I. Core / D. Dynamic programming and greedy algorithms

- Dynamic programming review and overview (last week Wednesday)
- Dynamic programming practice (last week Friday, this past Monday, and **today**)
- Greedy algorithms (**today**, Friday, and next week Monday)

Today:

- Work through company party problem
- Begin Greedy algorithms

15-6 Professor Stewart is consulting for the president of a corporation that is planning a company party. The company has a hierarchical structure; that is, the supervisor relation forms a tree rooted at the president. The personnel office has ranked each employee with a conviviality rating, which is a real number. In order to make the party fun for all attendees, the president does not want both an employee and his or her immediate supervisor to attend.

Professor Stewart is given the tree that describes the structure of the corporation *using the left-child right-sibling representation described in Section 10.4*. Each node of the tree holds, in addition to the pointers, the name of an employee and that employee's conviviality ranking. Describe an algorithm to make up a guest list that maximizes the sum of the conviviality ratings of the guests. Analyze the running time of your algorithm.

Dynamic programming vs greedy algorithms

Both are for optimization problems that have *optimal substructure*.

How are they different:

- Greedy algorithms make decisions that are locally optimal
- Greedy algorithms tend to be simpler, more straightforward to write
- The hard part of greedy algorithms is *determining whether an optimal greedy solution exists*

The activity selection problem (§16.1)

- Problem: S, the given, complete set of activities, $\{a_1, a_2, \ldots\}$.
- Subproblem: S_{ij}, the set of activities that fall between a_i and a_j.
- Solution to subproblem: A_{ij}, a "maximal" (in terms of cardinality) subset of S_{ij}

Claim (Theorem 16.1 in the book):

> *Let a_m be an activity with earliest finish time in S_k. There exists a maximal solution to S_k that includes a_m.*

Notation switch in the book: $S_k = S_{kn}$, $A_k = A_{kn}$.

Theorem 16.1. Let a_m be an activity with earliest finish time in S_k. There exists a maximal solution to S_k that includes a_m.

Let A_k be a maximal solution to subproblem S_k.

Suppose $a_m \notin A_k$

Let a_j be the element in A_k with earliest finish time.

Consider the set $(A_k - \{a_j\}) \cup \{a_m\}$.

Since

$$\begin{aligned} f_m &\leq f_j \\ &\leq s_x \end{aligned}$$

... for all $a_x \in A_k$, a_m does not conflict with anything in $(A_k - \{a_j\}) \cup \{a_m\}$.

$$\begin{aligned} |(A_k - \{a_j\}| &= |A_k| - 1 + 1 \\ &= |A_k| \end{aligned}$$

So $(A_k - \{a_j\}) \cup \{a_m\}$ is also. maximal. $\square$

Elements of the greedy strategy

find	1.	The optimal substructure	
develop	2.	A recursive solution	
prove	3/4.	The greedy choice	a. One subproblem remains b. It's safe to pick a local optimum
develop	5.	A recursive algorithm	
convert to	6.	An iterative algorithm	

Ex 16.2-1. Suppose we have items 1 through n, with v_i being the value of the whole thing and w_i being its weight. W is the capacity of the knapsack. Until the knapsack is full, (a) choose the item with highest value density ($\frac{v_i}{w_i}$) and take as much as will fit; (b) repeat with subknapsack $W - w_i$, assuming $w_i < W$.
Claim: For a given instance of the problem, there is a solution using this greedy approach.
Demonstration. Suppose $A = (a_1, a_2, \ldots a_n)$ is an optimal solution, indicated by the weight taken from each item. It must be that $a_1 + a_2 + \ldots a_n \leq W$, but assume that the total weight is in fact equal to W, since you can always increase the knapsack's value by adding something more. The value of the solution is

$$\sum_{k=1}^{n} a_i \frac{v_i}{w_i}$$

Suppose further that item m has the highest value density and that $a_m < w_m$. (We're also assuming $w_m < W$; the argument would be basically the same otherwise, just a little more complicated.)
Start removing items from the solution until you've removed $w_m - a_m$ weight, and then add the rest of item m. Since item m has the highest density, you now have a more valuable knapsack.

Ex 16.2-3. Always pick the smallest, most valuable one. This works because any solution that did not have the smallest, most valuable one can be made more valuable (without increaseing weight) by replacing one of the others with the smallest, most valuable one.

For next time:

Do Ex 16.2-4.

Read Sec 16.3. Note that the Huffman encoding was covered in CSCI 243 (unless you took it Spring '23, when we skipped it). I've also written a section about it for Algorithmic Commonplaces, though we don't cover it in CSCI 345. You can refer to either of my textbooks as an additional source to help understand what's happening in CLRS.

Our focus will not be on the premise (which I hope you remember...) but rather on the greedy choice property and other aspects of correctness and efficiency.

(Exercises 16.3-(2,4) will be assigned next time)