"Makes Change" Algorithm Implementation

Written by Edirisinghe Arachchige Isuru Priyanga, Dagher Joseph

The project repository can be accessed at : https://github.com/isuru117/MakesChange-UBFC

Design

The goal of this code is to implement two versions of the "Makes Change" algorithm for finding the minimum number of coins needed to make change for a given total amount.

The two versions are implemented as functions `greedy_iterative` and `greedy_recursive`.

`greedy_iterative(coin_values, total_amount)`

This function takes a list of coin denominations `coin_values` and a `total_amount` for which we need to make change. It uses an iterative approach to find the minimum number of coins required. It iterates through the coin denominations from the highest to the lowest, adding coins to the `change` list as long as the current coin value is less than or equal to the remaining `total_amount`. This process continues until the `total_amount` becomes zero or no suitable coins are left.

`greedy_recursive(coin_values, total_amount)`

This function also takes a list of coin denominations `coin_values` and a `total_amount`. It uses a recursive approach to find the minimum number of coins needed. It first checks if `total_amount` is zero, in which case it returns an empty list, indicating that no coins are needed. Otherwise, it identifies the largest coin denomination that is less than or equal to the `total_amount`, deducts it from the `total_amount`, and recursively calls itself with the updated `total_amount`. It continues this process until the `total_amount` becomes zero or no suitable coins are left.

Implementation

The code is implemented in Python and includes two functions, `greedy_iterative` and `greedy_recursive`, for solving the coin change problem using greedy algorithms. The code is structured as follows:

1. `greedy_iterative(coin_values, total_amount)` function implementation.
2. `greedy_recursive(coin_values, total_amount)` function implementation.

3. Definition of a list `L` representing coin denominations and a total amount `m` for which we want to make change.
4. Calling both `greedy_iterative` and `greedy_recursive` functions with `L` and `m` as inputs.
5. Printing the results.

Below is the code defition of each method:

```
def greedy_iterative(coin_values, total_amount):
    change = []
    for coin in coin_values:
        while total_amount >= coin:
            total_amount -= coin
            change.append(coin)
    return change
```

```
def greedy_recursive(coin_values, total_amount):
    if total_amount == 0:
        return []

    suitable_coins = [coin for coin in coin_values if coin <= total_amount]
    if not suitable_coins:
        return []

    max_coin = max(suitable_coins)
    remaining_amount = total_amount - max_coin

    return [max_coin] + greedy_recursive(coin_values, remaining_amount)
```

Evaluation

Below are the results obtained by executing the two methods for the recursive and iterative approach performed during this project.

Test Input

- List of coin denominations `L = [5, 2, 1, 0.5, 0.2, 0.1, 0.05]`.
- Total amount `m = 12.35`.

Output

- Greedy Iterative Change: `[5, 5, 2, 0.5, 0.2, 0.1]`
- Greedy Recursive Change: `[5, 5, 2, 0.5, 0.2, 0.1]`

The code has been tested with the provided input, and it correctly calculates the minimum number of coins needed to make change for the given total amount using both the iterative and recursive greedy algorithms.

Greedy Iterative Approach

Advantages:

1. **Efficiency**: The iterative approach is typically more efficient than the recursive one, especially for larger inputs. It avoids the overhead of function calls and stack management associated with recursion.
2. **Simplicity**: The code for the iterative approach is often simpler and more straightforward to understand, making it easier to maintain and debug.
3. **Predictable Space Complexity**: The space complexity of the iterative approach is constant (O(1)), as it does not rely on the call stack to store intermediate results.

Disadvantages:

1. **Lack of Optimality Guarantee**: Greedy algorithms, including the iterative approach, may not always yield the optimal solution for every problem. Depending on the coin denominations provided, it may produce suboptimal results.
2. **Dependency on Coin Order**: The order of the coin denominations in the list can affect the result. Changing the order may produce different results, which may or may not be optimal.

Greedy Recursive Approach

Advantages:

1. **Elegance and Readability**: Recursive code can be more elegant and readable, as it closely mirrors the mathematical definition of the problem. This can make the code easier to understand for some developers.
2. **General Applicability**: Recursive thinking can be applied to a wide range of problems, not just coin change. Learning the recursive approach can be valuable for understanding and solving various recursive algorithms.

Disadvantages:

1. **Inefficiency**: Recursive algorithms can be less efficient than iterative ones due to the overhead of function calls and stack memory usage. For large inputs, this can lead to a stack overflow error.
2. **Lack of Tail Recursion Optimization**: The recursive approach in the provided code is not optimized for tail recursion, which means it can be less efficient and may run into stack overflow errors for large inputs.

3. **Limited Applicability**: While recursion is a powerful concept, it may not be the best choice for every problem. In some cases, it can be challenging to convert a recursive solution into an iterative one.

The choice between the iterative and recursive approaches depends on factors such as efficiency, code readability, and the specific problem being solved. The iterative approach is generally preferred for efficiency and simplicity in this context.

Conclusion

In conclusion, this code demonstrates two different approaches (iterative and recursive) to solve the "makes change" problem using greedy algorithms. Both functions successfully find the minimum number of coins needed to make change for a given total amount using the provided coin denominations. The choice between the iterative and recursive approach may depend on factors such as code readability and performance for larger inputs. Greedy algorithms are efficient for solving this specific problem.