 3: 4, 4: 3, 5: 2} Input the desired amount: 29 There are 4 solutions: $1: 1 $3: 2 $4: 3 $5: 2 $2: 1 $3: 3 $4: 2 $5: 2 $2: 2 $3: 1 $4: 3 $5: 2 $3: 4 $4: 3 $5: 1 $ python3 change_making.py Input a dictionary whose keys represent coins face values with as value for a given key the number of coins that are available for the corresponding face value: {11:34, 12:34, 13: 234, 17:44, 18:54, 19: 3} Input the desired amount: 3422 There are 8 solutions: $11: 1 $12: 4 $13: 122 $17: 44 $18: 54 $19: 3 $11: 1 $13: 127 $17: 43 $18: 54 $19: 3 $11: 2 $12: 2 $13: 123 $17: 44 $18: 54 $19: 3 $11: 3 $13: 124 $17: 44 $18: 54 $19: 3 $12: 1 $13: 127 $17: 44 $18: 53 $19: 3 $12: 2 $13: 126 $17: 43 $18: 54 $19: 3 $12: 6 $13: 121 $17: 44 $18: 54 $19: 3 $13: 128 $17: 44 $18: 54 $19: 2 The natural approach makes use of the linear programming technique exemplified in the computation of the Levenshtein distance between two words discussed in Week 9 lectures.