CS 341: ALGORITHMS

Trevor Brown

trevor.brown@uwaterloo.ca

DC 2338, Office hour M3-4pm
THIS TIME

• Dynamic programming (DP) algorithms
  • 0-1 Knapsack
  • Coin Changing (when greedy fails)
Designing Dynamic Programming Algorithms for Optimization Problems

Optimal Structure
Examine the structure of an optimal solution to a problem instance $I$, and determine if an optimal solution for $I$ can be expressed in terms of optimal solutions to certain subproblems of $I$.

Define Subproblems
Define a set of subproblems $S(I)$ of the instance $I$, the solution of which enables the optimal solution of $I$ to be computed. $I$ will be the last or largest instance in the set $S(I)$.
Designing Dynamic Programming Algorithms (cont.)

Recurrence Relation
Derive a recurrence relation on the optimal solutions to the instances in $\mathcal{S}(I)$. This recurrence relation should be completely specified in terms of optimal solutions to (smaller) instances in $\mathcal{S}(I)$ and/or base cases.

Compute Optimal Solutions
Compute the optimal solutions to all the instances in $\mathcal{S}(I)$. Compute these solutions using the recurrence relation in a bottom-up fashion, filling in a table of values containing these optimal solutions. Whenever a particular table entry is filled in using the recurrence relation, the optimal solutions of relevant subproblems can be looked up in the table (they have been computed already). The final table entry is the solution to $I$. 
Express solution to problem size \( i \) in terms of problem sizes \( i-1 \) and \( i-2 \).

Recurrence relation that leads to this code:

\[
f(n) = \begin{cases} 
1 & : i = 1 \\
0 & : i = 0 \\
f(n-1) + f(n-2) & : i \geq 2 
\end{cases}
\]

Base cases are important: they totally determine the final solution.

Combining solutions to subproblems is easy in this case: just add (+)
Problem 5.1

0-1 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 $x_i \in \{0, 1\}$ for $1 \leq i \leq n$, and $\sum_{i=1}^{n} w_i x_i \leq M$.

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

Studied similar problem two lectures ago. Here we cannot take only part of an item.
Suppose the optimal solution \( O \) does not include this. Then with the \( O \) must achieve the best possible value using only items 1-3.

**Subproblem:** output max value for \( \leq 7kg \) out of these three items

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

This is a smaller subproblem: reduced # of items

Goal: create recurrence relation to describe optimal solution in terms of subproblems

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

Note: \( P[n, M] (= P[4, 7]) \) is the optimal profit

If \( O \) does not include the camera, then \( P[4, 7] = \) best we can do with the first three items and weight limit \( 7kg \)

That is, \( P[4, 7] = P[3, 7] \)
Suppose the optimal solution \( O \) includes this subproblem:

Subproblem: output max value for \( \leq 6 \text{kg} \) out of these three items

Problem: output maximum value one can get from taking \( \leq 7 \text{kg} \), out of these four items.

This is a smaller subproblem: reduced weight and # of items

Recall: \( P[i, m] = \) maximum profit using any subset of the items \( 1..i \), with weight limit \( m \)

If \( O \) includes the camera, then \( P[4, 7] = p_4 + \) best we can do with the first three items and weight limit \( 7 \text{kg} - w_4 = 6 \text{kg} \)

That is, \( P[4, 7] = p_4 + P[3, 6] \)

How to evaluate both possibilities: in & not in \( O \)?

Then with the remaining \( 7 \text{kg} - w_4 = 6 \text{kg} \), and items 1-3, \( O \) must achieve the best possible value.
Recall: \( P[i, m] = \) maximum profit using any subset of the items 1..\( i \), with weight limit \( m \)

<table>
<thead>
<tr>
<th></th>
<th>In general:</th>
</tr>
</thead>
<tbody>
<tr>
<td>If ( O ) does not include the camera, then</td>
<td>( P[4, 7] = P[3, 7] )</td>
</tr>
<tr>
<td>( P[4, 7] ) = best we can do with the</td>
<td>( P[i, m] = P[i-1, m] )</td>
</tr>
<tr>
<td>first three items and weight limit 7kg</td>
<td></td>
</tr>
<tr>
<td>If ( O ) includes the camera, then</td>
<td>( P[4, 7] = p_4 + P[3, 7-w_4] )</td>
</tr>
<tr>
<td>( P[4, 7] ) = best we can do with the</td>
<td>( P[i, m] = p_i + P[i-1, m-w_i] )</td>
</tr>
<tr>
<td>first three items and weight limit 7kg</td>
<td></td>
</tr>
<tr>
<td>(- w_4 = 6kg )</td>
<td></td>
</tr>
</tbody>
</table>

Try both and take the better result! (How?)

\[
P[4, 7] = \max\{P[3, 7], p_4 + P[3, 7-w_4]\}
\]

Note that \( \max\{P[i-1, m], p_i + P[i-1, m-w_i]\} \) is only valid if \( i \geq 2 \) and \( m \geq w_i \)

What to do when \( i = 1 \) or \( m < w_i \)? These are special cases.
<table>
<thead>
<tr>
<th>Special case 1: ( i \geq 2 ) and ( m &lt; w_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Since ( m &lt; w_i ), we <strong>cannot carry item i</strong>.</td>
</tr>
<tr>
<td>( P[i, m] = 0 ).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Special case 2: ( i = 1 ) and ( m \geq w_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Since ( i \leq 1 ), we <strong>can only use item 1</strong>.</td>
</tr>
<tr>
<td>Since ( m \geq w_i ), we <strong>can carry item 1</strong>.</td>
</tr>
<tr>
<td>So, ( P[i, m] = p_i ).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Special case 3: ( i = 1 ) and ( m &lt; w_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Since ( i \leq 1 ), we <strong>can only use item 1</strong>.</td>
</tr>
<tr>
<td>Since ( m &lt; w_i ), we <strong>cannot carry item 1</strong>.</td>
</tr>
<tr>
<td>So, ( P[i, m] = 0 ).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>General case: ( i \geq 2 ) and ( m \geq w_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Since ( m \geq w_i ), we <strong>can carry item i</strong>.</td>
</tr>
<tr>
<td>( P[i, m] = \max{P[i - 1, m], p_i + P[i - 1, m - w_i]} )</td>
</tr>
</tbody>
</table>

**Recurrence Relation:**

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

**In-class exercise!**
FILLING THE ARRAY

\[ P[i, m] = \begin{cases} 
  \max\{P[i - 1, m], p_i + P[i - 1, m - w_i]\} & \text{if } i \geq 2, m \geq w_i \\
  P[i - 1, m] & \text{if } i \geq 2, m < w_i \\
  p_i & \text{if } i = 1, m \geq w_1 \\
  0 & \text{if } i = 1, m < w_1.
\]
FILLING THE ARRAY

Suppose \( m < w_i \)

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

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

\[
P[i, m] = \begin{cases} 
\max\{P[i-1, m], p_i + P[i-1, m-w_i]\} & \text{if } i \geq 2, \ m \geq w_i \\
P[i-1, m] & \text{if } i \geq 2, \ m < w_i \\
p_1 & \text{if } i = 1, \ m \geq w_1 \\
0 & \text{if } i = 1, \ m < w_1 
\end{cases}
\]

Data dependency: need this cell already computed
FILLING THE ARRAY

\[ P[i, m] = \begin{cases} 
\max\{ P[i-1, m], p_i + P[i-1, m-w_i] \} & \text{if } i \geq 2, \ m \geq w_i \\
\max\{ P[i-1, m] \} & \text{if } i \geq 2, \ m < w_i \\
p_1 & \text{if } i = 1, \ m \geq w_1 \\
0 & \text{if } i = 1, \ m < w_1.
\end{cases} \]

Suppose \( m < w_i \)

Data dependency: need this cell already computed

Suppose \( m \geq w_i \)

We only look at the previous \( i \)-row!

Would this order work?
for \( i=1..n \), for \( m=M..1 \)

It is sufficient to fill:
row \( i=1 \) (2 cases), then for \( i=2..n \), for \( m=1..M \)

- \( i \)-axis (can use items in \( 1..i \))
- \( m \)-axis (remaining weight limit)
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:

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 1 |   |   |   |   |   | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |   |   |   |   |   |   |   |   |   |   |   |
| 2 |   |   |   |   | 1 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |   |   |   |   |   |   |   |   |   |   |   |
| 3 |   |   |   | 1 | 2 | 3 | 3 | 4 | 5 | 5 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |   |   |   |   |   |   |   |   |   |   |   |   |
| 4 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 5 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 6 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

\[ P[3, 16] = \]

? What do you think?
Algorithm: $0\text{-}1\text{Knapsack}(p_1, \ldots, p_n, w_1, \ldots, w_n, M)$

for $m \leftarrow 0$ to $M$
  if $m \geq w_1$
    then $P[1, m] \leftarrow p_1$
  else $P[1, m] \leftarrow 0$
for $i \leftarrow 2$ to $n$
  for $m \leftarrow 0$ to $M$
    if $m < w_i$
      do
        then $P[i, m] \leftarrow P[i - 1, m]$
      else $P[i, m] \leftarrow \max\{P[i - 1, m - w_i] + p_i, P[i - 1, m]\}$
return $(P[n, M])$;

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

Fill first row ($i=1$)

Last two cases

Fill other rows in our chosen order

First two cases

Read & return optimal profit

How about the optimal items?
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 ???

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 3 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 4 | 5 | 5 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 4 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 4 | 5 | 5 | 6 | 7 | 7 | 8 | 8 | 9 | 10 | 10 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 |
| 5 | 1 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 6 | 1 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

**Exercise:** continue, and determine which other items are in O

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 3 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 4 | 5 | 5 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 4 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 4 | 5 | 5 | 6 | 7 | 7 | 8 | 8 | 9 | 10 | 10 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 |
| 5 | 1 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| 6 | 1 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

**Same profit using items 1..4 or 1..5. So, item 5 is unused!**

**Best profit for remaining items + weight**

18 > 17, so item 6 is in O

Remaining weight is 14

Start at optimal profit
Algorithm: $\text{ComputeOptimalKnapsack}(p_1, \ldots, p_n, w_1, \ldots, w_n, M, P)$

$m \leftarrow M$
$p \leftarrow P[n, M]$

for $i \leftarrow n$ downto 2

\[ \begin{align*}
&\text{if } p = P[i - 1, m] \\
&\quad \text{then } x_i \leftarrow 0 \\
&\quad \text{do} \\
&\quad \quad \begin{cases} \\
&\quad \quad x_i \leftarrow 1 \\
&\quad \quad \text{else} \\
&\quad \quad \quad p \leftarrow p - p_i \\
&\quad \quad \quad m \leftarrow m - w_i \\
&\quad \end{cases} \\
&\end{align*} \]

if $p = 0$

\[ \begin{align*}
&\quad \text{then } x_1 \leftarrow 0 \\
&\quad \text{else } x_1 \leftarrow 1 \\
&\end{align*} \]

return $(X)$;
Complexity of the Algorithm

Suppose we assume the unit cost model, so additions / subtractions take time $O(1)$.

The complexity to construct the table is $\Theta(nM)$.

Is this a polynomial-time algorithm, as a function of the size of the problem instance?

We have

$$\text{size}(I) = \log_2 M + \sum_{i=1}^{n} \log_2 w_i + \sum_{i=1}^{n} \log_2 p_i.$$ 

Note in particular that $M$ is exponentially large compared to $\log_2 M$. So constructing the table is not a polynomial-time algorithm, even in the unit cost model.

What would the complexity of a recursive algorithm be?

- DP takes $\Theta(nM)$ time, which could be $\Theta(n2^n)$ for huge $M$.
- Huge $n$ is fine, but $M$ should be in $\text{poly}(n)$.
- $n$ must be very small.
- A recursive algorithm would take $\Theta(2^n)$ time.
There is a denomination with unit value!

What do you think?