Mobility Assignment

Mohammad SHREM
Mariam ALKHALAF

Presented to: Prof. CANALDA Philippe

October 07th, 2024

Problem Introduction

The goal of the **make change problem** is to minimize the number of coins used to make a specific amount ***m***, given a list of coin denominations ***L***.

For example, if L={5,2,1,0.5,0.2,0.1,0.05} and m=12.35, we need to find the smallest number of coins from the set L that sum to 12.35.

Greedy Algorithm Approach

The **greedy algorithm** works by selecting the largest denomination possible at each step. It keeps subtracting the largest possible coin from **m** until **m=0**. This approach is straightforward but may not always give the optimal solution depending on the coin denominations. However, for specific denominations (like standard currency systems), it often works well.

Steps to Solve

- **Input:** Coin denominations L={5,2,1,0.5,0.2,0.1,0.05} and m=12.35
- **Greedy algorithm:**
    - Start with the largest coin denomination in L.
    - Subtract this denomination from m until it no longer fits.
    - Move to the next largest denomination and repeat until the amount is reduced to 0.
- **Output:** The number of coins used and the specific denominations selected.

Manual Test

Let's apply the greedy algorithm with m=12.

1. Start with L={5,2,1,0.5,0.2,0.1,0.05} and m=12.35.

2. Use the largest coin, which is 5:
   12.35−5=7.35
   Number of 5 coins used: 1.

3. Use the next largest coin, which is 5 again:
   7.35−5=2.35
   Number of 5 coins used: 2.

4. Use the next largest coin, which is 2:
   2.35−2=0.35
   Number of 2 coins used: 1.

5. Use the next largest coin, which is 0.2:
   0.35−0.2=0.150.
   Number of 0.2 coins used: 1.

6. Use the next largest coin, which is 0.1:
   0.15−0.1=0.050
   Number of 0.1 coins used: 1.

7. Use the next largest coin, which is 0.005:
   0.05−0.005=0. ➔ **BEST SOLUTION**
   Number of 0.05 coins used: 1

8. Full Manual Result Table: (m = 12.35)

I	L	Quantity*	Quality**
1	5	1	2
2	2	2	1
3	1	3	0
4	0.5	4	0
5	0.2	5	1
6	0.1	6	1
7	0.05	**7**	1
Final Solution		**Coins Count Used**	**6**
		Coins Used List	**{5, 5, 2, 0.2, 0.1, 0.05}**

Footnote:
***number of solutions, time of execution**
****no bug or best solution, which solution and how many coins?**

Test Cases for Complexity & Linearity

We can test this algorithm with different values of mmm, such as:

Case 1: m=12.35

Expected result: The coins used should be [5, 5, 2, 0.2, 0.1, 0.05]

Case 2: m=10.70

Expected result: The coins used should be [5, 5, 0.5, 0.2]

- **Time Complexity:**

The greedy algorithm runs in **O(n)**, where n is the number of denominations in L. The time complexity is linear since we iterate through the list of denominations and subtract values from m.

- **Quality of Solution:**

The solution quality depends on the number of coins used to make m. The fewer coins used, the better the solution. In our example, we were able to make 12.35 using a combination of 5, 2, 0.2, 0.05.

- **Experimenting with Varying m:**

To test the performance, we can vary the number of digits in m from 1 to 10 and analyze how the number of coins and execution time change.

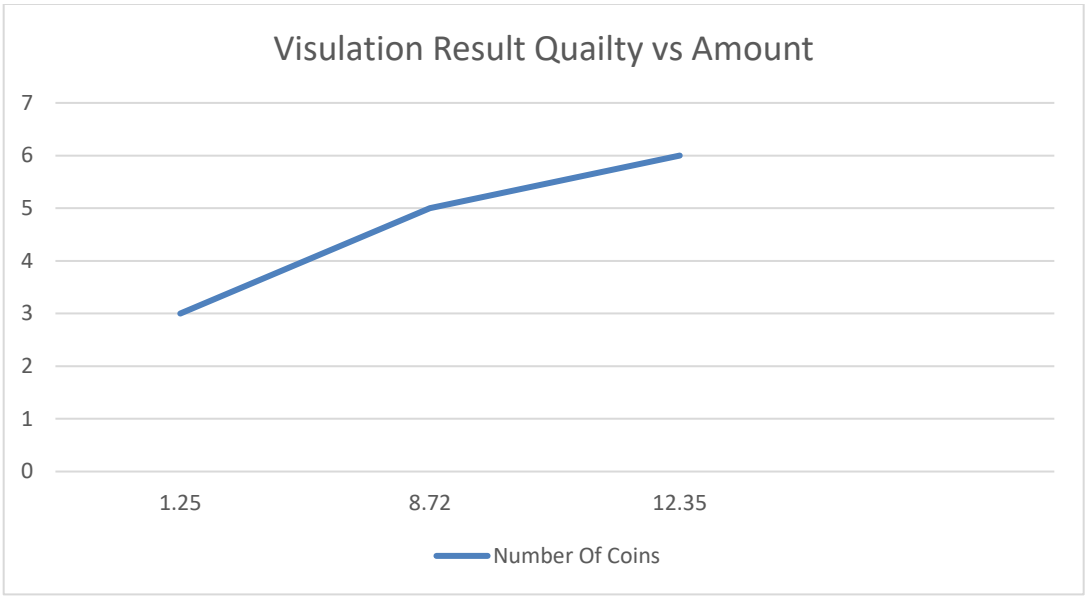
For example:

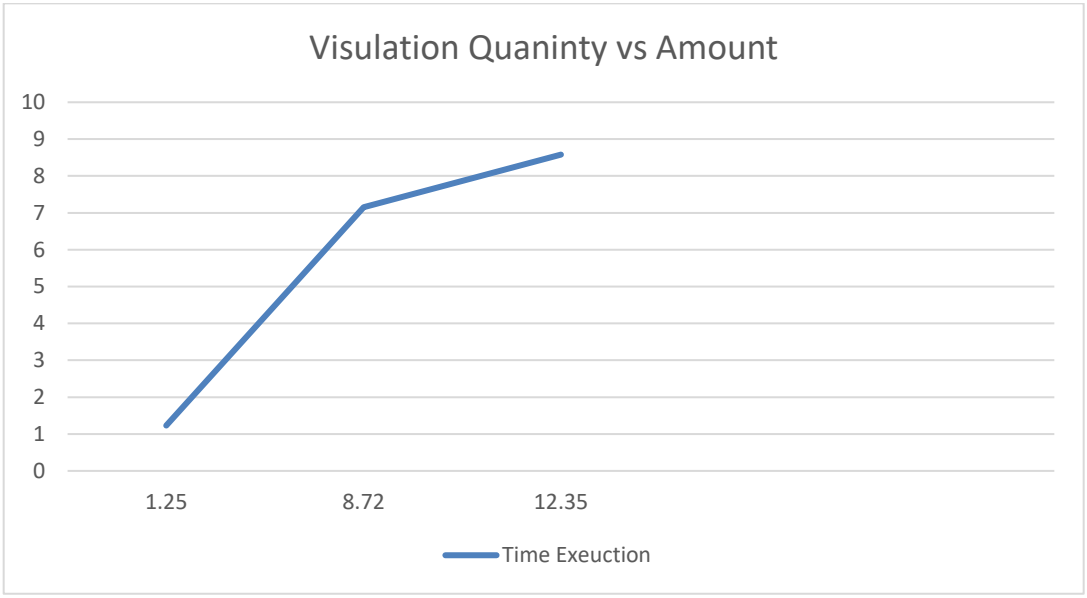
- m=1.23
- m=12.35
- m=123.456

- **Visualization (Table & Curves):**

You can visualize the number of coins required for different values of m using a table or curve. For instance:

m	Number of coins	Time
1.25	3	
8.72	5	
12.35	6	

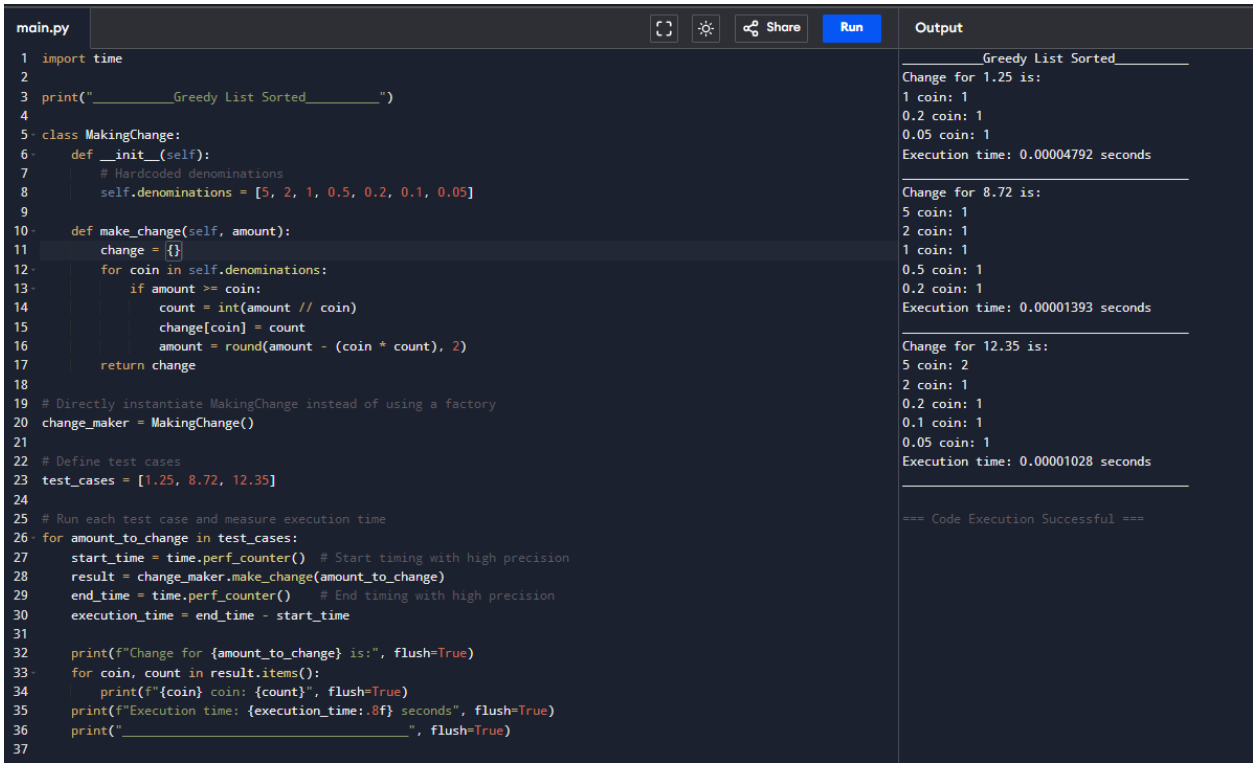

Visulation Result Quailty vs Amount
7
6
5
4
3
2
1
0
1.25
8.72
12.35
Number Of Coins


Visulation Quaninty vs Amount
10
9
8
7
6
5
4
3
2
1
0
1.25
8.72
12.35
Time Exeuction

Codes

- Python

```
import time

def change_quality(denominations, amount):
    coins_list = []
    for coin in sorted(denominations, reverse=True):
        while amount >= coin:
            amount = round(amount - coin, 4)
            coins_list.append(coin)
    return coins_list

denominations = [5, 2, 1, 0.5, 0.2, 0.1, 0.05]
amount = 12.35

start_time = time.time()
result = change_quality(denominations, amount)
end_time = time.time()
execution_time = round(end_time - start_time, 10)

print(f"Coins Used List: {result}")
print(f"Coins Count: {len(result)}")
print(f"Time of Execution: {execution_time} seconds")
```



- C#:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;

class Program
{
    static void Main()
    {
        List<double> denominations = new List<double> { 5, 2, 1, 0.5, 0.2, 0.1, 0.05 };
        double amount = 12.35;

        Stopwatch stopwatch = Stopwatch.StartNew();
        List<double> result = ChangeQuality(denominations, amount);
        stopwatch.Stop();
        double executionTime = stopwatch.Elapsed.TotalSeconds;

        Console.WriteLine($"Coins Used List: {string.Join(", ", result)}");
        Console.WriteLine($"Coins Count: {result.Count}");
        Console.WriteLine($"Time of Execution: {executionTime} seconds");
    }

    static List<double> ChangeQuality(List<double> denominations, double amount)
    {
        List<double> coinsList = new List<double>();
        denominations.Sort((a, b) => b.CompareTo(a));

        foreach (double coin in denominations)
        {
            while (amount >= coin)
            {
                amount = Math.Round(amount - coin, 4);
                coinsList.Add(coin);
            }
        }

        return coinsList;
    }
}
```

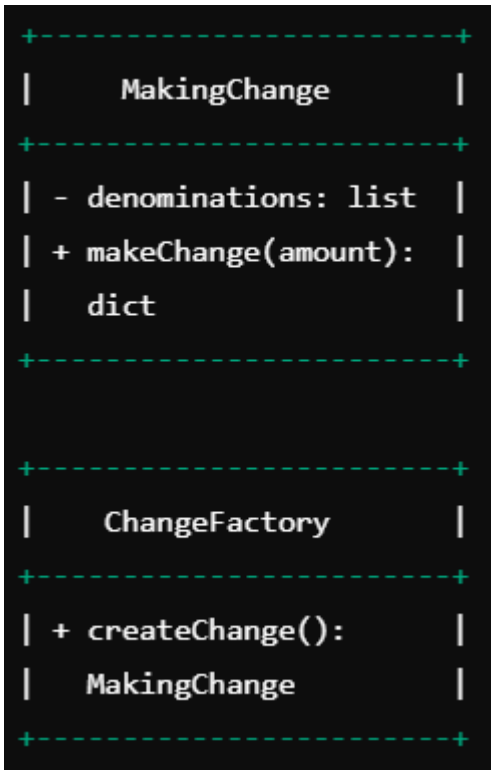
Main.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;

class Program
{
    static void Main()
    {
        List<double> denominations = new List<double> { 5, 2, 1, 0.5, 0.2, 0.1, 0.05 };
        double amount = 12.35;

        Stopwatch stopwatch = Stopwatch.StartNew();
        List<double> result = ChangeQuality(denominations, amount);
        stopwatch.Stop();
        double executionTime = stopwatch.Elapsed.TotalSeconds;

        Console.WriteLine($"Coins Used List: {string.Join(", ", result)}");
        Console.WriteLine($"Coins Count: {result.Count}");
        Console.WriteLine($"Time of Execution: {executionTime} seconds");
    }

    static List<double> ChangeQuality(List<double> denominations, double amount)
    {
        List<double> coinsList = new List<double>();
        denominations.Sort((a, b) => b.CompareTo(a));

        foreach (double coin in denominations)
        {
            while (amount >= coin)
            {
                amount = Math.Round(amount - coin, 4);
                coinsList.Add(coin);
            }
        }

        return coinsList;
    }
}
```

Output

```
mono /tmp/wGo2n3TYBz.exe
Coins Used List: 5, 5, 2, 0.2, 0.1, 0.05
Coins Count: 6
Time of Execution: 0.0014629 seconds

=== Code Execution Successful ===
```

- Class Diagram:



Greedy Algorithm Approach (L not sorted):

- **Input:** Coin denominations L={1 , 0.1, 5, 2, 0.5, 0.05, 0.02} and m=12.35
- **Greedy algorithm:**
    - Start with the coin denomination in L.
    - Subtract this denomination from m until it no longer fits.
    - Move to the next largest denomination and repeat until the amount is reduced to 0.
- **Output:** The number of coins used, and the specific denominations selected.
- **Full Manual Result Table:**

I	L	Quantity*	Quality**
1	1	1	12
2	0.1	2	3
3	5	3	0
4	2	4	0
5	0.5	5	0
6	0.05	6	1
7	0.02	**7**	0
Final Solution		**Coins Count Used**	**16**
		Coins Used List	**{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0.1, 0.1, 0.1, 0.05}**

Footnote:
***number of solutions, time of execution
**no bug or best solution, which solution and how many coins?

main.py

Share Run

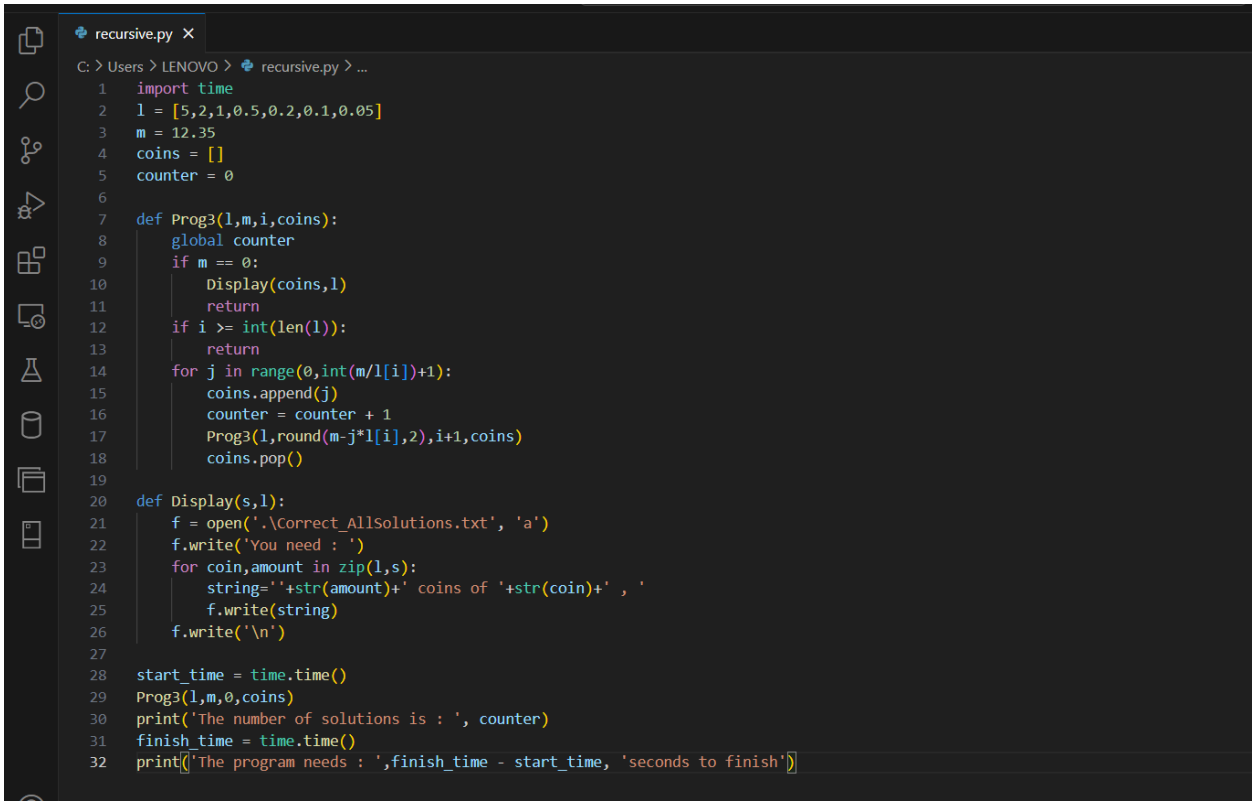
```python
#Program #1 (b)

import time

print("_________Greedy List Not Sorted________")

class MakingChange:
    def __init__(self):
        # Hardcoded denominations
        self.denominations = [1 , 0.1, 5, 2, 0.5, 0.05, 0.02]

    def make_change(self, amount):
        change = {}
        for coin in self.denominations:
            if amount >= coin:
                count = int(amount // coin)
                change[coin] = count
                amount = round(amount - (coin * count), 2)
        return change

# Directly instantiate MakingChange instead of using a factory
change_maker = MakingChange()

# Define test cases
test_cases = [1.25, 8.72, 12.35]

# Run each test case and measure execution time
for amount_to_change in test_cases:
    start_time = time.perf_counter()  # Start timing with high precision
    result = change_maker.make_change(amount_to_change)
    end_time = time.perf_counter()    # End timing with high precision
    execution_time = end_time - start_time

    print(f"Change for {amount_to_change} is:", flush=True)
    for coin, count in result.items():
        print(f"{coin} coin: {count}", flush=True)
    print(f"Execution time: {execution_time:.8f} seconds", flush=True)
    print("_____________________________________", flush=True)
```

Output

```
_________Greedy List Not Sorted________
Change for 1.25 is:
1 coin: 1
0.1 coin: 2
0.05 coin: 1
Execution time: 0.00004978 seconds
_____________________________________
Change for 8.72 is:
1 coin: 8
0.1 coin: 7
0.02 coin: 1
Execution time: 0.00001328 seconds
_____________________________________
Change for 12.35 is:
1 coin: 12
0.1 coin: 3
0.05 coin: 1
Execution time: 0.00001085 seconds
_____________________________________

=== Code Execution Successful ===
```

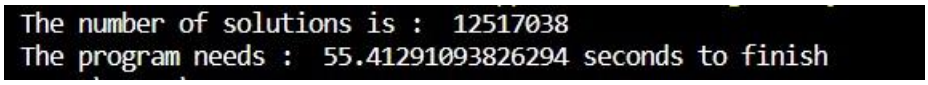
Recursive Method:

The recursive is better than the iterative for resources and time.



```python
import time
l = [5,2,1,0.5,0.2,0.1,0.05]
m = 12.35
coins = []
counter = 0

def Prog3(l,m,i,coins):
    global counter
    if m == 0:
        Display(coins,l)
        return
    if i >= int(len(l)):
        return
    for j in range(0,int(m/l[i])+1):
        coins.append(j)
        counter = counter + 1
        Prog3(l,round(m-j*l[i],2),i+1,coins)
        coins.pop()

def Display(s,l):
    f = open('.\Correct_AllSolutions.txt', 'a')
    f.write('You need : ')
    for coin,amount in zip(l,s):
        string=''+str(amount)+' coins of '+str(coin)+' , '
        f.write(string)
    f.write('\n')

start_time = time.time()
Prog3(l,m,0,coins)
print('The number of solutions is : ', counter)
finish_time = time.time()
print('The program needs : ',finish_time - start_time, 'seconds to finish')
```

Output:



```
The number of solutions is :  12517038
The program needs :  55.41291093826294 seconds to finish
```

- All programs requested with files generated are attached.
- GitHub link: https://github.com/MhmdBShreem/MakeChange

Thanks 😊