Greedy Algorithm

In this tutorial, you will learn what a Greedy Algorithm is. Also, you will find an example of a greedy approach.

A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment. It doesn't worry whether the current best result will bring the overall optimal result.

The algorithm never reverses the earlier decision even if the choice is wrong. It works in a top-down approach.

This algorithm may not produce the best result for all the problems. It's because it always goes for the local best choice to produce the global best result.

However, we can determine if the algorithm can be used with any problem if the problem has the following properties:

1. Greedy Choice Property

If an optimal solution to the problem can be found by choosing the best choice at each step without reconsidering the previous steps once chosen, the problem can be solved using a greedy approach. This property is called greedy choice property.

2. Optimal Substructure

If the optimal overall solution to the problem corresponds to the optimal solution to its subproblems, then the problem can be solved using a greedy approach. This property is called optimal substructure.

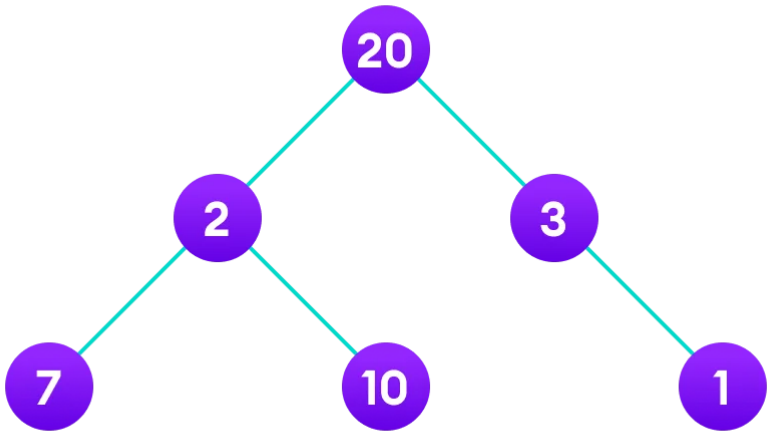
Advantages of Greedy Approach

- The algorithm is **easier to describe**.
- This algorithm can **perform better** than other algorithms (but, not in all cases).

Drawback of Greedy Approach

As mentioned earlier, the greedy algorithm doesn't always produce the optimal solution. This is the major disadvantage of the algorithm

For example, suppose we want to find the longest path in the graph below from root to leaf. Let's use the greedy algorithm here.



Apply greedy approach to this tree to find the longest route

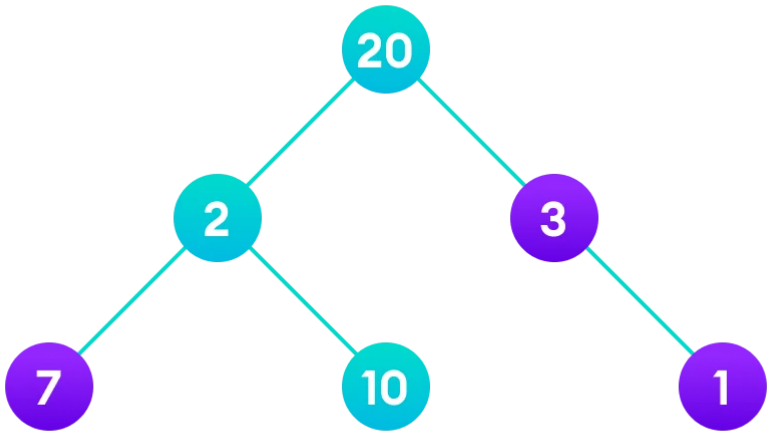
Greedy Approach

1. Let's start with the root node **20**. The weight of the right child is **3** and the weight of the left child is **2**.

2. Our problem is to find the largest path. And, the optimal solution at the moment is **3**. So, the greedy algorithm will choose **3**.

3. Finally the weight of an only child of **3** is **1**. This gives us our final result `20 + 3 + 1 = 24`.

However, it is not the optimal solution. There is another path that carries more weight (`20 + 2 + 10 = 32`) as shown in the image below.



Longest path

Therefore, greedy algorithms do not always give an optimal/feasible solution.

Greedy Algorithm

1. To begin with, the solution set (containing answers) is empty.
2. At each step, an item is added to the solution set until a solution is reached.
3. If the solution set is feasible, the current item is kept.
4. Else, the item is rejected and never considered again.

Let's now use this algorithm to solve a problem.

Example - Greedy Approach

```
Problem: You have to make a change of an amount using the smallest possible number of coins.
Amount: $18

Available coins are
  $5 coin
  $2 coin
  $1 coin
There is no limit to the number of each coin you can use.
```

Solution:

1. Create an empty `solution-set = { }`. Available coins are `{5, 2, 1}`.
2. We are supposed to find the `sum = 18`. Let's start with `sum = 0`.
3. Always select the coin with the largest value (i.e. 5) until the `sum > 18`. (When we select the largest value at each step, we hope to reach the destination faster. This concept is called **greedy choice property**.)
4. In the first iteration, `solution-set = {5}` and `sum = 5`.
5. In the second iteration, `solution-set = {5, 5}` and `sum = 10`.
6. In the third iteration, `solution-set = {5, 5, 5}` and `sum = 15`.
7. In the fourth iteration, `solution-set = {5, 5, 5, 2}` and `sum = 17`. (We cannot select 5 here because if we do so, `sum = 20` which is greater than 18. So, we select the 2nd largest item which is 2.)
8. Similarly, in the fifth iteration, select 1. Now `sum = 18` and `solution-set = {5, 5, 5, 2, 1}`.

Different Types of Greedy Algorithm

- Selection Sort
- Knapsack Problem
- Minimum Spanning Tree
- Single-Source Shortest Path Problem

- Job Scheduling Problem
- Prim's Minimal Spanning Tree Algorithm
- Kruskal's Minimal Spanning Tree Algorithm
- Dijkstra's Minimal Spanning Tree Algorithm
- Huffman Coding
- Ford-Fulkerson Algorithm