# CSCI-SHU 220: Algorithms
# Subset Sum and Knapsack

NYU Shanghai
Spring 2025

## Subset sum problems

**Problem (subset sum)**

Given a set $A = \{a_1, \ldots, a_n\}$ of integers and an integer $s$, we want to check whether there exists a subset $A' \subseteq A$ such that $s = \sum_{a \in A'} a$.

## Subset sum problems

**Problem (subset sum)**

Given a set $A = \{a_1, \ldots, a_n\}$ of integers and an integer $s$, we want to check whether there exists a subset $A' \subseteq A$ such that $s = \sum_{a \in A'} a$.

- **Some variants**
  - Finding a smallest subset $A'$ satisfying $s = \sum_{a \in A'} a$.

## Subset sum problems

**Problem (subset sum)**

Given a set $A = \{a_1, \ldots, a_n\}$ of integers and an integer $s$, we want to check whether there exists a subset $A' \subseteq A$ such that $s = \sum_{a \in A'} a$.

- **Some variants**
  - Finding a smallest subset $A'$ satisfying $s = \sum_{a \in A'} a$.
  - Finding a largest subset $A'$ satisfying $s = \sum_{a \in A'} a$.

## Subset sum problems

**Problem (subset sum)**

Given a set $A = \{a_1, \ldots, a_n\}$ of integers and an integer $s$, we want to check whether there exists a subset $A' \subseteq A$ such that $s = \sum_{a \in A'} a$.

- **Some variants**
  - Finding a smallest subset $A'$ satisfying $s = \sum_{a \in A'} a$.
  - Finding a largest subset $A'$ satisfying $s = \sum_{a \in A'} a$.
  - The variants where $A'$ can contain multiple copies of each $a_i$.
    (call it infinite subset sum for convenience)

## Subset sum problems

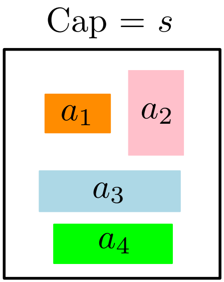
- How are these problems related to knapsack?

## Subset sum problems

- How are these problems related to knapsack?
- Suppose you have a knapsack of weight capacity $s$. There are $n$ items, where the $i$-th item has weight $a_i$.

## Subset sum problems

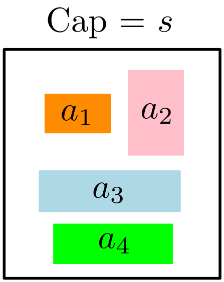
- How are these problems related to knapsack?
- Suppose you have a knapsack of weight capacity $s$. There are $n$ items, where the $i$-th item has weight $a_i$.



Can we fill the knapsack?

## Subset sum problems

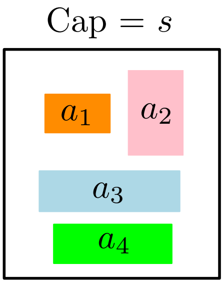
- How are these problems related to knapsack?
- Suppose you have a knapsack of weight capacity $s$. There are $n$ items, where the $i$-th item has weight $a_i$.



Fill the knapsack using fewest items?

## Subset sum problems

- How are these problems related to knapsack?
- Suppose you have a knapsack of weight capacity $s$. There are $n$ items, where the $i$-th item has weight $a_i$.



Fill the knapsack using most items?

## Knapsack problems

**Problem (0-1 knapsack)**

Given a knapsack of weight capacity $W$ and $n$ items where the $i$-th item has weight $w_i \in \mathbb{N}$ and value $v_i \in \mathbb{R}$, include in the knapsack items of total weight at most $W$ with maximum total value.

## Knapsack problems

**Problem (0-1 knapsack)**

Given a knapsack of weight capacity $W$ and $n$ items where the $i$-th item has weight $w_i \in \mathbb{N}$ and value $v_i \in \mathbb{R}$, include in the knapsack items of total weight at most $W$ with maximum total value.

- **Example**

  $W = 9$

  Item 1: $w_1 = 4, v_1 = 6$

  Item 2: $w_2 = 3, v_2 = 4$

  Item 3: $w_3 = 5, v_3 = 5$

  The optimal solution chooses Item 1 and Item 3.

## Knapsack problems

- Any relation between subset sum and 0-1 knapsack?

## Knapsack problems

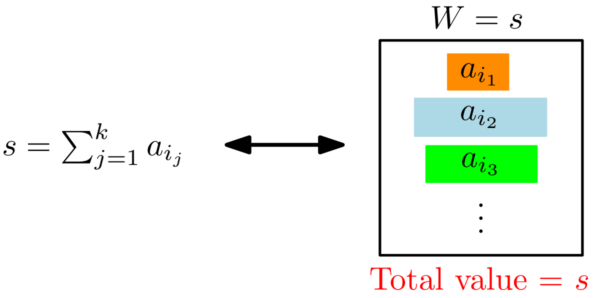
- Any relation between subset sum and 0-1 knapsack?
- Testing whether $s$ is the sum of a subset of $\{a_1, \ldots, a_n\}$

## Knapsack problems

- Any relation between subset sum and 0-1 knapsack?
- Testing whether $s$ is the sum of a subset of $\{a_1, \ldots, a_n\}$
  $W = s$, $w_i = v_i = a_i$ for all $i \in \{1, \ldots, n\}$

## Knapsack problems

- Any relation between subset sum and 0-1 knapsack?
- Testing whether $s$ is the sum of a subset of $\{a_1, \ldots, a_n\}$
  $W = s$, $w_i = v_i = a_i$ for all $i \in \{1, \ldots, n\}$



## Knapsack problems

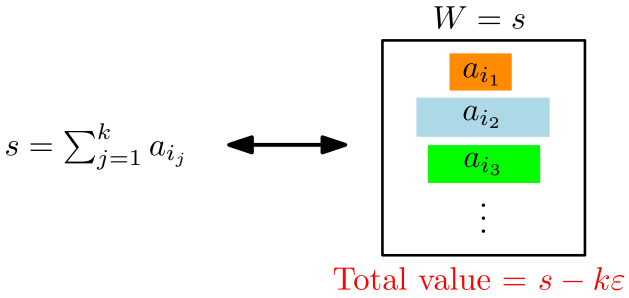
- Any relation between subset sum and 0-1 knapsack?
- Representing $s$ as the sum of a minimum subset of $\{a_1, \ldots, a_n\}$

## Knapsack problems

- Any relation between subset sum and 0-1 knapsack?
- Representing $s$ as the sum of a minimum subset of $\{a_1, \ldots, a_n\}$
  $W = s$, $w_i = a_i$ and $v_i = a_i - \varepsilon$ for all $i \in \{1, \ldots, n\}$

## Knapsack problems

- Any relation between subset sum and 0-1 knapsack?
- Representing $s$ as the sum of a minimum subset of $\{a_1, \dots, a_n\}$
  $W = s$, $w_i = a_i$ and $v_i = a_i - \varepsilon$ for all $i \in \{1, \dots, n\}$



$W = s$
$a_{i_1}$
$a_{i_2}$
$a_{i_3}$
$\vdots$
$s = \sum_{j=1}^{k} a_{i_j}$
Total value $= s - k\varepsilon$

## Knapsack problems

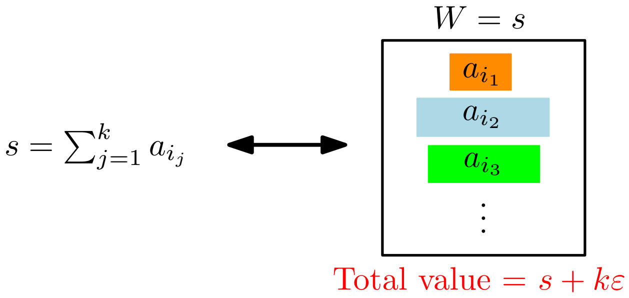
- Any relation between subset sum and 0-1 knapsack?
- Representing $s$ as the sum of a maximum subset of $\{a_1, \ldots, a_n\}$

## Knapsack problems

- Any relation between subset sum and 0-1 knapsack?
- Representing $s$ as the sum of a maximum subset of $\{a_1, \ldots, a_n\}$
  $W = s$, $w_i = a_i$ and $v_i = a_i + \varepsilon$ for all $i \in \{1, \ldots, n\}$

## Knapsack problems

- Any relation between subset sum and 0-1 knapsack?
- Representing $s$ as the sum of a maximum subset of $\{a_1, \ldots, a_n\}$
  $W = s$, $w_i = a_i$ and $v_i = a_i + \varepsilon$ for all $i \in \{1, \ldots, n\}$


$s = \sum_{j=1}^{k} a_{i_j}$
$W = s$
$a_{i_1}$
$a_{i_2}$
$a_{i_3}$
$\vdots$
Total value $= s + k\varepsilon$

## Knapsack problems

- In 0-1 knapsack each item has only one copy. What if each item has infinitely many copies?

## Knapsack problems

- In 0-1 knapsack each item has only one copy. What if each item has infinitely many copies?

**Problem (infinite knapsack)**

Given a knapsack of weight capacity $W$ and $n$ types of (infinite) items where the $i$-th type has weight $w_i \in \mathbb{N}$ and value $v_i \in \mathbb{R}$, include in the knapsack items of total weight at most $W$ with maximum total value.

## Knapsack problems

- In 0-1 knapsack each item has only one copy. What if each item has infinitely many copies?

**Problem (infinite knapsack)**

Given a knapsack of weight capacity $W$ and $n$ types of (infinite) items where the $i$-th type has weight $w_i \in \mathbb{N}$ and value $v_i \in \mathbb{R}$, include in the knapsack items of total weight at most $W$ with maximum total value.

- **Example**

  $W = 9$
  Item 1: $w_1 = 4, v_1 = 6$
  Item 2: $w_2 = 3, v_2 = 4.1$

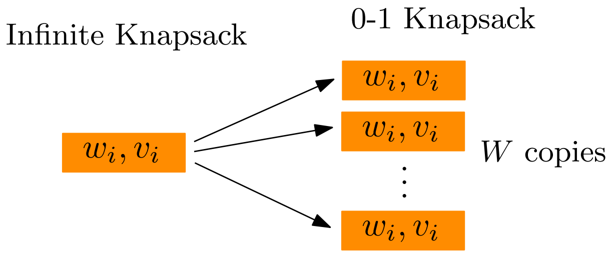
  The optimal solution chooses 3 copies of Item 2.

## Knapsack problems

- We can reduce infinite knapsack to 0-1 knapsack.

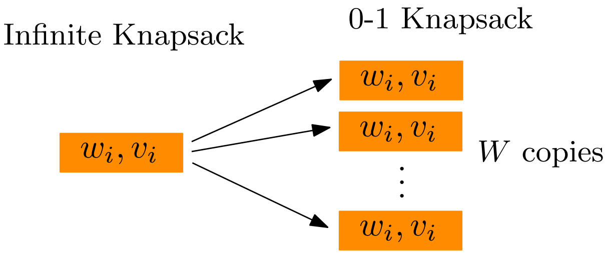
## Knapsack problems

- We can reduce infinite knapsack to 0-1 knapsack.



## Knapsack problems

- We can reduce infinite knapsack to 0-1 knapsack.



- We usually don't use this reduction, since it increases the number of items significantly from $n$ to $O(nW)$.

## Infinite knapsack

- **How to solve infinite knapsack**

## Infinite knapsack

- **How to solve infinite knapsack**
- Let's try some greedy algorithms first...

## Infinite knapsack

- **How to solve infinite knapsack**
- Let's try some greedy algorithms first...
  - Always pick the item with minimum weight.

## Infinite knapsack

- **How to solve infinite knapsack**
- Let's try some greedy algorithms first...
  - Always pick the item with minimum weight.
    $w1 = 1,\ v_1 = 0$
    $w2 = 2,\ v_2 = +\infty$

## Infinite knapsack

- **How to solve infinite knapsack**
- Let's try some greedy algorithms first...
  - Always pick the item with minimum weight.
    $w1 = 1,\ v_1 = 0$
    $w2 = 2,\ v_2 = +\infty$
  - Always pick the item with maximum value.

## Infinite knapsack

- **How to solve infinite knapsack**
- Let's try some greedy algorithms first...
  - Always pick the item with minimum weight.
    $w1 = 1,\ v_1 = 0$
    $w2 = 2,\ v_2 = +\infty$
  - Always pick the item with maximum value.
    $w1 = 1,\ v_1 = 1.9$
    $w2 = 2,\ v_2 = 2$

## Infinite knapsack

- **How to solve infinite knapsack**
- Let's try some greedy algorithms first...
  - Always pick the item with minimum weight.
    $w1 = 1,\ v_1 = 0$
    $w2 = 2,\ v_2 = +\infty$
  - Always pick the item with maximum value.
    $w1 = 1,\ v_1 = 1.9$
    $w2 = 2,\ v_2 = 2$
  - Always pick the item with maximum value/weight ratio.

## Infinite knapsack

- **How to solve infinite knapsack**
- Let's try some greedy algorithms first...
  - Always pick the item with minimum weight.
    $w1 = 1,\ v_1 = 0$
    $w2 = 2,\ v_2 = +\infty$
  - Always pick the item with maximum value.
    $w1 = 1,\ v_1 = 1.9$
    $w2 = 2,\ v_2 = 2$
  - Always pick the item with maximum value/weight ratio.
    $W = 9$
    $w_1 = 4, v_1 = 6$
    $w_2 = 3, v_2 = 4.2$

## Infinite knapsack

- **Dynamic programming**
- $P_w$ = getting the maximum value using items with total weight $w$

## Infinite knapsack

- **Dynamic programming**
- $P_w$ = getting the maximum value using items with total weight $w$
- $C(P_w) = \{j \in \{1, \ldots, n\} : w_j \leq w\}$

## Infinite knapsack

- **Dynamic programming**
- $P_w$ = getting the maximum value using items with total weight $w$
- $C(P_w) = \{j \in \{1, \ldots, n\} : w_j \leq w\}$
- $\text{opt}(P_w) = \max_{j \in C(P_i)} (\text{opt}(P_{w-w_j}) + v_j)$

## Infinite knapsack

- **Dynamic programming**
- $P_w$ = getting the maximum value using items with total weight $w$
- $C(P_w) = \{j \in \{1, \ldots, n\} : w_j \leq w\}$
- $\text{opt}(P_w) = \max_{j \in C(P_i)}(\text{opt}(P_{w-w_j}) + v_j)$
- Time complexity $= O(nW)$

## Infinite knapsack

- **Dynamic programming**
- $P_w$ = getting the maximum value using items with total weight $w$
- $C(P_w) = \{j \in \{1, \ldots, n\} : w_j \leq w\}$
- $\text{opt}(P_w) = \max_{j \in C(P_i)}(\text{opt}(P_{w-w_j}) + v_j)$
- Time complexity $= O(nW)$
- **Exercise**

  Improve the time cost to $O(nw^*)$, where $w^* = \max_{i \in \{1,\ldots,n\}} w_i^2$.

## 0-1 knapsack

- **How to solve 0-1 knapsack**

## 0-1 knapsack

- **How to solve 0-1 knapsack**
- Try the DP algorithm for infinite knapsack?

## 0-1 knapsack

- **How to solve 0-1 knapsack**
- Try the DP algorithm for infinite knapsack?
- Suppose I want to solve the subproblem $P_w$, and I pick the $j$-th item.

## 0-1 knapsack

- **How to solve 0-1 knapsack**
- Try the DP algorithm for infinite knapsack?
- Suppose I want to solve the subproblem $P_w$, and I pick the $j$-th item.
- Can we reduce the remaining problem to $P_{w-w_j}$?

## 0-1 knapsack

- **How to solve 0-1 knapsack**
- Try the DP algorithm for infinite knapsack?
- Suppose I want to solve the subproblem $P_w$, and I pick the $j$-th item.
- Can we reduce the remaining problem to $P_{w-w_j}$?
  No, because in the remaining problem we do not have the $j$-th item.

## 0-1 knapsack

- **How to solve 0-1 knapsack**
- Try the DP algorithm for infinite knapsack?
- Suppose I want to solve the subproblem $P_w$, and I pick the $j$-th item.
- Can we reduce the remaining problem to $P_{w-w_j}$?
  No, because in the remaining problem we do not have the $j$-th item.
- When doing DP, we need to remember which items are available.

## 0-1 knapsack

- **How to solve 0-1 knapsack**
- Try the DP algorithm for infinite knapsack?
- Suppose I want to solve the subproblem $P_w$, and I pick the $j$-th item.
- Can we reduce the remaining problem to $P_{w-w_j}$?
  No, because in the remaining problem we do not have the $j$-th item.
- When doing DP, we need to remember which items are available.
- **New definition of subproblems**

  $P_{w,A}$ = getting maximum value using items in $A$ with total weight $w$ for $w \in \{1, \ldots, W\}$ and $A \subseteq \{1, \ldots, n\}$.

## 0-1 knapsack

- **How to solve 0-1 knapsack**
- Try the DP algorithm for infinite knapsack?
- Suppose I want to solve the subproblem $P_w$, and I pick the $j$-th item.
- Can we reduce the remaining problem to $P_{w-w_j}$?
  No, because in the remaining problem we do not have the $j$-th item.
- When doing DP, we need to remember which items are available.
- **New definition of subproblems**

  $P_{w,A}$ = getting maximum value using items in $A$ with total weight $w$ for $w \in \{1, \dots, W\}$ and $A \subseteq \{1, \dots, n\}$.
- $\text{opt}(P_{w,A}) = \max_{j \in A}(\text{opt}(P_{w-w_j, A \setminus \{j\}}) + v_j)$.

## 0-1 knapsack

- This DP algorithm is inefficient when $n$ is large.

## 0-1 knapsack

- This DP algorithm is inefficient when $n$ is large.
- Any way to improve it?

## 0-1 knapsack

- This DP algorithm is inefficient when $n$ is large.
- Any way to improve it?
- **Key idea:**
  Consider the item in the solution with the largest index.

## 0-1 knapsack

- This DP algorithm is inefficient when $n$ is large.
- Any way to improve it?
- **Key idea:**
  Consider the item in the solution with the largest index.
- Suppose we want to solve the subproblem $P_{w,\{1,\ldots,n\}}$, i.e., achieve the maximum value with total weight $w$ when all items are available.

## 0-1 knapsack

- This DP algorithm is inefficient when $n$ is large.
- Any way to improve it?
- **Key idea:**
  Consider the item in the solution with the largest index.
- Suppose we want to solve the subproblem $P_{w,\{1,\ldots,n\}}$, i.e., achieve the maximum value with total weight $w$ when all items are available.
- What if we choose the $j$-th item as the one in our solution with the largest index? Then the remaining problem is $P_{w-w_j,\{1,\ldots,j-1\}}$.

## 0-1 knapsack

- $A_k$ = the set of items with indices $1, \ldots, k$
- $P_{w,k}$ = getting maximum value using items in $A_k$ with total weight $w$

## 0-1 knapsack

- $A_k$ = the set of items with indices $1, \ldots, k$
- $P_{w,k}$ = getting maximum value using items in $A_k$ with total weight $w$
- $\text{opt}(P_{w,k}) = \max\{\text{opt}(P_{w,k-1}), \text{opt}(P_{w-w_k,k-1}) + v_k\}$

## 0-1 knapsack

- $A_k$ = the set of items with indices $1, \ldots, k$
- $P_{w,k}$ = getting maximum value using items in $A_k$ with total weight $w$
- $\text{opt}(P_{w,k}) = \max\{\text{opt}(P_{w,k-1}), \text{opt}(P_{w-w_k,k-1}) + v_k\}$
- **Standard implementation**
- $\text{KNAPSACK}(W, w_1, \ldots, w_n, v_1, \ldots, v_n)$

```
(assume $\text{opt}[w,k] = -\infty$ for all $w$ and $k$ initially)
$\text{opt}[0,k] \leftarrow 0$ for all $k \in \{1, \ldots, n\}$
**for** $k = 1, \ldots, n$ **do**
    **for** $w = 1, \ldots, W$ **do**
        $\text{opt}[w,k] \leftarrow \max\{\text{opt}[w,k-1], \text{opt}[w-w_k, k-1] + v_k\}$
**return** $\max_{w \in \{1,\ldots,W\}} \text{opt}[w,n]$
```

## 0-1 knapsack

- **A better implementation**
- $\textsc{Knapsack}(W, w_1, \ldots, w_n, v_1, \ldots, v_n)$

```
(assume opt[$w$] $= -\infty$ for all $w$ initially)
opt[0] $\leftarrow 0$
for $k = 1, \ldots, n$ do
    for $w = W, \ldots, 1$ do
        opt[$w$] $\leftarrow \max\{\text{opt}[w], \text{opt}[w - w_k] + v_k\}$
return $\max_{i \in \{1,\ldots,W\}} \text{opt}[w]$
```

## 0-1 knapsack

- **A better implementation**
- 
```
KNAPSACK($W, w_1, \ldots, w_n, v_1, \ldots, v_n$)
  (assume opt[$w$] $= -\infty$ for all $w$ initially)
  opt[0] $\leftarrow 0$
  for $k = 1, \ldots, n$ do
      for $w = W, \ldots, 1$ do
          opt[$w$] $\leftarrow \max\{\text{opt}[w], \text{opt}[w - w_k] + v_k\}$
  return $\max_{i \in \{1,\ldots,W\}}$ opt[$w$]
```
- Why does this implementation work?

## 0-1 knapsack

- **A better implementation**
- $\textsc{Knapsack}(W, w_1, \ldots, w_n, v_1, \ldots, v_n)$

```
(assume opt[$w$] $= -\infty$ for all $w$ initially)
opt[0] $\leftarrow 0$
for $k = 1, \ldots, n$ do
    for $w = W, \ldots, 1$ do
        opt[$w$] $\leftarrow \max\{\text{opt}[w], \text{opt}[w - w_k] + v_k\}$
return $\max_{i \in \{1,\ldots,W\}} \text{opt}[w]$
```

- Why does this implementation work?
- Time complexity $= O(nW)$

## Fractional knapsack

- You can take a fraction of each item.

## Fractional knapsack

- You can take a fraction of each item.

**Problem (fractional knapsack)**

Given a knapsack of weight capacity $W$ and $n$ items where the $i$-th item has weight $w_i \in \mathbb{R}^+$ and value $v_i$, compute numbers $\rho_1, \ldots, \rho_n \in [0, 1]$ satisfying $\sum_{i=1}^{n} \rho_i w_i \leq W$ such that $\sum_{i=1}^{n} \rho_i v_i$ is maximized.

## Fractional knapsack

- You can take a fraction of each item.

**Problem (fractional knapsack)**

Given a knapsack of weight capacity $W$ and $n$ items where the $i$-th item has weight $w_i \in \mathbb{R}^+$ and value $v_i$, compute numbers $\rho_1, \ldots, \rho_n \in [0, 1]$ satisfying $\sum_{i=1}^{n} \rho_i w_i \leq W$ such that $\sum_{i=1}^{n} \rho_i v_i$ is maximized.

- **Example**

  $W = 6$
  Item 1: $w_1 = 4, v_1 = 5$
  Item 2: $w_2 = 3, v_2 = 4$
  Item 3: $w_3 = 5, v_3 = 5$

  The optimal solution is $\rho_1 = 75\%$, $\rho_2 = 100\%$, $\rho_3 = 0\%$.

## Fractional knapsack

- DP seems not work because the weights are no longer integers.

## Fractional knapsack

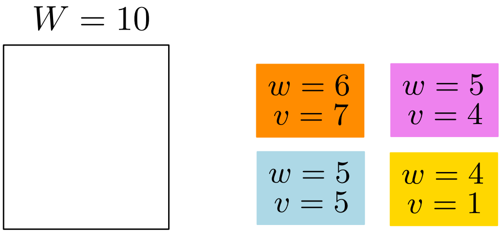
- DP seems not work because the weights are no longer integers.
- Try some greedy algorithm?

## Fractional knapsack

- DP seems not work because the weights are no longer integers.
- Try some greedy algorithm?
- Think about why the value/weight ratio greedy fails for 0-1 knapsack.

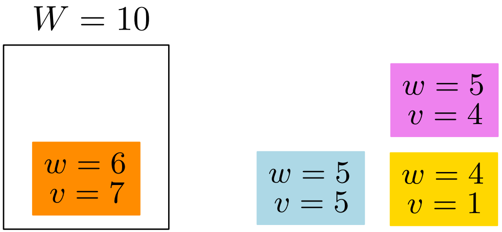
## Fractional knapsack

- DP seems not work because the weights are no longer integers.
- Try some greedy algorithm?
- Think about why the value/weight ratio greedy fails for 0-1 knapsack.



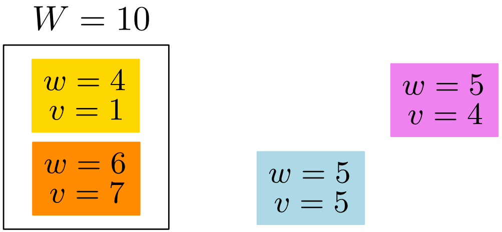
## Fractional knapsack

- DP seems not work because the weights are no longer integers.
- Try some greedy algorithm?
- Think about why the value/weight ratio greedy fails for 0-1 knapsack.



## Fractional knapsack

- DP seems not work because the weights are no longer integers.
- Try some greedy algorithm?
- Think about why the value/weight ratio greedy fails for 0-1 knapsack.



## Fractional knapsack

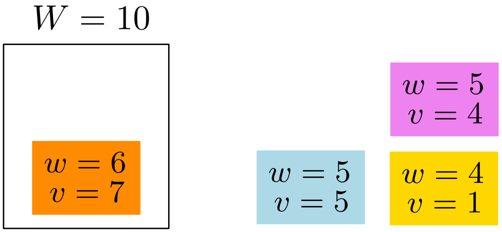
- Can this situation happen in fractional knapsack?

## Fractional knapsack

- Can this situation happen in fractional knapsack?
- Seems not, because we can "cut" the items.

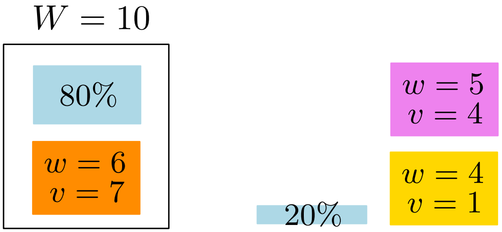
## Fractional knapsack

- Can this situation happen in fractional knapsack?
- Seems not, because we can "cut" the items.



## Fractional knapsack

- Can this situation happen in fractional knapsack?
- Seems not, because we can "cut" the items.



## Fractional knapsack

- Next let's formally describe the greedy strategy.

## Fractional knapsack

- Next let's formally describe the greedy strategy.
- Without loss of generality, suppose $\frac{v_1}{w_1} \geq \cdots \geq \frac{v_n}{w_n}$.

## Fractional knapsack

- Next let's formally describe the greedy strategy.
- Without loss of generality, suppose $\frac{v_1}{w_1} \geq \cdots \geq \frac{v_n}{w_n}$.
- Find the maximum index $k$ satisfying $\sum_{i=1}^{k} w_i \leq W$.

## Fractional knapsack

- Next let's formally describe the greedy strategy.
- Without loss of generality, suppose $\frac{v_1}{w_1} \geq \cdots \geq \frac{v_n}{w_n}$.
- Find the maximum index $k$ satisfying $\sum_{i=1}^{k} w_i \leq W$.
- $\rho_i = 1$ for all $i \in \{1, \ldots, k\}$
  $\rho_{k+1} = (W - \sum_{i=1}^{k} w_i)/w_{k+1}$
  $\rho_i = 0$ for all $i \in \{k+2, \ldots, n\}$

## Fractional knapsack

- Next let's formally describe the greedy strategy.
- Without loss of generality, suppose $\frac{v_1}{w_1} \geq \cdots \geq \frac{v_n}{w_n}$.
- Find the maximum index $k$ satisfying $\sum_{i=1}^{k} w_i \leq W$.
- $\rho_i = 1$ for all $i \in \{1, \ldots, k\}$
  $\rho_{k+1} = (W - \sum_{i=1}^{k} w_i)/w_{k+1}$
  $\rho_i = 0$ for all $i \in \{k+2, \ldots, n\}$
- **Proof of correctness**
  Assume there exists an optimal solution whose $\rho_i$ is the same as the greedy solution for all $i < t$, and show the existence of an optimal solution whose $\rho_i$ is the same as the greedy solution for all $i \leq t$.

## Another problem related to subset sum

**Problem (Fibonacci sum)**

Given a positive integer $s$...

1. Check whether $s$ is the sum of several distinct Fibonacci numbers.
2. Find fewest distinct Fibonacci numbers whose sum is $s$.
3. Find fewest Fibonacci numbers whose sum is $s$.

## Another problem related to subset sum

**Problem (Fibonacci sum)**

Given a positive integer $s$...

1. Check whether $s$ is the sum of several distinct Fibonacci numbers.
2. Find fewest distinct Fibonacci numbers whose sum is $s$.
3. Find fewest Fibonacci numbers whose sum is $s$.

- **Example**

  $s = 31$

  1. Yes
  2. $s = 2 + 8 + 21$
  3. $s = 5 + 13 + 13$

## Another problem related to subset sum

- **Question 1**

## Another problem related to subset sum

- **Question 1**
- Algorithm: Always output Yes!

## Another problem related to subset sum

- **Question 1**
- Algorithm: Always output Yes!

**Theorem**

*Every positive integer $s$ is the sum of several distinct Fibonacci numbers.*

## Another problem related to subset sum

- **Question 1**
- Algorithm: Always output Yes!

**Theorem**

*Every positive integer $s$ is the sum of several distinct Fibonacci numbers.*

- **Proof.** Induction on $s$

## Another problem related to subset sum

- **Question 1**
- Algorithm: Always output Yes!

**Theorem**

*Every positive integer $s$ is the sum of several distinct Fibonacci numbers.*

- **Proof.** Induction on $s$
  - **Base case:** $s = 1$. Trivial.

## Another problem related to subset sum

- **Question 1**
- Algorithm: Always output Yes!

**Theorem**

*Every positive integer s is the sum of several distinct Fibonacci numbers.*

- **Proof.** Induction on $s$
  - **Base case:** $s = 1$. Trivial.
  - **Induction hypothesis:** Suppose the statement holds for $1, \ldots, s-1$.

## Another problem related to subset sum

- **Question 1**
- Algorithm: Always output Yes!

**Theorem**

*Every positive integer $s$ is the sum of several distinct Fibonacci numbers.*

- **Proof.** Induction on $s$
  - **Base case:** $s = 1$. Trivial.
  - **Induction hypothesis:** Suppose the statement holds for $1, \ldots, s-1$.
  - Represent $s$ as the sum of distinct Fibonacci numbers?

## Another problem related to subset sum

- **Question 1**
- Algorithm: Always output Yes!

**Theorem**

*Every positive integer $s$ is the sum of several distinct Fibonacci numbers.*

- **Proof.** Induction on $s$
  - **Base case:** $s = 1$. Trivial.
  - **Induction hypothesis:** Suppose the statement holds for $1, \ldots, s-1$.
  - Represent $s$ as the sum of distinct Fibonacci numbers?
    $i =$ the largest index such that $F_i \leq s$
    $\implies s = F_i + s'$ where $s' < F_i$
    By our hypothesis, $s' = F_{j_1} + \cdots + F_{j_r}$ where $j_1, \ldots, j_r$ are distinct.
    $\implies s = F_i + F_{j_1} + \cdots + F_{j_r}$ and $i \notin \{j_1, \ldots, j_r\}$ as $s' < F_i$

## Another problem related to subset sum

- **Question 2**

## Another problem related to subset sum

- **Question 2**
- **Trivial algorithm**
  1. Reduce to the minimization version of subset sum.
     $A$ = set of Fibonacci numbers smaller than or equal to $s$
  2. Further reduce to the 0-1 knapsack problem.
  3. Solve the the 0-1 knapsack instance.

## Another problem related to subset sum

- **Question 2**
- **Trivial algorithm**
  1. Reduce to the minimization version of subset sum.
     $A =$ set of Fibonacci numbers smaller than or equal to $s$
  2. Further reduce to the 0-1 knapsack problem.
  3. Solve the the 0-1 knapsack instance.
- Time complexity?

## Another problem related to subset sum

- **Question 2**
- **Trivial algorithm**
  1. Reduce to the minimization version of subset sum.
     $A =$ set of Fibonacci numbers smaller than or equal to $s$
  2. Further reduce to the 0-1 knapsack problem.
  3. Solve the the 0-1 knapsack instance.
- Time complexity? $O(s \log s)$

## Another problem related to subset sum

- **Question 2**
- **Trivial algorithm**
  1. Reduce to the minimization version of subset sum.
     $A$ = set of Fibonacci numbers smaller than or equal to $s$
  2. Further reduce to the 0-1 knapsack problem.
  3. Solve the the 0-1 knapsack instance.
- Time complexity? $O(s \log s)$
- Can we do better?

## Another problem related to subset sum

- **Question 2**
- **Trivial algorithm**
  1. Reduce to the minimization version of subset sum.
     $A$ = set of Fibonacci numbers smaller than or equal to $s$
  2. Further reduce to the 0-1 knapsack problem.
  3. Solve the the 0-1 knapsack instance.
- Time complexity? $O(s \log s)$
- Can we do better?
- The answer to Question 1 gives us a natural greedy algorithm: Always take the largest Fibonacci number one can take.

## Another problem related to subset sum

- **Question 2**
- **Trivial algorithm**
  1. Reduce to the minimization version of subset sum.
     $A$ = set of Fibonacci numbers smaller than or equal to $s$
  2. Further reduce to the 0-1 knapsack problem.
  3. Solve the the 0-1 knapsack instance.
- Time complexity? $O(s \log s)$
- Can we do better?
- The answer to Question 1 gives us a natural greedy algorithm: Always take the largest Fibonacci number one can take.
- Time complexity for implementing this greedy algorithm?

## Another problem related to subset sum

- **Question 2**
- **Trivial algorithm**
  1. Reduce to the minimization version of subset sum.
     $A$ = set of Fibonacci numbers smaller than or equal to $s$
  2. Further reduce to the 0-1 knapsack problem.
  3. Solve the the 0-1 knapsack instance.
- Time complexity? $O(s \log s)$
- Can we do better?
- The answer to Question 1 gives us a natural greedy algorithm: Always take the largest Fibonacci number one can take.
- Time complexity for implementing this greedy algorithm? $O(\log s)$

## Another problem related to subset sum

- Our proof for Question 1 shows that this greedy algorithm always gives us a feasible solution.

## Another problem related to subset sum

- Our proof for Question 1 shows that this greedy algorithm always gives us a feasible solution.
- But does it always give you an optimal solution?

## Another problem related to subset sum

- Our proof for Question 1 shows that this greedy algorithm always gives us a feasible solution.
- But does it always give you an optimal solution?
  Yes, but this is not obvious...

## Another problem related to subset sum

- Our proof for Question 1 shows that this greedy algorithm always gives us a feasible solution.
- But does it always give you an optimal solution?
  Yes, but this is not obvious...
- **Key observations**
  1. An optimal solution does not contain adjacent Fibonacci numbers.

## Another problem related to subset sum

- Our proof for Question 1 shows that this greedy algorithm always gives us a feasible solution.
- But does it always give you an optimal solution?
  Yes, but this is not obvious...
- **Key observations**
  1. An optimal solution does not contain adjacent Fibonacci numbers.
  2. Let $S$ be a set of non-adjacent Fibonacci numbers in which the largest one is $F_i$. Then $F_{i+1} \geq \text{sum}(S)$.

## Another problem related to subset sum

- Our proof for Question 1 shows that this greedy algorithm always gives us a feasible solution.
- But does it always give you an optimal solution?
  Yes, but this is not obvious...
- **Key observations**
  1. An optimal solution does not contain adjacent Fibonacci numbers.
  2. Let $S$ be a set of non-adjacent Fibonacci numbers in which the largest one is $F_i$. Then $F_{i+1} \geq \text{sum}(S)$.
- The first observation can be proved by contradiction, and the second one can be proved by induction on $i$.

## Another problem related to subset sum

- If $s = F_i$ for some $i$, we are done.
- $i$ = the largest index such that $F_i < s$

## Another problem related to subset sum

- If $s = F_i$ for some $i$, we are done.
- $i =$ the largest index such that $F_i < s$
- Using the two key observations, we show that any optimal solution must contain the Fibonacci number $F_i$.

## Another problem related to subset sum

- If $s = F_i$ for some $i$, we are done.
- $i =$ the largest index such that $F_i < s$
- Using the two key observations, we show that any optimal solution must contain the Fibonacci number $F_i$.
- $S =$ an optimal solution
  $F_j =$ the largest number in $S$

## Another problem related to subset sum

- If $s = F_i$ for some $i$, we are done.
- $i =$ the largest index such that $F_i < s$
- Using the two key observations, we show that any optimal solution must contain the Fibonacci number $F_i$.
- $S =$ an optimal solution
  $F_j =$ the largest number in $S$
- We have $j \leq i$, for otherwise $F_j > s$ and thus $\text{sum}(S) > s$.

## Another problem related to subset sum

- If $s = F_i$ for some $i$, we are done.
- $i =$ the largest index such that $F_i < s$
- Using the two key observations, we show that any optimal solution must contain the Fibonacci number $F_i$.
- $S =$ an optimal solution
  $F_j =$ the largest number in $S$
- We have $j \leq i$, for otherwise $F_j > s$ and thus $\text{sum}(S) > s$.
- On the other hand, we have $j \geq i$, why?

## Another problem related to subset sum

- If $s = F_i$ for some $i$, we are done.
- $i =$ the largest index such that $F_i < s$
- Using the two key observations, we show that any optimal solution must contain the Fibonacci number $F_i$.
- $S =$ an optimal solution
  $F_j =$ the largest number in $S$
- We have $j \leq i$, for otherwise $F_j > s$ and thus $\text{sum}(S) > s$.
- On the other hand, we have $j \geq i$, why?
  - Observation 1 shows $S$ doesn't contain adjacent Fibonacci numbers.

## Another problem related to subset sum

- If $s = F_i$ for some $i$, we are done.
- $i =$ the largest index such that $F_i < s$
- Using the two key observations, we show that any optimal solution must contain the Fibonacci number $F_i$.
- $S =$ an optimal solution
  $F_j =$ the largest number in $S$
- We have $j \leq i$, for otherwise $F_j > s$ and thus $\mathrm{sum}(S) > s$.
- On the other hand, we have $j \geq i$, why?
  - Observation 1 shows $S$ doesn't contain adjacent Fibonacci numbers.
  - Observation 2 further shows that if $j < i$, then $s > F_i \geq \mathrm{sum}(S)$.

## Another problem related to subset sum

- If $s = F_i$ for some $i$, we are done.
- $i =$ the largest index such that $F_i < s$
- Using the two key observations, we show that any optimal solution must contain the Fibonacci number $F_i$.
- $S =$ an optimal solution
  $F_j =$ the largest number in $S$
- We have $j \leq i$, for otherwise $F_j > s$ and thus $\text{sum}(S) > s$.
- On the other hand, we have $j \geq i$, why?
  - Observation 1 shows $S$ doesn't contain adjacent Fibonacci numbers.
  - Observation 2 further shows that if $j < i$, then $s > F_i \geq \text{sum}(S)$.
- Thus, we have $i = j$. Now try to further prove the optimality of the greedy solution by induction.

## Another problem related to subset sum

- **Question 3**
- Now each Fibonacci number can be used multiple times...

## Another problem related to subset sum

- **Question 3**
- Now each Fibonacci number can be used multiple times...
- Let's consider the same greedy algorithm as before.

## Another problem related to subset sum

- **Question 3**
- Now each Fibonacci number can be used multiple times...
- Let's consider the same greedy algorithm as before.
- At a first glance, it shouldn't be correct, as it can only find solutions with distinct Fibonacci numbers.

## Another problem related to subset sum

- **Question 3**
- Now each Fibonacci number can be used multiple times...
- Let's consider the same greedy algorithm as before.
- At a first glance, it shouldn't be correct, as it can only find solutions with distinct Fibonacci numbers.
- Try some numbers in order to find a counterexample:

## Another problem related to subset sum

- **Question 3**
- Now each Fibonacci number can be used multiple times...
- Let's consider the same greedy algorithm as before.
- At a first glance, it shouldn't be correct, as it can only find solutions with distinct Fibonacci numbers.
- Try some numbers in order to find a counterexample:
  - $68 = 34 + 34$, $165 = 55 + 55 + 55$, $84 = 21 + 21 + 21 + 21$

## Another problem related to subset sum

- **Question 3**
- Now each Fibonacci number can be used multiple times...
- Let's consider the same greedy algorithm as before.
- At a first glance, it shouldn't be correct, as it can only find solutions with distinct Fibonacci numbers.
- Try some numbers in order to find a counterexample:
  - $68 = 34 + 34$, $165 = 55 + 55 + 55$, $84 = 21 + 21 + 21 + 21$
  - $68 = 55 + 13$, $165 = 144 + 21$, $84 = 55 + 21 + 8$

## Another problem related to subset sum

- **Question 3**
- Now each Fibonacci number can be used multiple times...
- Let's consider the same greedy algorithm as before.
- At a first glance, it shouldn't be correct, as it can only find solutions with distinct Fibonacci numbers.
- Try some numbers in order to find a counterexample:
  - $68 = 34 + 34$, $165 = 55 + 55 + 55$, $84 = 21 + 21 + 21 + 21$
  - $68 = 55 + 13$, $165 = 144 + 21$, $84 = 55 + 21 + 8$
- Now we should suspect that the greedy algorithm might be correct.

## Another problem related to subset sum

- The greedy algorithm is correct.

  $\Updownarrow$

  There's always an optimal solution with distinct numbers.

## Another problem related to subset sum

- The greedy algorithm is correct.

  $$\Updownarrow$$

  There's always an optimal solution with distinct numbers.

- Consider a simple case: $s = 2F_i$

## Another problem related to subset sum

- The greedy algorithm is correct.

  $$\Updownarrow$$

  There's always an optimal solution with distinct numbers.

- Consider a simple case: $s = 2F_i$

- $F_{i+1} = F_i + F_{i-1} = 2F_i - F_{i-2}$
  $\Longrightarrow 2F_i = F_{i+1} + F_{i-2}$
  $\Longrightarrow s = F_{i+1} + F_{i-2}$

## Another problem related to subset sum

- The greedy algorithm is correct.

  $\Updownarrow$

  There's always an optimal solution with distinct numbers.

- Consider a simple case: $s = 2F_i$

- $F_{i+1} = F_i + F_{i-1} = 2F_i - F_{i-2}$
  $\Longrightarrow 2F_i = F_{i+1} + F_{i-2}$
  $\Longrightarrow s = F_{i+1} + F_{i-2}$

- General case?

## Another problem related to subset sum

- The greedy algorithm is correct.

  $\Updownarrow$

  There's always an optimal solution with distinct numbers.
- Consider a simple case: $s = 2F_i$
- $F_{i+1} = F_i + F_{i-1} = 2F_i - F_{i-2}$
  $\implies 2F_i = F_{i+1} + F_{i-2}$
  $\implies s = F_{i+1} + F_{i-2}$
- General case?
  We can keep modifying $S$ using the equality $2F_i = F_{i+1} + F_{i-2}$.

## Another problem related to subset sum

- $S$ = an optimal solution

## Another problem related to subset sum

- $S$ = an optimal solution
- Keep doing the following until the numbers in $S$ are distinct:
  - Find $i \in \mathbb{N}$ such that $S$ contains two (or more) $F_i$'s.
  - Replace two $F_i$'s with $F_{i+1}$ and $F_{i-2}$.

## Another problem related to subset sum

- $S$ = an optimal solution
- Keep doing the following until the numbers in $S$ are distinct:
  - Find $i \in \mathbb{N}$ such that $S$ contains two (or more) $F_i$'s.
  - Replace two $F_i$'s with $F_{i+1}$ and $F_{i-2}$.
- We never change $|S|$ and $\text{sum}(S)$. So at the end, $S$ is still an optimal solution and must consist of distinct numbers.
  $\Longrightarrow$ There's always an optimal solution with distinct numbers.

## Another problem related to subset sum

- $S$ = an optimal solution
- Keep doing the following until the numbers in $S$ are distinct:
  - Find $i \in \mathbb{N}$ such that $S$ contains two (or more) $F_i$'s.
  - Replace two $F_i$'s with $F_{i+1}$ and $F_{i-2}$.
- We never change $|S|$ and $\text{sum}(S)$. So at the end, $S$ is still an optimal solution and must consist of distinct numbers.
  $\Longrightarrow$ There's always an optimal solution with distinct numbers.
- Any issue in the above argument?

## Another problem related to subset sum

- $S$ = an optimal solution
- Keep doing the following until the numbers in $S$ are distinct:
  - Find $i \in \mathbb{N}$ such that $S$ contains two (or more) $F_i$'s.
  - Replace two $F_i$'s with $F_{i+1}$ and $F_{i-2}$.
- We never change $|S|$ and $\text{sum}(S)$. So at the end, $S$ is still an optimal solution and must consist of distinct numbers.
  $\Longrightarrow$ There's always an optimal solution with distinct numbers.
- Any issue in the above argument?
- It's not clear whether the modification of $S$ will terminate or not.

## Another problem related to subset sum

- **Key idea:** Define a potential function on $S$ which keeps increasing (or decreasing) during the modification.

## Another problem related to subset sum

- **Key idea:** Define a potential function on $S$ which keeps increasing (or decreasing) during the modification.
- Observe that $F_i^2 + F_i^2 < F_{i+1}^2 + F_{i-2}^2$.

## Another problem related to subset sum

- **Key idea:** Define a potential function on $S$ which keeps increasing (or decreasing) during the modification.
- Observe that $F_i^2 + F_i^2 < F_{i+1}^2 + F_{i-2}^2$.
- So the square-sum function $f(S) = \sum_{x \in S} x^2$ is a perfect choice.

## Another problem related to subset sum

- **Key idea:** Define a potential function on $S$ which keeps increasing (or decreasing) during the modification.
- Observe that $F_i^2 + F_i^2 < F_{i+1}^2 + F_{i-2}^2$.
- So the square-sum function $f(S) = \sum_{x \in S} x^2$ is a perfect choice.
  - $f(S)$ always increases when we modify $S$.

## Another problem related to subset sum

- **Key idea:** Define a potential function on $S$ which keeps increasing (or decreasing) during the modification.
- Observe that $F_i^2 + F_i^2 < F_{i+1}^2 + F_{i-2}^2$.
- So the square-sum function $f(S) = \sum_{x \in S} x^2$ is a perfect choice.
    - $f(S)$ always increases when we modify $S$.
    - $f(S) \leq s^2$ for any solution $S$.

## Another problem related to subset sum

- **Key idea:** Define a potential function on $S$ which keeps increasing (or decreasing) during the modification.
- Observe that $F_i^2 + F_i^2 < F_{i+1}^2 + F_{i-2}^2$.
- So the square-sum function $f(S) = \sum_{x \in S} x^2$ is a perfect choice.
    - $f(S)$ always increases when we modify $S$.
    - $f(S) \leq s^2$ for any solution $S$.

$\Longrightarrow$ The modification procedure can have at most $s^2$ rounds.

## Another problem related to subset sum

- **Key idea:** Define a potential function on $S$ which keeps increasing (or decreasing) during the modification.
- Observe that $F_i^2 + F_i^2 < F_{i+1}^2 + F_{i-2}^2$.
- So the square-sum function $f(S) = \sum_{x \in S} x^2$ is a perfect choice.
  - $f(S)$ always increases when we modify $S$.
  - $f(S) \leq s^2$ for any solution $S$.

$\Longrightarrow$ The modification procedure can have at most $s^2$ rounds.

- Now we see there's always an optimal solution with distinct numbers.