Problem 1 (Knapsack, Take II, 50 pts)

Alice and Bob are siblings who share a collection of items. There are n items, and each item i has a size s_i, as well as separate valuations for Alice v_i^A and for Bob v_i^B. Each sibling has their own knapsack with a capacity S. Alice and Bob can pick disjoint subsets of items to maximize their combined valuation, with the constraint that the total size of the items they pick does not exceed the capacity of their respective knapsacks. Notice their subsets of items are disjoint because they cannot both pack the same item.

Put formally, we want to maximize $\sum_{i\in I_A} v_i^A + \sum_{i\in I_B} v_i^B$ subject to the constraints $\sum_{i\in I_A} s_i \le S$ and $\sum_{i\in I_B} s_i \le S$. $I_A, I_B \subseteq \{1, 2, \ldots, n\}$ represent the subsets of items that Alice and Bob each pick, and we require I_A and I_B to be disjoint ($I_A \cap I_B = \phi$). For simplicity, we only need to output the maximum combined valuation for Alice and Bob, not the subsets.

(a) Find the max combined value Alice and Bob can get for the following input:

- $n = 3$
- Item 1: $s_1 = 2, v_1^A = 3, v_1^B = 5$
- Item 2: $s_2 = 1, v_2^A = 4, v_2^B = 2$
- Item 3: $s_3 = 3, v_3^A = 7, v_3^B = 6$
- Capacity of knapsacks $S = 4$.

(b) Give a DP algorithm to solve this variant of the knapsack problem in $O(nS^2)$. Justify the correctness and runtime of your proposed algorithm.

(c) How would you adapt your algorithm in part (b) if Alice and Bob have different knapsack capacities, S_A and S_B, respectively? And how will the runtime of your algorithm change? You do not need to justify correctness for this part.

Solution

(a)

$$\begin{aligned} \text{Item } 1 &\rightarrow \text{Bob } (s_1 = 2,\ v_1^B = 5), \\ \text{Item } 2 &\rightarrow \text{Alice } (s_2 = 1,\ v_2^A = 4), \\ \text{Item } 3 &\rightarrow \text{Alice } (s_3 = 3,\ v_3^A = 7). \end{aligned}$$

Alice uses capacity $1 + 3 = 4$ and Bob uses capacity $2 \le 4$, so the max combined value is

$$5 + 4 + 7 = \boxed{16}.$$

(b)

1. **Subproblem:**
   We define

   $$K(i, a, b) = \text{the maximum combined valuation obtainable from items } i, i+1, \ldots, n,$$

   given that Alice has remaining capacity a and Bob has remaining capacity b (with $0 \le a, b \le S$). The solution is given by $K(1, S, S)$.

2. **Guess:**
   For each item i, we consider three possibilities:
   (a) Do not select item i: use $K(i+1, a, b)$.
   (b) If $a \geq s_i$, assign item i to Alice: obtain value v_i^A and reduce her capacity to $a - s_i$, i.e., $v_i^A + K(i+1, a-s_i, b)$.
   (c) If $b \geq s_i$, assign item i to Bob: obtain value v_i^B and reduce his capacity to $b - s_i$, i.e., $v_i^B + K(i+1, a, b-s_i)$.

3. **Recurrence:**

$$K(i,a,b) = \begin{cases} 0, & \text{if } i = n+1, \\ \max\left\{ \begin{array}{l} K(i+1,a,b), \\ \mathbf{1}_{\{a \geq s_i\}}\Big(v_i^A + K(i+1, a-s_i, b)\Big), \\ \mathbf{1}_{\{b \geq s_i\}}\Big(v_i^B + K(i+1, a, b-s_i)\Big) \end{array} \right\}, & \text{if } 1 \leq i \leq n. \end{cases}$$

   Here, $\mathbf{1}_{\{a \geq s_i\}}$ (and similarly for b) indicates that the corresponding option is considered only if the condition holds.

4. We use a memorization / bottom-up approach.

   Runtime Analysis : There are $(n+1)(S+1)(S+1) = O(nS^2)$ subproblems, and each is computed in $O(1)$ time.

   Correctness :

   Optimal Substructure:
   Consider the subproblem $K(i, a, b)$, which represents the maximum combined valuation obtainable from items $i, i+1, \ldots, n$ given remaining capacities a for Alice and b for Bob. In an optimal solution for $K(i, a, b)$, one of the following must occur:
   (a) Item i is not chosen. Then the optimal solution is exactly $K(i+1, a, b)$.
   (b) Item i is assigned to Alice (assuming $a \geq s_i$). Then the combined valuation is v_i^A plus an optimal solution for the remaining items with reduced capacity, namely $K(i+1, a-s_i, b)$.
   (c) Item i is assigned to Bob (assuming $b \geq s_i$). Then the combined valuation is v_i^B plus an optimal solution for the remaining items with reduced capacity, namely $K(i+1, a, b-s_i)$.

   Since in each case the solution for the subproblem $K(i, a, b)$ relies on an optimal solution to a smaller subproblem, the problem exhibits the optimal substructure property.

   Correct Base Case:
   When $i = n+1$, no items remain to be considered. Hence, regardless of the remaining capacities a and b, the maximum combined valuation is zero:

$$K(n+1, a, b) = 0 \quad \text{for all } 0 \leq a \leq S \text{ and } 0 \leq b \leq S.$$

5. **All set.**

(c)

1. **Subproblem:**
   (Re)define

   $$K(i, a, b) = \text{the maximum combined valuation obtainable from items } i, i+1, \ldots, n,$$

   where now $0 \le a \le S_A$ is the remaining capacity in Alice's knapsack and $0 \le b \le S_B$ is the remaining capacity in Bob's knapsack. Our goal becomes computing $K(1, S_A, S_B)$.

2. **Guess:**
   The choices remain the same:

   (a) Do not select item i: $K(i+1, a, b)$.
   (b) If $a \ge s_i$, assign item i to Alice: $v_i^A + K(i+1, a - s_i, b)$.
   (c) If $b \ge s_i$, assign item i to Bob: $v_i^B + K(i+1, a, b - s_i)$.

3. **Recurrence:**

$$K(i,a,b) = \begin{cases} 0, & \text{if } i = n+1, \\ \max\left\{\begin{array}{l} K(i+1,a,b), \\ \mathbf{1}_{\{a \ge s_i\}}\Big(v_i^A + K(i+1, a-s_i, b)\Big), \\ \mathbf{1}_{\{b \ge s_i\}}\Big(v_i^B + K(i+1, a, b-s_i)\Big) \end{array}\right\}, & \text{if } 1 \le i \le n. \end{cases}$$

4. We use a memorization / bottom-up approach.

   Runtime Analysis :

   The number of subproblems is $(n+1)(S_A+1)(S_B+1) = O(nS_AS_B)$ with each subproblem computed in constant time.

5. **All set.**

Problem 2 (Make Change, 50 points)

Given a set of coin denominations $S = \{s_1, s_2, \ldots, s_n\}$, consider the problem of making change for m cents using the fewest number of coins. Assume that the set of coin denominations consists of only positive integers and that for each denomination you can use an unlimited number of them. For example, if $S = \{1, 5, 10\}$ and $m = 17$, we can make the change of 17 cents by taking one 10, one 5, and two 1's, or by taking seventeen 1's, but the former is preferred, as it uses a minimal number of coins (4 in this case).

(a) Let $S = \{1, 5, 10, 25\}$, and $m = 83$, what is the minimal number of coins to make the change? Also, give the number of coins for each denomination used.

(b) Let $S = \{1, 7, 10, 15\}$, and $m = 36$, what is the minimal number of coins to make the change? Also, give the number of coins for each denomination used.

(c) Give a DP algorithm to solve the make change problem in $O(nm)$ time. Your algorithm only needs to output the minimal number of coins, not the detailed allocation. You may assume the denominations always include 1, so there is always a way to make the change. Justify the correctness and runtime of your proposed algorithm. *(Hint: Let $DP[i, j]$ denote the optimal solution for making a change of j cents using denominations $\{s_1, s_2, \ldots, s_i\}$)*

Solution

(a) $S = \{1, 5, 10, 25\}$ and $m = 83$,

$$\begin{aligned} \text{Number of 25-cent coins} &= 3 \quad (3 \times 25 = 75), \\ \text{Number of 5-cent coins} &= 1 \quad (1 \times 5 = 5), \\ \text{Number of 1-cent coins} &= 3 \quad (3 \times 1 = 3). \end{aligned}$$

This gives a total of

$$75 + 5 + 3 = 83 \text{ cents}$$

using

$$3 + 1 + 3 = 7 \text{ coins.}$$

Thus, the minimal number of coins is $\boxed{7}$.

(b) $S = \{1, 7, 10, 15\}$ and $m = 36$,

$$\begin{aligned} \text{Number of 15-cent coins} &= 1 \quad (1 \times 15 = 15), \\ \text{Number of 7-cent coins} &= 3 \quad (3 \times 7 = 21), \\ \text{Number of 10-cent coins} &= 0, \\ \text{Number of 1-cent coins} &= 0. \end{aligned}$$

This gives a total of

$$15 + 21 = 36 \text{ cents}$$

using

$$1 + 3 = 4 \text{ coins.}$$

Thus, the minimal number of coins is $\boxed{4}$.

(c)

We define $DP[i, j]$ to be the minimum number of coins needed to make change for j cents using the denominations $\{s_1, s_2, \ldots, s_i\}$.

Base Case:

$$DP[0, j] = \begin{cases} 0, & \text{if } j = 0, \\ \infty, & \text{if } j > 0. \end{cases}$$

This reflects that zero coins are needed to make change for 0 cents.

Recurrence: For $1 \leq i \leq n$ and $0 \leq j \leq m$,

$$DP[i, j] = \min \left\{ \begin{array}{ll} DP[i-1, j], & \\ 1 + DP[i,\, j - s_i] & \text{if } j \geq s_i \end{array} \right\}.$$

When considering denomination s_i, we have two options:

1. Do not use the coin s_i: in which case, the optimal solution is $DP[i-1, j]$.
2. Use at least one coin of denomination s_i: we then add one coin and solve the subproblem $DP[i,\, j - s_i]$.

Final Solution: The minimal number of coins needed is given by $DP[n, m]$.

Justification: We use a memorization / bottom-up approach.

Runtime Analysis:

We construct a DP table $DP[i, j]$ for $i = 0, 1, \ldots, n$ and $j = 0, 1, \ldots, m$. Thus, there are $(n + 1)(m + 1) = O(nm)$ subproblems. Since computing each entry involves only a constant number of operations (a minimum over two candidate values), the overall runtime of the algorithm is $O(nm)$.

Correctness Proof:

Optimal Substructure:

For any subproblem $DP[i, j]$, an optimal solution must satisfy one of the following:

1. **Excluding the ith coin:** In this case, the optimal solution is exactly $DP[i-1, j]$.
2. **Including the ith coin:** If $j \geq s_i$, then the solution includes at least one coin of denomination s_i. Removing one such coin reduces the problem to making $j - s_i$ cents with the same set of coins, whose optimal solution is $DP[i, j - s_i]$. Adding the one coin used gives a candidate solution of $1 + DP[i, j - s_i]$.

Thus, the recurrence is an optimal substructure of the problem.

Correct Base Case:

- When $j = 0$: No coins are needed to make 0 cents.
- When $i = 0$ and $j > 0$: With no coin denominations available, it is impossible to form any positive amount; hence, we define $DP[0, j] = \infty$ for $j > 0$.