Dynamic Programming Homework

In this assignment you will use the provided code template to implement solutions to each given problem. The solutions you submit must follow the guidelines provided in this document as well as any given in code.

Your solutions must implement non-recursive, bottom-up dynamic programming approaches to each given problem.

Restrictions

- Template code is provided and must be used.
  - ***Note: The templates are refreshed each semester. Be sure to use the current version.***
- Your code must be compatible with **Python 3.10**.
- Do not add imports for any libraries beyond the Python standard library.
- Do not modify the function name or definition.

Testing

For each problem, one or more base test cases are provided. A properly implemented solution will pass any provided base test cases, but their trivial nature is not intended to ensure the overall correctness of your algorithm.

These are the same test cases that will be used to provide assurance that your submitted solutions run using the autograder. We do not release any other test cases.

In problem instances where more than one solution is optimal, *any* optimal solution will be considered correct, assuming that is the solution your algorithm produces.

The test cases can - and should! - be expanded while developing your solutions.

You can run your test cases by issuing the `python3 -m unittest` command from the directory containing your solution files.

Submission

Submit only - and all of - your solution files (i.e., **cs6515_[problem_name].py**) to the Gradescope assignment on or before the posted due date. Do not submit a zip file or any other files except those matching the naming convention defined.

You may submit your solutions as many times as desired and the most recent submission is what will be graded. Execution of your code may take up to 60 minutes. If not completed before the due date, your previous submission will be used.

Faster and correct solutions are worth more credit.

After your submission completes execution, you will see a score of `- / 10` listed until grading is completed.

Tim's Woot-Off Mistake

Your head TA Tim has decided to hang up his podcasting dreams and take a second job working at a Woot distribution center. Unfortunately for him, they've just announced a Woot-Off and things are about to get **very busy**.

As part of this special event, `n` products from inventory were selected to hype up. Each product `i` has a weight `W[i]` and sells for `P[i]`.

Due to 98% of Woot's developers being reallocated to rewriting ELIZA in Rust, the Executive Management Team decided to make this their first-ever Regionalized Woot-Off, relying on in-person sales.

They identified the `m` top cities most likely to have buyers who still know about Woot, with plans to send a single loaded trailer to each city, with each trailer `j` having a hauling capacity of `C[j]`.

To everyone's surprise, the same Executive Managment Team would like to ensure that they're getting the maximum profit **in each city** during this event, and assures Tim (and the rest of the packagers) that "everything will sell!" and this will "totally be worth the extra hours!". A pizza party is planned for after the event.

Example: Given `W = [1, 2, 3]`, `P = [4, 9, 6]`, and `C = [2, 3, 5]`, then the total profit is **44** and the maximum profit for each city is **9, 13, and 22**.

Develop an efficient dynamic programming algorithm to obtain the total profit along with the maximum profit for each city.

Notes:

- You may assume Woot has at least 1 product to sell and 1 city to sell to.

Implement your algorithm for solving this problem in **cs6515_woot_off.py**.