Greedy Algorithm

Greedy algorithm greedily selects the best choice at each step and hopes that these choices will lead us to the optimal solution of the problem. Of course, the greedy algorithm doesn't always give us the optimal solution, but in many problems it does. For example, in the coin change problem of the Coin Change chapter (/course/algorithms-coin-change/), we saw that selecting the coin with the maximum value was not leading us to the optimal solution. But think of the case when the denomination of the coins are 1¢, 5¢, 10¢ and 20¢. In this case, if we select the coin with maximum value at each step, it will lead to the optimal solution of the problem.

Also as stated earlier, the fraction knapsack can also be solved using greedy strategy i.e., by taking the items with the highest $\frac{value}{weight}$ ratio first.

Thus, checking if the greedy algorithm will lead us to the optimal solution or not is our next task and it depends on the following two properties:

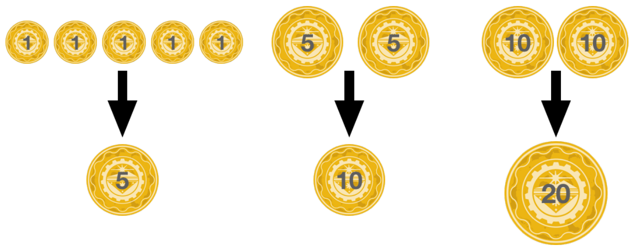
- **Optimal substructure** $\rightarrow$ If the optimal solutions of the sub-problems lead to the optimal solution of the problem, then the problem is said to exhibit the optimal substructure property.
- **Greedy choice property** $\rightarrow$ The optimal solution at each step is leading to the optimal solution globally, this property is called greedy choice property.

Implementation of the greedy algorithm is an easy task because we just have to choose the best option at each step and so is its analysis in comparison to other algorithms like divide and conquer but checking if making the greedy choice at each step will lead to the optimal solution or not might be tricky in some cases. For example, let's take the case of the coin change problem with the denomination of 1¢, 5¢, 10¢ and 20¢. As stated earlier, this is the special case where we can use the greedy algorithm instead of the dynamic programming to get the optimal solution, but how do we check this or know if it is true or not?

Let's suppose that we have to make the change of a number n using these coins. We can write n as $5x + y$, where x and y are whole numbers. It means that we can write any value as multiple of 5 + some remainder. For, example 4 can be written as $5 * 0 + 4$, 7 can be written as $5 * 1 + 2$, etc.

Now, the value of y will range from 0 to 4 (if it becomes greater than or equal to 5, then it will be covered in the $5x$ part) and we can check that any value between 0 to 4 can be made only by using all coins of value 1. So, we know that the optimal solution for the part y will contain coins of value 1 only. Therefore, we will consider for the optimal solution of the $5x$ part. As the problem exhibits optimal substructure, so the optimal solution to both the subproblems will lead to the optimal solution to the problem.

Since $5x$ is a multiple of 5, so it can be made using the values 5, 10 and 20 (as all three are multiples of 5). Also in 5, 10 and 20, the higher value is multiple of the lower ones. For example, 20 is multiple of 5 and 10 both and 10 is multiple of 5. So, we can replace the multiple occurrences of the smaller coins with the coins having higher value and hence, can reduce the total number of coins. For example, if 5 is occurring more than once, it can be replaced by 10 and if 10 is occurring more than once it can be replaced by 20. In other words, we can choose the coins with higher value first to reduce the total number of coins.



So, we have just checked that we can apply the greedy algorithm in this case.

</>
Take note that the method to verify if the greedy algorithm can be applied or not is more of a brain work than following any rules and one can use any different method (or different thinking) to verify the same. But make sure that you have verified it correctly.

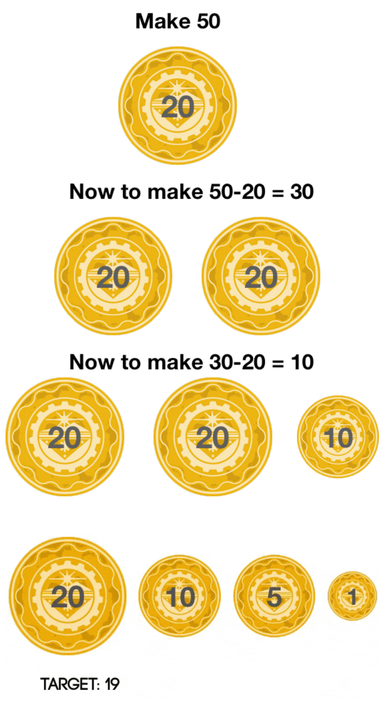
We are going to see more greedy algorithms in this course. So, you will become more comfortable with the greedy algorithm with the progress of this course.

Let's code the above coin change problem and get more familiar with the greedy algorithm.

Coin Change Problem with Greedy Algorithm

Let's start by having the values of the coins in an array in reverse sorted order i.e., `coins = [20, 10, 5, 1]`.

Now if we have to make a value of n using these coins, then we will check for the first element in the array (greedy choice) and if it is greater than n, we will move to the next element, otherwise take it. Now after taking one coin with value `coins[i]`, the total value which we have to make will become n-coins[i].



```
i = 0
while (n)
   if coins[i] > n
      i++
```

`if coins [i] > n` → We are starting from the 0th element (element with the largest value) and checking if we can use this coin or not. If the value of this coin is greater than the value to be made, then we are moving to the next coin - `i++`.

If the value of the coin is not greater than the value to be made, then we can take this coin. So, we will take it. Let's just print the value right here to indicate we have taken it, otherwise, we can also append these value in an array and return it.

```
while (n)
   ...
   else
      print coins[i]
```

Now, the value to be made is reduced by coins[i] i.e., `n-coins[i]`.

```
COIN-CHANGE-GREEDY(n)
  coins = [20, 10, 5, 1]
  i = 0

  while (n)
    if coins[i] > n
      i++
    else
      print coins[i]
      n = n-coins[i]
```

C **Python** **Java**

```c
#include <stdio.h>

void coin_change_greedy(int n) {
  int coins[] = {20, 10, 5, 1};
  int i=0;

  while(n) {
    if(coins[i] > n) {
      i++;
    }
    else {
      printf("%d\t",coins[i]);
      n = n-coins[i];
    }
  }
  printf("\n");
}

int main() {
  int i;
  for(i=1; i<=20; i++) {
    coin_change_greedy(i);
  }
  return 0;
}
```

Analysis of the Algorithm

We can easily see that the algorithm is not going to take more than linear time. As n is decreased by coins[i] at the end of the while loop, we can say that for most of the cases it will take much less than $O(n)$ time.

So, we can say that our algorithm has a $O(n)$ running time.

“ A positive attitude can really make dreams come true - it did for me. ”

- David Bailey

PREV (/course/algorithms-knights-tour-problem/) (/course/algorithms-activity-selection/) NEXT