

Dynamic Programming for greedy problems

Lecture 07.02
by Marina Barsky

Recap: subproblem approach to problem solving

- ❑ Technique of **breaking a problem into smaller subproblems** and using the solutions of the subproblems to construct the solution of the larger one is called ...
- ❑ When the algorithm solves the same subproblem repeatedly, **avoid recomputing values** that you already know by storing them in a table: *recursion with memoization*
- ❑ If we need the solutions to **all** subproblems, then we **compute the subproblems** starting from the simplest - **bottom-up**: *dynamic programming*

When to use Dynamic Programming

- ❑ Optimal value of the original problem can be computed easily from some subproblems.

$$\text{OPT}(w,i) = \text{max of two subproblems}$$

- ❑ There is a limited # of subproblems.

$$\{(i,j)\} \text{ for } i = 1, \dots, n \text{ and } j = 1, \dots, m$$

- ❑ There is a “natural” ordering of subproblems from smallest to largest such that you can obtain the solution for a subproblem by only looking at smaller subproblems.

Dynamic Programming: representative problems

- Edit distance
- Money change
- Knapsack 01

Making change

Recap: Change problem

Find the minimum number of coins needed to make change.



Formally:

Change problem

Input: An integer $money$ and positive integers $coin_1, \dots, coin_d$.

Output: The minimum number of coins with denominations $coin_1, \dots, coin_d$ that changes $money$.

Recap: greedy way

Algorithm *greedyChange*(*money*, *coins*)

change \leftarrow empty collection of coins

while *money* $>$ 0:

coin \leftarrow largest denomination from coins
 that does not exceed *money*

 add *coin* to *change*

money \leftarrow *money* - *coin*

return *change*

Greedy Money Change

in the US

$$40 \text{ cents} = 25 + 10 + 5$$

Greedy



1



5



10



25



50

Greedy Money Change

in Tanzania

$$40 \text{ cents} = 25 + 10 + 5 = 20 + 20$$

Greedy is not Optimal



Generalizing

Given the denominations $\text{coin}_1, \dots, \text{coin}_d$, what is the minimum number of coins needed to change n cents?

$$\text{MinNumCoins}(n) = \min \left\{ \begin{array}{l} \text{MinNumCoins}(n - \text{coin}_1) + 1 \\ \text{MinNumCoins}(n - \text{coin}_2) + 1 \\ \dots \\ \text{MinNumCoins}(n - \text{coin}_d) + 1 \end{array} \right.$$

Optimal for 1 cent

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 1 cent?

$$\text{MinNumCoins}(1) = \min \left\{ \begin{array}{l} \cancel{\text{MinNumCoins}(1 - 6) + 1} \\ \cancel{\text{MinNumCoins}(1 - 5) + 1} \\ \text{MinNumCoins}(1 - 1) + 1 \end{array} \right.$$

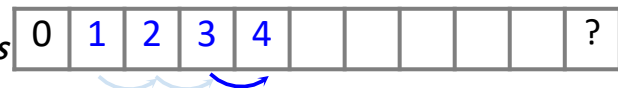
<i>money</i>		1	2	3	4	5	6	7	8	9	10
<i>MinNumCoins</i>	0	1									?

Optimal for 2,3,4 cents

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 2,3,4 cents?

$$\text{MinNumCoins}(4) = \min \left\{ \begin{array}{l} \cancel{\text{MinNumCoins}(4 - 6) + 1} \\ \cancel{\text{MinNumCoins}(4 - 5) + 1} \\ \text{MinNumCoins}(4 - 1) + 1 \end{array} \right.$$

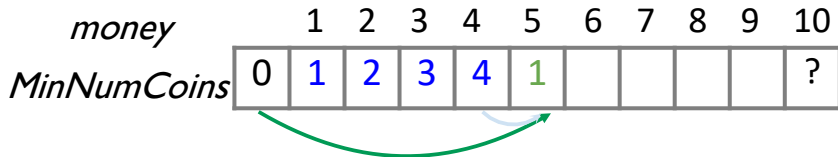
<i>money</i>		1	2	3	4	5	6	7	8	9	10
<i>MinNumCoins</i>	0	1	2	3	4						?



Optimal for 5 cents

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 5 cents?

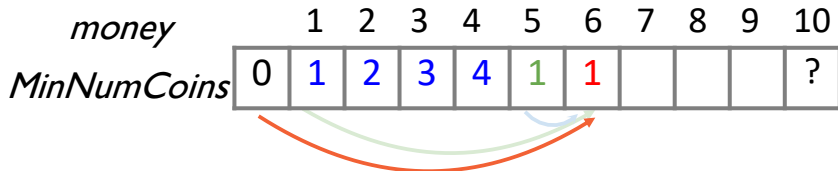
$$\text{MinNumCoins}(5) = \min \begin{cases} \text{MinNumCoins}(5 - 6) + 1 \\ \text{MinNumCoins}(5 - 5) + 1 \\ \text{MinNumCoins}(5 - 1) + 1 \end{cases}$$



Optimal for 6 cents

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 6 cents?

$$\text{MinNumCoins}(6) = \min \begin{cases} \text{MinNumCoins}(6 - 6) + 1 \\ \text{MinNumCoins}(6 - 5) + 1 \\ \text{MinNumCoins}(6 - 1) + 1 \end{cases}$$



Optimal for 7 cents

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 7 cents?

$$\text{MinNumCoins}(7) = \min \begin{cases} \text{MinNumCoins}(7 - 6) + 1 \\ \text{MinNumCoins}(7 - 5) + 1 \\ \text{MinNumCoins}(7 - 1) + 1 \end{cases}$$

<i>money</i>		1	2	3	4	5	6	7	8	9	10
<i>MinNumCoins</i>	0	1	2	3	4	1	1	2			?

Optimal for 8 cents

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 8 cents?

$$\text{MinNumCoins}(8) = \min \begin{cases} \text{MinNumCoins}(8 - 6) + 1 \\ \text{MinNumCoins}(8 - 5) + 1 \\ \text{MinNumCoins}(8 - 1) + 1 \end{cases}$$

<i>money</i>		1	2	3	4	5	6	7	8	9	10
<i>MinNumCoins</i>	0	1	2	3	4	1	1	2	3		?


The diagram shows a table with two rows. The first row is labeled 'money' and contains values from 1 to 10. The second row is labeled 'MinNumCoins' and contains values for each corresponding amount. The values are: 0 for 0, 1 for 1, 2 for 2, 3 for 3, 4 for 4, 1 for 5, 1 for 6, 2 for 7, 3 for 8, an empty cell for 9, and a question mark for 10. Three curved arrows are drawn below the table: a red arrow from the cell at (5, 1) to (8, 1), a green arrow from the cell at (3, 1) to (8, 1), and a blue arrow from the cell at (7, 1) to (8, 1).

Optimal for 9 cents

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 9 cents?

$$\text{MinNumCoins}(9) = \min \begin{cases} \text{MinNumCoins}(9 - 6) + 1 \\ \text{MinNumCoins}(9 - 5) + 1 \\ \text{MinNumCoins}(9 - 1) + 1 \end{cases}$$

<i>money</i>		1	2	3	4	5	6	7	8	9	10
<i>MinNumCoins</i>	0	1	2	3	4	1	1	2	3	4	?



Optimal for 10 cents

Given the denominations 6, 5, and 1, what is the minimum number of coins needed to change 10 cents?

$$\text{MinNumCoins}(10) = \min \begin{cases} \text{MinNumCoins}(10 - 6) + 1 \\ \text{MinNumCoins}(10 - 5) + 1 \\ \text{MinNumCoins}(10 - 1) + 1 \end{cases}$$

<i>money</i>		1	2	3	4	5	6	7	8	9	10
<i>MinNumCoins</i>	0	1	2	3	4	1	1	2	3	4	2

Algorithm DPchange(*money*, *coins*)

MinNumCoins \leftarrow array of length *money*

MinNumCoins [0] \leftarrow 0

for *m* from 1 to *money* :

MinNumCoins [*m*] \leftarrow ∞

 for *i* from 1 to *|coins|*:

 if $m \geq \text{coin}_i$:

NumCoins \leftarrow *MinNumCoins* [*m* - *coin*_{*i*}] + 1

 if *NumCoins* < *MinNumCoins* [*m*]:

MinNumCoins [*m*] \leftarrow *NumCoins*

return *MinNumCoins* [*money*]

Knapsack 01

Discrete items

Commercial placement problem

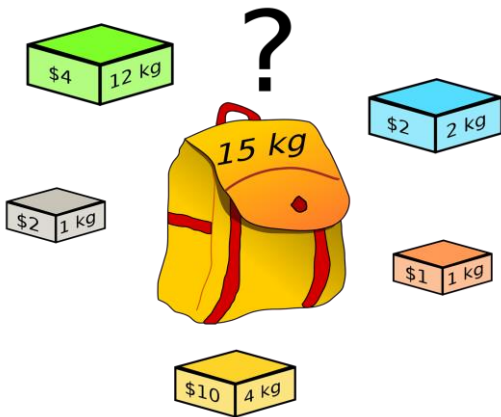
Select a set of TV commercials (each commercial has duration and cost) so that the total revenue is maximal while the total length does not exceed the length of the available time slot.

Data center performance problem

Purchase computers for a data center to achieve the maximum performance under limited budget.

Knapsack Problem

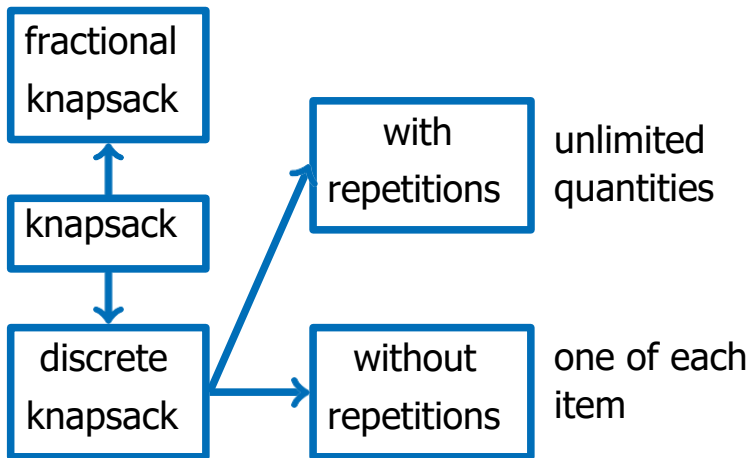
(knapsack is another word for backpack)



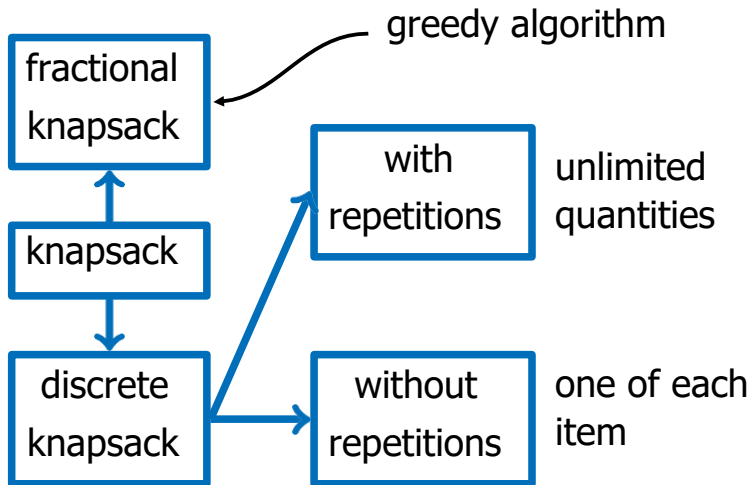
Goal

Maximize value (\$)
while limiting total weight (kg)

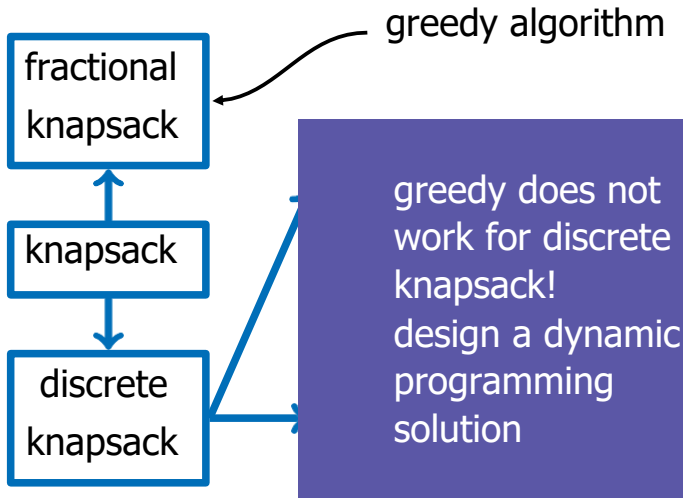
Knapsack Variations



Knapsack Variations



Knapsack Variations



Example

\$30

6

\$14

3

\$16

4

\$9

2

10

knapsack

Example

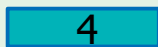
\$30



\$14



\$16



\$9



w/o repeats



total: \$46

w repeats



total: \$48

w fractions



total: \$48.5

With repetitions:
unlimited
quantities



Without
repetitions: one
of each item



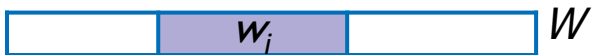
Knapsack with repetitions problem

Input: Weights w_1, \dots, w_n and values v_1, \dots, v_n of n items; total weight W of knapsack

Output: The maximum value of items whose weight does not exceed W . **Each item can be used any number of times.**

Identifying Subproblems

- Consider an optimal solution and an item in it:



- If we take this item out then we get an **optimal** solution for a knapsack of total weight $W - w_i$.

Generalizing

- Let $maxvalue(w)$ be the maximum value of knapsack of weight w .

$$maxvalue(w) = \max_{i: w_i \leq w} \{maxvalue(w - w_i) + v_i\}$$

- In other words, for each knapsack capacity w we try to fit each of the items by making sure that we have space for this item, and then selecting the max over all possible fittings
- We store this result as an optimal value for current capacity w

Example: $W = 10$

\$30

\$14

\$16

\$9

6

3

4

2

w

0

1

2

3

4

5

6

7

8

9

10

0

0

0

0

0

0

0

0

0

0

0

0

maxvalue(1)

Example: $W = 10$

\$30

\$14

\$16

\$9

6

3

4

2

w

0

1

2

3

4

5

6

7

8

9

10

0

0

9

0

0

0

0

0

0

0

0

0

maxvalue(2)

Example: $W = 10$

\$30

\$14

\$16

\$9

6

3

4

2

w

0

1

2

3

4

5

6

7

8

9

10

0

0

9

14

0

0

0

0

0

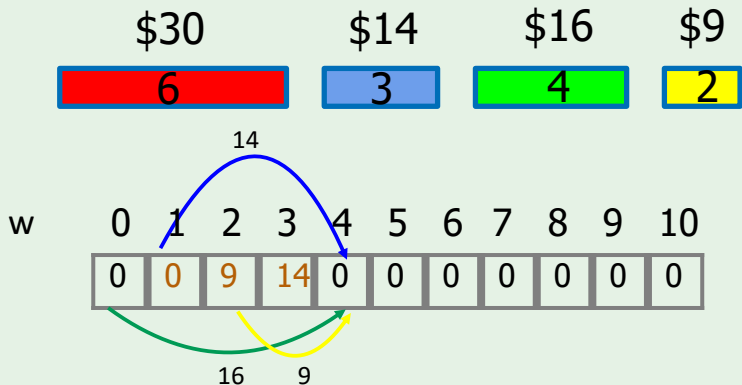
0

0

0

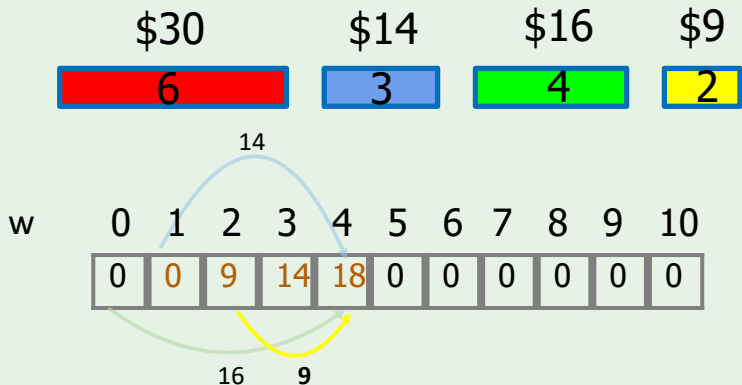
maxvalue(3)

Example: $W = 10$



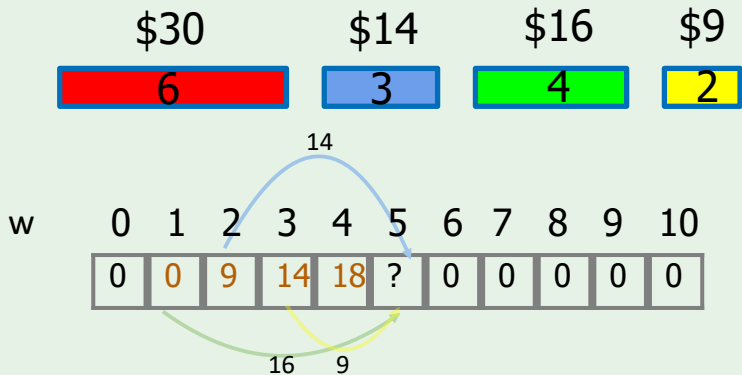
$\text{maxvalue}(4)$

Example: $W = 10$



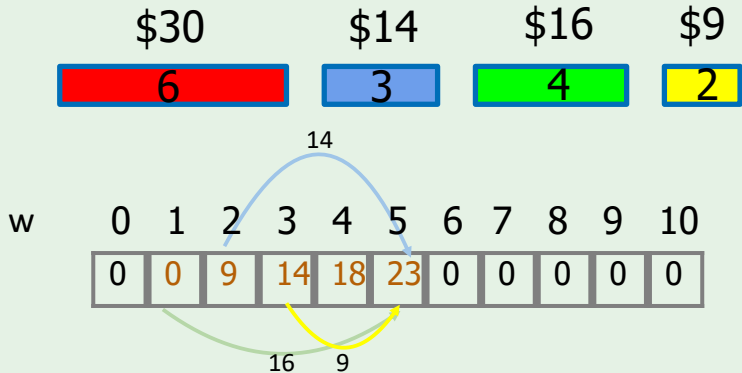
maxvalue(4)

Example: $W = 10$



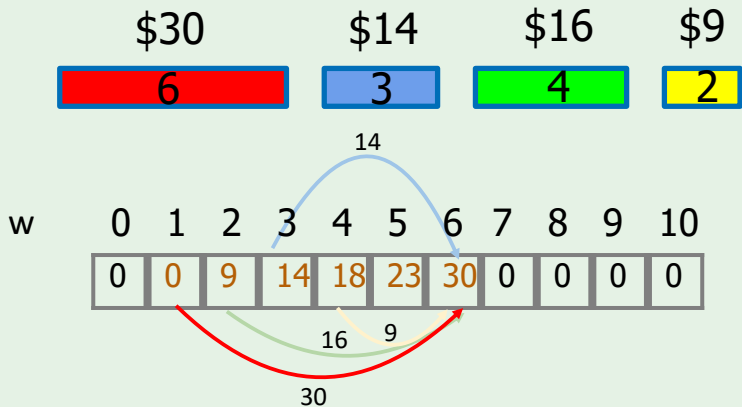
maxvalue(5)

Example: $W = 10$



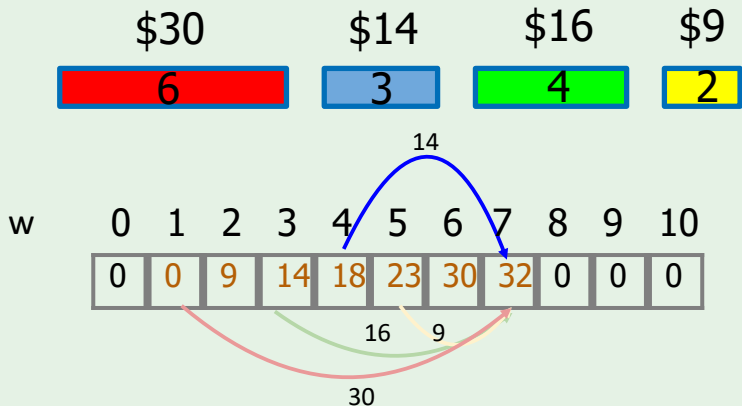
maxvalue(5)

Example: $W = 10$



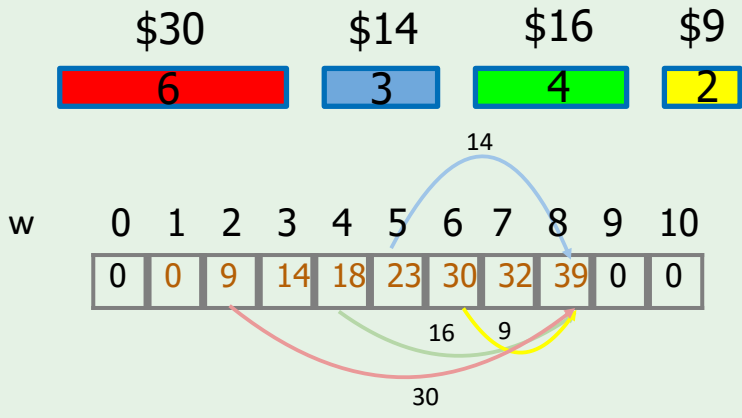
maxvalue(6)

Example: $W = 10$



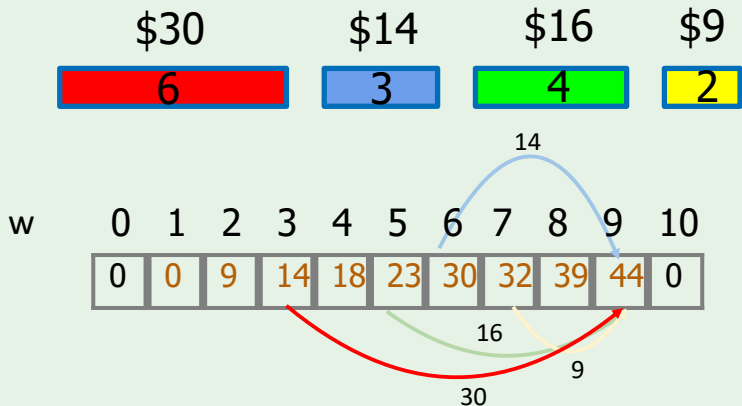
$\text{maxvalue}(7)$

Example: $W = 10$



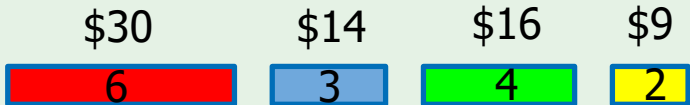
$\text{maxvalue}(8)$

Example: $W = 10$



maxvalue(9)

Example: $W = 10$



w	0	1	2	3	4	5	6	7	8	9	10
	0	0	9	14	18	23	30	32	39	44	0

$$\text{maxvalue}(10) = \max \left\{ \begin{array}{l} \text{maxvalue}(4)+30 \\ \text{maxvalue}(7)+14 \\ \text{maxvalue}(6)+16 \\ \text{maxvalue}(8)+9 \end{array} \right.$$

Example: $W = 10$

\$30

\$14

\$16

\$9

6

3

4

2

w 0 1 2 3 4 5 6 7 8 9 10

0	0	9	14	18	23	30	32	39	44	48
---	---	---	----	----	----	----	----	----	----	----

$$\text{maxvalue}(10) = \max \left\{ \begin{array}{l} 18+30=48 \\ 32+14=46 \\ 30+16=46 \\ 39+9=48 \end{array} \right.$$

Algorithm Knapsack(W , n items)

$maxvalue [0] \leftarrow 0$

for w from 1 to W :

$maxvalue [w] \leftarrow 0$

 for i from 1 to n :

 if $w_i \leq w$:

$val \leftarrow maxvalue [w - w_i] + v_i$

 if $val > maxvalue [w]$:

$maxvalue [w] \leftarrow val$

return $maxvalue [W]$

With repetitions:
unlimited
quantities



Without
repetitions:
one of each item



Knapsack without repetitions problem

Input: Weights w_1, \dots, w_n and values v_1, \dots, v_n of n items; total weight W

Output: The maximum value of items whose weight does not exceed W .

Each item can be used at most once.

Identifying Subproblems

- Consider an optimal solution and an item in it:



- If we take this item out then we get an **optimal** solution for a knapsack of total weight $W - w_i$.
- But because item i fills weight w_i in an optimal solution, **it cannot be used in an optimal solution for subproblem $W - w_i$!**

Identifying Subproblems

- In other words: if the i -th item is taken into an optimal solution:



- then what is left is an optimal solution for a knapsack of total weight $W - w_i$ **without item i**

Generalizing

For $0 \leq w \leq W$ and $0 \leq i \leq n$,
 $maxvalue(w, i)$ is the maximum value achievable
using a knapsack of weight w and items $1, \dots, i$.

The i -th item is either used or not:

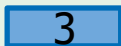
$$maxvalue(w, i) = \max \begin{cases} maxvalue(w - w_i, i-1) + v_i \\ maxvalue(w, i-1) \end{cases}$$

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0										
	0										
	0										
	0										

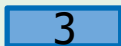
maxvalue(0, i)

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0										
	0										
	0										

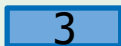
maxvalue(i, 1)

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14						
	0										
	0										

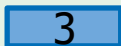
$\text{maxvalue}(3, 2), \text{maxvalue}(4, 2)$

Example: $W = 10$

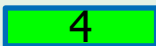
\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

0	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14						
	0										
	0										

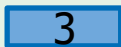
$$\text{maxvalue}(5, 2) = \max \left\{ \begin{array}{l} \text{maxvalue}(5-3, 1) + 14 \\ \text{maxvalue}(5, 1) \end{array} \right.$$

Example: $W = 10$

\$30



\$14



\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23					
	0										
	0										

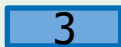
$$\text{maxvalue}(5, 2) = \max \begin{cases} 9+14 \\ 9 \end{cases}$$

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23	23	23	23	23	23
	0										
	0										

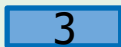
$$\text{maxvalue}(6, 2) = \max \left\{ \begin{array}{l} \text{maxvalue}(6-3, 1) + 14 \dots \\ \text{maxvalue}(6, 1) \end{array} \right.$$

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23	23	23	23	23	23
	0	0	9	14	16						
	0										

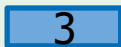
$$\text{maxvalue}(4, 3) = \max \begin{cases} \text{maxvalue}(4-4, 2) + 16 \\ \text{maxvalue}(4, 2) \end{cases}$$

Example: $W = 10$

\$30



\$14




\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23	23	23	23	23	23
	0	0	9	14	16	23					
	0										

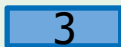
$$\text{maxvalue}(5, 3) = \max \begin{cases} \text{maxvalue}(5-4, 2) + 16 \\ \text{maxvalue}(5, 2) \end{cases}$$

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23	23	23	23	23	23
	0	0	9	14	16	23	25				
	0										

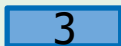
$$\text{maxvalue}(6, 3) = \max \begin{cases} \text{maxvalue}(6-4, 2) + 16 \\ \text{maxvalue}(6, 2) \end{cases}$$

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23	23	23	23	23	23
	0	0	9	14	16	23	25	30			
	0										

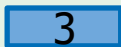
$$\text{maxvalue}(7, 3) = \max \left\{ \begin{array}{l} \text{maxvalue}(7-4, 2) + 16 \\ \text{maxvalue}(7, 2) \end{array} \right.$$

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23	23	23	23	23	23
	0	0	9	14	16	23	25	30	30		
	0										

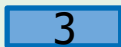
$$\text{maxvalue}(8, 3) = \max \begin{cases} \text{maxvalue}(8-4, 2) + 16 \\ \text{maxvalue}(8, 2) \end{cases}$$

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23	23	23	23	23	23
	0	0	9	14	16	23	25	30	30	39	39
	0										

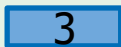
$$\text{maxvalue}(9, 3) = \max \begin{cases} \text{maxvalue}(9-4, 2) + 16 & \dots \\ \text{maxvalue}(9, 2) \end{cases}$$

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23	23	23	23	23	23
	0	0	9	14	16	23	25	30	30	39	39
	0	0	9	14	16	23	30				

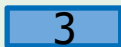
$$\text{maxvalue}(6, 4) = \max \left\{ \begin{array}{l} \text{maxvalue}(6-6, 3) + 30 \\ \text{maxvalue}(6, 3) \end{array} \right.$$

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23	23	23	23	23	23
	0	0	9	14	16	23	25	30	30	39	39
	0	0	9	14	16	23	30	30			

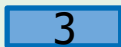
$$\text{maxvalue}(7, 4) = \max \begin{cases} \text{maxvalue}(7-6, 3) + 30 \\ \text{maxvalue}(7, 3) \end{cases}$$

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23	23	23	23	23	23
	0	0	9	14	16	23	25	30	30	39	39
	0	0	9	14	16	23	30	30	39		

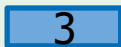
$$\text{maxvalue}(8, 4) = \max \begin{cases} \text{maxvalue}(8-6, 3) + 30 \\ \text{maxvalue}(8, 3) \end{cases}$$

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23	23	23	23	23	23
	0	0	9	14	16	23	25	30	30	39	39
	0	0	9	14	16	23	30	30	39	44	

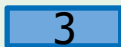
$$\text{maxvalue}(9, 4) = \max \begin{cases} \text{maxvalue}(9-6, 3) + 30 \\ \text{maxvalue}(9, 3) \end{cases}$$

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23	23	23	23	23	23
	0	0	9	14	16	23	25	30	30	39	39
	0	0	9	14	16	23	30	30	39	44	46

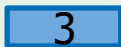
$$\text{maxvalue}(10, 4) = \max \begin{cases} \text{maxvalue}(10-6, 3) + 30 \\ \text{maxvalue}(10, 3) \end{cases}$$

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23	23	23	23	23	23
	0	0	9	14	16	23	25	30	30	39	39
	0	0	9	14	16	23	30	30	39	44	46

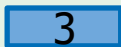
Reconstructing the items: included item 4 - **XXX1**

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23	23	23	23	23	23
	0	0	9	14	16	23	25	30	30	39	39
	0	0	9	14	16	23	30	30	39	44	46

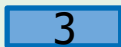
Reconstructing the items: included item 3 - XX**11**

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23	23	23	23	23	23
	0	0	9	14	16	23	25	30	30	39	39
	0	0	9	14	16	23	30	30	39	44	46

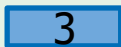
Reconstructing the items: no other items - 00**11**

Example: $W = 10$

\$30



\$14







\$16



\$9



w 0 1 2 3 4 5 6 7 8 9 10

	0	0	0	0	0	0	0	0	0	0	0
	0	0	9	9	9	9	9	9	9	9	9
	0	0	9	14	14	23	23	23	23	23	23
	0	0	9	14	16	23	25	30	30	39	39
	0	0	9	14	16	23	30	30	39	44	46

Maxvalue = 46



Knapsack(W , n items)

initialize all $maxvalue [0, i] \leftarrow 0$

initialize all $maxvalue [w, 0] \leftarrow 0$

for i from 1 to n :

 for w from 1 to W :

$maxvalue [w, i] \leftarrow maxvalue [w, i-1]$

 if $w_i \leq w$:

$val \leftarrow maxvalue [w - w_i, i - 1] + v_i$

 if $val > maxvalue [w, i]$:

$maxvalue [w, i] \leftarrow val$

return $maxvalue [W, N]$

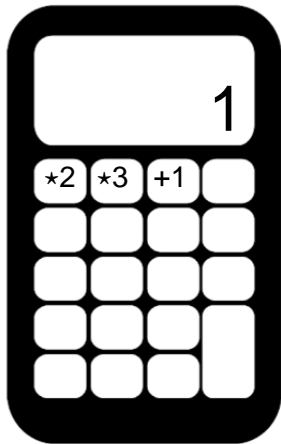
running time $O(nW)$

Calculator

Assignment 6

Problem: primitive calculator

- You are given a primitive calculator that can perform the following three operations with the current number x : multiply x by 2, multiply x by 3, or add 1 to x .
- Your goal is given a positive integer n , find the **minimum** number of operations needed to obtain the number n starting from the number 1.



Would greedy work?

- ❑ Going from 1 to n is the same as going from n to 1, each time either dividing the current number by 2 or 3 or subtracting 1 from it.
- ❑ Since we would like to go from n to 1 as fast as possible it is natural to repeatedly reduce n as much as possible.

That is, at each step we replace n by $\min\{n/3, n/2, n - 1\}$ (the terms $n/3$ and $n/2$ are used only when n is divisible by 3 and 2, respectively).

- ❑ We do this until we reach 1.

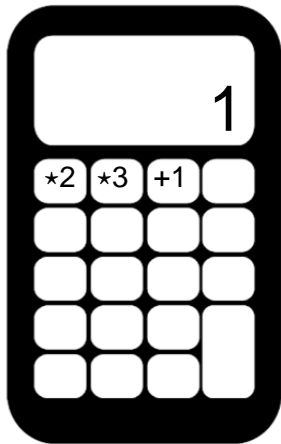
Is this algorithm correct?

Try $n=20$

[starter code](#)

Dynamic programming: subproblems

- The minimum number of operations to obtain n , can be easily determined if we know optimal solutions for $n-1$, for $n/2$, and for $n/3$.
- How can we find these optimal solutions?
- How do we go from the solutions to subproblems to the overall solution for n ?



Assignment 6. Min number of operations