Greedy Algorithms

Josh Wilcox (jw14g24)

April 28, 2025

Table of Contents

Overview of Greedy Algorithms

What is the Greedy Strategy

- The **greedy** strategy is a **design paradigm**
- It uses algorithms that make **locally optimal** choices at each step
  - No planning ahead - just what's directly around you
- You **hope** that each of these local choices will *eventually* lead to a **globally** optimal solution
- Greedy algorithms tend to be very efficient
- Greedy techniques are good for approximation of harder problems - to know *wherabouts* the actual optimal solution should be

Job Scheduling

Assume we have n jobs, each with weight w_i and length l_i, $1 \leq i \leq n$. The jobs share some resource (i.e. CPU) so they must be run sequentially. If we run the jobs in the order 1, 2, 3, ... then job i has completion time $c_i = \sum_{k=1}^{i} l_k$. Our goal is to minimize the quantity $f = \sum_{k=1}^{n} w_k \cdot c_k$. In particular, we would like to have a greedy algorithm that minimizes f.

Special Case 1 - Assume all jobs have same weight

$$\begin{aligned} f &= w(l_1 + (l_1 + l_2) + (l_1 + l_2 + l_3) + \ldots + (l_1 + l_2 + \ldots l_n)) \\ &= w\left(n \cdot l_1 + (n-1) \cdot l_2 + (n-2) \cdot l_3 + (n-3) \cdot l_4 + \ldots + 1 \cdot l_n\right) \end{aligned}$$

- f will be minimized here if the order is selected such that:

$$l_1 \leq l_2 \leq \ldots \leq l_n$$

Special Case 2 - Assume all jobs have the same length

$$\begin{aligned} f &= (w_1 \cdot l) + (w_2 \cdot 2l) + (w_3 \cdot 3l) + \ldots + (w_n + nl) \\ &= \sum_{i=1}^{n} (w_i \cdot i \cdot l) = l \cdot \sum_{i=1}^{n} (w_i \cdot i) \end{aligned}$$

- f will be minimized here if the order is selected such that:

$$w_1 \geq w_2 \geq \ldots \geq w_n$$

- Choose the biggest weight first

General Case - Optimal Solution

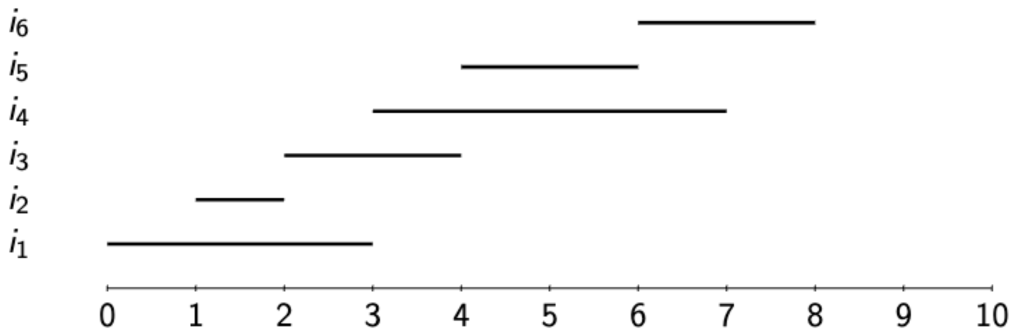
- The **optimal solution** arises from using the **ratio** of weight-length
- Run the jobs with the **largest weight-length** ratio first

Interval Scheduling

Consider a set of n intervals (s, e) where s and e are the starting and ending time respectively. We would like to choose a non-overlapping subset of those intervals such that the total number of selected intervals is maximum.

The Greedy Solution

- Choose the next compatible interval with the **soonest** completion time



- In the above example, choose i_2 first, as it has the soonest completion time, the next **non-overlapping** interval with the soonest completion is i_3, then i_5, then i_6

Fractional Knapsack

Maximize:

$$\sum_{i=1}^{n} x_i \cdot v_i$$

Subject to the condition:

$$\sum_{i=1}^{n} x_i \cdot w_i \leq C$$

Where $0 \leq x_i \leq 1$ is a fraction of item i that is used

- Sort items by their descending ratio of $\frac{value}{weight}$
- Go through this sorted list and add them to the knapsack until one of the items does not fit
- For the first item that does not fit, enter the **fraction** of that item that will fit

Huffman Coding

An introduction to Symbol Encoding

- **Symbol Encoding** - Attaching a binary value to letters in an alphabet in order to send a message
- Variable length encoding - Uses shorter codes for the most frequent symbols and longer codes for the least frequent
- However, it is still required that **each letter can be uniquely seen** - a boundary of sorts between each letter
- This is where **prefix codes** come in
  - For example $a = 0$ and $b = 01$ is not a valid encoding as a is a prefix of b and could not be deciphered

The Greedy solution to Huffman Coding

- Create a node for each symbol, with the character and frequency stored in a node
- Add each node to a **queue** Q
- While $|Q| > 1$:
  - Select the two nodes with the minimum frequency, $x < y$
  - Define a new node whos left child is x and right child is y
    - This node's frequency $z.f \leftarrow x.f + y.f$
  - Insert this new node to the Queue