**(1) DPV Problem 6.17.**

Given an unlimited supply of coins of denominations $x_1, x_2, ..., x_n$, we wish to make change for a value $v$; that is, we wish to find a set of coins whose total value is $v$. This might not be possible: for instance, if the denominations are 5 and 10 then we can make change for 15 but not for 12. Give an $O(nv)$ dynamic-programming algorithm for the following problem.

Input: $x_1, ..., x_n; v$.

Question: Is it possible to make change for $v$ using coins of denominations $x_1, ..., x_n$?

(1) Subproblems

For values $t$ where $1 \leq t \leq v$, $C(t) = c$ where $c$ is either true(1) or false(0) and whether it can be made with the given denominations or not. Additionally, set $C(0) = 1$

(2) Recurrence and algorithm

Let the recurrence relation be:

$C(t) = max_{x \in \{x_1, ..., x_n\}} C(t - x)$

With rough pseudocode:

initialize $C$
$C[0] = 1$
$C[1 : v] = 0$
for ($t$ from 1 to $v$) {
    $C[t] = max_{x \in \{x_1, ..., x_n\}} C[t - x]$
}
return $C[v]$

(3) Explanation of recurrence

It is possible to make change for $t$ if there is a value $t - x$ where $max_{x \in \{x_1, ..., x_n\}}$ because This signifies just adding a new coin of some existing denomination to an already found and proven value.

**(2) DPV Problem 6.19. Your algorithm should run in time at most $O(nvk)$ to get full credit. In fact, this is also possible to do in time $O(nv)$.**

Given an unlimited supply of coins of denominations $x_1, x_2, ..., x_n$, we wish to make change for a value $v$ using at most $k$ coins; that is, we wish to find a set of $\leq k$ coins whose total value is $v$. This might not be possible: for instance, if the denominations are 5 and 10 and $k = 6$, then we can make change for 55 but not for 65. Give an efficient dynamic-programming algorithm for the following problem.

Input: $x_1, ..., x_n; k; v$.

Question: Is it possible to make change for $v$ using at most $k$ coins, of denominations $x_1, ..., x_n$?

(1) Subproblems

For values $t$ where $1 \leq t \leq v$, $C(t) = c$ where $c$ is the smallest number of coins required to create the $t$, with the given denominations. If it is not possible, then let $C(t) = \infty$. Additionally, set $C(0) = 0$

(2) Recurrence and algorithm

Let the recurrence relation be:

$C(t) = min_{x \in \{x_1,...,x_n\}} C(t - x) + 1$

With rough pseudocode:

initialize $C$
$C[0] = 0$
$C[1 : v] = \infty$
for ($t$ from 1 to $v$) {
    $C[t] = min_{x \in \{x_1,...,x_n\}} C[t - x] + 1$
}
if ($C[v] \leq v$) {
    return true
} else {
    return false
}

(3) Explanation of recurrence

The first part involves initializing $C(0)$ to 0. Following this the recurrence relation can be used from $C(1)$ onward because the number of coins at a position $C(t)$ will be found using the previous values except in the case that previous value is $\infty$, which will signify that it is not possible to make this value using these coins.

**(3) DPV Problem 6.22.**

Give an $O(nt)$ algorithm for the following task.
Input: A list of n positive integers $a_1, a_2, ..., a_n$; a positive integer $t$.
Question: Does some subset of the $a_i$s add up to $t$? (You can use each $a_i$ at most once.)
(Hint: Look at subproblems of the form does a subset of $a_1, a_2, ..., a_i$ add up to $s$? )

(1) Subproblems
For values $x$ where $1 \leq x \leq n$ ad a value $y$ where $1 \leq y \leq t$, $S(x, y) = c$ where $c$ either true(1) or false(0) and whether there is a subset or not. Additionally, set $S(i, 0) = 1$ and $S(0, i) = 0$

(2) Recurrence and algorithm
Let the recurrence relation be:
$S(x, y) = (S(x-1, y)) \;||\; ((a_j \leq y) \;\&\&\; S(x-1, y-a_j))$

With rough pseudocode:
initialize $S$
$C[0:n, 0] = 1$
$C[0, 0:t] = 0$
for ($x$ from 1 to $n$) {
    for ($y$ from 1 to $t$) {
        if ($y \geq a_j$) {
            $S[x, y] = S[x-1, y] \;||\; S[x-1, y-a_j]$
        }
        if ($y < a_j$) {
            $S[x, y] = S[x-1, y]$
        }
    }
}
return $S[n, t]$

(3) Explanation of recurrence
The recurrence works because there are two specific cases. In the first, if $y \geq a_j$, then this signifies that t divided by $a_j$ would leave a remainder and would have to be equal to $y - a_j$ which would prove possible only if $S(x-1, y-a_j)) = 1$. On the other hand, if $y < a_j$, then there will be a subset that sums up to $y$ and would be possible if $S(x-1, y) = 1$.

**(4) DPV Problem 7.18, parts (a) and (c). For part (a), show how to use the original max-flow problem to solve this variation; for part (c) show how to use linear programming to solve this.**

(a) There are many sources and many sinks, and we wish to maximize the total flow from all sources to all sinks.

This can be found by implementing an additional special source vertex (I call $S1$) and an additional special sink vertex (I call $S2$). Have all the source vertices connect to $S1$ while all the sinks connect to $S2$. Following this, find the max flow from $S1$ to $S2$.

(c) Each edge has not only a capacity, but also a lower bound on the flow it must carry.

The original problem has a constraint for each edge $e$ such that $0 \leq f(e) \leq c(e)$ where each edge has a flow of $f(e)$ and a maximum constraint of $c(e)$. This total constraint can be modified for every edge so that the new constraint attached to each edge is $l(e) \leq f(e) \leq c(e)$ where $l(e)$ is the lower bound for $e$.