OBJECT ORIENTED ANALYSIS & DESIGN
DATA STRUCTURES & ALGORITHMS

Greedy Algorithms

Optimization Problems

- A computational problem in which the objective is to find the best of all possible solutions. More formally, find a solution in the feasible region which has the minimum (or maximum) value of the objective function.

- Shortest path is an example of an optimization problem: we wish to find the path with lowest weight.

General Characteristics of Optimization Problems

- **Ingredients:**

  Instances: The possible inputs to the problem.

  Solutions for Instance: Each instance has an exponentially large set of valid solutions.

  Cost of Solution: Each solution has an easy-to-compute cost or value.

- **Specification**

  Preconditions: The input is one instance.

  Postconditions: A valid solution with optimal cost. (minimum or maximum)

Greedy Solutions to Optimization Problems

In order to get what you want, just start grabbing what looks best.

Surprisingly, many important and practical optimization problems can be solved this way.

To optimize, you start with the most desirable solution at that point of time. You don't worry about whether this choice will hurt your optimization in the coming future.

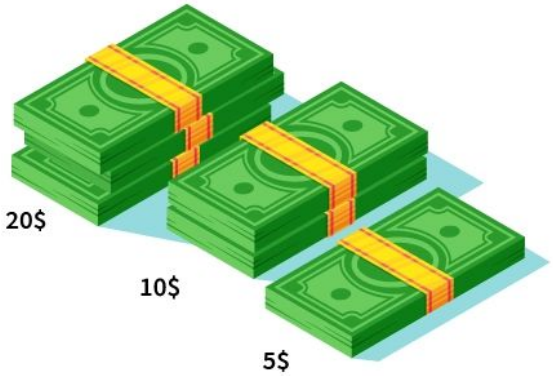
A greedy algorithm refers to any algorithm employed to solve an optimization problem where the algorithm proceeds by making a locally optimal choice (that is a greedy choice) in the hope that it will result in a globally optimal solution.

Example 1: Giveaway

Imagine a situation where a company is having a promotional giveaway event where you can take as much money as possible in one minute from 3 piles of INR 5, INR 10, and INR 20 currency notes. Each pile has a finite number of currency notes.

The catch is that you can take only one currency note off the pile at one time.

Your Bag



Example 1: Solution by Greedy Algorithm

The Greedy Choice: Commit to the object that looks the “best''

Solution: To maximize the amount you take you will obviously start with the INR 20 pile. If you're able to finish that pile in under one minute, then you'll move to the INR 10 pile (and not the INR 5 pile). If you're extremely fast here too, then finally you'll go for the INR 5 pile.

Points to note

- The greedy solution to an optimization problem is very **intuitive** sometimes. For example, in the example discussed above, it just felt right that we choose the notes with the higher denomination to maximize our profit. Lucky for us, it worked! But usually, it does not yield the globally optimal solution.

- Hence, it is always advised to prove the correctness of the greedy algorithm before using it to solve any problem.

- As intuitive as it is, it is not always easy to prove mathematically that the greedy algorithm will lead to the globally optimal solution.

Algorithms and protocols using Greedy approach

Some common algorithms and protocols that use a Greedy approach are:

1. Prim-Jarnik's and Kruskal's **Minimal Spanning Tree (MST) algorithms**
2. Djikstra's **Shortest Path algorithm**
3. Activity selection problem
4. Huffman Coding (used for lossless data compression)
5. **Open-Shortest-Path-First** routing protocol (it uses Djikstra's algorithm)

Characteristics of Greedy approach

- Any greedy algorithm consists of a sequence of choices/ decisions (for the problem we're solving) and each choice made should be the best possible one at that time

- The choice made at any point cannot be altered later.

- The greedy algorithm may not always yield the globally optimal solution and may even result in the worst global solution (such as in the Travelling Salesman Problem).

- Greedy algorithms are very fast as compared to their alternatives, such as dynamic programming. This is because dynamic programming consider all possible cases locally, while the greedy approach goes ahead with only one optimal choice.

- Greedy algorithms are very intuitive, but it is very hard to prove mathematically the correctness of the algorithm.

- The greedy algorithm may lead to a solution that is very close to the optimal solution in problems where there is no efficient solution. It is to be noted that the obtained solution in such cases is not an exact one, rather an approximate one.

Why to use Greedy Algorithm then?

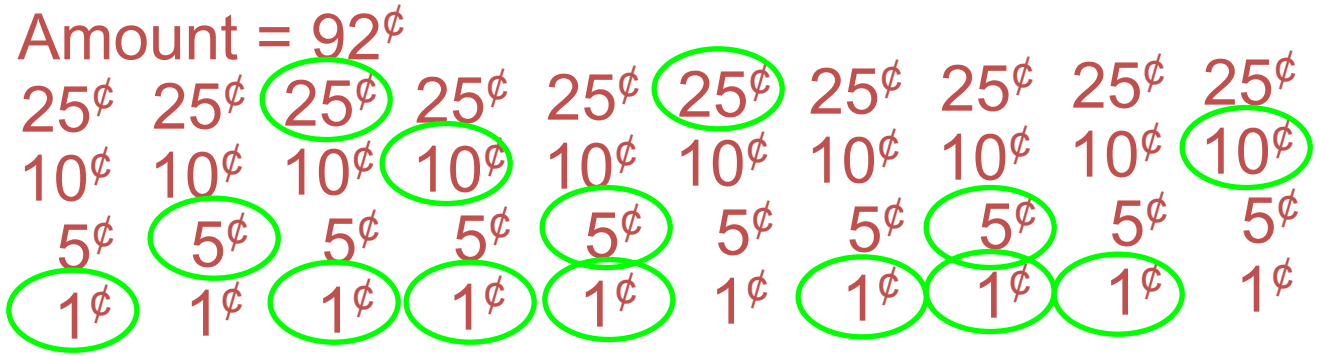
- In many cases greedy algorithm is the most intuitive approach to solve a given optimization problem, that is, it is often the first guessed solution and is easy to formulate.

- **Correct greedy algorithms** are often more powerful and fast as compared to other approaches such as dynamic programming. This is because the greedy approach has to consider only one locally optimal choice while moving ahead with the solution, while dynamic programming must check all possible cases.

- Greedy algorithms work for a wide range of problems (such listed here), many of which have real-life applications in fields such as designing networking protocols and scheduling multiple events.

Making Change Example

Instance: A drawer full of coins and an amount of change to return



Solutions for Instance: A subset of the coins that total the amount.

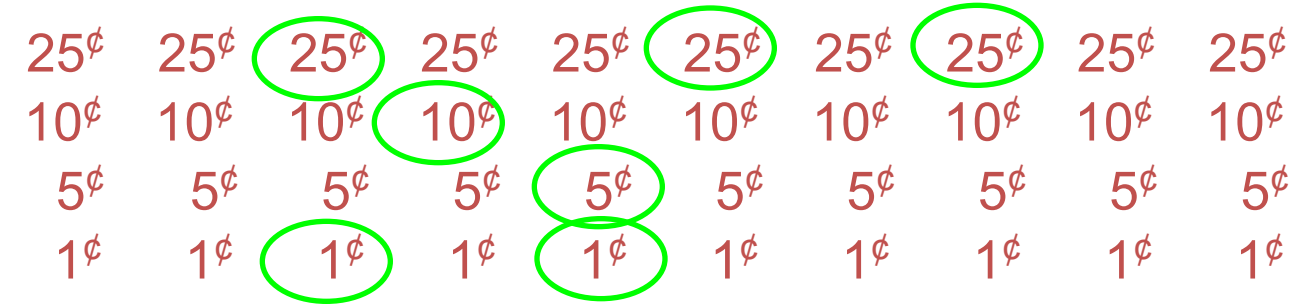
Cost of Solution: The number of coins in the solution = 14

Goal: Find an optimal valid solution.

Making Change Example

Instance: A drawer full of coins and an amount of change to return

Amount = 92¢



Greedy Choice:

Start by grabbing quarters until exceeds amount, then dimes, then nickels, then pennies.

Does this lead to an optimal number of coins?

Cost of Solution: 7

Architecture of a Greedy Algorithm

The basic architecture of the implementation of any greedy algorithm is as follows:

Input to Algorithm: Function = F, Set of Inputs = C
Output: Solution Set Which Maximizes or Minimizes Function

Steps:

```
1. Initialize an empty solution set, S = {}
2. While (C is not empty):
    # step 1: choose an item from C
    current_choice = Select(C)

    # step 2: if current_choice is feasable, add to S
    if(isFeasable(current_choice)):
        S.add(C)

    # step 3: remove the current_choice from C
    C.remove(current_choice)
3. Return S
```

- Select() function makes a greedy choice from the input set C
- isFeasable() function checks whether the choice made by Select() function leads to an optimal value of F

Properties of a Greedy Algorithm

There is no defined mathematical way to check the correctness of a given greedy algorithm. But it has been observed that many optimization problems that can be solved using some greedy algorithm satisfy 2 properties as given below:

1. Greedy choice property
2. Optimal substructure property

To understand these 2 properties in a better way, let us take an example problem.

The Indian Coin Change Problem

If we want to make a change for XX rupees, and we have an infinite supply of each of the denominations in Indian currency, that is, {1, 2, 5, 10, 20, 50, 100, 500, 1000} valued coins, what is the minimum number of coins needed to make the change?

Properties of a Greedy Algorithm

- A problem is said to exhibit greedy choice property if the globally optimal solution can be reached by making locally optimal (greedy) choices. These choices are not dependent on our future choices, and can not be altered after they have been made.
- It is equivalent to showing that there is an optimal solution to the problem that is obtained by only greedy choices.

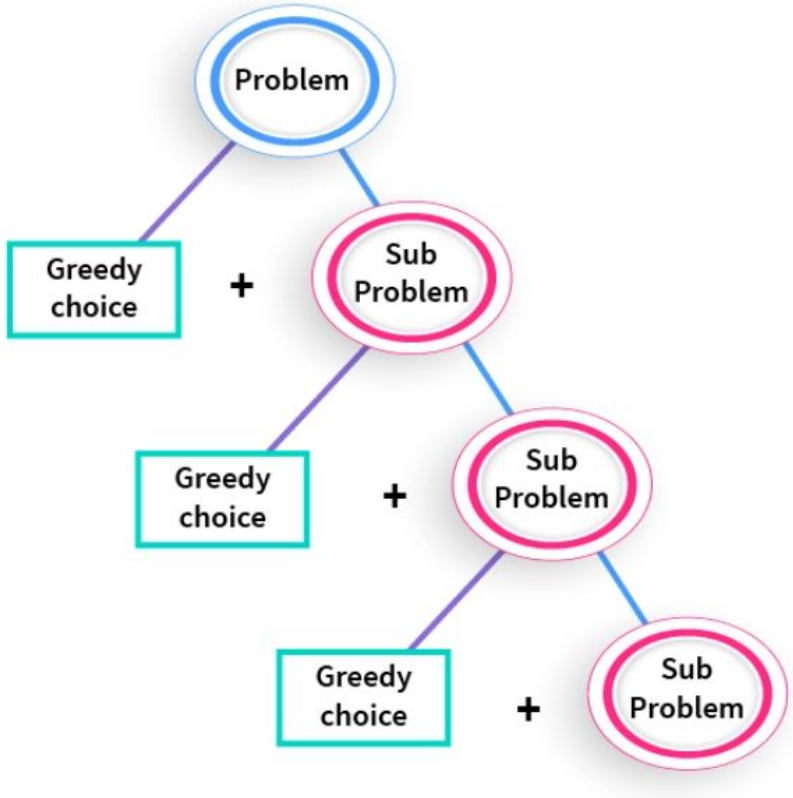
It can be seen using brute force solutions that the optimal solution for some XX, say 76, is made up of greedy choices only as follows:

\ce76 ->[-50] 26->[-20] 6->[-5] 1->[-1]0

So, the solution set comprising of the minimum number of coins required for exchange is \{1,5,20,50\}{1,5,20,50}. As we can see, it consists of all the greedy choices, that is, choosing the coin of the largest denomination that is equal to or less than the value of the amount left.

Optimal Substructure

A problem is said to exhibit optimal substructure property if the global optimal solution to the said problem can be constructed by combining the greedy choice made with optimal solutions to the subproblem of the original problem.



Schematic Diagram of Optimal Substructure

Let's break the above problem (for X = 76) into 2 parts:

- The greedy choice: Choosing the largest possible coin denomination ≤ 76, that is the coin with value 50.
- The Subproblem: Applying our greedy algorithm on Y = X-50 =26. As was seen above, the solution set for the subproblem is {20, 5, 1}.
- Combining Solutions: Combining the greedy choice with the solution set of the subproblem, we get

  {50} U {20, 5, 1} = {50, 20, 5, 1}

Output :

- that was the globally optimal solution for X=76X=76 as was shown in the above section.

- Since combining greedy choice with the subproblem's optimal solution gives the global optimal solution to the original problem, we can say this problem exhibits optimal substructure.

Key Terminologies used in Greedy Algorithms

- Objective Function
- Candidate Set
- Selection Function
- Feasibility Function

Advantages of Greedy Algorithm

1. All greedy algorithms are very intuitive and can be guessed easily.

2. Greedy algorithms are generally easy to understand as well as implement.

3. If correct, greedy algorithms find the optimal solution much faster as well more efficiently than its alternatives, such as dynamic programming.

Disadvantages of Greedy Algorithm

1. Though very intuitive, it is very hard to prove mathematically the correctness of a greedy algorithm to solve the given problem at hand.

2. A greedy approach to solve optimization problems may not always be correct, may even lead to the worst possible solution (as in the case of the Travelling Salesman Problem).