20191003

More Greed

Greedy Algorithm for Activity Selection

Suppose we have a set of activities to choose from, but some of them overlap. We want to choose the largest possible set of non-overlapping activities. We will say that two activities x and y overlap if there is any time t such that $t \in [start\_time(x) \ldots finish\_time(x)]$ and $t \in [start\_time(y) \ldots finish\_time(y)]$. Note that using this definition of overlap means that two activities overlap even if one of them ends at the exact moment the other begins. We could define overlap so that this situation (one activity starting exactly when another ends) doesn't count as an overlap. It really wouldn't change anything important in our analysis. As an exercise you might want to work out exactly what the (trivial) change to the algorithm would be. In class I set up a scenario involving an imaginary Film Festival with movies starting and ending at odd times. I'll spare you the details here and just refer to generic "activities".

A greedy algorithm always starts by sorting the objects – but it can be a challenge to find the right sorting criterion. In class we experimented with different criteria for sorting the activities:

1. Sort the activities by start time. Then repeatedly choose the next activity if it doesn't overlap with the ones already chosen.

This algorithm fails on this example

Task	1	2	3
Start Time	8:00	8:01	8:05
Finish Time	9:00	8:02	8:06

2. Sort the activities by length, shortest first. Then repeatedly choose the next activity if it doesn't overlap with the ones already chosen.

This algorithm fails on this example

Task	1	2	3
Start Time	8:58	8:55	9:01
Finish Time	9:02	8:59	9:05

3. Sort the activities by length, longest first.   Then repeatedly choose the next activity if it doesn't overlap with the ones already chosen.   (I mentioned this idea very briefly in class – it seems silly – and it is.)

This algorithm fails on this example

Task	1	2	3
Start Time	8:00	8:01	8:05
Finish Time	9:00	8:02	8:06

None of these work ... but undaunted by our repeated failures, we finally hit upon another alternative:

4. Sort the activities by **finish time**, earliest first. Then repeatedly choose the next activity if it doesn't overlap with the ones already chosen.

In pseudo-code:

```
Greedy Activity Selection:
    Let {$x_1, x_2, ...x_n$} be the set of activities after sorting by finish time.
    Select $x_1$
    f_time = finish_time($x_1$)
    for i = 2 to n:
        if start_time($x_i$) > f_time:
            select $x_i$
            f_time = finish_time($x_i$)
```

Proof of correctness:

We will use proof by induction. We apply induction to n, the number of activities.

Base case: Let n = 0. Clearly the algorithm will choose no activities, which is the optimal solution.

Inductive Hypothesis: Assume the algorithm always finds the optimal solution when there are $\leq$ n activities, where n is some integer $\geq$0.

Let the number of activities be n+1. Apply the algorithm (sort by finish time, and select as above). Let the Greedy Algorithm solution be A = {$a_1, a_2, \ldots a_t$}. Note that a_1 must have the earliest of all finish times, due to the sort.

Stage 1: We need to show that there is an optimal solution containing a_1. Let O be any optimal solution, with its elements sorted by finish time. Let O = {$o_1, o_2, \ldots o_s$} We know that start_time(o_2) > finish_time(o_1) $\geq$ finish_time(a_1) because a_1 has the earliest finish time of all the activities. Thus O* = {$a_1, o_2, \ldots o_s$} is a feasible solution, and |O*| = |O|, so O* is an optimal solution that contains a_1.

Stage 2 and Stage 3: Now we need to show that A is optimal. By the inductive hypothesis {$a_2, a_3, \ldots a_t$} is an optimal (ie largest) solution to the reduced problem of choosing activities that start later than finish_time(a_1) because there must be $\leq$n such candidate activities.

There's an important thing to notice here. We *must* include the constraint that after choosing

a_i we are only interested in activities that start after finish_time(a_i). If we don't do that, the algorithm would continue by simply choosing the activity with the next finish_time, and then the next one after that ... in fact it would choose all the activities, regardless of any overlaps.

Thus while it is correct to say that a Greedy Algorithm never looks back at its previous choices and changes them, here we are seeing an example of a Greedy Algorithm that needs a little bit of information about the most recent decision (specifically, the finish time of the most recently selected activity). It is a common feature of Greedy Algorithms that the subproblem we reduce to is dependent on the most recent selection.

By enforcing this "don't overlap with the most recent selection" we guarantee that the algorithm's selected activities will form a feasible solution. This is really the proof of Stage 3, where we show that the algorithm's first choice can be combined with the rest of the algorithm's choices, but it seemed reasonable to put it here.

To continue with the proof of optimality ...

Let O* be the optimal solution we constructed above, so O* = $\{a_1, o_2, \ldots o_s\}$.

Observe that $\{o_2, o_3, \ldots o_s\}$ is also a solution to the problem of choosing activities that start later than finish_time(a_1).

Because $\{a_2, a_3, \ldots a_t\}$ is an **optimal** solution to this reduced problem, we know

$|\{a_2, a_3, \ldots a_t\}| \geq |\{o_2, o_3, \ldots o_s\}|$

Thus $|A| \geq$ |O*|

Since O* is an optimal solution, |A| > |O*| is not possible, so $|A|$ = |O*|. Thus A is an optimal solution, and the Greedy Algorithm finds the optimal solution whenever there are n+1 activities.

Therefore the Greedy Algorithm finds the optimal solution to the Activity Selection Problem for all sets of activities.

In class I stressed the point that even though in our proof of correctness for Greedy Algorithms we say "Let O be an optimal solution", our proof does not need to actually construct O – we just make use of properties that all optimal solutions must have in common.

The algorithm simply constructs the set A – it never constructs some other solution O and swaps elements of A and O – that is just part of the proof that A is an optimal solution.

Here's another idea for sorting the activities: sort the activities in order of the number of "overlaps" (so an activity that overlaps with two others would be sorted in front of one that overlaps with three others). Now apply the Greedy algorithm exactly as described above (really, all Greedy algorithms look alike): work through the sorted list and add each activity to the chosen set iff it has no overlap with the ones already chosen.

Question 1: Can you prove that this sorted order always leads to an optimal solution, or can you find a set of activities where this approach fails to find an optimal solution?

Question 2: Regardless of whether or not this works, what is the computational complexity of this version of the algorithm?