Greedy algorithms – Recap

- A greedy algorithm makes the choice that looks best at the moment, without regard for future consequence.
Greedy algorithms – Recap

- A greedy algorithm makes the choice that looks best at the moment, without regard for future consequence.
- The proof of the greedy algorithm producing an optimal solution is based on the following two key properties:

  1. The greedy-choice property: a globally optimal solution can be arrived at by making a locally optimal (greedy) choice.
  2. The optimal substructure property: an optimal solution to the problem contains within it optimal solutions to subproblems.
Greedy algorithms – Recap

- A greedy algorithm makes the choice that looks best at the moment, without regard for future consequence.
- The proof of the greedy algorithm producing an optimal solution is based on the following two key properties:
  - **The greedy-choice property**
    - *globally* optimal solution can be *arrived at* by making a *locally* optimal (greedy) choice.
Greedy algorithms – Recap

- A greedy algorithm makes the choice that looks best at the moment, without regard for future consequence.
- The proof of the greedy algorithm producing an optimal solution is based on the following two key properties:
  - **The greedy-choice property**
    a *globally* optimal solution can be *arrived at* by making a *locally* optimal (greedy) choice.
  - **The optimal substructure property**
    an optimal solution to the problem *contains* within it optimal solution to subprograms.
Greedy algorithms – Recap

- A greedy algorithm makes the choice that looks best at the moment, without regard for future consequence.

- The proof of the greedy algorithm producing an optimal solution is based on the following two key properties:
  
  - **The greedy-choice property**
    a *globally* optimal solution can be arrived at by making a *locally* optimal (greedy) choice.
  
  - **The optimal substructure property**
    an optimal solution to the problem *contains* within it optimal solution to subprograms.

- Greedy algorithms do not always yield optimal solutions, but for many problems they do.
0-1 knapsack problem

Problem statement:

- Given \( n \) items \( \{1, 2, \ldots, n\} \)
- Item \( i \) is worth \( v_i \), and weight \( w_i \)
- Find a most valuable subset of items with total weight \( \leq W \)
0-1 knapsack problem

Problem statement:

- Given $n$ items $\{1, 2, \ldots, n\}$
- Item $i$ is worth $v_i$, and weight $w_i$
- Find a most valuable subset of items with total weight $\leq W$

Rule: have to either take an item or not take it ("0-1 Knapsack") – cannot take part of it.
0-1 knapsack problem

Problem statement:

- Given $n$ items $\{1, 2, \ldots, n\}$
- Item $i$ is worth $v_i$, and weight $w_i$
- Find a most valuable subset of items with total weight $\leq W$

Rule: have to either take an item or not take it ("0-1 Knapsack") – cannot take part of it.

Example:

- Given

<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 $W = 5$

- Find a most valuable subset of items with total weight $\leq W = 5$
Problem statement, mathematically – version 1:

Find a subset $S \subseteq \{1, 2, \ldots, n\}$ such that

maximize $\sum_{i \in S} v_i$

subject to $\sum_{i \in S} w_i \leq W$
0-1 knapsack problem

Problem statement, *mathematically* – version 2:

*Let* $x = (x_1, x_2, \ldots, x_n)$, *and*

$$x_i = \begin{cases} 
1 & \text{i-th item is in the knapsack} \\
0 & \text{i-th item is not in the knapsack}
\end{cases}$$

*Then the knapsack problem is*

\[
\text{maximize } \sum_{i=1}^{n} v_i x_i \\
\text{subject to } x_i \in \{0, 1\} \\
\sum_{i=1}^{n} w_i x_i \leq W
\]
0-1 knapsack problem

The brute-force algorithm
0-1 knapsack problem

The brute-force algorithm

- $2^n$ feasible solutions
0-1 knapsack problem

The brute-force algorithm

- $2^n$ feasible solutions
- Total cost $= O(n \cdot 2^n)$
0-1 knapsack problem

Three possible greedy strategies:

1. Greedy by highest value \( v_i \)
0-1 knapsack problem

Three possible greedy strategies:

1. Greedy by highest value $v_i$

2. Greedy by least weight $w_i$
0-1 knapsack problem

Three possible greedy strategies:

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

Example

<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 $W = 5$

Question: how about greedy by highest value? by least weight?
0-1 knapsack problem

Example

<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
- Leftover capacity = 2
0-1 knapsack problem

Example

<table>
<thead>
<tr>
<th></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  
- Leftover capacity = 2

Optimal solution  
- take items 2 and 3.  
- value = 22, weight = 5  
- no leftover capacity
0-1 knapsack problem

Example

<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
► Leftover capacity = 2

Optimal solution

► take items 2 and 3.
► value = 22, weight = 5
► no leftover capacity

Question: how about greedy by highest value? by least weight?
0-1 knapsack problem

Another example

Given the following six items with $W = 100$:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$v_i$</th>
<th>$w_i$</th>
<th>$v_i/w_i$</th>
<th>Greedy by</th>
<th>optimal solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>value</td>
<td>weight</td>
</tr>
<tr>
<td>1</td>
<td>40</td>
<td>100</td>
<td>0.4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>50</td>
<td>0.7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>45</td>
<td>0.4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>20</td>
<td>0.2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>5</td>
<td>0.4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Total value</td>
<td></td>
<td></td>
<td>40</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>Total weight</td>
<td></td>
<td></td>
<td>100</td>
<td>80</td>
</tr>
</tbody>
</table>
### 0-1 knapsack problem

#### Another example

Given the following six items with \( W = 100 \):

<table>
<thead>
<tr>
<th>( i )</th>
<th>( v_i )</th>
<th>( w_i )</th>
<th>( v_i/w_i )</th>
<th>( v_i/w_i )</th>
<th>( v_i/w_i )</th>
<th>( v_i/w_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td>100</td>
<td>0.4</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>50</td>
<td>0.7</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>45</td>
<td>0.4</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>20</td>
<td>0.2</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>5</td>
<td>0.4</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total value</td>
<td>40</td>
<td>34</td>
<td>51</td>
<td>55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total weight</td>
<td>100</td>
<td>80</td>
<td>85</td>
<td>100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All three greedy approaches generate feasible solutions, but none of them generate the optimal solution. Greedy algorithms doesn't work for the 0-1 knapsack problem!