0-1 knapsack problem revisited

Problem:

Input: $n$ items $\{1, 2, \ldots, n\}$

Item $i$ is worth $v_i$ and weight $w_i$

Total weight $W$
0-1 knapsack problem revisited

Problem:

**Input:** $n$ items $\{1, 2, \ldots, n\}$

- Item $i$ is worth $v_i$ and weight $w_i$
- Total weight $W$

**Output:** a subset $S \subseteq \{1, 2, \ldots, n\}$ such that

$$\sum_{i \in S} w_i \leq W \quad \text{and} \quad \sum_{i \in S} v_i \quad \text{is maximized}$$

Equivalently, the problem can be cast as follows:

$$\max x_i \in \{0, 1\} \quad \sum_{i=1}^{n} v_i x_i$$

s.t.

$$\sum_{i=1}^{n} w_i x_i \leq W$$
0-1 knapsack problem revisited

Problem:

*Input:* $n$ items \{1, 2, \ldots, $n$\}

*Item $i$ is worth $v_i$ and weight $w_i$*

*Total weight $W$*

*Output:* a subset $S \subseteq \{1, 2, \ldots, n\}$ such that

$$\sum_{i \in S} w_i \leq W \quad \text{and} \quad \sum_{i \in S} v_i \quad \text{is maximized}$$

Equivalently, the problem can be cast as follows:

$$\max_{x_i \in \{0, 1\}} \sum_{i=1}^{n} v_i x_i$$

s.t. $$\sum_{i=1}^{n} w_i x_i \leq W$$
0-1 knapsack problem revisited

Greedy solution strategy: three possible greedy approaches:

1. Greedy by highest value $v_i$

2. Greedy by least weight $w_i$

3. Greedy by largest value density $\frac{v_i}{w_i}$
0-1 knapsack problem revisited

Greedy solution strategy: three possible greedy approaches:

1. Greedy by highest value $v_i$

2. Greedy by least weight $w_i$

3. Greedy by largest value density $\frac{v_i}{w_i}$

All three approaches generate feasible solutions. However, cannot guarantee to always generate an optimal solution!
0-1 knapsack problem revisited

Example 1:

<table>
<thead>
<tr>
<th>i</th>
<th>$v_i$</th>
<th>$w_i$</th>
<th>$v_i/w_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total weight $W = 5$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Greedy by value density $v_i/w_i$:

- take items 1 and 2.
- value = 16, weight = 3
0-1 knapsack problem revisited

Example 1:

<table>
<thead>
<tr>
<th>i</th>
<th>$v_i$</th>
<th>$w_i$</th>
<th>$v_i/w_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Total weight $W = 5$

Greedy by value density $v_i/w_i$:

- take items 1 and 2.
  - value = 16, weight = 3

Optimal solution – by inspection

- take items 2 and 3.
  - value = 22, weight = 5
0-1 knapsack problem revisited

The knapsack problem exhibits the optimal substructure property:

1. $S' = S \setminus \{i_k\}$ is an optimal solution for weight $W - w_{i_k}$ and items $\{i_1, ..., i_{k-1}\}$.
2. The value of the solution $S$ is $v_{i_k} +$ the value of the subproblem solution $S'$. 
0-1 knapsack problem revisited

The knapsack problem exhibits the optimal substructure property:

Let \( i_k \) be the highest-numbered item in an optimal solution \( S = \{i_1, \ldots, i_{k-1}, i_k\} \). Then

1. \( S' = S - \{i_k\} \) is an optimal solution for weight \( W - w_{i_k} \) and items \( \{i_1, \ldots, i_{k-1}\} \)
0-1 knapsack problem revisited

The knapsack problem exhibits the optimal substructure property:

Let $i_k$ be the highest-numberd item in an optimal solution $S = \{i_1, \ldots, i_{k-1}, i_k\}$, Then

1. $S' = S - \{i_k\}$ is an optimal solution for weight $W - w_{i_k}$ and items $\{i_1, \ldots, i_{k-1}\}$

2. the value of the solution $S$ is

$$v_{i_k} + \text{the value of the subproblem solution } S'$$
0-1 knapsack problem revisited

- Define
  \[ c[i, w] = \text{value of an optimal solution for items } \{1, \ldots, i\} \]
  and maximum weight \( w \).
0-1 knapsack problem revisited

- Define
  
  \[ c[i, w] = \text{value of an optimal solution for items } \{1, \ldots, i\} \text{ and maximum weight } w. \]

- Then we have the following two cases for the item \( i > 0 \):
  
  - **Case 1** \((w_i > w)\): the weight of item \( i \) is larger than the weight limit \( w \), then item \( i \) cannot be included, and
    
    \[ c[i, w] = c[i - 1, w] \]
0-1 knapsack problem revisited

- Define
  
  \[ c[i, w] = \text{value of an optimal solution for items } \{1, \ldots, i\} \]
  
  and maximum weight \( w \).

- Then we have the following two cases for the item \( i > 0 \):

  - **Case 1** \((w_i > w)\): the weight of item \( i \) is larger than the weight limit \( w \), then item \( i \) cannot be included, and

    \[ c[i, w] = c[i - 1, w] \]

  - **Case 2** \((w_i \leq w)\): we have two choices:
    - **choice 1**: includes item \( i \), in which case it is \( v_i \) plus a subproblem solution for \( i - 1 \) items and the weight excluding \( w_i \).
    - **choice 2**: does not include item \( i \), in which case it is a subproblem solution of \( i - 1 \) items and the same weight.
0-1 knapsack problem revisited

- Define
  \[ c[i, w] = \text{value of an optimal solution for items } \{1, \ldots, i\} \]
  and maximum weight \( w \).

- Then we have the following two cases for the item \( i > 0 \):
  - **Case 1** (\( w_i > w \)): the weight of item \( i \) is larger than the weight limit \( w \), then item \( i \) cannot be included, and
    \[ c[i, w] = c[i - 1, w] \]
  - **Case 2** (\( w_i \leq w \)): we have two choices:
    - choice 1: includes item \( i \), in which case it is \( v_i \) plus a subproblem solution for \( i - 1 \) items and the weight excluding \( w_i \).
    - choice 2: does not include item \( i \), in which case it is a subproblem solution of \( i - 1 \) items and the same weight.

The better of these two choices should be made., that is

\[
c[i, w] = \max\{ v_i + c[i - 1, w - w_i], \ c[i - 1, w] \}
\]

- choice 1
- choice 2
In summary,

\[
c[i, w] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } w = 0 \\
 c[i - 1, w] & \text{if } i > 0 \text{ and } w_i > w \\
 \max \{v_i + c[i - 1, w - w_i], c[i - 1, w]\} & \text{if } i > 0 \text{ and } w_i \leq w 
\end{cases}
\]
0-1 knapsack problem revisited

▶ In summary,

\[ c[i, w] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } w = 0 \\
 c[i - 1, w] & \text{if } i > 0 \text{ and } w_i > w \\
 \max \{v_i + c[i - 1, w - w_i], c[i - 1, w]\} & \text{if } i > 0 \text{ and } w_i \leq w 
\end{cases} \]

▶ The value of an optimal solution = \( c[n, W] \).
0-1 knapsack problem revisited

- In summary,

\[
c[i, w] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } w = 0 \\
c[i - 1, w] & \text{if } i > 0 \text{ and } w_i > w \\
\max\{v_i + c[i - 1, w - w_i], c[i - 1, w]\} & \text{if } i > 0 \text{ and } w_i \leq w
\end{cases}
\]

- The value of an optimal solution = \( c[n, W] \).
- The set of items to take can be deduced from the \( c \)-table by starting at \( c[n, W] \) and tracing where the optimal values came from as follows:
0-1 knapsack problem revisited

- In summary,

\[
c[i, w] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } w = 0 \\
\max \{v_i + c[i - 1, w - w_i], c[i - 1, w]\} & \text{if } i > 0 \text{ and } w_i \leq w \\
c[i - 1, w] & \text{if } i > 0 \text{ and } w_i > w 
\end{cases}
\]

- The value of an optimal solution = \(c[n, W]\).

- The set of items to take can be deduced from the \(c\)-table by starting at \(c[n, W]\) and tracing where the optimal values came from as follows:

  - If \(c[i, w] = c[i - 1, w]\), item \(i\) is not part of the solution, and we continue tracing with \(c[i - 1, w]\).
0-1 knapsack problem revisited

▶ In summary,

\[
c[i, w] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } w = 0 \\
c[i - 1, w] & \text{if } i > 0 \text{ and } w_i > w \\
\max \{v_i + c[i - 1, w - w_i], c[i - 1, w]\} & \text{if } i > 0 \text{ and } w_i \leq w
\end{cases}
\]

▶ The value of an optimal solution = \(c[n, W]\).

▶ The set of items to take can be deduced from the \(c\)-table by starting at \(c[n, W]\) and tracing where the optimal values came from as follows:

▶ If \(c[i, w] = c[i - 1, w]\), item \(i\) is not part of the solution, and we continue tracing with \(c[i - 1, w]\).

▶ If \(c[i, w] \neq c[i - 1, w]\), item \(i\) is part of the solution, and we continue tracing with \(c[i - 1, w - w_i]\).
0-1 knapsack problem revisited

- In summary,

\[ c[i, w] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } w = 0 \\
 c[i - 1, w] & \text{if } i > 0 \text{ and } w_i > w \\
 \max \{v_i + c[i - 1, w - w_i], c[i - 1, w]\} & \text{if } i > 0 \text{ and } w_i \leq w 
\end{cases} \]

- The value of an optimal solution = \( c[n, W] \).
- The set of items to take can be deduced from the \( c \)-table by starting at \( c[n, W] \) and tracing where the optimal values came from as follows:
  - If \( c[i, w] = c[i - 1, w] \), item \( i \) is not part of the solution, and we continue tracing with \( c[i - 1, w] \).
  - If \( c[i, w] \neq c[i - 1, w] \), item \( i \) is part of the solution, and we continue tracing with \( c[i - 1, w - w_i] \).

- Running time: \( \Theta(nW) \):
  - \( \Theta(nW) \) to fill in the \( c \) table
    \((n + 1)(W + 1)\) entries each requiring \( \Theta(1) \) time
  - \( O(n) \) time to trace the solution
    starts in row \( n \) and moves up 1 row at each step.
0-1 knapsack problem revisited

Example 1:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$v_i$</th>
<th>$w_i$</th>
<th>$v_i/w_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Total weight $W = 5$

By dynamic programming, we generate the following $c$-table:

<table>
<thead>
<tr>
<th>$i \backslash w$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>16</td>
<td>18</td>
<td>22</td>
</tr>
</tbody>
</table>
0-1 knapsack problem revisited

Example 1:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$v_i$</th>
<th>$w_i$</th>
<th>$v_i/w_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Total weight $W = 5$

By dynamic programming, we generate the following $c$-table:

<table>
<thead>
<tr>
<th>$i \backslash w$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>16</td>
<td>18</td>
<td>22</td>
</tr>
</tbody>
</table>

By the table, we have

- The optimal solution (the items to take): $S = \{3, 2\}$
Example 2: We have $n = 9$ items with
  - value $= v = [2, 3, 3, 4, 4, 5, 7, 8, 8]$
  - weight $= w = [3, 5, 7, 4, 3, 9, 2, 11, 5]$;
  - Total allowable weight $W = 15$

DP generates the following $c$-table:

\[
\begin{array}{cccccccccccc}
& 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
2 & 0 & 0 & 2 & 2 & 3 & 3 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\
3 & 0 & 0 & 2 & 2 & 3 & 3 & 5 & 5 & 5 & 5 & 6 & 6 & 6 & 6 & 6 & 6 \\
4 & 0 & 0 & 2 & 4 & 4 & 6 & 6 & 7 & 7 & 7 & 7 & 9 & 9 & 9 & 9 & 9 \\
5 & 0 & 0 & 2 & 4 & 4 & 6 & 6 & 7 & 7 & 7 & 9 & 9 & 10 & 10 & 10 & 10 \\
6 & 0 & 0 & 2 & 4 & 4 & 6 & 6 & 7 & 9 & 9 & 10 & 10 & 11 & 11 & 11 & 11 \\
7 & 0 & 0 & 2 & 4 & 4 & 6 & 9 & 9 & 10 & 10 & 11 & 11 & 11 & 11 & 13 & 13 \\
8 & 0 & 0 & 2 & 4 & 4 & 9 & 9 & 10 & 10 & 11 & 11 & 11 & 11 & 13 & 13 & 13 \\
9 & 0 & 0 & 2 & 4 & 9 & 9 & 10 & 10 & 11 & 11 & 11 & 11 & 13 & 13 & 13 & 13 \\
10 & 0 & 0 & 2 & 9 & 9 & 10 & 10 & 11 & 11 & 11 & 11 & 11 & 13 & 13 & 13 & 13 \\
\end{array}
\]
Example 2: We have \( n = 9 \) items with

- value \( = v = [2, 3, 3, 4, 4, 5, 7, 8, 8] \)
- weight \( = w = [3, 5, 7, 4, 3, 9, 2, 11, 5] \);
- Total allowable weight \( W = 15 \)

DP generates the following \( c \)-table:

<table>
<thead>
<tr>
<th>( i/w )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>13</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>17</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>13</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>11</td>
<td>11</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>19</td>
<td>19</td>
<td>19</td>
<td>21</td>
<td>23</td>
</tr>
</tbody>
</table>
Example 2: We have $n = 9$ items with

- value $= v = [2, 3, 3, 4, 4, 5, 7, 8, 8]$
- weight $= w = [3, 5, 7, 4, 3, 9, 2, 11, 5]$;
- Total allowable weight $W = 15$

DP generates the following $c$-table:

<table>
<thead>
<tr>
<th>$i/w$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>13</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>17</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>13</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>17</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>11</td>
<td>11</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>19</td>
<td>19</td>
<td>21</td>
<td>23</td>
</tr>
</tbody>
</table>

By the table, we have

- The set of items to take $S = \{9, 7, 5, 4\}$. 
Dynamic Programming – Summary

- Not a specific algorithm, but a technique (like Divide-and-Conquer and Greedy algorithms)
Dynamic Programming – Summary

▶ Not a specific algorithm, but a technique (like Divide-and-Conquer and Greedy algorithms)
▶ Four-step (two-phase) technique:
  1. Characterize the structure of an optimal solution
  2. Recursively define the value of an optimal solution
Dynamic Programming – Summary

- Not a specific algorithm, but a technique (like Divide-and-Conquer and Greedy algorithms)

- Four-step (two-phase) technique:
  1. Characterize the structure of an optimal solution
  2. Recursively define the value of an optimal solution
  3. Compute the value of an optimal solution in a bottom-up fashion
  4. Construct an optimal solution from computed information
Dynamic Programming – Summary

Elements of DP:

1. **Optimal substructure:**
   
   the optimal solution to the problem **contains** optimal solutions to subprograms \(\Rightarrow\) recursive algorithm

Example: LCS, recursive formulation and tree
Dynamic Programming – Summary

Elements of DP:

1. **Optimal substructure:**
   the optimal solution to the problem contains optimal solutions to subprograms \(\implies\) **recursive algorithm**
   
   Example: LCS, recursive formulation and tree

2. **Overlapping subproblems:**
   There are few subproblems in total, and many recurring instances of each. *(unlike divide-and-conquer, where subproblems are independent)*
   
   Example: LCS has only \(mn\) distinct subproblems
Dynamic Programming – Summary

Elements of DP:

1. **Optimal substructure:**
   
   the optimal solution to the problem contains optimal solutions to subprograms $\Rightarrow$ recursive algorithm

   Example: LCS, recursive formulation and tree

2. **Overlapping subproblems:**

   There are few subproblems in total, and many recurring instances of each. (*unlike divide-and-conquer, where subproblems are independent*)

   Example: LCS has only $mn$ distinct subproblems

3. **Memoization:**

   after computing solutions to subproblems, store in table, subsequent calls do table lookup.

   Example: LCS has running time $\Theta(mn)$