1. Recall the coin changing algorithm described in class:

\[\text{CoinChange}(d, N) \quad \text{(Pre: The supply of coins in each denomination is unlimited)}\]

\[
\begin{align*}
&1. \quad n \leftarrow \text{length}[d] \\
&2. \quad \text{for } i \leftarrow 1 \text{ to } n \\
&3. \quad C[i,0] \leftarrow 0 \\
&4. \quad \text{for } i \leftarrow 1 \text{ to } n \\
&5. \quad \text{for } j \leftarrow 1 \text{ to } N \\
&6. \quad \text{if } i = 1 \text{ and } j < d[i] \\
&7. \quad C[1, j] \leftarrow \infty \\
&8. \quad \text{else if } i = 1 \\
&9. \quad C[1, j] \leftarrow 1 + C[1, j-d[i]] \\
&10. \quad \text{else if } j < d[i] \\
&11. \quad C[i, j] \leftarrow C[i-1, j] \\
&12. \quad \text{else} \\
&13. \quad C[i, j] \leftarrow \min(C[i-1, j], 1 + C[i, j-d[i]]) \\
&14. \quad \text{return } C[n,N]
\end{align*}
\]

a. Write a recursive algorithm which given the table \(C[1...n;0...N]\) created by the above algorithm, prints out a sequence of \(C[n,N]\) coin values which disburse \(N\) monetary units. In the case \(C[n,N]=\infty\), print a message to the effect that no such disbursal is possible.

b. Modify the above algorithm to allow for the possibility that the supply of coins in some denominations is limited. Let \(L[1...n]\) be an input array giving the limits on each denomination, i.e. at most \(L[i]\) coins of denomination \(d[i]\) are to be used \((1 \leq i \leq n)\), where \(0 \leq L[i] \leq \infty\).

2. Recall the 0-1 Knapsack Problem described in class. A thief wishes to steal \(n\) objects having values \(v_i > 0\) and weights \(w_i > 0\) \((1 \leq i \leq n)\). His knapsack, which will carry the stolen goods, holds at most a total weight \(W\). Let \(x_i \in [0,1]\) to be the indicator variable for this problem, i.e. for \(1 \leq i \leq n\), \(x_i=1\) if object \(i\) is taken, \(x_i=0\) if object \(i\) is not taken. The thief’s goal is then to maximize the total value \(\sum_{i=1}^{n} x_i v_i\), subject to the capacity constraint \(\sum_{i=1}^{n} x_i w_i \leq W\).

a. Write pseudo-code for a dynamic programming algorithm which solves this problem. Your algorithm should take as input the value and weight arrays \(v[1..n]\) and \(w[1..n]\), and the weight limit \(W\). It should generate a table \(V[1..n;0..W]\) of intermediate results. Each entry \(V[i,j]\) is the maximum value of the objects which can be stolen from the set \(\{1,...,i\}\) \((1 \leq i \leq n)\) using a knapsack having capacity \(j\) \((0 \leq j \leq W)\). Your algorithm should return the maximum possible value of the goods which can be stolen from the full set of objects, i.e. the value \(V[n,W]\). (Alternatively you may write your algorithm to return the whole table.)

b. Write an algorithm which given the filled table generated in part (a), prints out a list of exactly which objects are to be stolen.