

Computing Algorithms 2801ICT

Assignment 1 – Pay In Coins



April 22, 2019

Sam grumley

S5048240

# Algorithm Design

## Overview

Write a program that can calculate the total number of ways a given amount can be paid using a specified number of coins where coins can be the value of any prime number. The given total and number of coins are given in the form of a txt file that can include three different types of input. An input may have a single integer which represents the money you have to pay using all possible combinations of coins. An Input may have two integers where the first integer represents the amount to be paid and the second integer represents the number of coins you should use to pay the amount. Additionally, if an input contains 3 integers, the first integer is the amount to be paid, the second and the third integers represent the range of number of coins that can be used. The output will also be in the form of a txt file that will simply reflect how many combinations there are for that input line. This amount of combinations does not include duplicates regardless of order e.g. 2,1,5 = 1,2,5

## Algorithm Description

My implementation to this problem builds off certain rules to minimise the search space:

### Column iteration:

To avoid getting duplicate combinations, the further most column iterated will be the new lowest value for any column to the right.

e.g. 7 = max prime number

iteration 5: 1 + 1 + 1 + 7

iteration 6: 1 + 1 + 2 + 2

iteration x: 1 + 1 + 7 + 7

iteration x+1: 1 + 2 + 2 + 2

### Prime gap:

Referring to the table below you can see that the gap between prime numbers is never significantly big compared to the prime number. This means that it is very unlikely that the highest prime number below the total will be in many of the solutions. With this fact we can find the difference between the total and the max prime number and search with n – 1 for all possible solutions.

e.g. Total = 16, n =3, highest prime = 13

diff = total – hp = 3

Search (n=2, total=3, primeValuesUnderDiff=[1,2,3])

This changes the state space from to for the main search

Where = prime numbers up to total and = prime number up to diff

Using the example above you will end up with 343 – 225 = 118 less iterations

A screenshot of a cell phone

Description generated with very high confidence

### Limitation rule (pruning):

This simply determines if the difference between the next prime and current prime is greater than the difference between the total and the current prime.

e.g. total = 9, current combination = 1+1+1+5 = 8

Next node – current node = 7 – 5 = 2

Total – current combination = 1

2 > 1 therefore don’t iterate the column to the left

New combination = 1 + 1 + 2 + 2

### Difference Array:

As seen in the pruning part of the algorithm the difference needs to be calculated repeatedly for every iteration. To minimise the number of computations all differences are calculated as soon as the prime numbers are known.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P[] | 1 | 2 | 3 | 5 | 7 | 11 | 13 | 17 |
| Difference | 2-1 | 3-2 | 5-3 | 7-5 | 11-7 | 13-11 | 17-13 |  |
| diffArray[] | 1 | 1 | 2 | 2 | 4 | 2 | 4 |  |

## Pseudo Code

### Driver:

The Driver will simply determine the input and run the search with regards to the parameters

ReadIn()

For every line in file

define n, total

sum += search(total, n)

Write results to file

### runFunc:

runFunc Collects all sets of data needed for the algorithm to complete its search

p[] <- getPrimeNums()

if n = 1

return 1 # gold coin

primeGap, PrimeGapList <- getPrimeGap() # solve sub problem

solutions = algorithm(n-1, primeGap, primeGapList, difArr)

Soloutions += algorithm(n, total, p, difArr)

Return Solutions

### algorithm:

The algorithm function contains 2 main purposes. Firstly, to control a brute force approach: when one n has maxed out plus one to the value left and start the iterations again. Secondly, to make informed decisions that remove as much as the search space as possible. The actual concepts implemented here can be found above under the heading ‘Algorithm Description’.

n\_val = [0 for i in range(n)] # This represented the combination of p values via index

while (the first value in n\_val is not the max prime)

atLimit, maxColPos = colCheck(p,nval) # check if any of the values have reached the max

while(atLimit = True)

increment the n\_val value to the left of the maxed value

for every value on the right

set them equal to the incremented value

atLimit, maxColPos = colCheck(p,nval)

if atLimit = False

sum = summedValues

limit = total – sum

if limit = 0

numSolutions += 1

increment the second right most n\_val

else if difArr > limit

increment the second right most n\_val

else

increment the right most n\_val

return solutions

# Results and Analysis

## Results

|  |  |  |
| --- | --- | --- |
| Input | Python 3 Time (s) | Output |
| 5 | 0.0 | 6 |
| 6 2 5 | 0.0 | 7 |
| 6 1 6 | 0.0 | 9 |
| 8 3 | 0.0 | 2 |
| 8 2 5 | 0.000999 | 10 |
| 20 10 15 | 2.375999 | 57 |
| 100 5 10 | 214.2400 | 14837 |

## Performance Analysis

Several ideas / features were tested to see if they made an improvement.

### Test Binary vs iterative

This involved using a binary search instead of iterating the right most value. Each other value would increment as usual but the value iterating the most was replaced with a binary search. Unfortunately, this lead to finding duplicate combinations which would have been a defining factor in it’s poor performance.

|  |  |  |
| --- | --- | --- |
| Input | Binary search Time (s) | Iterative Time (s) |
| 100 10 | 307.904 | 255.894 |
| 200 6 | 13.1189 | 10.7925 |

### Difference Array vs Calculating

This simply show the time saved by pre calculating all the differences

|  |  |  |
| --- | --- | --- |
| Input | Difference Array Time (s) | Calculating Time (s) |
| 200 8 | 739.407 | 748.310 |
| 200 7 | 90.6614 | 93.3520 |

### Comparison of Python versions

Although not algorithm specific. I would recommend running this program with python 2.7 for the most time efficient solution.

|  |  |  |
| --- | --- | --- |
| Input | Python 3 Time (s) | Python 2.7 Time (s) |
| 5 | 0.0 | 0.001999 |
| 6 2 5 | 0.0 | 0.001999 |
| 6 1 6 | 0.0 | 0.003000 |
| 8 3 | 0.0 | 0.002000 |
| 8 2 5 | 0.000999 | 0.001999 |
| 20 10 15 | 2.375999 | 1.815999 |
| 100 5 10 | 214.2400 | 161.2899 |