CS 341: ALGORITHMS
Lecture 8: dynamic programming II
Readings: see website
Trevor Brown
https://student.cs.uwaterloo.ca/~cs341
trevor.brown@uwaterloo.ca

ROD CUTTING
A "REAL" DYNAMIC PROGRAMMING EXAMPLE
• Input:
  • $n$: length of rod
  • $p_1, ..., p_n$: $p_i =$ price of a rod of length $i$
• Output:
  • Max income possible by cutting the rod of length $n$
    into any number of integer pieces (maybe no cuts)

$\text{Example output: 10}$

DYNAMIC PROGRAMMING APPROACH
• High level idea (can just think recursively to start)
  • Given a rod of length $n$
    • Either make no cuts,
      or make a cut and recurse on the remaining parts
  • Where should we cut?

DYNAMIC PROGRAMMING APPROACH
• Try all ways of making that cut
  • i.e., try a cut at positions $1, 2, ..., n - 1$
  • In each case, recurse on two rods $[0, i]$ and $[i, n]$
  • Take the max income over all possibilities (each i / no cut)

Critical step! Must define what $M(k)$ means, semantically!

EXCEL TABLE

<table>
<thead>
<tr>
<th>$i$</th>
<th>$M(i)$</th>
<th>$M(i+1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>$p_2$</td>
<td>$p_2$</td>
</tr>
<tr>
<td>3</td>
<td>$M(1) + p_3$</td>
<td>$M(2) + p_3$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$n$</td>
<td>$M(n-1)$</td>
<td>$M(n)$</td>
</tr>
</tbody>
</table>

Recurrence: $M(k) = \max[p_k, \max_{i=1}^{k-1}(M(i) + M(k-i))]$

Computing solutions bottom-up

Critical step! Must define what $M(k)$ means, semantically!
Recall, semantically, $W(k) =$ maximum income for rod of length $k$.

**Recurrence:** $W(k) = \max \{0, \max_{i \leq 1} W(k-i) \}$

- **RedCutting:**
  ```java
  int W[] = new int[7];
  // compute each entry W[k]
  for (k = 1; k <= 7; k++)
    for (i = 1; i <= k; i++)
      // try each cut in I_i(k)
      W[k] = \max \{W[k], W[i-1] + W[k-i] \}
  return W[7];
  ```

- **Time complexity (unit cost)?** $O(\epsilon)$

**Solution to 0-1 Knapsack**

Recall, semantically, $P(i,m) =$ maximum profit using any subset of the items $1 \ldots i$, with weight limit $m$.

**Problem:** output maximum value one can get from taking $\leq 7$ items out of these four items.

Let $P(i,m) =$ maximum profit using any subset of the items $1 \ldots i$, with weight limit $m$.

- **Subproblem:** output maximum value for $\leq 7$ items out of these four items.

First, what if the camera is included in $O$?

Then with the $O$, we must achieve the best possible value using only items 1–3.

Suppose the optimal solution $O$ does not include the camera, then $P(4,7)$ is best we can do with the first three items and weight limit $7$kg.

— This is a smaller subproblem, reduced # of items.

Then, with the $O$ we must achieve the best possible value using only items 1–2, and items 1–2 $O$ must achieve the best possible value.

If $O$ includes the camera, then $P(4,7)$ is best we can do with the first three items and weight limit $7$kg.

— This is a smaller subproblem, reduced weight and # of items.

Recall, for a maximum profit using any subset of the items 1–i, with weight limit m:

- suppose the optimal solution $O$ includes the camera. Then with the remaining $7kg - w_4 = 6kg$, and items 1–2 $O$ must achieve the best possible value.

How to evaluate both possibilities, in & not in $O$?

Recall, $P(i,m) =$ maximum profit using any subset of the items 1–i, with weight limit m.

- **In general**:
  - If $O$ does not include the camera, then $P(4,7)$ is best we can do with the first three items and weight limit $7$kg.
    - $P(4,7) = P(3,7) + p_4$.
    - $P(3,7) = P(2,7) + p_3$.
    - $P(2,7) = P(1,7) + p_2$.
    - $P(1,7) = P(0,7) + p_1$.
  - If $O$ includes the camera, then $P(4,7)$ is the best we can do with the first three items and weight limit $7$kg.
    - $P(4,7) = P(3,7)$.
    - $P(3,7) = P(2,7) + p_3$.
    - $P(2,7) = P(1,7) + p_2$.
    - $P(1,7) = P(0,7) + p_1$.
  - Try both and take the better result (How?)

Note that once $i \geq 2$ and $m \geq w_i$, we can stop the recursion. These are special cases.
Suppose we have profits 1, 2, 3, 5, 7, 10, weights 2, 3, 5, 8, 13, 16, and capacity 30.

The following table is computed:

**m-axis**

<table>
<thead>
<tr>
<th>Items</th>
<th>( P[3, 16] )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**EXERCISE**

\[
\max \{ \{ P[i-1, m], p_i + P[i-1, m - w_i] \} \mid i \geq 2, m \geq w_i, P[i-1, m] \} 
\]

Would the following fill order work? (items)

<table>
<thead>
<tr>
<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>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FILLING THE ARRAY:**

\[
P[m, w] = \begin{cases} 
\max(P[i-1, m], p_i + P[i-1, m-w_i]) & \text{if } i \geq 2, m \geq w_i \\
0 & \text{if } i = 1, m \geq w_1 \\
0 & \text{if } i = 1, m < w_1 \\
P[i-1, m] & \text{if } i \geq 2, m < w_i \\
\end{cases} 
\]

**General case:** \( i \geq 2 \) and \( m \geq w_i \)

Since \( m \geq w_i \), we can carry item \( i \).

\[
P[i, m] = \max(P[i-1, m], p_i + P[i-1, m-w_i]) 
\]

**Special case 1:** \( i \geq 2 \) and \( m < w_i \)

Since \( m < w_i \), we cannot carry item \( i \).

\[
P[i, m] = 0 
\]

**Special case 2:** \( i = 1 \) and \( m \geq w_1 \)

Since \( i = 1 \), we can only use item 1.

\[
P[i, m] = P[i-1, m] 
\]

**Special case 3:** \( i = 1 \) and \( m < w_1 \)

Since \( i = 1 \), we can only use item 1.

\[
P[i, m] = 0 
\]

**Recurrence Relation:**

\[
P[i, m] = \begin{cases} 
\max(P[i-1, m], p_i + P[i-1, m-w_i]) & \text{if } i \geq 2, m \geq w_i \\
0 & \text{if } i \geq 2, m < w_i \\
0 & \text{if } i = 1, m \geq w_1 \\
P[i-1, m] & \text{if } i = 1, m < w_1 \\
\end{cases} 
\]

We only ever look at \( P[1, \cdot] \), for \( m \geq w_1 \).

We only ever look at the previous row.

Depending how many zeros we have in the top row, and how far back we're looking, might start to get cells containing \( p_i + P[i-1, m] \).

To satify data dependencies, we can fill entries in the order:

1. We start from \( P[1, \cdot] \) or \( P[2, \cdot] \).
2. For each cell \( P[i, m] \), we only look at cells \( P[i-1, m] \), \( P[i-1, m-w_i] \), and \( P[i-2, m] \) if \( i \geq 2 \).
3. We fill cells in the order of increasing \( m \) values.

Data dependency:

\[
\text{We only ever look at cells already computed.} 
\]

Data dependency:

\[
\text{We only ever look at cells already computed.} 
\]

Suppose \( i \) does not fit until the \( m \) value \( w_i \).

Where is slot \( (i-1) \), \( w_i \)?

Data dependency:

\[
\text{We only ever look at cells already computed.} 
\]

Consider the cell where \( i \) cannot fit.

So, what value should be stored in this entry?

We only ever look at the previous row.

Data dependency:

\[
\text{We only ever look at cells already computed.} 
\]

Would the following fill order work? (items)

<table>
<thead>
<tr>
<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>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EXERCISE

max\{P[i - 1, m], P[i - 1, m - w_i]\} if \( i \geq 2, m \geq w_i \)

\[ P[i - 1, m] \]

if \( i \geq 2, m < w_i \)

Suppose we have profits 1, 2, 3, 5, 7, 10, weights 2, 3, 5, 8, 13, 16, and capacity 30.
The following table is computed:

<table>
<thead>
<tr>
<th>( m )-axis (weight)</th>
<th>( i )-axis (items)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1, 2, 3, 5, 7, 10</td>
</tr>
<tr>
<td>3</td>
<td>1, 2, 3, 5, 7, 10</td>
</tr>
<tr>
<td>5</td>
<td>1, 2, 3, 5, 7, 10</td>
</tr>
<tr>
<td>8</td>
<td>1, 2, 3, 5, 7, 10</td>
</tr>
<tr>
<td>13</td>
<td>1, 2, 3, 5, 7, 10</td>
</tr>
<tr>
<td>16</td>
<td>1, 2, 3, 5, 7, 10</td>
</tr>
<tr>
<td>20</td>
<td>1, 2, 3, 5, 7, 10</td>
</tr>
<tr>
<td>22</td>
<td>1, 2, 3, 5, 7, 10</td>
</tr>
<tr>
<td>25</td>
<td>1, 2, 3, 5, 7, 10</td>
</tr>
<tr>
<td>27</td>
<td>1, 2, 3, 5, 7, 10</td>
</tr>
<tr>
<td>28</td>
<td>1, 2, 3, 5, 7, 10</td>
</tr>
<tr>
<td>29</td>
<td>1, 2, 3, 5, 7, 10</td>
</tr>
</tbody>
</table>

\[ P[3, 16] = \max\{P[2, 16], P[2, 11] + 3\} = \max(3, 3 + 3) = 6. \]

Recall: To satisfy data dependencies, we can fill entries in the order:

for \( i = 1 \ldots n \), for \( m = 0 \ldots M \).

OUTPUTTING CONTENTS OF THE OPTIMAL KNAPSACK \( \mathcal{O} \)
The optimal solution is computed by tracing back through the table.

For the previous example, consisting of profits 1, 2, 3, 5, 7, 10, weights 2, 3, 5, 8, 13, 16, and capacity 30, the optimal solution is \( \{1, 2, 3, 5, 7, 10\} \).

19 > 17, so \( \mathcal{O} \) must take item 6.

Remaining weight = 14.

Best profit for remaining items + weight = 18 or 17. So, any optimal solution must take item 6. Consider \( \mathcal{O} \).

Exercise: continue, and determine which other items are in \( \mathcal{O} \).
SIMPLIFYING BASE CASES

**m-axis** (remaining weight limit)

**i-axis** (can use items in 1...i)

For **m = 0**, we have **P[1...m] = 0**

For **m < w[i]**, we have **P[1...m] = P[1...m-w[i]]**

For **m ≥ w[i]**, we have **P[1...m] = P[1...m-w[i]] + p[i]**

We get much simpler code!

SAVING SPACE

We never look at **P[i-2]**[…]. Just keep two arrays representing **P[i]** and **P[i-1]**

COIN CHANGING

There is a denomination with unit value!

In 0-1 knapsack, we only considered two subproblems in our recurrence: taking an item, or not.

Final recurrence relation

Let **N[i, t]** denote the optimal solution to the subproblem consisting of the first **i** coin denominations **d_1, ..., d_i** and target sum **t**.

Exploring: some sensible base case(s)?

General case:

What are the different ways we could use coin denomination **d_j**?

What subproblems / solutions should we use?

Final recurrence relation
Let $N[i, t]$ denote the optimal solution to the subproblem consisting of the first $i$ coin denominations $d_1, \ldots, d_i$, and target sum $t$.

Since $d_1 = 1$, we immediately have $N[1, t] = t$ for all $t$.

General case:

What are the different ways we could use coin denomination $d_i$?

What subproblems / solutions should we use?

Final recurrence relation

Also $N[i, 0] = 0$ for all $i$.

Consider cell $N[i, t] = \min \{N[i-1, t] - j \cdot d_i : 0 \leq j \leq \lfloor t/d_i \rfloor \}$

If $i \geq 2$.

We only look at the previous $i$-row!

It is sufficient to fill row $i = 1$ (base case), then for $i = 2, \ldots$, for $t = 0, 1, \ldots$.

Compute $\min(\ldots)$ over $j = 0, \ldots, \lfloor t/d_i \rfloor$.
If $\sigma$ is constant, $O, \in = T \log \in$. So, $O = I \sigma$ is allowed to vary, $O(n)$ is fixed at 10 and $T = n$. However, if $\log T = \in$, And $\log c = \in$ has a linear runtime in $n$, where this DP solution shines! (O, I)$ is a polynomial in our input size $S$. Recall $R(T) \in O(n^2)$, and $S \in O(n \log T)$.

If $T \in O(n)$, then $S \in O(n \log n)$, and $R(T) \in O(n^2)$. Note $O(n^2)$ is a smaller runtime than $O(S^2) = O(n \log n)$. And $S^2$ is polynomial in $S$, so $O(n^2)$ is a polynomial runtime. So, for some inputs with relatively small $T$, we can get polynomial runtimes!

In particular, for $T \in O(n^k)$ where $k$ is constant, $R(T) \in O(n(n^k)^2) = O(n^{2k+1})$ and $S \in O(n \log n^k) = O(n \log n)$. And $R(T) \in O(n^{2k+1}) \leq O\left(\lceil \log n \rceil^{2k+1}\right) = O(3^{2k+1})$