Greedy Algorithms in DSA: Beginner to Advanced Guide

Subtitle: Learn the greedy approach to solve optimization problems efficiently and understand when it guarantees an optimal solution.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Greedy algorithms solve problems by making the **best local choice at each step**. They are used in optimization problems like interval scheduling, minimum spanning trees, and coin change. Understanding greedy strategy helps in solving problems **quickly and efficiently**.

Step 1: What is a Greedy Algorithm?

- Make a **locally optimal choice** at each step
- Hope that **global optimum** is reached
- Works when problem has **Greedy Choice Property** and **Optimal Substructure**

Example: Coin Change Problem (with standard denominations)

```
let coins = [25,10,5,1];

let amount = 63;

let count = 0;



for (let coin of coins) {

  count += Math.floor(amount/coin);

  amount %= coin;

}

console.log(count); // 6 coins
```

Step 2: Steps to Approach Greedy Problems

1. **Understand the problem** and check if greedy works
2. **Identify the greedy choice** → the choice that looks best now
3. **Prove optimality** if possible (optional for practice)

4. **Implement iteratively** or using sorting/priority queue

Step 3: Common Greedy Patterns

1. Interval Scheduling / Activity Selection

- Sort intervals by **finish time**
- Pick earliest finishing activity that doesn't overlap

2. Minimum Spanning Tree (MST)

- **Kruskal's Algorithm** → sort edges, union-find
- **Prim's Algorithm** → grow tree from starting node

3. Huffman Encoding

- Build binary tree using **priority queue** for compression

4. Fractional Knapsack

- Take items with **max value/weight ratio** first

Step 4: Example Problems

1. Interval Scheduling

```
let intervals = [[1,3],[2,4],[3,5]];

intervals.sort((a,b)=>a[1]-b[1]);



let last = -Infinity, count = 0;

for (let [start,end] of intervals) {

  if (start >= last) {

    count++;

    last = end;

  }

}

console.log(count);
```

2. Fractional Knapsack

```
let items = [{w:10,v:60},{w:20,v:100},{w:30,v:120}];
```

```
let W = 50;

items.sort((a,b)=>b.v/a.w - a.v/b.w);


let total = 0;

for (let item of items) {

  if (W >= item.w) {

    total += item.v;

    W -= item.w;

  } else {

    total += item.v * (W/item.w);

    break;

  }

}

console.log(total);
```

3. Job Sequencing Problem

- Assign jobs with deadlines to maximize profit

Step 5: Common Mistakes

- Assuming greedy works **for all problems** (not always optimal)
- Ignoring **sorting or priority queue requirement**
- Missing edge cases like **equal weights or intervals**
- Not verifying **greedy choice property**

Step 6: Practice Plan for Greedy Algorithms

Day 1:

- Coin change, activity selection → 5 problems

Day 2:

- Fractional knapsack → 5 problems

Day 3:

- MST: Kruskal and Prim → 5–7 problems

Day 4:

- Job sequencing → 5 problems

Day 5–6:

- Mixed greedy problems → 10–12 problems

Mini Exercises

1. Minimum number of platforms at railway station
2. Maximum meetings in a room
3. Minimize number of coins for amount (given denominations)
4. Huffman coding simulation for a string

Key Takeaways

- Greedy algorithms make **local optimal choices**
- Works only if problem has **greedy choice property**
- Sorting and priority queues are commonly used
- Excellent for **optimization problems with simple patterns**
- Practice to **recognize greedy-friendly problems**

Visit haas.dev for more DSA greedy algorithm guides, problem sets, and interview preparation resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app