Programming Assignment 3 — Container Placement

Harbors face container placement challenges on a daily basis. Many containers arrive every day and they have to be placed in the most optimal way to reduce the costs of moving them.

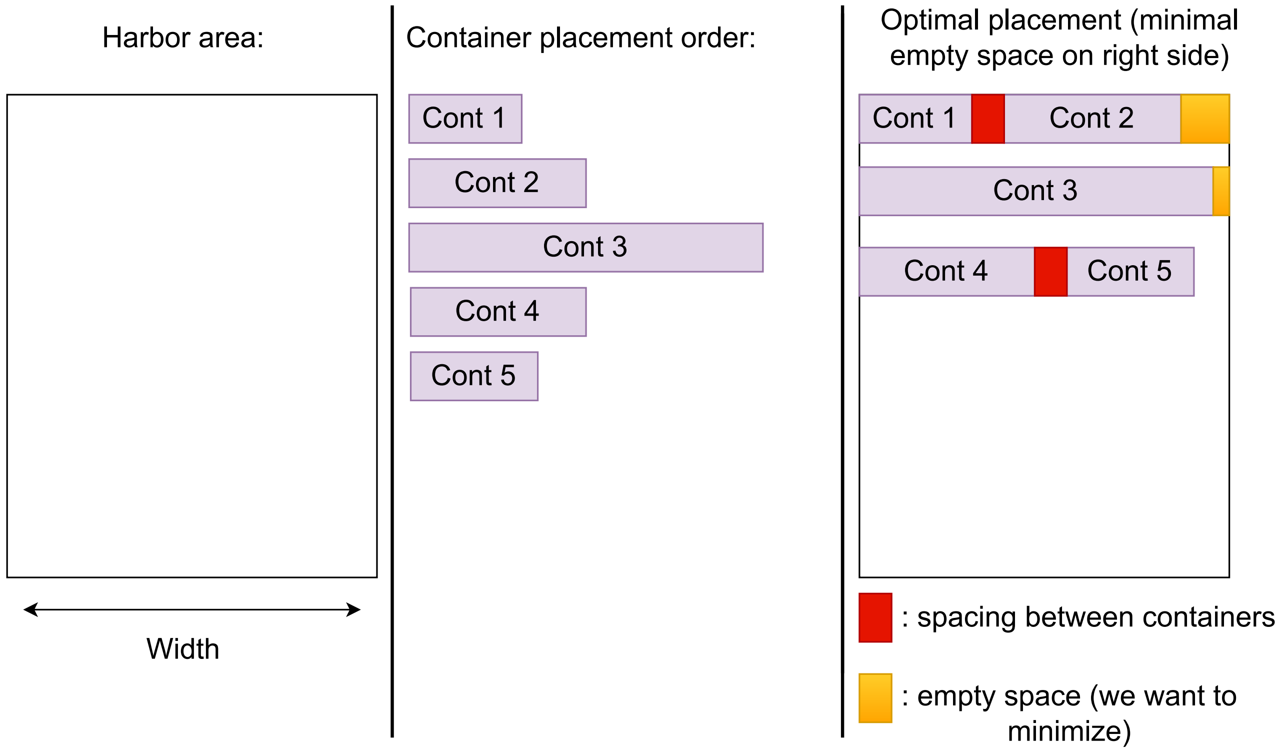
Harbor area:
Width
Container placement order:
Cont 1
Cont 2
Cont 3
Cont 4
Cont 5
Optimal placement (minimal empty space on right side)
Cont 1
Cont 2
Cont 3
Cont 4
Cont 5
: spacing between containers
: empty space (we want to minimize)

Figure 1: Illustration of the container placement problem.

Problem description

The container placement problem is shown in Figure 1. We have an area with a fixed width to our disposal on which we can temporarily place containers (left of figure). We are given a container placement order (center of figure), which consists of containers $c_1, \ldots, c_n$ of varying widths $w_1, \ldots, w_n$ that have just arrived and need to be placed on the harbor area. You can assume that all container widths w_i are smaller or equal to the total harbor width w. The containers **must** be placed in the given order (1,2,3,4,5 in this case) from left to right and from top to bottom. **Between** two adjacent containers, we need to have **exactly** s horizontal spacing to allow the cranes to place and retrieve the containers easily. It is not allowed to have containers exceeding the width of the harbor area: containers **must fit within the given width of the harbor**.

We are also given different cranes $o_1, o_2, \ldots, o_m$ which we can use to place the containers. Placing containers $c_i, \ldots, c_j$ on a single row with a given crane o_k has a certain *row operation* cost, which we denote $row\_opcost(i, j, k) = \sum_{s=i}^{j} opcost(s, k)$ (the sum of the costs of placing the individual containers). Importantly, the cranes **must be used in order from left to right**. This means that we cannot use o_k after having used o_c for $c > k$. For example, suppose that we are given 3 containers and 2 cranes. Then, we can use the first crane to place the first two containers, and the second crane to place the remaining container. However, it is **not** allowed to place the first two containers with crane 2 and the last container with crane 1. It is allowed to skip cranes: we can use crane 2 to place all containers without ever using crane 1.

The goal is to place all of the containers, taking into account the given order we have to place them in, in such a way that the *cost* of empty spaces on the right-hand side of all used rows **except the last one**, plus the cost of placing the containers with the cranes, is minimized. These empty spaces that we want to minimize are illustrated as yellow boxes in the right part of the figure.

Suppose that we want to place containers $c_i, c_{i+1}, \ldots, c_j$ on a single row, leaving an empty space $\xi(i,j)$ on the right-hand side of the row. Then, the cost of this empty space is $\xi(i,j)^5$ ($\xi(i,j)$ to the power of 5). We want to minimize the placement costs with the cranes plus the sum of these emptiness-costs over all rows without violating the constraints of the problem.

Solution example Suppose that the right-hand side of the figure is the optimal placement of the containers. A solution has 2 parts: 1) how are the containers placed, and 2) which cranes were used to place the containers. We can represent part 1) as a list of the left-most containers of every row. In this way, we can represent the solution as [Cont 1, Cont 3, Cont 4] because Cont 1 is the leftmost container on row 1, Cont 3 the leftmost on row 2, and Cont 4 the leftmost on row 3. We use this approach also in our implementation. Part 2) can be represented as a list of the cranes that were used for every container. Suppose that containers 1, 2, and 3 were placed with crane 1, and containers 4 and 5 with crane 2. Then, we represent the `crane list` as [1, 1, 1, 2, 2].

Dynamic programming approach

In order to solve this problem, we will make use of bottom-up dynamic programming. This requires that we can divide the problem into smaller overlapping subproblems, and use the solutions of the smaller problems to solve the complete problem.

Step 1: simplified version Start by assuming that there is only a single crane that can be used **without** operation cost. In that case, we want to to compute $\mathit{total\_cost}(j)$ of placing containers $c_1, c_2, \ldots, c_j$, using our previously computed solutions $\mathit{total\_cost}(k)$ for $k < j$. Write down the recurrent formula for this. Hint: first define a function $\mathit{row\_cost}(i,j)$ that gives the cost of placing containers $c_i, c_{i+1}, \ldots, c_j$ on a single row. We define $\mathit{row\_cost}(i,j)$ to be ∞ when the containers $c_i, c_{i+1}, \ldots, c_j$ do not fit on a row. Also keep in mind the special case for the last row (zero cost).

Step 2: multi-crane version After you have succeeded in finding the recurrent formula above, think what happens if we have multiple cranes instead of one. Then, we are interested in computing $\mathit{total\_cost}(j,k)$, meaning: the optimal cost of placing containers $c_1, \ldots, c_j$ using cranes $o_1, \ldots, o_k$. We can represent this function as a 2D matrix (see `self.total_cost` in the code. What is the base case(s)? You already know how to fill a single column of the memory (from the step above). How can we use this to define a recurrent formula for $\mathit{total\_cost}(j,k)$ in terms of $\mathit{total\_cost}(i,w)$ for $i < j$ and $w < k$. Hint: do not forget about the cost of placing containers with the different cranes in this step: $\mathit{row\_opcost}(i,j,k)$ (cost of placing containers $c_i, \ldots, c_j$ with crane o_k). **Note that your dynamic programming function must use the**

recurrent equation from this step: your program must solve the multi-crane version of the problem.

Programming

In short, the goal of the assignment is to write a dynamic programming algorithm that determines the optimal placement of the containers and the corresponding lowest total cost.

We provide you with a template program. It consists of two files:

- `containers.py`: The main assignment file
- `test_containers.py`: A Unit Test file

The file `containers.py` contains various functions that are not implemented yet (in total: 5). Read the documentation about these functions and implement them. Most of the functions require less than 20 lines of code. None of the functions require more than 30 lines of code. Do not change the headers of the functions provided. You are allowed to add functions yourself if you document them, but we strongly suggest you use the existing template for efficiency.

The file `test_containers.py` consists of several test functions. You can use these to give you an idea of whether functions are implemented correctly. Note that we do not test everything and a passing unit test does not mean that everything is implemented correctly! For this reason, we also ask you to come up with unit tests for (tricky) cases that the provided tests do not check for. The unit tests can be executed by running the following command:
`python -m unittest test_containers.py` **All public unit tests must be passed in order to get a passing grade.**

Make sure to have the right packages installed. Do not use other packages than those present in the template. Do not alter the unit tests that are provided. If your program does not succeed on all unit tests that are provided, it is likely that there is still a problem in your code. Make sure that all the unit tests succeed, before submitting the code.

Additionally, also hand in a file named `test_private.py`. In here, you can create additional unit tests to verify the working of your program. Write at least 3 additional unit tests, and hand these in along with the assignment. Also keep in mind that all unit tests should be able to run within a matter of seconds on any computer.

Report

Write a report in LaTeX(at most 3 pages) that is **neatly formatted** and addresses the following points / research questions:

- Give a complete explanation of the problem and write the final goal: what do we want to know? Introduce relevant notation that you use throughout the report.
- Write down the recurrent formulation in the case that we have a single crane (step 1). This means that you have to express the total cost $total\_cost(j)$ of placing containers $c_1, \ldots, c_j$ in terms of the total cost of placing containers $c_1, \ldots, c_k$ for $k < j$ **and explain** what this means. Ignore the operation cost for this part.
- Write down the recurrent formulation for $total\_cost(j, k)$ in the case where we have multiple cranes $o_1, \ldots, o_k$ (step 2). How does the recurrent formulation change? Make sure to not forget about *opcost*.

- What is the time complexity of the dynamic programming method? Use Θ-notation and use n for the number of containers.
- What is the space complexity of the method? Again, use Θ-notation and use n for the number of containers.
- How can we deduce the optimal placement of containers after running our dynamic programming function? (Explain backtrace_solution in natural language—without using code in your report!)

The deadline for this assignment is the **25th of May 2022, 23:59 CET (Central European Time)**.