Simple Knapsack Problem (SKS)

Input: $d, w_1, \ldots, w_d, C \in \mathbb{N}$ (where C is the capacity)

Output: $\max_S\{K(S)|K(S) \leq C\}$, $S \subseteq \{1, \ldots, d\}$, $K(S) = \sum_{i \in S} w_i$.

1st step: define an array of subproblems

Only consider first i objects summing up to an *intermediate* weight limit j.

$$R(i,j) = \begin{cases} True & \text{if } \exists S \subseteq \{1, \ldots, i\} \text{ s.t. } K(S) = j \\ False & \text{otherwise} \end{cases}$$

$0 \leq i \leq d$, $0 \leq j \leq C$. We can compute the solution from R as follows:

$$M = \max_{j \leq C}\{j|R(d,j) = True\}$$

2nd step: Define a recurrence

- Initialize: $R(0,j) = False$, $j > 0$; $R(i,0) = True$, $i = 0, 1, \ldots, d$.
- Write a recurrence for $i, j \geq 1$
  - Suppose we do not include object i.
    Then $R(i,j) = True$ iff $R(i-1,j) = True$.
  - Suppose object i is included.
    Then $R(i,j) = True$ iff $R(i-1, j-w_i) = True$, $j - w_i \geq 0$ (i.e., there is a subset S of objects $\{1, ..., i-1\}$ s.t. the sum of weights $K(S)$ is exactly $j - w_i$, $j \geq w_i$).

$\therefore$ For $i, j \geq 1$, $R(i,j) = True$ iff

$$R(i-1,j) = True \quad \vee \quad j \geq w_i \wedge R(i-1, j-w_i) = True$$

3rd step: Write the program

We will use the same space saving trick as before by using one-dimensional array $S(j)$ for $R(i,j)$.

$S(0) \longleftarrow True$ /* *Initialize array for* $i = j = 0$ */
for $(j = 1; j \leq C; j{+}{+})$
    $S(j) \longleftarrow False$ /* *Initialize array for* $i = 0$ *and* $j = 1, \ldots, C$ */
for $(i = 1; i \leq d; i{+}{+})$
    for $(j = C; j \geq 1; j{-}{-})$
        if $(j \geq w_i$ and $S(j - w_i) = True)$ then
            $S(j) \longleftarrow True$
    /* *at this point* $S(j) = R(i,j)$ *for all* j */

- Overwriting does not create a problem here, because the values of j are considered in decreasing order. Thus the array position $S(j - w_i)$ has not yet been updated when the reference is made.

- **Ex.** Find an input for which the program would make an error if the decreasing "for" were changed to an increasing "for".

- The assertion $S(j) = R(i, j)$ can be proved by induction on the number of times the inner loop body is executed. This assertion implies that upon termination of the algorithm, $S(j) = R(d, j)$ for all j.

- **Ex.** What is the time complexity of the above algorithm? Is this polynomial time (in the length of an input instance $d, w_1, \ldots, w_d, C$, where $w_i \leq C$)?
(The understanding is that binary notation is always used for the input.)