CSE 202 NOTES FOR NOVEMBER, 2002

Greedy Algorithms

Greedy algorithms are basically backtracking algorithms with a "greedy rule". This greedy rule governs the decisions that you make at each backtracking point, and limits the search space by a huge factor. It is not unusual for a greedy algorithm to take an exponential backtracking algorithm and turn it into an efficient linear time search.

Greedy algorithms are generally very fast. And they are generally very wrong. There are two important points regarding the design of greedy algorithms:

- You must carefully prove that your greedy algorithm actually finds the optimal solution. There are several proof templates that we will see later; generally you should use one of these templates tailored to your particular problem.
- Your algorithm will change a lot between your initial idea and your final solution. The basic greedy algorithm is just a starting point—you will probably need to restructure it and start using a good data structure at some point.

The following steps illustrate how to design a greedy algorithm.

Find a generic backtracking algorithm: This may involve a more general problem. Generally the backtracking algorithm will have the following shape:

```
pick *any* decision point
Options ← $\{\mathrm{op}_1, \mathrm{op}_2, \ldots, \mathrm{op}_k\}$
Best-So-Far ← BT(subproblem, $\delta \leftarrow \mathrm{op}_1$)
for $i = 2$ upto $k$ do
   Next-Case ← BT(subproblem, $\delta \leftarrow \mathrm{op}_k$)
   if Value(Next-Case) > Value(Best-So-Far) then
      Best-So-Far ← Next-Case
   end if
end for
return Best-So-Far
```

As in backtracking, starting with a simple algorithm usually helps in finding a solution.

Find a property E that makes a decision easy: This often involves imposing order on the decision points. The basic idea is that by applying property E you no longer have to consider the other decision options at that decision point.

The greedy rule generally has the form "If a decision point has property E, only consider the greedy option; don't consider anything else."

Show that E is sound: Sound means that if a decision point d has E then some optimal solution picks the greedy option for d. There is another notion, which we might call Completeness: if every decision point has an option with property E, then E is considered complete. One can consider a truth-table for soundness and completeness—if a rule is not complete and not sound, it is useless; if a rule is complete and not sound, then it may be helpful for greedy heuristics in an approximate solution to the problem; if it is not a complete property but it is sound, then it may help a backtracking solution through pruning; if it is complete and sound, then it can be used to create a greedy algorithm. At this point you should have some idea of why these algorithms turn out to be wrong more often than they turn out to be right—there are a lot of restrictions on when a greedy algorithm can be used, and you're lucky if you find an algorithm that can be converted to greedy.

Conference Room scheduling (or fixed-boundary job scheduling)

Consider a new problem. You can think of this as picking the largest set of events that can be scheduled in a single conference room (this is is quite different than maximizing the amount of time the conference room is in use). More formally,

Instance: Set E of events $e = (s, f)$, where s is a start time and f is a finish time ($f \geq s$).

Solution Format: $S \subset E$

Constraints: No two events overlap (but they can be adjacent) in time. That is, no two events e_i and e_j exist such that $e_i.s \leq e_j.s$ and $e_i.f > e_j.s$.

Objective: Maximize $|S|$.

For example, let $E = \{(\mathrm{M}, \mathrm{W}), (\mathrm{Tu}, \mathrm{Tu}), (\mathrm{W}, \mathrm{F}), (\mathrm{F}, \mathrm{Su}), (\mathrm{Su}, \mathrm{Su})\}$.

Yes, I am notationally stupid because I use E to denote the set of events here, not the greedy property. Don't get confused. I'd feel bad.

As a first candidate rule, consider picking the event with the lowest initial time, removing all conflicting events, and then repeating. This doesn't work because the earliest-starting event could go on for a long time blocking a bunch of other events, so this isn't a sound rule (consider $E_1 = \{(\mathrm{M}, \mathrm{Su}), (\mathrm{Tu}, \mathrm{Tu}), (\mathrm{W}, \mathrm{W})\}$—clearly the last two events would be a better choice than the first). Since looking at the end time didn't help, let's consider instead the same rule, but using the smallest range as the decision metric. This doesn't help much either because one short event could be very inconveniently scheduled and block other schedules. This rule is not sound either.

What about picking the event that has the first *ending* time (and removing all conflicting events from the set)? This seems very similar to the first rule, but it's hard to think of a schedule that will break it.

What are the decision points?

Events.

What is property E?

Consider event e if $e_{\text{end}} \leq e'_{\text{end}} \forall e' \in E$.

What is the greedy rule?

Put e into S.

Is this sound? Let e be a decision point with $e_{\text{end}} \leq e'_{\text{end}}, \forall e' \in E$. Let S' be an optimal solution of non-conflicting events such that $e \notin S'$. The goal will be to construct an S such that $|S| \geq |S'|$ and $e \in S$. The key insight here is that e must finish before the first event e'_0 in S' finishes, so just do $S = S' - \{e'_0\} \cup \{e\}$, and clearly $|S| \geq |S'|$ and $e \in S$. In other words, let $e' = (s', f')$ be the first event in S'

(events are ordered unambiguously in time because they cannot overlap). Since no event in S' overlaps with e' we can switch e' and e to create the new S.

So why is this new rule sound whereas the first rule we considered was not sound? After all, isn't the problem symmetric with respect to start time and end time? It is, but the *entire problem* is symmetric: you choose the first end time, so you choose the last start time. Is this rule sound? Let e be a decision point with $e_{\text{start}} \geq e'_{\text{start}}, \forall e' \in E$. Let S' be an optimal solution of non-conflicting events such that $e \notin S'$. If we replace the *last* event in S' with e, you will see that this all works out the same way. The key is that you work backwards using either the start time or the end time.

Notice also that your first two guesses would probably have been wrong, but might have seemed speciously right. Greedy algorithms are dangerous, which is why you have to be diligent in proving them.

So far I haven't restated the algorithm formally. Basically you can do it in a stupid way: find the earliest-start-time-element in E and put it into S and delete conflicting elements, and repeat until E is empty. This takes time $n + n - 1 + n - 2 + \cdots 1$, or $T(n) \in O(n^2)$. Alternatively you can sort E by the ending time and make a single pass through. This is $T(n) \in O(n \log n + n) \in O(n \log n)$.