GREEDY ALGORITHM

By Harsh Kelawala, Abhipsa Sahoo & Aryan Gupta

DESCRIPTION

Greedy algorithms allow to find a globally optimal solution for a given problem by making successive locally optimal choices (hence the term greedy). That strategy doesn't always lead to a global optima, but works for several well known problems and gives reasonably good approximations for others.

Greedy Algorithm as the name itself implies is an algorithm that is always greedy in taking decisions at each step of process. That is it chooses the best solution(either maximum or minimum / known as local optimum in technical terms) at each step of process assuming that you will end up with a best solution(known as global optimum in technical terms) for the whole problem in the end.

Problem Used:

CHANGE MAKING PROBLEM

DESCRIPTION

The Problem involves finding the minimum number of coins for making the change of a given amount of money. Usually, this problem is referred to as the change-making problem.At first, we’ll define the change-making problem with a real-life example. Next, we’ll understand the basic idea of the solution approach to the change-making problem and illustrate its working by solving our example problem.Then, we’ll discuss the pseudocode of the greedy algorithm and analyze its time complexity. Finally, we’ll point out the limitation of the discussed algorithm and suggest an alternative to overcome it.

Continued:

In the change-making problem, we’re provided with an array D = {d1,d2,d3….dm} of m distinct coin denominations, **where each denomination has an infinite supply**. We need to find an array having a minimum number of coins that add up to a given amount of money , provided that there exists a viable solution. Let’s consider a real-life example for a better understanding of the change-making problem.

Let’s assume that we’re working at a cash counter and have an infinite supply of D = { 1,2,10 } valued coins. Now, we need to return one of our customers an amount of n = 15 using the minimum number of coins.

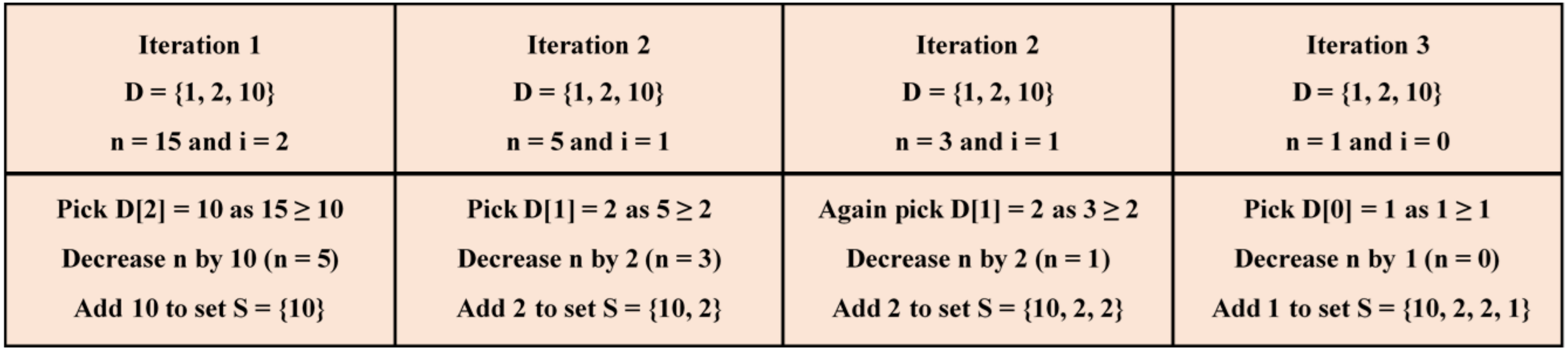
SOLUTION
APPROACH

The greedy algorithm finds a feasible solution to the change-making problem iteratively.

At each iteration, it selects a coin with the largest denomination, say **D[i]** ∈ **D** , such that **n >= D[i]** . Next, it keeps on adding the denomination **D[i]** to the solution array ***S*** and decreasing the amount ***n*** by **D[i]** as long as **n >= D[i]** . This process is repeated until ***n*** becomes zero.

Continued:

Let’s now try to understand the solution approach by solving the example above. The image below shows the step-by-step solution to our problem:



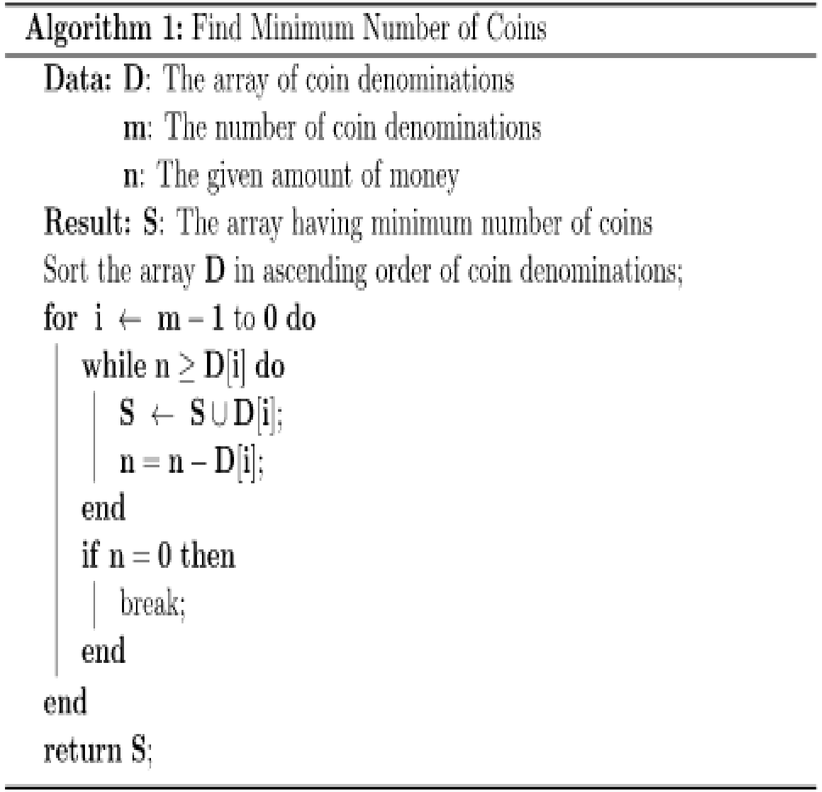
Iteration 1
D = {1, 2, 10}
n = 15 and i = 2
	Iteration 2
	D = {1, 2, 10}
	n = 5 and i = 1
		Iteration 2
		D = {1, 2, 10}
		n = 3 and i = 1
			Iteration 3
			D = {1, 2, 10}
			n = 1 and i = 0

Pick D[2] = 10 as 15 ≥ 10 Decrease n by 10 (n = 5) Add 10 to set S = {10}	Pick D[1] = 2 as 5 ≥ 2 Decrease n by 2 (n = 3) Add 2 to set S = {10, 2}	Again pick D[1] = 2 as 3 ≥ 2 Decrease n by 2 (n = 1) Add 2 to set S = {10, 2, 2}	Pick D[0] = 1 as 1 ≥ 1 Decrease n by 1 (n = 0) Add 1 to set S = {10, 2, 2, 1}

Hence, we require minimum four coins to make the change of amount ***n = 15*** and their denominations are ***S = { 10 , 2 , 2 , 1 }*** .

Pseudocode:

Now that we know the basic idea of the solution approach, let’s take a look at the pseudocode of the algorithm:



```
Algorithm 1: Find Minimum Number of Coins
Data: D: The array of coin denominations
      m: The number of coin denominations
      n: The given amount of money
Result: S: The array having minimum number of coins
Sort the array D in ascending order of coin denominations;
for i ← m − 1 to 0 do
    while n ≥ D[i] do
        S ← S ∪ D[i];
        n = n − D[i];
    end
    if n = 0 then
        break;
    end
end
return S;
```

To begin with, we sort the array of coin denominations in ascending order of their values.

Next, we start from the last index of the array and iterate through it till the first index. At each iteration, we add as many coins of each denomination as possible to the solution array and decrement by the denomination for each added coin.

Once becomes zero, we stop iterating and return the solution array as an outcome.

TIME COMPLEXITY ANALYSIS

Let's now analyze the time complexity of the algorithm above. We can sort the array **D** of coin denominations in ***O(m*(log2m))*** time.

Similarly, the for loop takes ***O(n)*** time, as in the worst case, we may need coins to make the change. Hence, the **overall time complexity of the greedy algorithm becomes *O(n)*** since ***m <<< n*** . Although, we can implement this approach in an efficient manner with ***O(m*(log2m))*** time.

CONCLUSION:

In this article, we’ve studied a greedy algorithm to find the least number of coins for making the change of a given amount of money and analyzed its time complexity. Further, we’ve also illustrated the working of the discussed algorithm with a real-life example

THANK YOU