CIS 575. Introduction to Algorithm Analysis
Material for April 8, 2024

Greedy Algorithms: What and When

The design principle covered in this set of notes is the topic of *Cormen*'s Chapter 15.

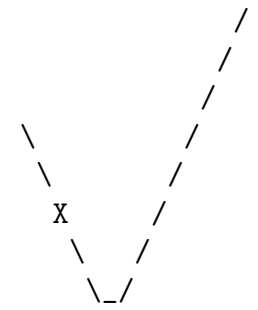
1 The Greedy Approach

Numerous problems in computer science involve that we explore various paths in a search space, so as to find a path that eventually produces a reward that is at least as good as what any other path can produce. We have already seen several examples, such as how to decide which items to pick when packing a knapsack. We have also seen that *Dynamic Programming* often enables us to substantially reduce the cost of computation, in that each subproblem needs to be solved only once even though it appears along multiple paths.

Still, an algorithm based on Dynamic Programming will in many situations need to explore (at least) two options, and only at the end know which is the best. For example, if a given item fits in the knapsack, we may pick it, or omit it, without knowing in advance which choice enables us to pack the highest value.

It would be very convenient if at each point we could know "where to set our foot next". This is the essence of a **greedy algorithm**; it attempts to build a solution that is *globally optimal* by repeatedly extending it based on *local* information, and *never backtracks*.

In subsequent notes, we shall investigate situations where greedy strategies are guaranteed to be successful. You may recall we have already seen one such situation; there is a greedy algorithm for giving optimal exact change for the US coin set, though that algorithm will not give an optimal solution for a coin set without nickels. In general, it is often the case that the globally optimal solution can only be reached through steps that from a local perspective may appear disadvantageous, as illustrated by the simple figure



where from X, assuming the goal is to get as high as possible, one must move *right* (though that appears to hinder our goal) rather than *left* (though that appears to advance our goal).

For a specific example of a solution that is *locally* optimal but *not globally* optimal, consider the following *scheduling problem*. The goal is to line up a number of customers, where each customer j has been given a number $P(j)$, in a way that aims for neighbors to have similar P-numbers. More precisely, for each neighbor pair (i, j) we will assign a penalty of $\min(|P(i) - P(j)|, 5)$, and want to minimize the total penalty. If we are given 10 customers, with their P-numbers assuming all values from 1 to 10, it is clearly optimal to line them up in increasing (or decreasing) order in which case the total penalty is **9**. But consider the schedule

$$6, 7, 8, 9, 10, 1, 2, 3, 4, 5$$

which is far from optimal in that the total penalty is $4+\min(9, 5)+4 = 13$. Still, that schedule is *locally* optimal in that if we swap two customers then the penalty will be *increased*.