Coin Change Problem

Authors:
Asad Muhammad, Ahmed Ali Aun Muhammad, Syed Nisar Hussain

ABSTRACT

The coin change problem is a fundamental problem in computer science and algorithm design. It involves finding the minimum number of coins required to make up a given amount of money, given a set of coin denominations. This problem has various practical applications, such as in cash registers, vending machines, and financial transactions.. This problem has the overlapping subproblems and optimal substructure property which is why it can be solved through dynamic programming.

OBJECTIVE

Analyze and compare the performance of different design techniques for solving the coin change problem.

RESULTS/FINDINGS

This section demonstrates how the execution time of various algorithms utilized scale in response to increasing target values.

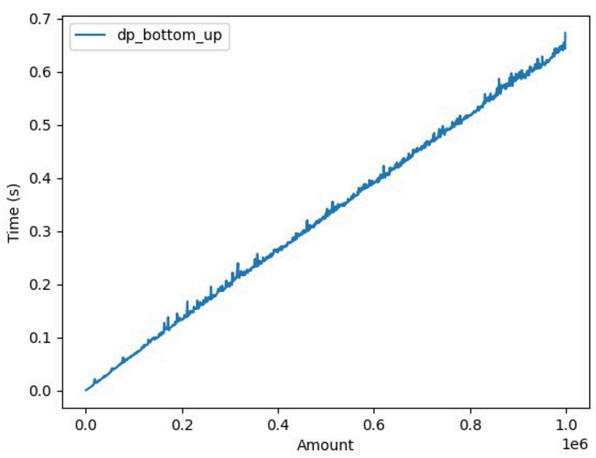

Fig. 1: Time Analysis Graph for DP Approach

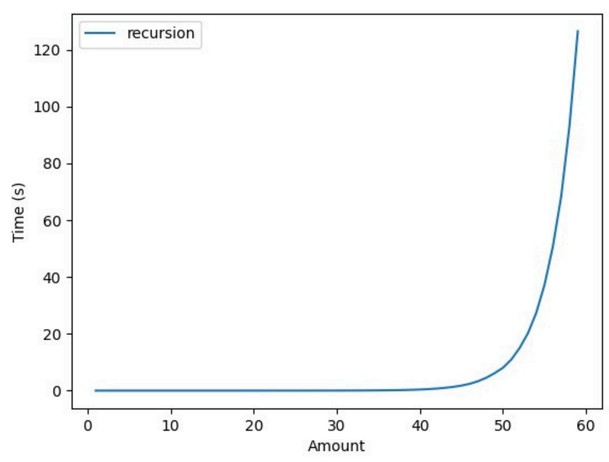

Fig. 2: Time Analysis Graph for Recursive/Naive Approach

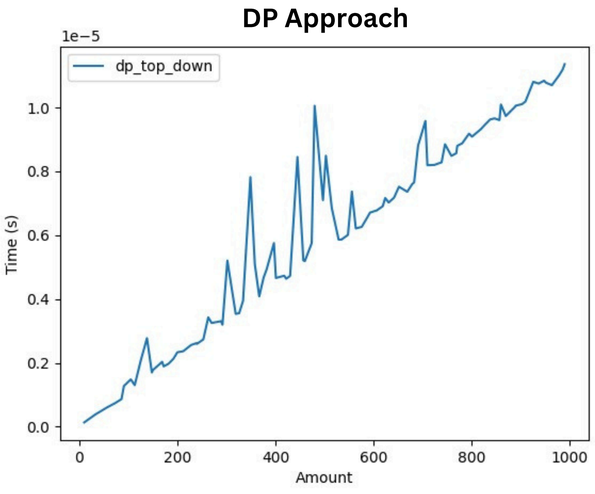

Fig. 3: Time Analysis Graph for Recursive/Memoized Approach

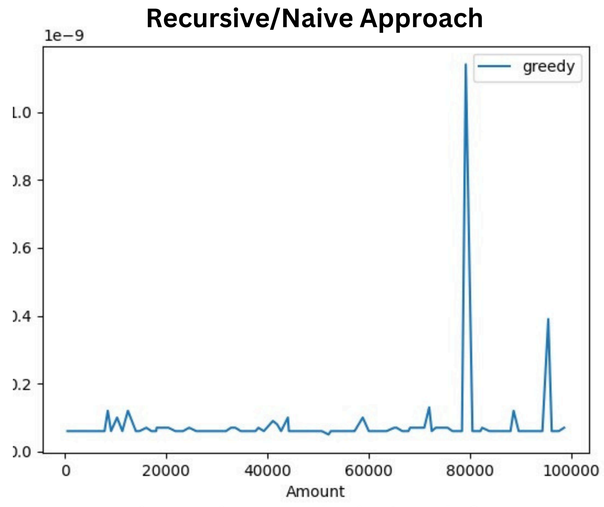

Fig. 4: Time Analysis Graph for Greedy Approach

EMPIRICAL ANALYSIS

- Bottom-Up Dynamic Programming and Top-Down Memoization both show strong performance in terms of both time efficiency and accuracy for large values of n (number of denominations) and W (target sum). They maintained polynomial complexities but were significantly more reliable than the other methods for achieving the minimum number of coins.
- Naive Recursion, as expected from its exponential time complexity, demonstrated poor scalability. It became impractical for larger values due to increased computation time and stack overflow risks.
- Greedy Method displayed the fastest runtimes but at the expense of accuracy. It failed to find the optimal solution in several cases, indicating its limited applicability when precision is crucial.

DESIGN TECHNIQUES & ANALYSIS OF THEIR THEORETICAL TIME COMPLEXITIES

The following design techniques will form the foundation of our empirical analysis. Note that n denotes the number of different coin denominations and W denotes the target sum:

Naive Recursion O(n^W):
Exponential complexity. Highly inefficient for large inputs due to recursive decomposition without any optimization.

Top-Down Memoization O(n*W):
Polynomial complexity. Optimizes naive recursion by caching results, reducing redundant calculations hence lowering time complexity.

Bottom-Up Dynamic Programming O(n*W):
Polynomial complexity. Iteratively builds the solution from smaller subproblems, optimizing space and time by avoiding recursion.

Greedy O(W):
Linear complexity. Fastest in scenarios where it finds a near-optimal solution quickly but does not guarantee the optimal solution in all cases.

METHODOLOGY

Input: Uses a fixed set of coin denominations and randomly generated target values from 10^2 to 10^6.

Procedure: Measures execution time of each algorithm for each target value.

Visualization: Plots execution time against target value for algorithm comparison.

CONCLUSION

The empirical results reinforce the theoretical predictions about the efficiencies and limitations of each algorithm.

Dynamic Programming (both Top-Down and Bottom-Up) emerges as the most effective approach for the coin change problem, balancing between computational efficiency and solution accuracy. These methods are suitable for applications requiring precise outcomes and can handle larger problem sizes effectively.

Naive Recursion is impractical for real-world applications due to its poor performance with increasing problem size, making it unsuitable except for educational purposes or very small datasets.

Greedy Method, while quickest, should only be employed when approximate solutions are sufficient or when computational resources are severely constrained.