Greedy Algorithms

Process Scheduling

You have a computer and n processes with processing times $t_1, ..., t_n$. You have to pick the order in which to run the processes. Let p_i denote the i^{th} process you run. Then, the completion time C_i for process p_i is defined as $C_{p_i} = \sum_{j=1}^{i} t_{p_j}$, i.e., the sum of times for all the processes up till this one ends. You have to minimize the average completion time, i.e. $\sum_{i=1}^{n} C_{p_i}$.

Greedy solution

This problem has a well known greedy solution, known as the Shortest Processing Time First (SPTF) rule. We perform the processes in order of lowest processing time. Let us prove this.

Assume $t_{p_i} \geq t_{p_j}$ and $i < j$. If we interchange p_i and p_j, then the completion time of everything from process i to j reduces by $t_{p_i} - t_{p_j}$, which is non-negative, and the completion time for everything process j onwards remains unchanged. Thus, the interchanged order of processes is less than or equal to the original.

In this manner, we can sort the process times by performing two-swaps one by one, and we will only decrease our average completion time. We can do this in $O(n \log n)$ time.

Online version

This problem has the same solution when processes can be added dynamically. If a process with a lower processing time than the remaining processing time of the current one is added, we switch to that one and complete it first. The proof is similar.

Event Overlap problem

You have n events on your calendar, defined as intervals with a start time s_i and a finish time f_i. The events might overlap, and you want to attend all the events, so you are going to create k clones of yourself to achieve this. You want to minimize the number of clones you need, k. A clone can attend a certain non-overlapping subset of events.

Greedy solution

We sort the intervals by start time. We start with 0 clones and dynamically assign each interval to one of these clones. We maintain the finish time of the last interval assigned to each of these clones.

As we iterate through the sorted list, for each interval, if it starts after the last finished event for one of the clones, we can assign this interval to the clone without an overlap. So we assign this event to it and update its finish time. If it is not compatible with any of the clones, we create a new clone and assign this event to it.

Correctness

Let us consider the event that corresponds to adding the m^{th} clone. Suppose it happens when considering interval (s_i, f_i). This means that $m - 1$ previously considered intervals overlap with this. But, since they all start before s_i (since we sorted by start time), that means that at time s_i, there are at least m concurrent intervals. This means that the optimal solution uses $\geq m$ clones.

Implementation

Here is an $O(n \log n)$ implementation. We maintain a min-heap of finish times of each clone's last interval. When adding an interval (s_i, f_i), if the minimum finish time on the heap is $\geq s_i$, all of the clones are incompatible, and so we create a new clone by adding f_i to the heap. If the minimum finish time on the heap $< s_i$, we can add this interval to it, and so we pop from the heap and add f_i.

Note This problem can in fact be generalized to decomposing any partial ordering into chains by Dilworth's theorem.

Fractional Make-Change

Let us consider the make-change problem from last recitation, with a few modifications. Instead of coins, we have m kinds of metals. Given a value N, we want to make change for N cents. Metal type i has value S_i per kilogram, and we have n_i kilograms of it. As these metals are in molten form, and we have an infinite precision scale, we can give out non-integral weights also.

This means that if we choose to use k_i kilograms of type i, where $0 \leq k_i \leq n_i$, the value given out will be $S_i k_i$. The objective now is to minimize the total weight of the metals used $\sum k_i$

Greedy solution

Our greedy intuition for the original make change problem now works. We take the most valuable metal, and try to fulfill as much of the remaining value with it as possible.

In other words, we sort the metals in decreasing order of value per kilogram and set remaining value $r = N$. As we iterate through the sorted list from $i = 1$ to m, if $S_i n_i < r$, we set $r = r - S_i n_i$ and add n_i kg of metal i to our set. Otherwise, if $S_i n_i \geq r$, we add $\frac{r}{S_i}$ kg of metal i to our collection, and set $r = r - S_i \frac{r}{S_i} = 0$. We break at this point, as our requirement is fulfilled.

Correctness

The proof follows by cut-and-paste. Let's say we have w kg unused metal i and we are using w kg of metal j in our optimal solution, such that $S_i > S_j$. Then, we could replace the w of metal j with $w\frac{S_j}{S_i}$ more of metal i. As $S_i > S_j$, $w\frac{S_j}{S_i} < w$, so we have that much of metal i by hypothesis. Further, our total weight strictly decreases. This contradicts the assumption that our solution was optimal.

We conclude that we will always exhaust the more valuable metal before using a less valuable one, so our greedy algorithm is correct.