CS 341: ALGORITHMS

Lecture 9: greedy algorithms

Slides by Trevor Brown (some material from Doug Stinson)

trevor.brown@uwaterloo.ca
(http://tbrown.pro)
CONTINUING INTERVAL COLOURING
Algorithm: GreedyIntervalColouring(\(A\))

1. sort the intervals so that \(s_1 \leq \cdots \leq s_n\)
2. \(d \leftarrow 1\)
3. \(colour[1] \leftarrow 1\)
4. \(finish[1] \leftarrow f_1\)

for \(i \leftarrow 2\) to \(n\)

\[
\text{flag} \leftarrow \text{false}\]
\[
c \leftarrow 1\]

while \(c \leq d\) and (not flag)

\[
\text{if} \ finish[c] \leq s_i \text{ then}
\]
\[
\begin{cases}
\text{colour}[i] \leftarrow c \\
\text{finish}[c] \leftarrow f_i \\
\text{flag} \leftarrow \text{true}
\end{cases}
\]

else \(c \leftarrow c + 1\)

if not flag then

\[
\begin{cases}
\text{if} \ not \ flag \text{ then}
\end{cases}
\]

\[
\text{d} \leftarrow d + 1
\]
\[
\text{colour}[i] \leftarrow d
\]
\[
\text{finish}[d] \leftarrow f_i
\]

return \((d, \text{colour})\)

\[\text{finish}[c] = \text{finish time of last interval to receive colour } c\]

\(d = \text{# of colours used so far}\)
Correctness of the Algorithm

The correctness of this greedy algorithm can be proven inductively as well as by a “slick” method—we give the “slick” proof:

Let $D$ denote the number of colours used by the algorithm.

Let $F_D$ be the first interval that has colour $D$

Let $L_c$ be the last interval that has colour $c$ and starts before $F_D$ ends

We prove $F_D$ overlaps every interval $L_c$ for all $c < D$
Let $F_D$ be the first interval that has colour $D$.

Let $L_c$ be the last interval that has colour $c$ and starts before $F_D$ ends.

We prove $F_D$ overlaps every interval $L_c$ for all $c < D$.

Let's argue $L_1$ overlaps $F_D$:

Note $L_1$ must exist (otherwise greedy would just use colour 1 for $F_D$).

And $\text{finish}[L_1]$ must be after $F_D$ starts (same reason).

So $L_1$ finishes during $F_D$.

Same argument applies to $L_2, ..., L_{D-1}$.

So, $F_D$ overlaps intervals with $D - 1$ different colours!

Moreover, every interval in $\{L_1, ..., L_{D-1}\}$ contains the starting point of $F_D$!

So, we must use $D$ colours!
**Algorithm: GreedyIntervalColouring(\(A\))**

1. sort the intervals so that \(s_1 \leq \cdots \leq s_n\)
2. \(d \leftarrow 1\)
3. \(colour[1] \leftarrow 1\)
4. \(finish[1] \leftarrow f_1\)
5. for \(i \leftarrow 2\) to \(n\)
   - \(\text{flag} \leftarrow \text{false}\)
   - \(c \leftarrow 1\)
   - while \(c \leq d\) and (not flag)
     - do
       - if \(\text{finish}[c] \leq s_i\) then
         - \(\text{flag} \leftarrow \text{true}\)
         - \(\text{colour}[i] \leftarrow c\)
         - \(\text{finish}[c] \leftarrow f_i\)
       - else \(c \leftarrow c + 1\)
   - if not flag then
     - \(d \leftarrow d + 1\)
     - \(\text{colour}[i] \leftarrow d\)
     - \(\text{finish}[d] \leftarrow f_i\)
6. return \((d, \text{colour})\)

**TIME COMPLEXITY?**

- Total \(O(n \log n + nd)\)
- Could be \(O(n \log n)\) if only a constant number of colours are needed (or even \(\log n\) colours!)
- Could be \(O(n^2)\) if \(n\) colours are needed
- Most accurate complexity statement is \(\Theta(n \log n + nD)\)

What inefficiencies exist in this algorithm? Could we make it faster with clever data structure usage?
IMPROVING THIS ALGORITHM

• Current greedy algorithm:
  • For each interval $A_i$, compare its start time $s_i$ with the $finish[c]$ times of all colours introduced so-far
  • Why? Looking for a $finish[c]$ time that is earlier than $s_i$
  • We are doing linear search... Can we do better?
  • Use a priority queue to keep track of the earliest $finish[c]$ at all times in the algorithm
    • Then we only need to look at minimum element
EXAMPLE: HEAP-BASED ALGORITHM

Initial state

Min element: NULL

Heap
EXAMPLE: HEAP-BASED ALGORITHM

Min element: NULL

Heap

Iteration i=1  Check heap minimum  Empty, so a new colour is needed

<table>
<thead>
<tr>
<th>A_1</th>
<th>A_2</th>
<th>A_3</th>
<th>A_4</th>
<th>A_5</th>
<th>A_6</th>
<th>A_7</th>
<th>A_8</th>
<th>A_9</th>
<th>A_10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x-axis
EXAMPLE: HEAP-BASED ALGORITHM

Min element: finish at time 3
Heap finish at time 3

Iteration $i=1$, Check heap minimum, Empty, so a new colour is needed

<table>
<thead>
<tr>
<th>A_1</th>
<th>A_2</th>
<th>A_3</th>
<th>A_4</th>
<th>A_5</th>
<th>A_6</th>
<th>A_7</th>
<th>A_8</th>
<th>A_9</th>
<th>A_10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Heap elements are filled as follows:
- $A_1$ filled at time 3
- $A_2$, $A_3$, $A_4$, $A_5$, $A_6$, $A_7$, $A_8$, $A_9$, $A_{10}$ filled at time 3

Finish at time 3 for both Min element and Heap.
EXAMPLE: HEAP-BASED ALGORITHM

Min element: finish at time 3

Heap: finish at time 3

Iteration i=2
Check heap minimum
Check if finish time 3 is before $s_2$
No. New colour!

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
<th>A7</th>
<th>A8</th>
<th>A9</th>
<th>A10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Finish at time 3

No. New colour!
**EXAMPLE: HEAP-BASED ALGORITHM**

- **Min element:** finish at time 3
- **Heap:** finish at time 3
- **finish at time 7**

<table>
<thead>
<tr>
<th>A_1</th>
<th>A_2</th>
<th>A_3</th>
<th>A_4</th>
<th>A_5</th>
<th>A_6</th>
<th>A_7</th>
<th>A_8</th>
<th>A_9</th>
<th>A_10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Iteration i=2**
- Check heap minimum
- Check if finish time 3 is before $s_2$
- No. New colour!
**EXAMPLE: HEAP-BASED ALGORITHM**

- **Min element:** finish at time 3
- **Heap:** finish at time 3
  - finish at time 7

### Heap Structure

<table>
<thead>
<tr>
<th>Iteration i=3</th>
<th>Check heap minimum</th>
<th>Check if finish time 3 is before $s_3$</th>
<th>No. New colour!</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$ 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_2$ 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_3$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_4$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_5$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_6$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_7$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_8$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_9$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_{10}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**x-axis**

- 0 2 4 6 8 10 12 14 16 18 20
EXAMPLE: HEAP-BASED ALGORITHM

- **Min element:** finish at time 3
- **Heap:** finish at time 3
- **finish at time 7** finish at time 5

**Iteration i=3:**
- Check heap minimum
- Check if finish time 3 is before $s_3$
- No. New colour!

<table>
<thead>
<tr>
<th></th>
<th>A_1</th>
<th>A_2</th>
<th>A_3</th>
<th>A_4</th>
<th>A_5</th>
<th>A_6</th>
<th>A_7</th>
<th>A_8</th>
<th>A_9</th>
<th>A_10</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Finish at time 3
- Finish at time 7
- Finish at time 5
EXAMPLE: HEAP-BASED ALGORITHM

Min element: finish at time 3

Heap:
- finish at time 3
- finish at time 7
- finish at time 5

<table>
<thead>
<tr>
<th>Iteration i=4</th>
<th>Check heap minimum</th>
<th>Check if finish time 3 is before $s_4$</th>
<th>Yes. Reuse colour, deleteMin and insert new finish time into heap!</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_2 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_3 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A_10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x-axis

0 2 4 6 8 10 12 14 16 18 20
EXAMPLE: HEAP-BASED ALGORITHM

Iteration i=4
Check heap minimum
Check if finish time 3 is before $s_4$
Yes. Reuse colour, deleteMin and insert new finish time into heap!

Min element: 
finish at time 5

Heap

<table>
<thead>
<tr>
<th>A₁</th>
<th>A₂</th>
<th>A₃</th>
<th>A₄</th>
<th>A₅</th>
<th>A₆</th>
<th>A₇</th>
<th>A₈</th>
<th>A₉</th>
<th>A₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Min element: 
finish at time 5

finish at time 7

finish at time 5

x-axis
EXAMPLE: HEAP-BASED ALGORITHM

Iteration i=4
Check heap minimum
Check if finish time 3 is before $s_4$
Yes. Reuse colour, deleteMin and insert new finish time into heap!

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
<th>A7</th>
<th>A8</th>
<th>A9</th>
<th>A10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Min element: finish at time 5
Heap finish at time 9
finish at time 7 finish at time 5

Iteration $i=4$ finish at time 7 finish at time 5

Finish at time 9

x-axis
**EXAMPLE: HEAP-BASED ALGORITHM**

- **Min element:** finish at time 5
- **Heap:** finish at time 9
  - finish at time 7
  - finish at time 5

<table>
<thead>
<tr>
<th>A₁</th>
<th>A₂</th>
<th>A₃</th>
<th>A₄</th>
<th>A₅</th>
<th>A₆</th>
<th>A₇</th>
<th>A₈</th>
<th>A₉</th>
<th>A₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

- **Iteration i = 5:**
  - Check heap minimum
  - Check if finish time 5 is before $s_5$

- **Yes.** Reuse colour, `deleteMin` and insert new finish time into heap!
## Example: Heap-Based Algorithm

### Iteration i=5
- Check heap minimum
- Check if finish time 5 is before $s_5$

<table>
<thead>
<tr>
<th></th>
<th>A_1</th>
<th>A_2</th>
<th>A_3</th>
<th>A_4</th>
<th>A_5</th>
<th>A_6</th>
<th>A_7</th>
<th>A_8</th>
<th>A_9</th>
<th>A_10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finish at time</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Min element: finish at time 7

Heap: finish at time 9

Finish at time 7

Yes. **Reuse** colour, **deleteMin** and insert new finish time into heap!

-x-axis
### Example: Heap-Based Algorithm

#### Min element:
- Finish at time 7

#### Heap:
- Finish at time 9
- Finish at time 7
- Finish at time 13

#### Table:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Graph:
- **Iteration i=5**
- **Check heap minimum**
- **Check if finish time 5 is before \( s_5 \)**
- **Yes. Reuse colour, deleteMin and insert new finish time into heap!**

**x-axis**

**0 2 4 6 8 10 12 14 16 18 20**
 EXAMPLE: HEAP-BASED ALGORITHM

Min element: finish at time 7

Heap

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
<th>A7</th>
<th>A8</th>
<th>A9</th>
<th>A10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Iteration i=6
Check heap minimum
Check if finish time 7 is before $s_6$

Yes. Reuse colour, deleteMin and insert new finish time into heap!

Finish at time 7
Finish at time 9
Finish at time 7
Finish at time 13
EXAMPLE: HEAP-BASED ALGORITHM

Min element: finish at time 9
Heap finish at time 9

- Iteration i=6: Check heap minimum
- Check if finish time 7 is before $s_6$: Yes. Reuse colour, deleteMin and insert new finish time into heap!

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
<th>A7</th>
<th>A8</th>
<th>A9</th>
<th>A10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

x-axis

- finish at time 9
- finish at time 13
**EXAMPLE: HEAP-BASED ALGORITHM**

<table>
<thead>
<tr>
<th>Iteration i=6</th>
<th>Check heap minimum</th>
<th>Check if finish time 7 is before $s_6$</th>
<th>Yes. <strong>Reuse</strong> colour, <strong>deleteMin</strong> and insert new finish time into heap!</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Min element:** finish at time 9  

**Heap**  
- finish at time 9  
- finish at time 11  
- finish at time 13  

**And so on, and so forth...**

---

The diagram shows a timeline with a check heap minimum process indicated at iteration i=6. The finish times for each element are shown on the x-axis, with A1 finishing at time 1, A2 finishing at time 2, and so on. The min element is highlighted as finishing at time 9. The process of checking if the finish time is before $s_6$ is confirmed to be yes, leading to the reuse of colour and the deletion followed by insertion of a new finish time into the heap.
TIME COMPLEXITY

Whenever we colour interval $A_i$ with colour $c$, we insert $(f_i, c)$ into the priority queue (here $f_i$ is the “key”).

When we want to colour the next interval $A_i$, we look at the minimum key $f$ in the priority queue. If $f \leq s_i$, then we do a deletemin operation, yielding the pair $(f, c)$ and we use colour $c$ for interval $A_i$. If $f > s_i$, we introduce a new colour.

Note that each interval is inserted once and deleted once from the priority queue. Therefore, the complexity of this approach is $O(n \log D)$. Since $D \leq n$, it is $O(n \log n)$. 

$O(\log S)$ where $S = \text{size(priority queue)}$

$O(1)$

$O(\log D)$

$O(\log D)$
KNAPSACK PROBLEMS
Problem 4.4

Knapsack

Instance: Profits $P = [p_1, \ldots, p_n]$; weights $W = [w_1, \ldots, w_n]$; and a capacity, $M$. These are all positive integers.

Feasible solution: An $n$-tuple $X = [x_1, \ldots, x_n]$ where $\sum_{i=1}^{n} w_ix_i \leq M$. 

Gotta respect the weight limit…
Knapsack

Instance: Profits $P = [p_1, \ldots, p_n]$; weights $W = [w_1, \ldots, w_n]$; and a capacity, $M$. These are all positive integers.

Feasible solution: An $n$-tuple $X = [x_1, \ldots, x_n]$ where $\sum_{i=1}^{n} w_i x_i \leq M$.

In the 0-1 Knapsack problem (often denoted just as Knapsack), we require that $x_i \in \{0, 1\}$, $1 \leq i \leq n$.

In the Rational Knapsack problem, we require that $x_i \in \mathbb{Q}$ and $0 \leq x_i \leq 1$, $1 \leq i \leq n$.

Find: A feasible solution $X$ that maximizes $\sum_{i=1}^{n} p_i x_i$.

0-1 Knapsack: NP Hard. Probably requires exponential time to solve...

Rational knapsack: Can be solved in polynomial time by a greedy alg!

Let's discuss this now... other one later
Possible Greedy Strategies for Knapsack Problems

Consider the items in decreasing order of profit (i.e., the local evaluation criterion is \( p_i \)).

Consider the items in increasing order of weight (i.e., the local evaluation criterion is \( w_i \)).

Consider the items in decreasing order of profit divided by weight (i.e., the local evaluation criterion is \( p_i/w_i \)).

Does one of these strategies yield a correct greedy algorithm for the Rational Knapsack problem?
Consider the following instance of the **Rational Knapsack Problem**:

<table>
<thead>
<tr>
<th>profits</th>
<th>50</th>
<th>90</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>weights</td>
<td>50</td>
<td>60</td>
<td>10</td>
</tr>
<tr>
<td>capacity</td>
<td>100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If we consider the objects in order of decreasing profit, we obtain the solution $(4/5, 1, 0)$, yielding profit 130.

If we consider the objects in order of increasing weight, we obtain the solution $(1, 2/3, 1)$, yielding profit 130.

If we consider the objects in order of decreasing profit / weight ratio, then we obtain the solution $(3/5, 1, 1)$, yielding profit 140.

At least for this instance, the third strategy is better than the other two strategies.
Algorithm: GreedyRationalKnapsack\((P, W : \text{array}; M : \text{integer})\)

sort the items so that \(p_1/w_1 \geq \cdots \geq p_n/w_n\)

\(X \leftarrow [0, \ldots, 0]\)

\(i \leftarrow 1\)

\(\text{CurW} \leftarrow 0\)

while \((\text{CurW} < M) \text{ and } (i \leq n)\)

\(\text{if } \text{CurW} + w_i \leq M\)

\(\quad \{\)

\(\quad \quad x_i \leftarrow 1\)

\(\quad \quad \text{CurW} \leftarrow \text{CurW} + w_i\)

\(\quad \quad i \leftarrow i + 1\)

\(\quad \}\)

\(\text{else}\)

\(\quad \{\)

\(\quad \quad x_i \leftarrow (M - \text{CurW})/w_i\)

\(\quad \quad \text{CurW} := M\)

\(\quad \}\)

return \((X)\)

No items are chosen yet

Current weight of knapsack

Until full, or no more items

If \text{whole item} fits

Put it in the knapsack

Else put in as much of the item as you can, to exactly fill the knapsack

Either \(X=(1,1,\ldots,1,0,\ldots,0)\) or \(X=(1,1,\ldots,1,x_i,0,\ldots,0)\) where \(x_i \in (0,1)\)

Time complexity?
NEXT TIME

• Time complexity
• Optimality proof
• New problem(s)