Greedy Algorithms

A greedy algorithm is an algorithm that makes the choice that looks the best at a given moment. Ie, it makes a locally optimal choice, in the hope that it will lead to a globally optimal solution. However, it is worth noting that Greedy Algorithms do not always yield the most optimal solution. Despite that, the greedy method is quite powerful and many algorithms are based on it. Minimal Spanning Tree Algorithms, Djikstra's Algorithm, etc are some examples of Greedy Algorithms.

Activity Selection Problem

Description of the Problem

Here, we try to schedule a set of several competing activities, that require exclusive use of a common resource, with a goal of selecting a maximum number of mutually exclusive activities.

For instance, imagine you are in charge of scheduling a conference room.

You are presented with $S = \{a_1, a_2 ... a_n\}$ a set of n proposed activities, which require the exclusive use of the conference room. Each activity has a start time s_i and end time f_i where $0 \leq s_i < f_i < \infty$.

If selected, the activity will take place during the time interval $[s_i, f_i)$. We say a_i and a_j are **compatible activities** if their respective intervals are not overlapping.

In the activity selection problem, the goal is to select a subset of S such that all the elements of the subset are mutually compatible, and the the corresponding subset is the largest such subset.

For simplicity, we may assume that the provided set of activities S are sorted monotonically by their finish times. That is, $f_i \leq f_j$ for all $i < j$

Constructing a Solution

From intuition, if we choose the activity that'll leave the resource available for most other activities as possible. Of all the activities we select, one of them has to finish first, so we choose one that will finish the earliest. Since the activities are sorted monotonically by their finish times, the first entry in our activities list will be a_1.

Once we've made that choice, we just look for activities that start the earliest after the first entry in our activities list ends.

We now choose a set $S_k = \{a_i \in S | s_i \geq f_k\}$. This gives us the set of activities that are starting after a_k finishes. If by our greedy choice, we choose a_1, the only problem we have to solve is S_1.

Consider a subproblem S_k, with the first element in this set being a_m, ie a_m has the earliest finish time in S_k. We can claim, a_m is included in the maximum-size subset of mutually compatible activities of S_k. The proof of this claim is non-trivial and is left as an exercise for the reader because I'm too lazy to type it out just read CLRS if you want.

This intuition suggests a recursive approach to solving the problem. This has a top-down approach, where we make a choice, and then solve a sub-problem, rather than a bottom-up approach, like a Dynamic Programming problem may have.

Pseudocode

We define a procedure, `RECURSIVE-ACTIVITY-SELECTOR` that takes the start times and the finish times of the activities, and and returns the maximal set of mutually compatible activities in S_k.

The procedure assumes that the input n activities are all ordered by their finishing time f_i. If it is not, you will need to first sort it using merge sort, which will take $O(n \log n)$ time.

In order to start, we begin with the fictitious activity a_0 which ends at $f_0 = 0$, so the subproblem S_0 is the entire set of activities S provided.

The initial call, that will solve the entire problem is `RECURSIVE-ACTIVITY-SELECTOR(s,f,0,n)`.

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
m = k + 1
while m <= n and s[m] <= f[k]
    m = m + 1 \\ with this we're finding the first activity
if m < n:
    return {a_m} + RECURSIVE-ACTIVITY-SELECTOR(s, f,m,n)
else
    return {}
```

Lines 1, 2, and 3 are looking for the first activity that is finishing in the set. This activity, a_m is selected, and added to the list. This is added to the set of activities, and then the recursive algorithm is again called for the elements that will start after the selected a_m ends.

Time Complexity

Assuming all the activities are sorted by finish times, the running time of the `RECURSIVE-ACTIVITY-SELECTOR` is $O(n)$, where n is the number of activities. This is because, no while loop will go through the whole list, unless it terminates. And each while loop starts after the previous one ends.