1. Recall: Knapsack problem

   (a) Given a set of items with an associated weight and value, and a maximum weight \( W \), choose a subset with total weight at most \( W \) of maximum value.

   (b) Key idea: use subproblems indexed by both the set of items remaining (first \( j \)) and the remaining capacity of the knapsack \( i \).

   (c) Analysis by cases: either we can’t take the \( j \)th item, we don’t need to take the \( j \)th item, or we have to take the \( j \)th item.

      i. In this last case, an optimal solution on the first \( j \) gives an optimal solution on the first \( j - 1 \) with capacity reduced by the weight of the \( j \)th item.

      ii. Otherwise, we can just use the solution on the first \( j - 1 \) items with the same capacity.

   (d) We can reconstruct the set from the values by checking whether or not we need to take the \( j \)th item to remain optimal, and including it only if so.

   (e) This algorithm is \textit{pseudopolynomial time} because it runs in time polynomial in the value of the weight bound, not in the size of its representation (which is the number of digits).

2. Problem: sequence alignment

   (a) Given costs for insertions, deletions, and substitutions, compute the minimum cost transformation of an input string \( X \) into another input string \( Y \).

   (b) Key lemma: the last character is transformed by either a substitution (including possibly leaving the character alone), an insertion, or a deletion. Any one of these results in a problem of transforming a pair of strings of strictly shorter total length (prefixes of the original strings).

   (c) Thus, the subproblems are the minimum cost transformations of the first \( i \) characters of \( X \) into the first \( j \) characters of \( Y \). Using the lemma, this leads to a straightforward dynamic programming algorithm.