

Greedy Programming

Greedy is an *algorithmic paradigm* that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So the problems where choosing locally optimal also leads to global solution are best fit for Greedy.

If a Greedy Algorithm can solve a problem, then it generally becomes the best method to solve that problem as they in general are more efficient than other techniques like Dynamic Programming. But Greedy algorithms cannot always be applied e.g 0-1 Knapsack vs fractional Knapsack.



Assume that you have an objective function that needs to be optimized (either maximized or minimized) at a given point. A Greedy algorithm makes greedy choices at each step to ensure that the objective function is optimized. The Greedy algorithm has only one shot to compute the optimal solution so that it never goes back and reverses the decision.

Greedy algorithms have some advantages and disadvantages:

- It is quite easy to come up with a greedy algorithm (or even multiple greedy algorithms) for a problem.
- Analyzing the run time for greedy algorithms will generally be much easier than for other techniques (like Divide and conquer). For the Divide and conquer technique, it is not clear whether the technique is fast or slow. This is because at each level of recursion the size of gets smaller and the number of sub-problems increases.
- The difficult part is that for greedy algorithms you have to work much harder to understand correctness issues. Even with the correct algorithm, it is hard to prove why it is correct. Proving that a greedy algorithm is correct is more of an art than a science. It involves a lot of creativity.

The greedy algorithms are sometimes also used to get an approximation for Hard optimization problems. For example, *Traveling Salesman Problem* is a NP-Hard problem. A Greedy choice for this problem is to pick the nearest unvisited city from the current city at every step. This solutions don't always produce the best optimal solution but can be used to get an approximately optimal solution.

Popular Greedy Programming questions

- Fractional Knapsack Problem 
- Activity Selection Problem 
- Job Sequencing Problem

Greedy Algorithms in Graph (to be done with Graphs chapter)

- Kruskal's Minimum Spanning Tree
- Prim's Minimum Spanning Tree
- Dijkstra's shortest path algorithm
- Graph Coloring