CHECKING GREEDY ALGORITHM FOR COIN SETS

Problem:
Design an efficient algorithm to determine whether the greedy algorithm will give a minimum sized collection of coins for a given coin set $i_1, i_2, \ldots, i_n$ for all amounts.

Characterization:
We can classify the coin sets based on their distinct characteristics.
Let the coin set be represented by $i_1, i_2, \ldots, i_n$ where $i_1 < i_2 < i_3 < \ldots < i_{n-1} < i_n$.

Case 1: When
- $i_1$ is a factor of $i_2$,
- $i_2$ is a factor of $i_3$,
- $\ldots$,
- $i_k$ is a factor of $i_{k+1}$,
- $\ldots$,
- $i_{n-1}$ is a factor of $i_n$.

Claim: When a coin set has the above-mentioned characteristics, we can say that Greedy algorithm will give an optimal solution for all amounts.

Proof: If $N$ is the total amount of which we need to find coins,
Let $c_1=i_2/i_1, c_2=i_3/i_2, \ldots, c_{n-1}=i_n/i_{n-1}$.

Whenever the amount $N > i_n$, we need to select coins of $i_n$ so that the remaining amount $N_r$ will become less than $i_n$.
And whenever the amount $N > i_k$ and $N < i_{k+1}$ we need to select the coin $i_k$ otherwise for representing $N$ we need to take $c_{k-1}$ coins of $i_{k-1}$. And it will become even worse if we leave $i_{k-1}$ also.
So, whenever amount either initial or subsequent amount $N > i_k$ then we have to select the coin $i_k$ for obtaining optimal solution.

As the Greedy algorithm selects the coins in the exact way as described above i.e., if $N > i_k$ and $N < i_{k+1}$ then it selects $i_k$, the Greedy algorithm gives an optimal solution when the above-mentioned characteristics are satisfied by the coin set.

Example:
<table>
<thead>
<tr>
<th>Coin set</th>
<th>Amount</th>
<th>Greedy solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 5 10 25</td>
<td>30</td>
<td>25,5</td>
</tr>
</tbody>
</table>
Case 2: When \( i_{k-1} \) is not a factor of \( i_k \)

Then there arises different situation. The greedy algorithm may or may not give optimal solution for all the amounts. We can check the greedy solution for all the amounts from \( i_k \) to either \( i_{k+1} \) (if one exists) or \( i_k + i_{k-1} \).

Examples:

<table>
<thead>
<tr>
<th>Coin set</th>
<th>Amount</th>
<th>Solution by Greedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 10, 25</td>
<td>34(25+10-1)</td>
<td>25,1,1,1,1,1,1,1,1,1(total=10 coins)</td>
</tr>
</tbody>
</table>

If we won’t select 25, then 10,10,1,1,1,1,1 (total=7 coins), So, Greedy is not giving an optimal solution.

1, 3, 10 12(10+3-1) 10,1,1(total=3 coins)

If we won’t select 10 then 3,3,3 (total=4 coins), So, Greedy algorithm is giving an optimal though \( i_{k-1} \) is not a factor of \( i_k \).

In my algorithm I am checking for all the amounts less than \( i_n+i_{n-1}-1 \). If greedy algorithm succeeds for all the amounts less than \( i_n+i_{n-1}-1 \) there exists no amount for which greedy algorithm fails.

Claim: If Greedy algorithm gives optimal solution for all the amounts less than or equal to \( i_n+i_{n-1}-1 \) then it will give optimal solution for any amount.

I will prove it by contradiction.

Let \( N \) be the smallest amount for which Greedy algorithm doesn’t give an optimal solution. That is the optimal selection doesn’t contain the coin \( i_n \). \( N \) must be greater than \( i_n+i_{n-1}-1 \) since greedy algorithm gives optimal solution for all amounts less than \( i_n+i_{n-1}-1 \).

Let the optimal selection selects \( i_k \) (other than \( i_n \)) first. Then the amount \( N \) will become \( N-i_k \) (say \( N_1 \)).

\[
N_1 = N - i_k > i_n+i_{n-1}-1-i_k \quad \text{Since} \quad N > i_n+i_{n-1}-1
\]

Since \( i_k \) is less than or equal to \( i_{n-1} \) \( N_1 \) is greater than \( i_n \). But the greedy algorithm gives optimal solution for all the amounts less than \( N \) and hence the optimal solution will contain the coin \( i_n \). This is a contradiction to our assumption that optimal solution doesn’t contain \( i_n \) and hence our assumption that there exists \( N \) for which greedy algorithm fails is FALSE.

Thus Greedy algorithm gives optimal solution for all the amounts.
Based on the above study we can write an algorithm in which we check for all coins whether \( i_{k-1} \) is a factor of \( i_k \) and whenever its not a factor we check for the amounts from \( i_k \) to \( i_{k+1} \) (if one exists) or to \( i_k + i_{k-1} \) and decide based on the situation whether greedy algorithm will give optimal for those amounts or not. If its not we can conclude that greedy algorithm won’t give optimal for all the amounts for the given set of coins.

**Algorithm:**

Algorithm to find whether the Greedy algorithm will give an optimal solution for a given coin set or not. Let the \( V[1,2,3,…..,n] \) be the input coin set in the non decreasing order for which we need to find whether the greedy algorithm gives optimal solution or not. \( V[i] \) is the value of the \( i^{th} \) coin value.

**GREEDYCHECK (V[n])**

```plaintext
{ 
For i ← 3 to n do {
    If ( V[i] mod V[i-1] = 0 ) then continue;
    Else {
        If i=n then \( V_e=V_i+V_{i-1} \) else \( V_e=V_{i+1} \);
        For k ← V[i] to V[e] do {
            count1 ← SOLN(k,i);
            count2 ← SOLN(k,i-1);
            If (count2 < count1) then {
                print (“Greedy algorithm doesn’t give optimal for this set”); 
                return();
            }
        }
    }
}
print (“Greedy algorithm gives optimal for this set”);
}
```

**SOLN (k, i)**

/*function that gives coin count for ‘k’ with using coins from V[1] to V[i] based on Greedy algorithm.*/

```plaintext
{ count ← 0;
While (k > 0) {
    If (V[i] <= k) then {
        n = \lceil k/V[i] \rceil ;
        count ← count + n;
        k ← k – n*V[i];
    }
    Else i ← i – 1;
}
Return (count);
}
```
**Time Complexity:**

**SOLN (k, i):**

The time complexity of the function depends on i and is O(i) since the loop in the function is exactly iterated i times. The maximum value of i can be n where n is the number of coins in the coin set.

Hence the worst case time complexity of SOLN is O(n)

**GREEDYCHECK( ):**

In the worst case when \( i_{k-1} \) is not a factor of \( i_k \) for all \( k \leq n-1 \), the function SOLN is called at most \( i_n + i_{n-1} - 1 \) times.

Worst case time complexity of GREEDYCHECK( ) \( \in O((i_n + i_{n-1} - 1) \times n) \)
\( \in O(i_n \times n) \)

**References:**  *Fundamentals of Algorithms, by Brassard and Bratley,
Course Material for CIS775, by Rodney R Howell.*