Greedy Algorithms Overview

Greedy Algorithms are algorithms that make decisions one at a time based on what is the highest weighed / most immediately rewarding choice. This means it takes the best short term decision but lacks longer term oversight. But in some conditions this is a good decision as the knowledge of other options might be limited or the ease of taking the immediate best decision might be easy / good enough for the task at hand.

Activity Selection

This highlights the key difference on the greedy algorithm. When we have a series of activities - n and a limited time we need to select the activities that are the most optimal to complete. Now we have other methods for approaching this problem, for example dynamic programming which seeks to instead not repeat any actions and plan out each path to then find the optimal one using recursion. Or divide and conquer to break down each problem into smaller piecemeal problems that mean they are less intensive to tackle but they can be done more efficiently. While both may be good to produce globally optimised solutions (best overall) at the cost of local optimisation (taking the immediately best action). These can both be computationally and time costly whereas the greedy algorithm has significantly less time processing the next action making it computationally fast by opting for local optimisation, which might come at the cost of global optimisation but this process can be faster.

Job Scheduling

This is another problem that we can give our greedy algorithm. We have n tasks to get done and each task (i) takes (ti) hours. So we need to find the optimal substructure to get as much done as possible given our time constraints. So our course of action is the same between dynamic programming and greedy algorithms. Either way we will go for the best tasks first, whether that be the shortest or largest. But we can also add in another weight to find a more optimal path. Rather than take on just the largest or the smallest we can work out the value of the task and the time it takes by just simply adding the weight of cost over the time. This is the strength of a greedy algorithm since we can use the proxy to give it weight in a direction that is more optimal for our use case. And this process takes us O(nlog(n).

Huffman Coding

Huffman coding looks to have prefix free coding. It is a way to make more commonly used data points become cheaper so they are less taxing when being used. The example we are covering is using the alphabet in regards to English since some letters like vowels are used significantly more than consonants like ‘z’. We can think of this construction method by using a tree. Then we can create a measured value to each leaf in the tree. For example the probability x the depth within our tree. This gives us a weight we will want to minimise since we want to quickly grab the most common elements within our tree. We build this tree by

taking our lowest values and combining two with a super node then adding the nest layer with the next lowest node / nodes. This then greedily builds our tree.

Prim’s Algorithm - Minimum Spanning Trees

We want to create a way to span a graph using the minimum expenditure and we want to have no cycles in how we traverse that graph. This algorithm also starts from an arbitrary node within our graph. This method also helps us to select non repeating sub-problems as we can avoid going back to the same nodes and looking for the shortest edge. We then looked at two implementations of Prim's algorithm, a slower alternative that looks at each of the individual options available at each step. This adds a lot of time complexity to the algorithm, its overall complexity is O(nm). The faster alternative instead of ‘scanning’ it adds each potential node and takes from the shortest potential next node. That way it saves from searching and makes the O(m log n)

Kruskal's Algorithms

This algorithm rather than looking at the perspective from each node it looks at each connection. Then it adds those connections to its path. When it does this though it needs to make sure it isn't creating a loop or cycle. But it goes through each connection between each node rather than each node itself.