Greedy Algorithms: Fractional Knapsack

Michael Levin

Higher School of Economics

Algorithmic Toolbox
Data Structures and Algorithms

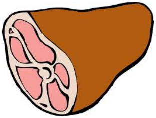
Outline

1. Long Hike
2. Fractional Knapsack
3. Pseudocode and Running Time

Long Hike

Long Hike

Long Hike



Outline

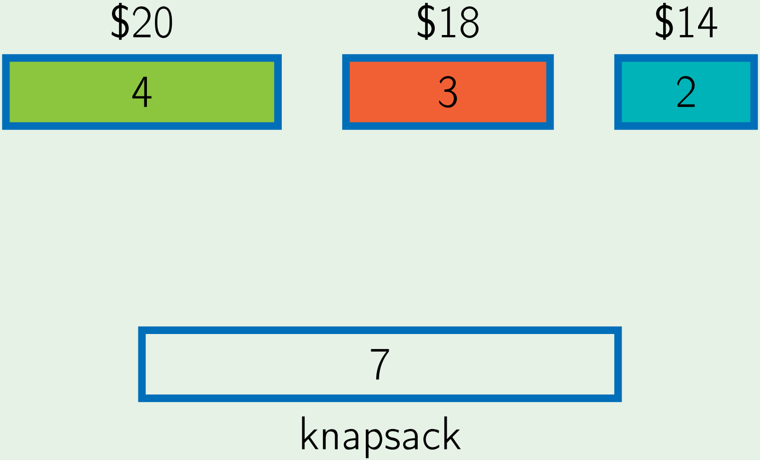
1. Long Hike
2. Fractional Knapsack
3. Pseudocode and Running Time

Fractional knapsack

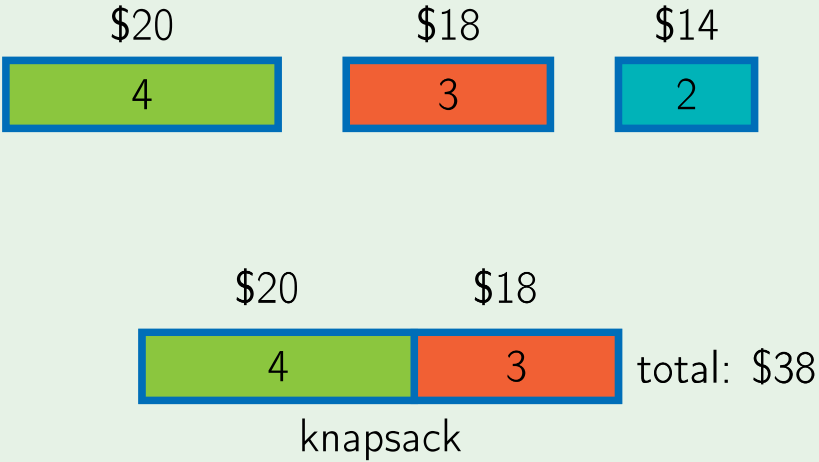
Input: Weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$ of n items; capacity W.

Output: The maximum total value of fractions of items that fit into a bag of capacity W.

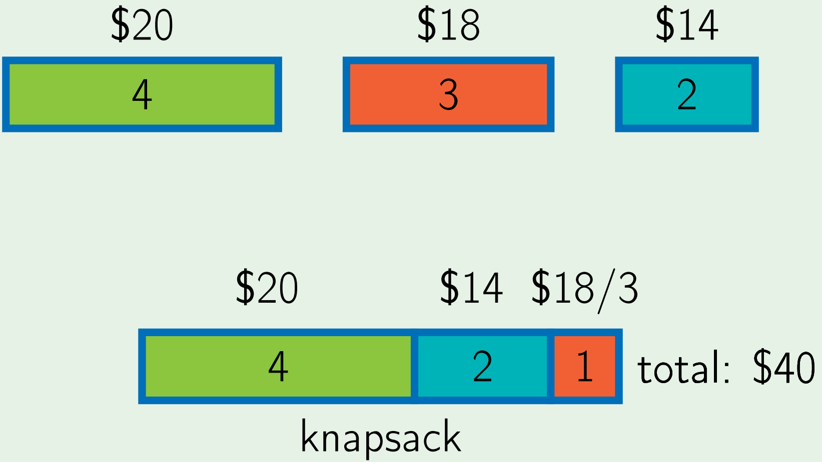
Example



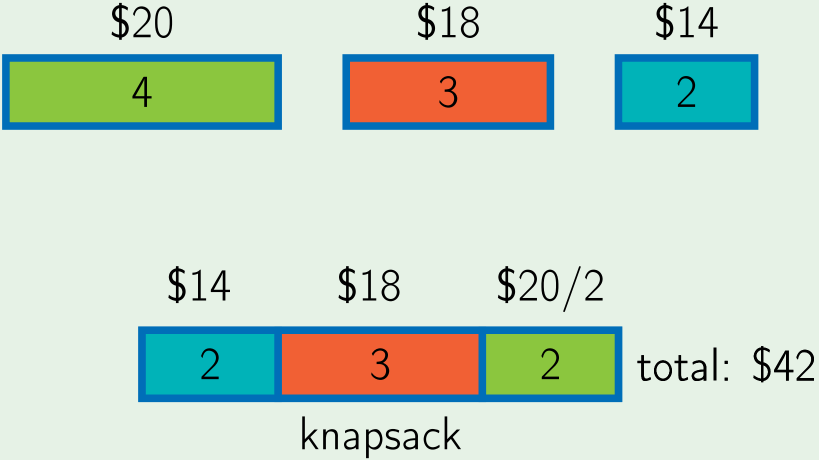
Example



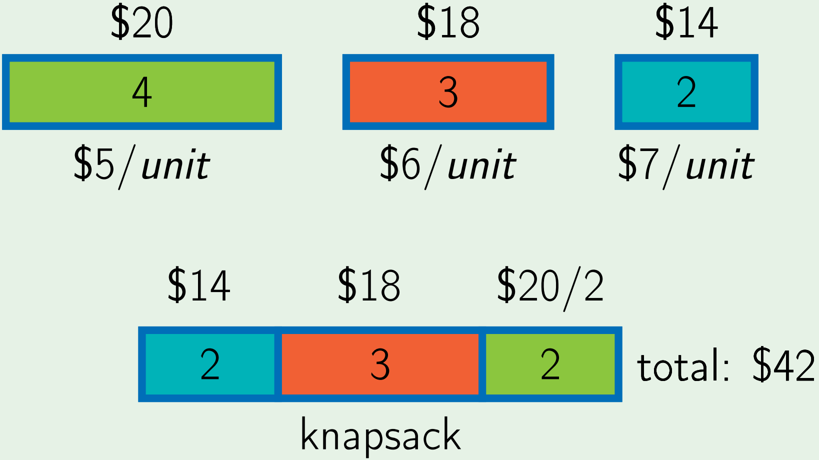
Example



Example



Example



Safe move

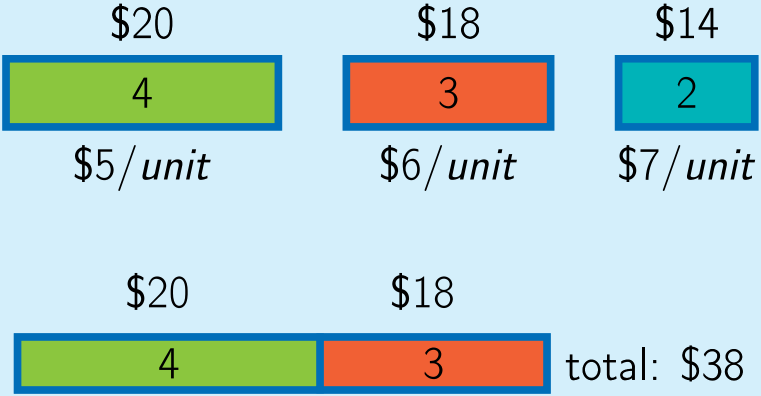
Lemma

There exists an optimal solution that uses as much as possible of an item with the maximal value per unit of weight.

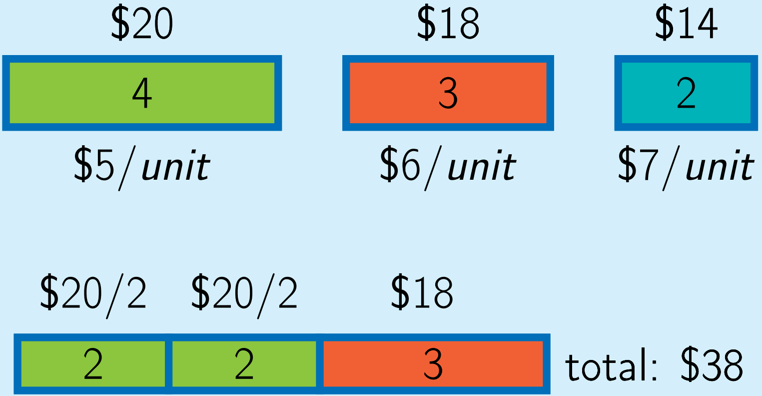
Proof

$20	$18	$14
4	3	2
$5/*unit*	$6/*unit*	$7/*unit*

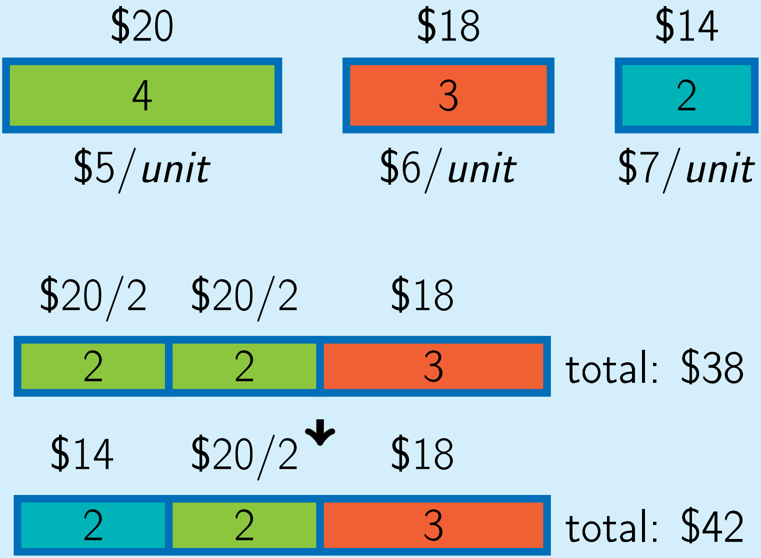
Proof



Proof



Proof


$20
4
$5/unit
$18
3
$6/unit
$14
2
$7/unit
$20/2
$20/2
$18
2
2
3
total: $38
$14
$20/2
$18
2
2
3
total: $42

Greedy Algorithm

- While knapsack is not full

Greedy Algorithm

- While knapsack is not full
- Choose item i with maximum $\frac{v_i}{w_i}$

Greedy Algorithm

- While knapsack is not full
- Choose item i with maximum $\frac{v_i}{w_i}$
- If item fits into knapsack, take all of it

Greedy Algorithm

- While knapsack is not full
- Choose item i with maximum $\frac{v_i}{w_i}$
- If item fits into knapsack, take all of it
- Otherwise take so much as to fill the knapsack

Greedy Algorithm

- While knapsack is not full
- Choose item i with maximum $\frac{v_i}{w_i}$
- If item fits into knapsack, take all of it
- Otherwise take so much as to fill the knapsack
- Return total value and amounts taken

Outline

1. Long Hike
2. Fractional Knapsack
3. Pseudocode and Running Time

Greedy Algorithm

- While knapsack is not full
- Choose item i with maximum $\frac{v_i}{w_i}$
- If item fits into knapsack, take all of it
- Otherwise take so much as to fill the knapsack
- Return total value and amounts taken

Knapsack($W, w_1, v_1, \ldots, w_n, v_n$)

```
A ← [0, 0, ..., 0], V ← 0
repeat n times:
  if W = 0:
    return (V, A)
  select i with w_i > 0 and max v_i/w_i
  a ← min(w_i, W)
  V ← V + a·v_i/w_i
  w_i ← w_i − a, A[i] ← A[i] + a, W ← W − a
return (V, A)
```

Lemma

The running time of `Knapsack` is $O(n^2)$.

Lemma

The running time of `Knapsack` is $O(n^2)$.

Proof

- Select best item on each step is $O(n)$

Lemma

The running time of `Knapsack` is $O(n^2)$.

Proof

- Select best item on each step is $O(n)$
- Main loop is executed n times

Lemma

The running time of `Knapsack` is $O(n^2)$.

Proof

- Select best item on each step is $O(n)$
- Main loop is executed n times
- Overall, $O(n^2)$ $\square$

Optimization

- It is possible to improve asymptotics!

Optimization

- It is possible to improve asymptotics!
- First, sort items by decreasing $\frac{v}{w}$

Assume $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$

```
Knapsack(W, w_1, v_1, ..., w_n, v_n)
```

$A \leftarrow [0, 0, \ldots, 0]$, $V \leftarrow 0$
for i from 1 to n:
  if $W = 0$:
    return (V, A)
  $a \leftarrow \min(w_i, W)$
  $V \leftarrow V + a\frac{v_i}{w_i}$
  $w_i \leftarrow w_i - a$, $A[i] \leftarrow A[i] + a$, $W \leftarrow W - a$
return (V, A)

Asymptotics

- Now each iteration is $O(1)$
- `Knapsack` after sorting is $O(n)$
- Sort + `Knapsack` is $O(n \log n)$