Castillo, Anjelica M.
MCSCC 123-MSCS12S1 - Advanced Data Structures and Algorithms

Assessment ask

The problem is, your crates can't weigh more than 1000 kilograms, and that's a hard limit. To make things harder, you can choose only from a limited set of things, already packed in boxes:

- Potatoes, 800 kgs
- Rice, 300 kgs
- Wheat flour, 400 kgs
- Peanut butter, 20 kgs
- Tomato cans, 300 kgs
- Beans, 300 kgs
- Strawberry jam, 50 kgs

Table 1.1 A recap of the available goods, with their weight and calories

Food	Weight (kgs)	Total calories
Potatoes	800	1,501,600
Wheat flour	400	1,444,000
Rice	300	1,122,000
Beans (can)	300	690,000
Tomatoes (can)	300	237,000
Strawberry jam	50	130,000
Peanut butter	20	117,800

Problem:

- We have **n** items, each with a given **weight** and **caloric value**.
- We can **either take or leave** an item (no fractions allowed).
- The total weight cannot exceed **1000 kg**.
- The goal is to **maximize the total calories** while staying within the weight limit.

I first experimented with a trial-and-error approach, similar to a brute force method, to see how different combinations of items and weights would affect the total calorie count while staying within the weight limit. Initially, I attempted to include as many high-calorie items as possible without considering an optimized strategy. However, this method was inefficient and time-consuming, as it required manually adjusting the weights and recalculating the total calories multiple times.

Castillo, Anjelica M.
MCSCC 123-MSCS12S1 - Advanced Data Structures and Algorithms

```python
import itertools
import numpy as np
import pandas as pd

define_items = [
    {"name": "Potatoes", "weight": 800, "calories": 1501600},
    {"name": "Wheat flour", "weight": 400, "calories": 1444000},
    {"name": "Rice", "weight": 300, "calories": 1122000},
    {"name": "Beans (can)", "weight": 300, "calories": 690000},
    {"name": "Tomatoes (can)", "weight": 300, "calories": 237000},
    {"name": "Strawberry jam", "weight": 50, "calories": 130000},
    {"name": "Peanut butter", "weight": 20, "calories": 117800}
]
MAX_WEIGHT = 1000 #The max_weight allowed
```

✓ 0.6s

Afterward, I explored structured approaches like dynamic programming and the fractional knapsack method, which provided optimal or near-optimal solutions. While these methods improved efficiency and maximized calories, they had a drawback—they sometimes removed entire items to achieve the best outcome. I wanted a solution that retained all items while still maximizing calorie intake.

```python
def dynamic_knapsack(items, max_weight):
    """Solves the problem using dynamic programming."""
    n = len(items)
    dp = np.zeros((n + 1, max_weight + 1))
    for i in range(1, n + 1):
        for w in range(max_weight + 1):
            if items[i - 1][1] <= w:
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - items[i - 1][1]] + items[i - 1][2])
            else:
                dp[i][w] = dp[i - 1][w]
    return dp[n][max_weight]
```

✓ 0.1s

Adjusting weights to maximize calories is the solution that I find most effective because it allows me to include all the items without completely removing any of them. Instead of excluding lower-priority items, it intelligently reduces their weights while still maximizing the total calorie intake within the given weight limit. This way, I can ensure a balanced distribution of food items, making the solution more practical and efficient in real-world scenarios where completely discarding certain resources is not ideal.

Castillo, Anjelica M.
MCSCC 123-MSCS12S1 - Advanced Data Structures and Algorithms

Results:

```
Adjusted Item Weights and Calories:
Peanut butter: 1 kg, 5890 calories
Rice: 70 kg, 261800 calories
Wheat flour: 180 kg, 649800 calories
Strawberry jam: 1 kg, 2600 calories
Beans (can): 80 kg, 184000 calories
Potatoes: 580 kg, 1088660 calories
Tomatoes (can): 80 kg, 63200 calories

Total Calories: 2255950
```

I really appreciate how this approach maintains variety while optimizing for the highest possible calorie count.

Code:

```python
def adjust_weights_to_maximize_calories(items, max_weight):
    """Adjusts item weights to fit within max_weight while maximizing total calories without removing any item."""
    total_weight = sum(item[1] for item in items)
    if total_weight <= max_weight:
        return items, sum(item[2] for item in items)

    # Sort items by calorie density (calories per kg)
    items_sorted = sorted(items, key=lambda x: x[2] / x[1], reverse=True)

    # Reduce weights proportionally while keeping all items with at least 1 kg weight
    while total_weight > max_weight:
        for i in range(len(items_sorted)):
            if total_weight <= max_weight:
                break
            if items_sorted[i][1] > 1:  # Ensure item retains at least 1 kg
                reduction = min(10, items_sorted[i][1] - 1)  # Reduce in small steps, keeping at least 1 kg
                new_weight = items_sorted[i][1] - reduction
                new_calories = items_sorted[i][2] * (new_weight / items_sorted[i][1])
                items_sorted[i] = (items_sorted[i][0], new_weight, new_calories)
                total_weight -= reduction

    total_calories = sum(item[2] for item in items_sorted)
    return items_sorted, total_calories

# Define items as (name, weight, calories)
items = [
    ("Potatoes", 800, 1501600),
    ("Wheat flour", 400, 1444000),
    ("Rice", 300, 1122000),
    ("Beans (can)", 300, 690000),
    ("Tomatoes (can)", 300, 237000),
    ("Strawberry jam", 50, 130000),
    ("Peanut butter", 20, 117800)
]

max_weight = 1000

# Run the weight adjustment algorithm
adjusted_items, total_calories = adjust_weights_to_maximize_calories(items, max_weight)

print("Adjusted Item Weights and Calories:")
for item in adjusted_items:
    print(f"{item[0]}: {item[1]} kg, {int(item[2])} calories")

print("\nTotal Calories:", int(total_calories))
```