Programming Assignment 2 — Tegelaar

Tegelaar is a Dutch tiling business located in the United States where they are chasing the American dream. They are growing very fast and can no longer create invoices manually. Therefore, they have contacted you to automate this process.

Description

Tegelaar is a company that creates and executes designs for the tiling of places such as new houses. Because of the fast growth in recent years, they want to automate i) the assessment of the feasibility of a given project, ii) the design of tiling patterns, and iii) the creation of invoices (how much money it will cost to execute the project).

A project consists of a flat rectangular area that has to be filled with tiles. In addition, the customer has given a budget limit (the maximum amount of dollars that they want to spend on the tiling). The tiles should fill the entire area and their can be no empty gaps in between. Tegelaar has an inventory of different rectangular tiles of various sizes and prices. It is your task to design and implement an algorithm that checks whether a given project can be fulfilled taking into account the following constraints:

- The complete rectangular area must be filled with tiles and there can be **no gaps** and **no tiles that go out-of-bounds** of the provided area
- The used tiles must be present in the inventory of the company
- The total cost of the project (sum of the prices of the tiles) must be less than the budget of the customer
- It is not allowed to cut tiles because that will weaken the special material that is used and reduce the quality significantly
- Tiles cannot overlap

Goal: This is a perfect problem for backtracking, which you will implement for Tegelaar. Additionally, you will write a greedy algorithm that attempts to obey these requirements.

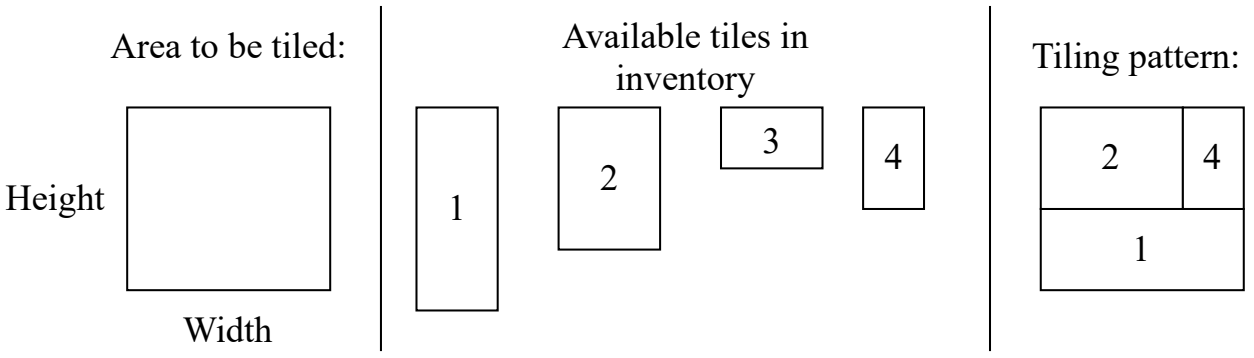

Figure 1: An example of an area to be tiled (left), the inventory of Tegelaar at the time of the project (middle) and a solution (right) using the tiles from the inventory. Note that tiles can be rotated.

Programming

For the project description, you are given the width and height of the rectangular area that has to be tiled as well as the budget of the customer. Moreover, you have access to the inventory of Tegelaar, which is implemented as a simple list of tuples (width, height). Every tuple is a tile with the given dimensions. Lastly, you are also given the prices of all of the tiles, implemented as a dictionary that maps a tile to its corresponding selling price. Note that the cost of a tile (width, height) will be the same as a tile of size (height, width) because they refer to the same tile (albeit rotated). Remember that the total cost of the project is the sum of the tiles being used. **Importantly:** you do **not** have to search for the cheapest solution because the customer has said it is OK as long as the total price is below the budget.

When there is no solution (because there are no tiles in the inventory with the right sizes to fill the entire area or because the tiling is more expensive than the budget) your program should return None.

We provided a template program. It consists of two files:

- `tegelaar.py`: The main file in which you will implement the solution.
- `test_usecases.py`: A Unit Test file with project requests that Tegelaar has done before.

The file `tegelaar.py` contains various functions that are not implemented yet. Note that you are not required to implement `build_schedule_backtracking`, this function makes a call to `_build_schedule_recursive`. Read the written documentation about these functions, and implement these functions. Most of these functions require less than 10 lines of code. Do not change the headers of the functions provided, except for `_build_schedule_recursive`. You are allowed to add functions yourself, if you feel that that makes it easier. Make sure to document these consistently.

The file `test_usecases.py` consists of several test functions. We provided these functions to help programming and testing the code. By running the following command, the unit tests can be executed:

```
python -m unittest test_usecases.py
```

Make sure to have the right packages installed. Do not use other packages than those present in the template. Feel free to add more unit tests. 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 other 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.

Example of a solution

In order to represent a tiling pattern, we use a dictionary where the keys are the starting positions of the tiles (bottom-left corners) and the values are the tiles themselves. For example,

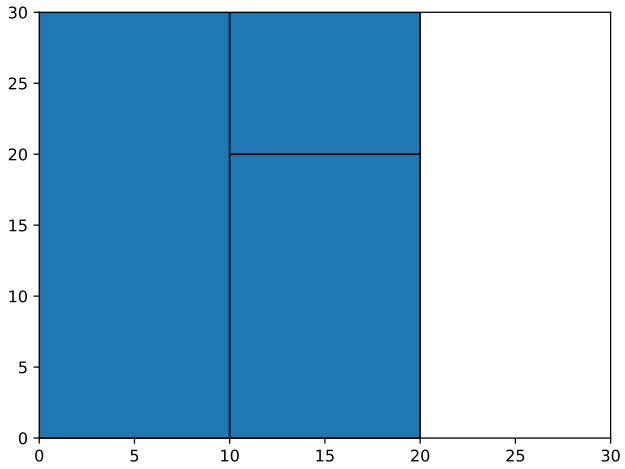

Figure 2: The tiling pattern given by the encoding in Equation 1.

suppose we have a tiling pattern

$$\begin{aligned} \texttt{solution} = \{&\text{pos}(0,0) : (10,30) \\ &\text{pos}(10,0) : (10,20) \\ &\text{pos}(10,20) : (10,10)\}. \end{aligned} \tag{1}$$

This pattern is shown in Figure 2. To see this, note that at position (0,0) we put the tile with a width of 10 and height of 30 (the tile (10,30)). At position (10,0) we place the tile (10,20) with a width of 10 and height of 20, and at position (10,20) the tile (10,10).

Report

Write a report in LaTeX(at most 3 pages), addressing the following points / research questions:

- Introduction: describe the problem. Describe a state and action.
- Draw the state-space of a small example, e.g., using an area with a width of 20 and height of 30, and available tiles (10,30), (10,10), (5,5), and (10,10). You may assume that the customer has an infinite budget. Also indicate the total cost of the partial solutions. You are allowed to draw this on paper and scan the result.
- Could you think of a better way of representing the inventory of Tegelaar (instead of a list)? How would that save time?
- Analysis of optimal method: How much time does the optimal method take on schedules with varying number of courses? What is the limit that you can run within 10 minutes?
- How well does the greedy approach work? How often and how much does the answer differ from the result of the optimal algorithm? Include an adversarial test-case in your report where the greedy approach does not yield an optimal schedule.
- Summary and Discussion. Always end a report with some summarizing remarks.

The deadline for this assignment is **the 22th of April 2022, 23:59 CET.**