Make change REPORT

Killian MAXEL
Master IoT 2023-2024

Teacher: Mr. Philippe CANALDA

Introduction

The Make Change problem is a well-known problem in computer science, it involves finding the most efficient way to give change for a given amount using a set of coins. In this study, we explore two distinct approaches to solving this problem: an iterative greedy solution and a recursive one. We also need to return the best solution, a solution and all possible combinations. To do so we will use Java using IntelliJ IDE.

Background and specifications

Before starting any implementation, here are keywords to keep in mind:

Iterative Greedy Solution:

- Greedy approach, always choosing the largest coin denomination.
- Limitation: May not always yield the optimal solution.

Modified Iterative Approach:

- Considers decreasing L (remaining amount) for optimized coin selection.
- Orders combinations for efficient processing.

Stopping Criteria:

- Controls the search process in the iterative approach.
- Balances computational efficiency with solution quality.

Recursive Approach:

- Provides a complementary, memory-efficient alternative.
- May have higher runtime complexity.

The iterative greedy solution is a straightforward approach, where the algorithm selects the largest available coin denomination at each step until the target amount is reached. However, this method may not always yield the optimal solution.

Using what is expected, we have to implement

- a greedy algorithm using iterative approach, generating a solution (may not be the right one)
- an algorithm using iterative approach generating all possible combinations
- a greedy algorithm using iterative approach generating the BEST solution
- a recursive algorithm generating all possible solutions

Implementation and Results

Iterative Greedy Solution (**makeChangeBest**):

- This algorithm uses a greedy approach by always selecting the largest coin denomination available to reduce the remaining amount.
- It starts by converting the given amount to cents and initializes an array to store the minimum number of coins needed for each amount.
- The algorithm iterates through each coin denomination and calculates the minimum number of coins needed to make change for each amount from the smallest to the target amount.
- Finally, it keeps the best combination of coins found to achieve the target amount with the minimum number of coins.
- We need to be sure that it is the best solution (not a random solution)

Modified Iterative Approach (**makeChangeAllIterative**):

- This algorithm explores all possible combinations of coins iteratively.
- It starts with the highest coin denomination and loops through all possibilities to find combinations that sum up to the target amount.
- The result is a list of all valid combinations.

Recursive Approach (**makeChangeAllRecursive**):

- This algorithm uses recursion to find all possible combinations of coins that sum up to the target amount.
- It explores different combinations by recursively adding coins to the current combination until the target amount is reached.
- The result is a list of all valid combinations.

Now, let's analyze the results:

1) Iterative Greedy Solution (**makeChangeBest**):

We conducted an investigation to assess whether the ordering of the coin set affected the selection of the best solution. For an amount of 12.35, we tested two different coin sets: {5, 2, 1, 0.5, 0.2, 0.1, 0.05} and {2, 1, 0.5, 0.2, 0.1, 0.05, 5}. The experiment confirmed that the algorithm identifies the same best solution, even when the coin sets are not ordered identically. The following two combinations represent the best solution for the given amount:

[5.0, 5.0, 2.0, 0.2, 0.1, 0.05]
[2.0, 0.2, 0.1, 0.05, 5.0, 5.0]

This demonstrates the robustness of the algorithm in consistently providing the optimal change, regardless of the ordering of the coin denominations.

2) Modified Iterative Approach (**makeChangeAllIterative**):

Because of memory errors with the previous set (long runtime), we decided to work on a smaller inputs, with an amount of 10 and a set of coins equals to {5, 2, 0.5}, the algorithm provides several valid combinations of coins that sum up to 10, such as [2.0, 2.0, 2.0, 2.0, 0.5, 0.5, 0.5, 0.5]. The total number of combinations is 10.

3) Recursive Approach (**makeChangeAllRecursive**):

Because of the limitations of the previous algorithm, and to compare results with this recursive method, we decided to use the same small set as before. The algorithm provides the same combinations as the modified iterative approach, so to compare with the previous algorithm, here are the different runtimes using System.nanoTime():

- **makeChangeAllIterative: 68 000 * 10^-9s**
- **makeChangeAllRecursive: 110 800* 10^-9 s**

Obviously the recursive approach is time consuming, but when coming to larger input, contrary to the iterative approach, we have a lower memory consuming approach.
Indeed, even if there were memory constraints for our iterative algorithm, we decided to test the recursive one with the initial inputs of 12.35 and coins {5, 2, 1, 0.5, 0.2, 0.1, 0.05}, the recursive algorithm finds a total of 266,724 different combinations and is working well (without memory error)

In summary, when generating the best solution, the iterative greedy solution provides the right amount with the minimum number of coins, Whereas when we generate all solutions, the iterative solution has memory constraints while recursive approaches find all possible combinations for a given amount without problem.

Here is our output:

for amount=12.35
for coins={5, 2, 1, 0.5, 0.2, 0.1, 0.05}
Best Change:
[5.0, 5.0, 2.0, 0.2, 0.1, 0.05]
Best Change2 (the set of coins order is changed) :
[2.0, 0.2, 0.1, 0.05, 5.0, 5.0]

Because iterative approach is memory greedy, we choose to work on a small set (amount=10, coins = {5,2,0.5}
All Possible Changes in iterative (max for printing = 10 coins):
[2.0, 2.0, 2.0, 2.0, 0.5, 0.5, 0.5, 0.5]
[2.0, 2.0, 2.0, 2.0, 2.0]
[5.0, 2.0, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[5.0, 2.0, 2.0, 0.5, 0.5]
[5.0, 5.0]
nbr of combinations in iterative: 10

To compare with iterative approach we have: (**amount=10, coins = {5,2,0.5})**

All Possible Changes in recursive (max for printing = 10 coins):
[5.0, 5.0]
[5.0, 2.0, 2.0, 0.5, 0.5]
[5.0, 2.0, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[2.0, 2.0, 2.0, 2.0, 2.0]
[2.0, 2.0, 2.0, 2.0, 0.5, 0.5, 0.5, 0.5]
nbr of combinations in recursive: 10
nbr of combinations in iterative: 10

We go back to amount=12.35 and coins={5, 2, 1, 0.5, 0.2, 0.1, 0.05}

All Possible Changes in recursive (printing limited):
[5.0, 5.0, 2.0, 0.2, 0.1, 0.05]
[5.0, 5.0, 2.0, 0.2, 0.05, 0.05, 0.05]
[5.0, 5.0, 2.0, 0.1, 0.1, 0.1, 0.05]
[5.0, 5.0, 2.0, 0.1, 0.1, 0.05, 0.05, 0.05]
[5.0, 5.0, 2.0, 0.1, 0.05, 0.05, 0.05, 0.05, 0.05]
[5.0, 5.0, 2.0, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05]
[5.0, 5.0, 1.0, 1.0, 0.2, 0.1, 0.05]
[5.0, 5.0, 1.0, 1.0, 0.2, 0.05, 0.05, 0.05]
[5.0, 5.0, 1.0, 1.0, 0.1, 0.1, 0.1, 0.05]
[5.0, 5.0, 1.0, 1.0, 0.1, 0.1, 0.05, 0.05, 0.05]
[5.0, 5.0, 1.0, 1.0, 0.1, 0.05, 0.05, 0.05, 0.05, 0.05]
[5.0, 5.0, 1.0, 0.5, 0.5, 0.2, 0.1, 0.05]
...
nbr of combinations in recursive: 266724

Conclusion

In conclusion, our study provides valuable insights into solving the Make Change problem using iterative and recursive approaches.

While the iterative greedy solution offers simplicity, it may suffer from memory inefficiency for larger inputs, in our case, when we generate all solutions. Modifying our iterative approach may be a solution to optimize memory usage, but further research is needed. We could also try the algorithm's implementation using C language, which may be more effective, especially for its runtime and its memory garbage management.