CC
GROUP

Process Improvement Series

Greedy Algorithms

By Farid ALVI

Corporate Consulting Group

Introduction

Greedy algorithmic paradigm builds up a solution piece by piece, always choosing the next most favourable piece that seems offering the most obvious and immediate benefit. These algorithms are used for optimization problems.

Some of the popular Greedy Algorithms are:

- Fractional Knapsack,
- Dijkstra's algorithm,
- Kruskal's algorithm,
- Huffman coding and
- Prim's Algorithm

Characteristics of Greedy Algorithm

- Simple and easy to implement.
- Efficient in terms of **time complexity,**
- Mostly providing quick solutions
- Do not reconsider previous choices
- Make decisions based on current information without looking ahead
- The **greedy algorithm is not always the optimal solution** for every optimization problem, as shown in the example ahead.

How does the Greedy approach work?

1. Starting with the initial state of the problem, you begin making choices
2. Evaluate and Consider all possible choices you can make from that specific moment
3. Choose the option that seems best at that moment, regardless of future consequences. (This is the "greedy" part - you take the best option available now, even if it might not be the best in the long term)
4. Move to the new state based on your chosen option. This becomes your new starting point for the next iteration.
5. Repeat steps 2-4 until you reach the goal state or no further progress is possible. (Choosing the best local choices until you reach the end of the problem or get stuck).

Example

Let's say you have a set of coins with values [1, 2, 5, 10] and you need to give minimum number of coin to someone for a total of 39.

- *Step-1: Start with the highest coin value that is less than or equal to the amount to be given. In this case, the largest coin less than to 39 is 10.*
- *Step- 2: Subtract the largest coin value from the amount to be paid and add the coin to the solution. In this case, subtracting 10 from 39 gives 29, and we add one 10-coin to the solution.*
- *Repeat steps 1 and 2 until the amount to be paid becomes 0.*

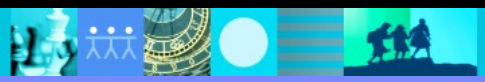
Example

- Needing to pay 39, first of all you pay a coin of 10
- That leaves 39 - 10 = 29 to pay more
- Then you pay a coin of 10, leaving 29 - 10 = 19 to pay more
- You pay another coin of 10, leaving 19 – 10 = 9 to pay more
- Now to pay 9 the nearest choice is 5, leaving 9 – 5 = 4 to pay more
- Paying the nearest choice 2, leaving 4 – 2 = 2 to pay more
- Lastly paying a coin of 2, leaves 2 – 2 = 0 to pay more
- This completes the solution

Example

Considering that you have to pay a total of 39, while having coins in denomination of 10, 5, 2, 1.

Stages →	1	2	3	4	5	6
To pay	**= 39**	**29**	**19**	**9**	**4**	**2**
Paid	**= 10**	**10**	**10**	**5**	**2**	**2**
Leaving to pay	**= 29**	**19**	**9**	**4**	**2**	**0**

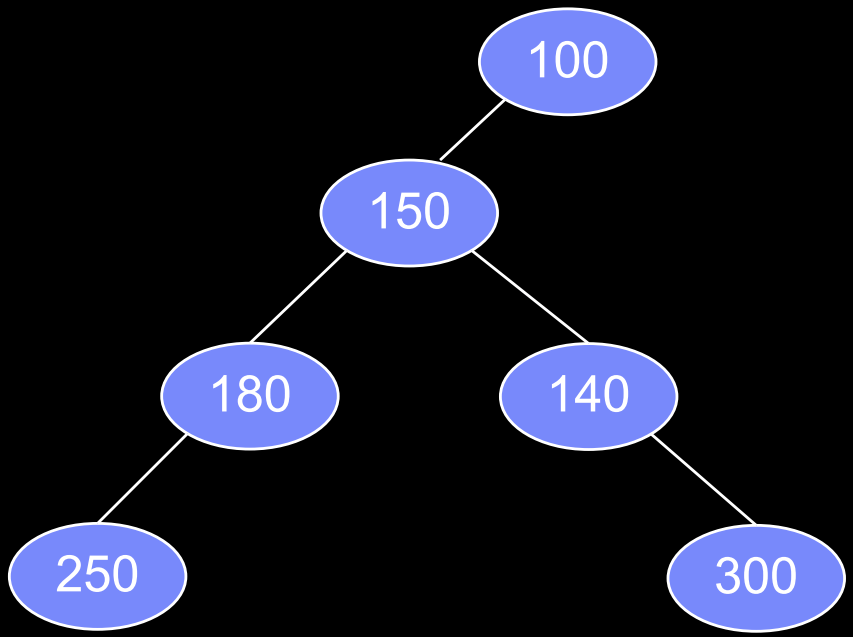


Example

- A taxi driver comes to the taxi stand, where he drops a passenger who pays him rupees 100.
- At this stand he gets two new rides one to pay 150 and other for 120. He chooses the ride for 150.
- At the drop of this ride he again gets two options. One for 180 and other for 140. He picks up the ride for 180.
- When he drops this ride he gets another for 250. He takes this as the last ride.
- His total income today is 100 + 150 + 180 + 250 = 680
- Do you think he choose all best options? Let see what other options could yeild.
- Had he choosen 100 + 150 + 140 + 300 = 690. In this option he would have earned more. So to say that his choosen route was not the best option.

Example Illustration

Taxi Routing Problem



100 + 150 + 180 + 250 = 680

100 + 150 + 140 + 300 = 690

Conclusion

- Greedy algorithms are effective tools for optimization. Some well-known problems that are commonly solved using greedy strategies include the Coin Change Problem, the 0/1 Knapsack Problem, the Fractional Knapsack Problem, and Dijkstra's Shortest Path Algorithm.
- Considering the above example, Greedy Algorithmic approach is not always optimum.
- The greedy algorithm can occasionally fail based on the choices it makes. It is effective when making locally optimal choices that consistently lead to the best overall solution. However, it may not succeed when a short-term decision blocks access to a better long-term option. It can fail sometimes, depending upon the choices it makes.

Conclusion

- A greedy algorithm solves problems by making the best possible choice at each step, aiming for a globally optimal solution.
- Greedy algorithms are fast and efficient, often running in linear or logarithmic time, but they do not always guarantee the best possible solution for all problems.
- Once a choice is made, it is final and not revisited. This makes greedy algorithms faster but less flexible.
- Greedy algorithms are used for optimization problems like the Coin Change problem, Fractional Knapsack, and Dijkstra's Shortest Path Algorithm.
- Greedy algorithms are commonly applied to problems such as resource allocation, pathfinding, and making decisions under constraints.
- They can fail if a locally optimal choice blocks access to a better long-term solution.