Greedy Algorithms

Always go for the best locally. In a sense these algorithms harps on the idea that locally optimal choice leads to global optimality.

A trivial example. Suppose we have a limited capacity W, and a set S of objects A_i to choose from where A_i has weight $w_i > 0$.

Problem. Choose a subset $T \subseteq S$ so that

$$\sum_{A_i \in T} w_i \leq W \text{ and}$$

$|T|$ is maximized.

The greedy strategy is to choose the objects with the smallest weight.

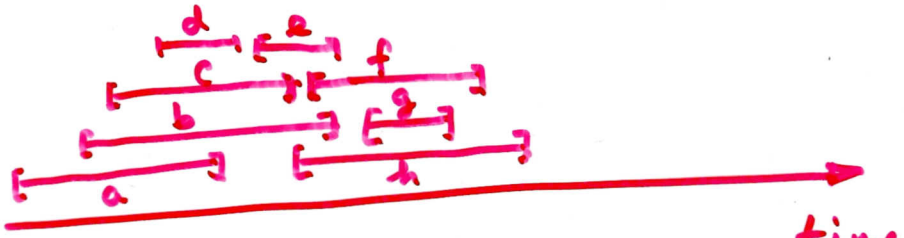
1. Sort so that $w_i \leq w_j$ if $i < j$
2. $T := \emptyset$; $w(T) := 0$; $i := 1$;
   while $w(T) + w_i \leq W$ do
   $T := T \cup \{A_i\}$; $w(T) := w(T) + w_i$;
   $i := i + 1$
   endwhile

$O(?)$

2. A scheduling problem.

Let $S = \{1, 2, \ldots, n\}$ be a set of activities. Activity i has start time s_i and finish time f_i. If activity i is scheduled, it occupies the resource in the time interval $[s_i, f_i)$, $s_i < f_i$. Problem is to maximize the # of activities scheduled.

Ex.



Best schedule is a, e, g

Strategy: Always choose the one that ends earliest.

1. Sort so that $f_i \leq f_j$ if $i < j$
2. T := {1}; last := 1;
   for i := 2 to n do
   if $f_{last} \leq s_i$ then
   T := T ∪ {i}; last := i
   endif
   endfor

The difficult part of the greedy algorithms is usually to prove that it works. Or, if it does not produce the optimal result, how well does it approximate the optimal.

Claim. The presented greedy algorithm schedules the largest number of activities.

proof. First observe that activity 1 can always be chosen. This is because the first activity of any schedule (also any optimal schedule) can be replaced by 1 giving another schedule with same number of activities.

After deleting all activities i with $[s_1, f_1] \cap [s_i, f_i] \neq \emptyset$ (which is equivalent to $s_i < f_1$) the problem can be solved recursively, which is what the above algorithm does.