Module 8 - Greedy Algorithms

# What is a Greedy Algorithm:

Greedy algorithms use a metric (for instance value/weight ratio) to make a sequence of decisions. Essentially it is an algorithm that uses a decision-proxy to find an optimal solution quickly.

# Why is it called Greedy:

The name Greedy algorithm is not accurate since it is not computationally greedy nor greedy in that it tends to over-estimate – a greedy algorithm finds an optimal solution. The name comes from the use of the word greedy to reflect a Fear-Of-Missing-Out sense of the word, where the algorithm always chooses the nicest looking immediate option rather than what may be optimal in the long term.

# Activity Selection Problem:

A group of red text

Description automatically generatedActivity Selection algorithms select a set of activities based on conflicts between the activities (usually time):

Greedy algorithms are good Activity Selection algorithms as you can make Finishing Time the decision-proxy and simply select the activity with the soonest finishing time.

This does raise a question about the effectiveness of greedy algorithms in terms of solving for the most optimal solution. A good greedy algorithm is one whose decision-proxy yields a set of potential solutions that always includes the optimal solution.

# Greedy Algorithms VS Memorisation:

Greedy Algorithms act as shortcuts to memorisation algorithms. Where a memorisation solution would form a sort of tree of all optimal solutions a greedy algorithm uses its decision-proxy to traverse a branch of that tree directly to an optimal solution.

Greedy algorithms are not always a substitute to memorisation solutions.

# Greedy VS Memorisation VS Divide-and-Conquer:

Another related algorithm solution is divide-and-conquer which similarly forms a tree of sub-solutions. Greedy solves for problems where we only ever solve a sub-problem once. Divide-and-conquer solves for problems where sub-problems are independent of one another. And memorisation solves for problems where we will need to reuse solutions to sub-problems.

# Huffman Coding: