Greedy Search

The primary focus of discussion this week and next week is optimization. Greedy search is a common approach to optimizing some function and provides a nice "introduction" to the more complex concepts we will need for dynamic programming. A greedy algorithm is one that always makes a choice that "looks best" at a given point in the search process. Therefore, the approach involves making "locally optimal" choices in the hope that the collection of choices will yield a globally optimal result in the end. Frequently, greedy search does indeed find a globally optimal solution, but such guarantees only come with certain types of problems. One example problem we have examined that made use of such greedy choices was finding the set of shortest paths from a single source. Both Dijkstra's algorithm and the Bellman-Ford algorithm apply greedy search.

Definition: The greedy choice property is the property of a problem whereby a globally optimal solution can be found by making locally optimal choices. Such choices do not depend on solutions to subproblems.

Definition: A problem exhibits optimal substructure if an optimal solution to the problem contains optimal solutions to the subproblems within it. This is sometimes referred to as the principle of optimality.

Let's consider two variations to a seemingly simple problem where we will able a greedy strategy. In one case, we will find that the greedy strategy succeeds in finding the optimal solution but the greedy strategy fails in the other.

Knapsack Problems

Suppose a thief is robbing a store and finds n items of interest in the store. Assume the ith element is worth v_i dollars and weights w_i pounds. Assume also that the thief wants to steal the items with maximal value, but the thief's knapsack only holds W pound total. The knapsack problem involves determining which items the thief should steal to maximize payoff.

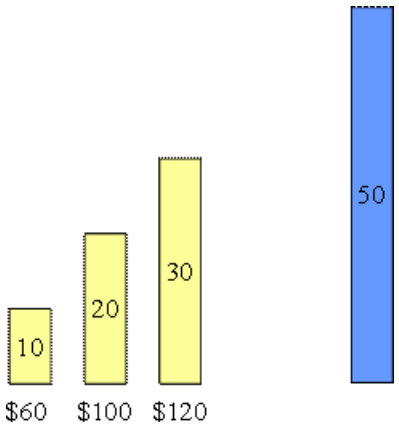
There are two versions of the knapsack problem of interest to us. First, the 0/1 knapsack problem assumes that the thief must steal whole items. The fractional knapsack problem, on the other hand, assumes the thief can steal partial items, and the value of a partial item is proportional to the amount of the item that is stolen. As it turns out, a simple greedy strategy works when solving the fractional knapsack problem, but it fails when solving the 0/1 knapsack problem. In fact, the 0/1 knapsack problem is "NP-complete."

0/1 Knapsack Problem

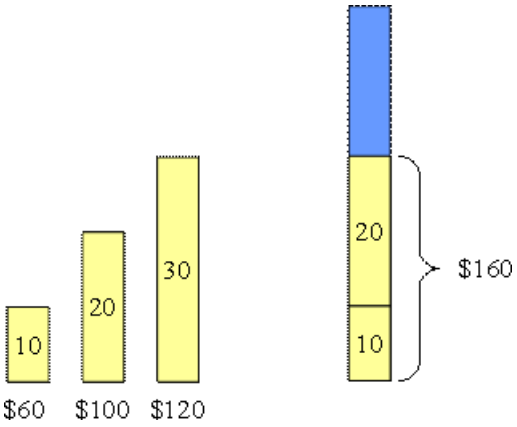
Let's consider a greedy approach to solving the 0/1 knapsack problem to demonstrate the failure of the greedy strategy. To solve this problem with greedy search, we will do the following:

1. Compute the value per pound of each item in the store (i.e., v_i/w_i).
2. Follow a greedy strategy by taking as many of an item with maximum value per pound that fits in the knapsack.
3. Upon exhausting a particular item, move on to the next highest value per pound.
4. Continue doing this until the knapsack is full.

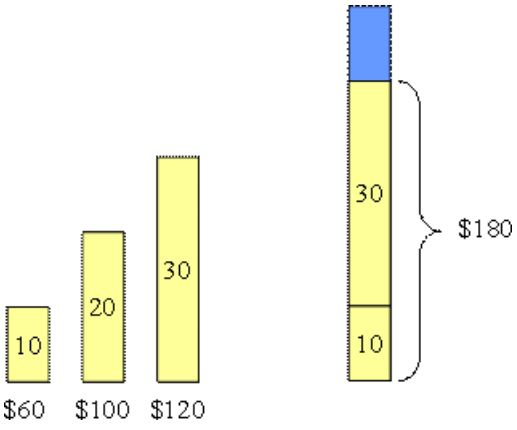
On the surface, this appears to be a perfectly reasonable approach. However, to illustrate the greedy strategy's failure on this problem, suppose the thief has a knapsack that will hold 50 pounds and is presented with three objects to steal. These objects weigh 10, 20, and 30 pounds respectively, and they are worth \$60, \$100, and \$120 respectively. Thus the values per pound of the three objects are 6, 5, and 4 respective. We can depict this problem graphically as follows where the yellow rectangles are the objects to steal and the blue rectangle is the knapsack:



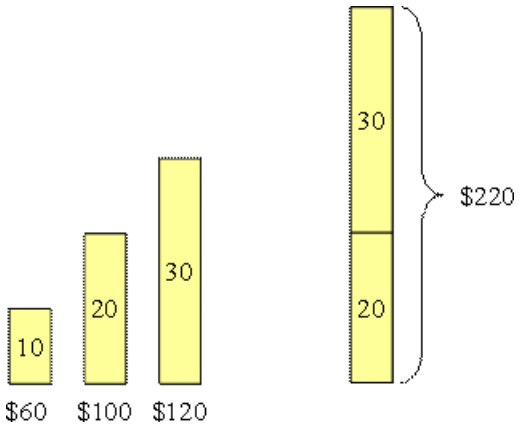
If we apply the greed strategy above, we will first pick the \$60 item and then the \$100 item. We will not have room for the \$120 item, so search will terminate:



Notice, however, that if we were to choose the \$120 item instead of the \$100 item, we would yield a better result for the thief.



But even this is not the best we can do. If we had considered the objects in reverse order (which would have selected based on value alone), we would yield the optimal result.



This would seem to suggest that we should optimize based on value alone and not consider the weight, but we can easily construct another counter-example leading to failure of greedy search based on value alone.