

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:

## Pseudo Code

# Results and Analysis

## Results

Table or results

## Performance Analysis

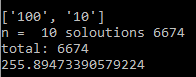
Table of results compared to task sheet

N=10 Total=100

Binary Search:



Iterating:



N=6 t=200

Bin search

result 41061

13.118966341018677

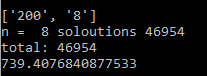
Iter:

total: 8231

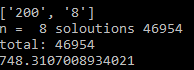
10.792451620101929

N=8 t=200

diffArr:



diffCalc:



N=7 t=200

diffArr:

total: 7900

90.66143774986267

diffCalc:

total: 7900

93.35204362869263