Analysis of the Coin Change Problem CS412 Algorithms: Design & Analysis

Syed Nisar Hussain
Habib University
Karachi, Sindh, Pakistan
sh07126@st.habib.edu.pk

Asad Muhammad
Habib University
Karachi, Sindh, Pakistan
am07127@st.habib.edu.pk

Ahmed Ali Aun Mohammad
Habib University
Karachi, Sindh, Pakistan
aa07600@st.habib.edu.pk

Figure 1: Multiple stacks of coins (https://shorturl.at/aACFH)

ABSTRACT

This paper presents an analysis of the time complexity of four design techniques for solving the Coin Change problem: recursion, top-down memoization, dynamic programming (DP) bottom-up approach, and greedy strategy. We explore the theoretical time complexity of each technique. Empirical analyses validate these findings, offering insights into practical performance. Our study provides a concise framework for selecting appropriate algorithms based on their theoretical properties and empirical efficiency.

ACM Reference Format:
Syed Nisar Hussain, Asad Muhammad, and Ahmed Ali Aun Mohammad. 2024. Analysis of the Coin Change Problem CS412 Algorithms: Design & Analysis. In *Proceedings of Algorithms: Design & Analysis Project (CS412' Spring 24).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The Coin Change problem is a fundamental challenge in algorithmic design, demanding the calculation of the fewest number of coins necessary to provide change for a specified amount. Despite its initial appearance of simplicity, it carries substantial significance across a spectrum of domains. This paper delves into an analysis of the Coin Change problem by employing four distinct design methodologies: recursion, top-down memoization, DP bottom-up approach (tabulation), and the greedy strategy. Through a combination of theoretical analysis and empirical assessment, we will look into how computationally efficient these algorithms are. Our overarching objective is to provide insights into the performance attributes of these methodologies for the Coin Change Problem.

CS412' Spring 24, May 02-06, 2024, Karachi, PK

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2 COIN CHANGE PROBLEM

The Coin Change problem is a classical problem in computer science and mathematics, involving the determination of the minimum number of coins required to make change for a given amount of money. It has a rich history dating back centuries and finds applications in various fields, including finance, commerce, and algorithmic research.

2.1 Mathematical Definition

The Coin Change problem involves representing coin denominations as a collection of n unique positive integers, arranged in ascending order from w_1 to w_n. The problem is: given an amount W, also a positive integer, to find a set of non-negative (positive or zero) integers $x_1, x_2, ..., x_n$, with each x_j representing how often the coin with value w_j is used, which minimize the total number of coins $f(W)$ [5]

$$f(W) = \sum_{j=1}^{n} x_j$$

subject to

$$\sum_{j=1}^{n} w_j x_j = W$$

2.2 Applications

The Coin Change problem finds applications in various fields:

(1) **Financial Transactions:** Efficient algorithms for coin change are essential in banking and commerce for minimizing the number of coins dispensed by vending machines, ATMs, and cash registers.
(2) **Optimization Problems:** As it is a variation of the 0/1 Knapsack problem with repetitions, it serves as a prototype for various optimization problems.
(3) **Sports:** Variations of the Coin Change problem arise in different sports such as computing a nine-dart finish in a game of darts. [5]
(4) **Chemistry:** The Coin Change problem can also be used in computing the possible atomic (or isotopic) composition of a given mass/charge peak in mass spectrometry. [5] Mass spectrometry is a widely used analytical technique in chemistry for identifying and quantifying molecules based on their mass-to-charge ratio. [4]

3 DESIGN TECHNIQUES

3.1 Naive Recursive

Description. The naive top-down recursion approach involves recursively exploring all possible combinations of coins to find the minimum number of coins required for the given change amount.

Asymptotic Time Complexity. The time complexity for the naive recursive method is exponential, $O(n^W)$. The exponential growth tells us that naive recursive is not the ideal solution for the Coin Change problem. [2]

3.2 Top-Down Memoized

Description. Top-down memoized recursion optimizes the naive recursive approach by storing the solutions to subproblems in a memoization table to avoid redundant computations.

Asymptotic Time Complexity. The time complexity for the top-down memorized method is pseudo-polynomial, $O(nW)$. It performs better than the naive recursive method and, is considered the optimal solution for the Coin Change Problem. [2, 5]

3.3 Bottom-Up DP

Description. Dynamic programming bottom-up approach iteratively computes the minimum number of coins required for each change amount from 0 to the desired amount, building up the solution incrementally.

Asymptotic Time Complexity. Like the memorized method, the time complexity for the bottom-up tabulation method is also pseudo-polynomial, $O(nW)$. It is considered as the optimal method to solve the Coin Change problem. [2, 5]

3.4 Greedy

Description. The greedy approach for the Coin Change Problem selects the largest denomination coin that can be used at each step until the desired change amount is reached, without considering future consequences.

Asymptotic Time Complexity. The time complexity for the greedy method is linear, $O(W)$. It has the best running time, however, it doesn't always guarantee the the correct solution due to the nature of the algorithm. [3]

4 EXPERIMENT DESIGN

To empirically evaluate the time complexities of the algorithms, the experiments were conducted with the following:

- Input sizes: The US coin denomination set was used throughout the experiments which contain 1¢, 5¢, 10¢, 25¢, 50¢, and $1(100¢) coins. As for the total amount, those were varied for each design technique due to the computing limitations. They are described in detail in the later sections.
- Programming language: Python3 was chosen to implement the algorithms and plot the results.
- Hardware: The experiments were conducted on a machine with an Intel Core i5 13420H processor, and 16GB RAM.

5 EMPIRICAL ANALYSIS

In this section, we present the empirical analysis of each of the four design techniques employed in solving the Coin Change problem. Through this experiment, we aim to evaluate their performance characteristics and gain insights into their practical efficiency. It is important to note that our empirical analysis only involves measuring the execution times.

5.1 Naive Recursive

Setup. For the naive recursion method, our range of values of W for the experiment was from 1 to 59.

Plot. The following figure is the plot for solving the Coin Change problem using recursion.

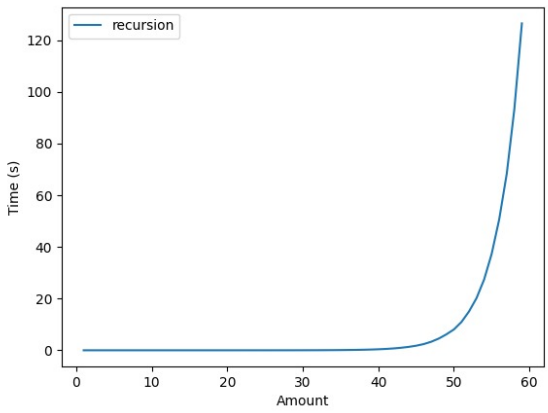

Figure 2: Running time vs amount with recursion

Analysis. The results from our experiment show a similarity with the asymptotic time complexity stated earlier. The plot shows exponential growth as the amount(W) increases. For the smaller amount values, the plot is relatively flat which means the time taken is low however as soon the amount values start getting bigger than 50, a very sharp increase in running time can be observed. Exponential time complexity is considered inefficient for the same reason that the time required to solve the problem grows extremely rapidly for larger input values. Therefore, this makes the recursive solution impractical for large instances of the Coin Change problem.

Constraints. Given that recursion is very expensive computationally, we were unable to test this method on larger amount(W) values. The machine used in this experiment threw `Max Recursion Depth` error while running for larger values, and with trial and error, we ended up restricting our input values to below 60.

5.2 Top-Down Memoized

Setup. For the top-down memoization method, our range of values of W for the experiment was from 10 to 1000 with intervals of 10.

Plot. The following figure is the plot for solving the Coin Change problem using top-down memoization.

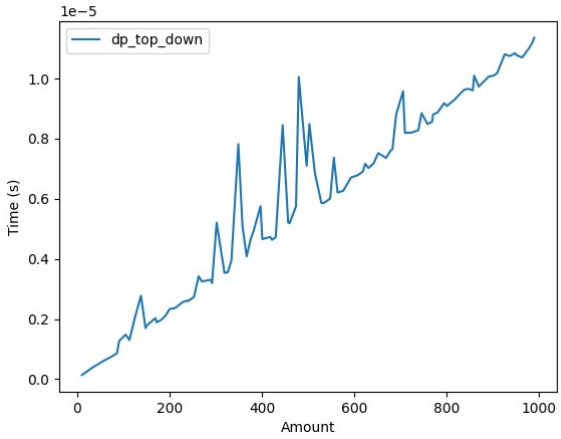

Figure 3: Running time vs amount with memoization

Analysis. The results with the memoization method show a significant improvement over the naive recursion method. The plot exhibits an overall increasing trend, however there are noticeable fluctuations in the plot. The plot is not exactly linear but also does not match its asymptotic pseudo-polynomial time complexity. The reason behind this could be that our experiment takes only one set of coin denominations, meaning considering n as constant, ultimately making the algorithm run in $O(W)$ time.

Constraints. Even though storing and reusing the results gave a significant performance improvement over the naive recursion method, we still faced the `Max Recursion Depth` problem when input values got larger, and therefore the range of input values is still not very big.

5.3 Bottom-Up DP

Setup. For the bottom-up DP (tabulation) method, our range of values of W for the experiment was from 100 to 1000000 with intervals of 1000.

Plot. The following figure is the plot for solving the Coin Change problem using the tabulation method.

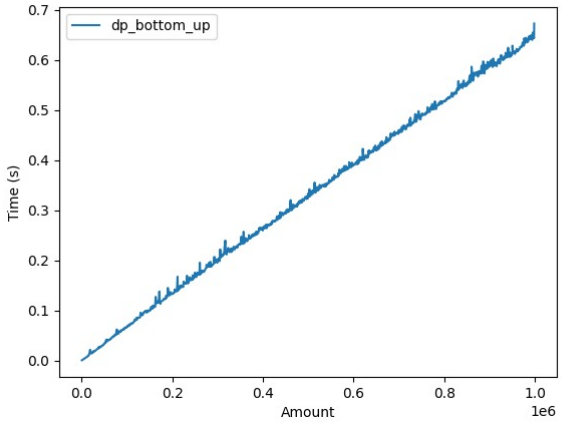

Figure 4: Running time vs amount with tabulation

Analysis. The data points form an approximately linear curve, indicating that the time complexity of the algorithm grows linearly with the input size (target sum). Theoretically, the time complexity of the bottom-up dynamic programming approach for the minimum coin change problem is $O(nW)$. Since the number of coin denominations is generally considered a constant, the time complexity can be simplified to $O(W)$, which is linear in the input size. The empirical data we collected aligns with this theoretical expectation. The linear growth of time taken as the target sum increases suggests that the algorithm's performance scales linearly with the input size, as predicted by the theoretical analysis.

Constraints. It is worth noting that the empirical data may have been influenced by various factors, such as the specific implementation details, the computational environment, and the range of input values tested. Additionally, for very large input sizes, the algorithm's performance may eventually deviate from the linear behavior due to hardware limitations or other practical considerations.

5.4 Greedy

Setup. For the greedy method, our range of values of W for the experiment was from 100 to 100000 with intervals of 1000.

Plot. The following figure is the plot for solving the Coin Change problem using the greedy method.

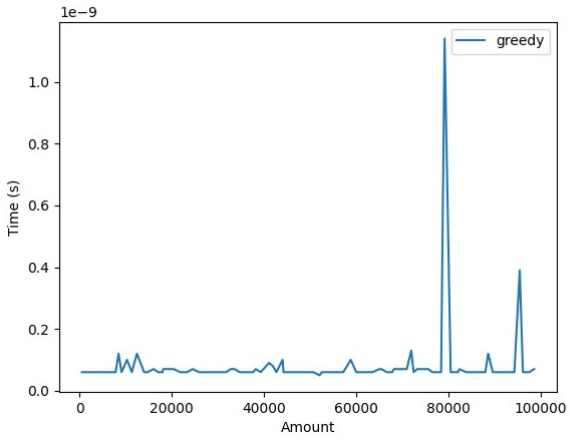

Figure 5: Running time vs amount with greedy approach

Analysis. As shown in the plot, the greedy method displayed the fastest run times. However, it should be noted that this came at the expense of accuracy. It failed to find the optimal solution in several cases, indicating its limited applicability when precision is crucial. It is important to note that since the greedy approach doesn't always guarantee an accurate solution, it requires some form of input validation for correctness. Checking for this correctness is referred to as checking whether a coin system is canonical. A coin system is called canonical if the greedy algorithm always solves its Coin Change problem optimally[5]. The algorithm for checking whether a coin system is canonical or not run in polynomial time [1].

Constraints. The greedy approach does not guarantee an optimal solution for all possible sets of coin denominations and target values. It works by repeatedly selecting the largest possible coin denomination that is less than or equal to the remaining amount. This strategy can sometimes lead to a suboptimal number of coins being used.

6 CONCLUSION

The empirical findings support the theoretical hypotheses on the effectiveness and constraints of every algorithm. The coin change problem is best approached using dynamic programming, both top-down and bottom-up, since it strikes a balance between the correctness of the solution and computing efficiency. These techniques work well for applications that need exact results and can effectively handle bigger problem sizes. Because of its poor performance as the problem size increases, Naive Recursion is not ideal for real-world applications; it is best suited for instructional purposes or relatively tiny datasets. Even though the greedy method is the fastest, it should only be used when there is a severe lack of processing resources or when approximations of solutions are sufficient. These observations not only support the use of dynamic programming approaches to challenging optimization issues, but they also emphasize how crucial it is to select the appropriate algorithm in light of particular limitations and requirements. Subsequent research endeavours may investigate more enhancements in dynamic programming methodologies or hybrid approaches that may present superior performance trade-offs.

ACKNOWLEDGMENTS

To Dr. Shah Jamal Alam, for the wonderful experience in the CS412 course.

REFERENCES

[1] Xuan Cai. 2009. Canonical Coin Systems for Change-Making Problems. arXiv:0809.0400 [cs.DS]
[2] Geeks for Geeks. 2023. *Find minimum number of coins to make a given value (Coin Change).* https://www.geeksforgeeks.org/find-minimum-number-of-coins-that-make-a-change/?ref=lbp
[3] Geeks for Geeks. 2023. *Greedy Algorithm to find Minimum number of Coins.* https://www.geeksforgeeks.org/greedy-algorithm-to-find-minimum-number-of-coins/
[4] Broad Institute. 2024. *What is Mass Spectrometry?* https://www.broadinstitute.org/technology-areas/what-mass-spectrometry
[5] Wikipedia. 2024. *Change-making problem.* https://en.wikipedia.org/wiki/Change-making_problem

A RESOURCES

A.1 GitHub Repository

The following URL is of the GitHub repository containing all the codes for the implementation of the algorithms, and the plots from the empirical analysis:
https://github.com/NisarSyed/coin-change-ADA

A.2 Research Poster

The following URL is for the research poster designed on Canva, that summarizes the key findings from this project:
https://shorturl.at/csT05