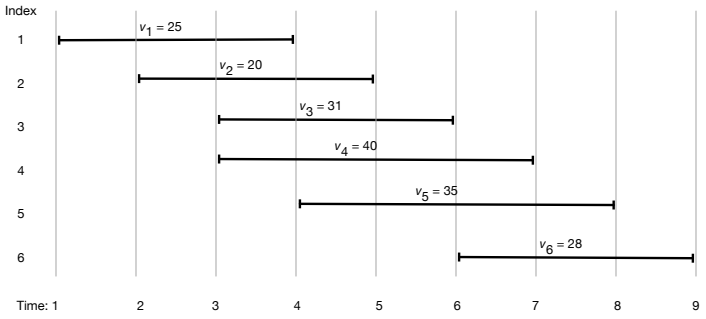
CSE 417
Algorithms & Computational Complexity
Assignment #7 (rev. a)
Due: Friday, 2/25/22

1. The solution to the Weighted Interval Scheduling Problem (KT 6.1, p 252) uses the auxiliary function $p(j)$, giving the latest-ending potential "predecessor" of job j. Computing $p(j)$ for all j and saving their values in a table is a reasonable step to take. The "obvious" way to do this takes $\Theta(n^2)$ time—an outer loop for j from 1 to n, and an inner loop for i from $j-1$ down to 1, testing compatibility of job j with i in each case. Give a faster algorithm for this problem. As usual, include a sketch of correctness and an analysis of runtime.

2. Consider the instance of the Weighted Interval Scheduling Problem (KT 6.1, p 252) given in the following table and sketched in the figure.

Index j	Start s_j	Finish f_j	Value v_j
1	1	4	25
2	2	5	20
3	3	6	31
4	3	7	40
5	4	8	35
6	6	9	28



Build a table like that shown on slide 19 of my "DP-Scheduling" slides to illustrate the execution of the dynamic programming algorithm for solving this problem ("Iterative-Compute-Opt", KT 6.2, p259 or my slide 17; *not* the "recursive" or "memoized" versions given earlier in the book/slides). Your table should should repeat j, s_j, f_j, and v_j from above and show, for each $1 \le j \le 6$

(a) The value of $p(j)$, (cf. p 253, slide 10) for this problem instance, as well as

(b) The "max" expression and resulting $Opt[j]$ values (slide 19).

In addition:

(c) *Find* the optimal solution (not just its value) by running the "traceback" algorithm (p 257–8 or slide 20)

(d) Decorate your table with arrows to illustrate the traceback as in my slides 21–2.

(e) Assume we change the problem so that $v_6 = 30$. Explain how, if at all, the table would change, how the Opt value would change, and how the solution and traceback would change.

3. Show the OPT table (Figures 6.11, 6.12) that would be produced by the Subset Sum algorithm (KT, Section 6.4, pp 266-271) when run on the sequence of weights $w_1 = 5$, $w_2 = 2$, $w_3 = 4$, $w_4 = 3$, and $w_5 = 6$, with bound $W = 16$. What is the optimum solution found (both its total value and the selected weights)? Summarize the traceback (e.g., in part by overlaying a few arrows on the table) that establishes this optimum solution.

4. Here's a variant of the knapsack problem: You are given a knapsack of capacity W, and an *unlimited* supply of each of n kinds of item, where the i-th kind of item has integer weight $w_i > 0$ and value $v_i > 0$, $1 \le i \le n$. Give an $O(nW)$ time algorithm to find how many of each item to carry so as to maximize value without exceeding capacity. I.e., find non-negative integers $m_i, 1 \le i \le n$, maximizing $\sum_{1 \le i \le n} m_i v_i$ subject to $\sum_{1 \le i \le n} m_i w_i \le W$. (Note that the best solution might not completely fill the knapsack.) Include timing and correctness, as usual.

5. Define a *socially distant* subset of the integers $S = \{1, \ldots, n\}$ to be a set $D \subseteq S$ such that for all $i, j \in D$, if $i < j$ then $i < j - 1$ (i.e., no two elements of D are numerically adjacent). Given a vector $x_1, x_2, \ldots x_n$ of positive real numbers, the *Jumbo Socially Distant Subset Problem* (JSDSP) is to find a socially distant subset $D \subseteq S$ such that $\sum_{i \in D} x_i$ is as large as possible. E.g., given $x = (12.3,\ 23.4,\ 34.5,\ 45.6)$, the best socially distant subset is $D = \{2, 4\}$.

(a) Prof. F. L. Ashbulb says "Oh, that's easily solved by the following greedy algorithm: You should simply pick $D = \{1, 3, 5, \dots\}$ or $D' = S - D = \{2, 4, 6, \dots\}$, whichever gives the larger sum." Prove them wrong.

(b) Their arch-nemesis, Prof. P. Optart says "Ha Ha Ha, you fool! That's cosmically stupid! You wouldn't recognize a proper greedy algorithm if it drained your bitcoin account. Here's how you solve it: sort the x's in decreasing order, and successively take the largest remaining value that is not adjacent to a previously selected item!" Prove them wrong as well.

(c) Having listened to Ruzzo's stunningly brilliant lectures on dynamic programming, you have a better idea, that actually *does* solve arbitrary JSDSP instances. Find the solution, not just the maximal sum; i.e. include traceback. As usual, describe your algorithm (briefly, in plain English, perhaps with succinct pseudocode), argue its correctness, and analyze its run time.

(d) Give an example, with $n \leq 8$, where your algorithm succeeds (and simulate it to show its result), but both other algorithms fail (and explain why your example is a counter-example to their correctness).

6. Suppose you own a luxury vacation condo in Hawaii that you rent to tourists through Fair B&B Bay. You offer "Economy Package" and "Deluxe Package" weekly rentals; potential renters bid for the package and week of their choice. You have to choose which bids to accept for the upcoming vacation season. Of course, you can't rent your one condo to two different bidders in any given week, and to maximize your income, you want to rent to the highest bidder each week, but there's one complication: you can't rent the condo to anyone during the week *before* a Deluxe rental because of the extra time it takes to fumigate, remove empty beer cans, clean the carpets, etc. in preparation for your Deluxe guest. When bidding closes for the season, FBBB gives you a list of the highest bid for each package for each week, i.e. a list $(e_1, e_2, \dots e_n)$ of the highest bids for an Economy stay for each of weeks 1 through n, and similarly a list $(d_1, d_2, \dots d_n)$ of the highest Deluxe bids per week. Your computational problem is to devise an algorithm to decide whether to take an economy or deluxe bid, or no bid, each week, so as to maximize your income, subject to the constraints described above. Mathematically, I can express this as: find $x_i, y_i \in \{0, 1\}, 1 \leq i \leq n$ so as to maximize $\sum_{i=1}^{n} x_i * e_i + y_i * d_i$ subject to the constraints $x_i + y_i \leq 1 - y_{i+1}, 1 \leq i \leq n - 1$ and $x_n + y_n \leq 1$. (E.g., $x_1 = 1$ means accept the economy bid for week 1; $y_1 = 1$ means accept the deluxe bid; you can't do both, and you can't do either if $y_2 = 1$.) For example, given the following bids:

	Week 1	Week 2	Week 3	Week 4
Econo	300	600	650	1500
Deluxe	1000	1500	0	2000

the optimal solution would be to accept the Deluxe bid for week 1 and the Econo bids for the remaining weeks, for a total rental income of 3750.

Give a dynamic programming algorithm for this problem (including traceback), argue correctness and analyze run time.

7. **Extra Credit:** KT Chapter 6, problem 6, page 317. (.jpg image) Assume no word is longer than L. Include correctness and timing, as usual. [Side note: this algorithm, invented by Knuth, is a core feature of TEX/LATEX typesetting system.]

Revision History:

Rev a: Reminders to include timing, correctness. — Twosday, 2/22/2022.