Interview Cake

You are a renowned thief who has recently switched from stealing precious metals to stealing cakes because of the insane profit margins. You end up hitting the jackpot, breaking into the world's largest privately owned stock of cakes—the vault of the Queen of England.

While Queen Elizabeth has a *limited number of types of cake*, she has an *unlimited supply of each type*.

Each type of cake has a weight and a value, stored in a tuple with two indices:

0. An integer representing the **weight** of the cake in kilograms
1. An integer representing the **monetary value** of the cake in British pounds

For example:

```python
# weighs 7 kilograms and has a value of 160 pounds
(7, 160)

# weighs 3 kilograms and has a value of 90 pounds
(3, 90)
```

You brought a duffel bag that can hold limited weight, and you want to make off with the most valuable haul possible.

Write a function `max_duffel_bag_value()` that takes **a list of cake type tuples** and **a weight capacity**, and returns **the *maximum monetary value* the duffel bag can hold.**

For example:

```python
cake_tuples = [(7, 160), (3, 90), (2, 15)]
capacity    = 20


max_duffel_bag_value(cake_tuples, capacity)
# returns 555 (6 of the middle type of cake and 1 of the last type of cake)
```

Weights and values may be any non-negative integer. Yes, it's weird to think about cakes that weigh nothing or duffel bags that can't hold anything. But we're not just super mastermind criminals—we're also meticulous about keeping our algorithms flexible and comprehensive.

Gotchas

Does your function work if the duffel bag's weight capacity is 0 kg?

Does your function work if any of the cakes weigh 0 kg? Think about a cake whose weight and value are *both* 0.

We can do this in $O(n * k)$ time and $O(k)$ space, where n is the number of types of cakes and k is the duffel bag's capacity!

Breakdown

Upgrade now ➜ (/upgrade)

Solution

This part requires full access (/upgrade)

You've already used up your free full access questions!

If you subscribe to our weekly newsletter (/free-weekly-coding-interview-problem-newsletter), you'll get another free full access question every week.

To see the full solution and breakdown for *all* of our questions, you'll have to upgrade to full access (/upgrade).

Upgrade now ➜ (/upgrade)

Complexity

$O(n * k)$ time, and $O(k)$ space, where n is number of types of cake and k is the capacity of the duffel bag. We loop through each cake (n cakes) for every capacity (k capacities), so our runtime is $O(n * k)$, and maintaining the list of $k + 1$ capacities gives us the $O(k)$ space.

Congratulations! Because of dynamic programming, you have successfully stolen the Queen's cakes and made it big.

Keep in mind: in some cases, it might *not* be worth using our optimal dynamic programming solution. It's a pretty slow algorithm—without any context (not knowing how many cake types we have, what our weight capacity is, or just how they compare) it's easy to see $O(n * k)$ potentially being as bad as $O(n^2)$ if n is close to k.

If we cared about *time*, like if there was an alarm in the vault and we had to move quickly, it might be worth using a *faster algorithm that gives us a **good** answer, even if it's not always the **optimal** answer*. Some of our first ideas in the breakdown were to look at cake values or value/weight ratios. Those algorithms would probably be faster, taking $O(n \lg n)$ time (we'd have to start by sorting the input).

Sometimes an efficient, *good* answer might be more *practical* than an inefficient, *optimal* answer.

Bonus

1. We know the *max value we can carry*, but **which cakes should we take, and how many?** Try adjusting your answer to return this information as well.
2. What if we check to see if all the cake weights have a **common denominator**? Can we improve our algorithm?
3. A cake that's both *heavier* and *worth less* than another cake would *never* be in the optimal solution. This idea is called **dominance relations**. Can you apply this idea to save some time? Hint: dominance relations can apply to *sets of cakes*, not just individual cakes.
4. What if we had a tuple for *every individual cake* instead of *types of cakes*? So now there's not an unlimited supply of a type of cake—there's exactly one of each. This is a *similar but harder* problem, known as the **0/1 Knapsack** problem.

What We Learned

Upgrade now ➜ (/upgrade)

Want more coding interview help?

Check out **interviewcake.com** for more advice, guides, and practice questions.