Summary:

My Learnings on Greedy Algorithms

I've learned that **Greedy Algorithms** work by making the simplest, most obvious "best" choice at every step to solve a problem, trusting that these local decisions will lead to a globally optimal solution. The key to making them work is identifying the right measure, or **proxy**, for making that greedy choice.

I saw this principle in four different applications:

1. **Activity Selection**: To schedule the most activities, the best strategy is to always pick the one with the **earliest finish time**. This leaves the maximum amount of time open for other activities.
2. **Job Scheduling**: When tasks have different durations and penalties, the ideal proxy isn't just time. I learned to prioritize jobs based on the highest **cost-to-time ratio (c_i/t_i)** to minimize overall costs.
3. **Huffman Coding**: To create the most efficient data compression codes, the greedy approach is to build a binary tree by repeatedly merging the two characters with the **lowest frequencies**. This ensures common characters get the shortest codes.
4. **Kruskal's Algorithm**: For finding a Minimum Spanning Tree in a graph, the strategy is to always add the **cheapest available edge**, with the important rule that you must skip any edge that would form a cycle.

My main takeaway is that for certain problems, you don't need a complex, all-encompassing strategy like Dynamic Programming. If you can find a reliable proxy, you can achieve an optimal result by simply and repeatedly making the choice that looks best at the moment.

Task 3: Critical Evaluation

Based on my test results, I've concluded that the greedy algorithm is a fast but unreliable shortcut, while the dynamic programming (DP) approach is slower but always finds the best answer.

When the Greedy Approach Works

My first test case ([1, 5, 10, 25] for amount 99) showed that for standard coin systems, the greedy algorithm works perfectly. It produced the exact same optimal result as the DP solution. In these "canonical" systems, the greedy choice of taking the largest coin is always the right move.

When the Greedy Approach Fails

The algorithm's weakness became clear in my second test ([1, 3, 4] for amount 6).

- **My Greedy Result:** [4, 1, 1] (3 coins)
- **My DP (Optimal) Result:** [3, 3] (2 coins)

Here, I saw the greedy algorithm fail. By immediately taking the 4, it locked itself into a suboptimal path. It never considered that starting with a smaller coin (3) would lead to a better overall solution. It has tunnel vision and can't see the bigger picture.

Conclusion

My analysis shows that the greedy algorithm's effectiveness depends entirely on the coin system. It’s useful for currencies where the greedy choice is always the best one. For any other case where a truly optimal solution is required, the more thorough dynamic programming method is the only reliable choice.