Greedy Strategies Group Exercises -- **KEY**

Please work the following problems in groups of 2 or 3. Use additional paper as needed, and staple the sheets together before turning them in. **ONLY TURN IN 1 WORKSHEET/ANSWERS PER TEAM.**

Assume that we have a set of coins, and we have an unlimited number of each coin. For our example we will restrict the set of coins to the US coins: \( c = \{c_1, c_2, c_3, c_4, c_5\} \) where \( c_5 \) is worth 1¢, \( c_4 \) is worth 5¢, \( c_3 \) is worth 25¢, \( c_2 \) is worth 50¢, and \( c_1 \) is worth 100¢. Note the coins are sorted in descending order: \( c_1 > c_2 > c_3 > c_4 > c_5 \). Anytime we pay for something with cash and have change returned, the goal of the cashier is to give us the fewest coins possible to make up the value of the change.

Your task is to explore this problem and the various ways to solve it, including divide and conquer, dynamic programming, and greedy strategies: Given an amount \( A \) and the set of \( n \) coins \( c = \{c_1, c_2, ..., c_n\} \), find the minimum number of coins needed to make the amount \( A \). Call this minimum set \( MC \). **We will restrict the problem to US coins.**

1. Assume we are going to solve the problem using recursion and the divide and conquer approach. Characterize the subproblem structure. Answer the following questions.

   a. What are the subproblems that is/are smaller instances of the overall problem?

   **Checking each coin against the amount.** Start with the largest value \( (c_1) \) coin, include it or not, if you do, then look at the rest with respect to the amount \( A-c_1 \). If you don’t then look at the rest with respect to \( A \).

   b. How did you create the subproblems?

   **By checking the coins in order from highest to lowest, and making the amount smaller when a coin is chosen.**

   c. How many subproblems are there? 2

   d. Is the subproblem structure optimal? Why/why not?

   **Yes, because if we find the smallest set of coins for one of the subproblems, it will be included in the smallest set of coins for the overall problem. By contradiction, assume \( MC_{ij} \) is an optimal (minimum size) set of coins, with size \( |MC_{ij}| \). We know that \( MC_{ij} \) was created from 2 subproblems: \( MC_{ik} \) and \( MC_{kj} \). Now assume we found a smaller set of coins for one of the subproblems, so that \( |MC'_{kj}| < |MC_{kj}| \). This means we can use \( MC'_{kj} \) in our solution, so that the solution is now \( MC_{ik} \cup MC'_{kj} \) and has a size \( |MC_{ik}| + |MC'_{kj}| \) which is < \( |MC_{ij}| \). But this is a contradiction because we said that \( MC_{ij} \) is an optimal solution to the overall problem. Thus, the solution to the subproblem \( MC_{kj} \) must be included in the optimal solution \( MC_{ij} \).**

2. Write the recurrence for the solution.

   \[
   MC(i, A) = \begin{cases} 
   \phi & A = 0 \\
   MC(i-1,A) & c_i > A \\
   MC(i, A-c_i) & c_i \leq A \\
   \end{cases}
   \]

   **\( MC(A) = \min_{i=1...n}(MC(A-v_i)) \)**

3a. Since we are first exploring using a divide and conquer approach, what is one potential complexity problem we might have? (Hint: Think about what the call graphs you drew on the dynamic programming worksheet for the recursive version of the rod cutting problem.)
One problem is that checking whether to add or not each coin and then check all the other coins for each decision is an exponential problem. A second problem is that we’re taking the highest coin that is still less than or equal to the value we’re trying to create, so we don’t have to check all the combinations.

3b. What approach could we use instead of divide and conquer to solve the problem you wrote about in 3a?

Dynamic programming could help this.

4. How can we further simplify/reduce the number of subproblems, based on the choice of the coin that we try?

We can take the highest coin that is still less than or equal to the value we’re trying to create, so we don’t have to check all the combinations. Recognizing that we’re going to take the biggest coin possible means that we’re just left with 1 subproblem.

5. What technique does the simplification suggest we should try?

We could still use dynamic programming, but greedy should be investigated.

6. We’ve already done steps 1-3 of the greedy strategy. What are these and which of the problems above addressed them?

1. Determine optimal problem substructure – we did this in #1 above.
2. Develop recursive solution – we did this in #2 above.
3. Show that if we make the greedy choice only 1 subproblem remains – we did this in #4 above.

7. What do we need to prove for the next step of the greedy strategy? What should the central idea of this argument be?

If \( A > 0 \) and there is a coin \( c_i \) that is the largest in value of the set of coins where \( c_i \leq A \), then \( c_i \) is in the minimum set of coins (MC) needed to make the amount \( A \).

8. Write pseudo-code for a greedy algorithm to determine the minimum set of coins needed to make an amount. The coin set is \( c = \{c_1, c_2, ..., c_n\} \), the amount is \( A \), and the minimum set of coins is \( MC \).

```python
x = amount to be changed
S = φ
while \( x > 0 \) {
    let \( k \) be smallest integer such that \( c_k \leq x \)
    if (there is no such \( k \))
        return "no solution found"
    \( x \leftarrow x - c_k \)
    \( S \leftarrow S \cup \{k\} \)
}
return S
```

9. If you were going to code this in Python3, what data structures would you use, and for what purposes?

A list of the coin values, sorted in descending order
A dictionary with the key being the coin value and the value being the number of this coin used in the solution