Middle East Technical University Department of Computer Engineering

CENG 315

Algorithms

Fall 2024-2025

Take-Home Exam 4

Chocolate Value Problem (Dynamic Programming)

Due date: 16 November 2026, Sunday, 23:59

1 Problem Definition

While escaping the Queen of Hearts, Alice ends up at the Mad Hatter's workshop of magical chocolates. To avoid being handed over, she must help the Mad Hatter cut a large rectangular chocolate bar the way he likes.

But the Mad Hatter is picky: he does not care how many pieces the chocolate becomes. For him, a piece of chocolate is valuable only if its exact dimensions appear in his list of "sellable cuts."

You are given a 2D integer array `values` such that:

$$\texttt{values[w][h]} = \begin{cases} -1, & \text{if a } w \times h \text{ chocolate piece cannot be sold at all,} \\ \text{a positive integer,} & \text{if a } w \times h \text{ piece can be sold for that market value.} \end{cases}$$

Here, $1 \le w \le \texttt{chocolate\_width} - 1$ and $1 \le h \le \texttt{chocolate\_height} - 1$.

- If values[w][h] = -1, the piece is not sellable.
- If values[w][h] > 0, that piece can be sold for that market value.
- Rotation is forbidden — a 5×4 piece is not the same as 4×5.
- When updating the DP table, replace an entry only if the newly computed value is *strictly greater* than the current value. If equality occurs, you must keep the existing value.

Since the chocolate bar has a pattern on it, Mad Hatter forbids Alice to rotate it. He wants the maximum total market value of the sellable pieces. The waste is simply, the total area of chocolate that cannot be sold after optimal cutting.

Formally, you are required to implement the function:

```
std::array<unsigned int, 2> chocolate_value(unsigned short chocolate_width,
    unsigned short chocolate_height, int **values);
```

In this exam, you are tasked with implementing the method chocolate value to return the maximum total market value of the sellable chocolate slices and the waste amount of chocolates. The chocolate bar is represented as a 2D rectangle with width chocolate width and height chocolate height. The dimensions of the sellable slices are given to you with the values 2D array of integers, where each value stands for the value of a slice with the indices of the array cell as dimensions. For example, if values[4][5] > 0, then a slice with width 4 and height 5 is a sellable slice (if cut properly). However, this does not mean that a slice with width 5 and height 4 is sellable, since the orientation of the chocolate is important.

Your implementation should return the maximum total market value of the sellable chocolate slices and the waste amount. You are expected to solve this task using a dynamic programming approach. Test cases and VPL limits will cause recursive solutions to not terminate on most inputs.

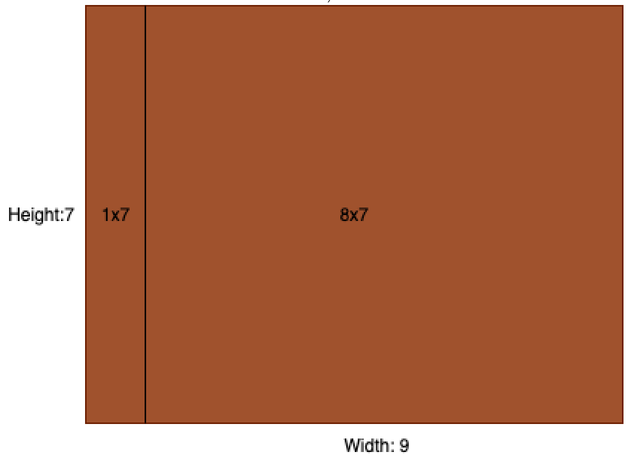
1.1 Example Run

The following is only an illustrative example, not a hidden test case.

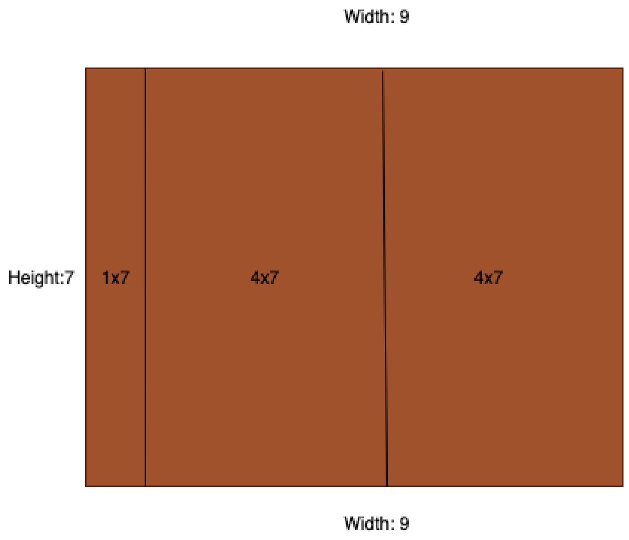
```
chocolate_width = 9
chocolate_height = 7
values[4][2] = 6
values[1][3] = 1
values[w][h] = -1 for all other pairs
```

Step by step, Alice can cut the chocolate bar as follows:

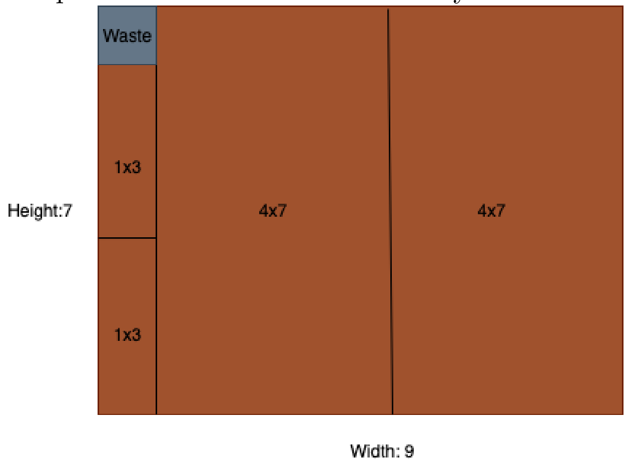
With the vertical first cut, the bar is divided into two pieces:



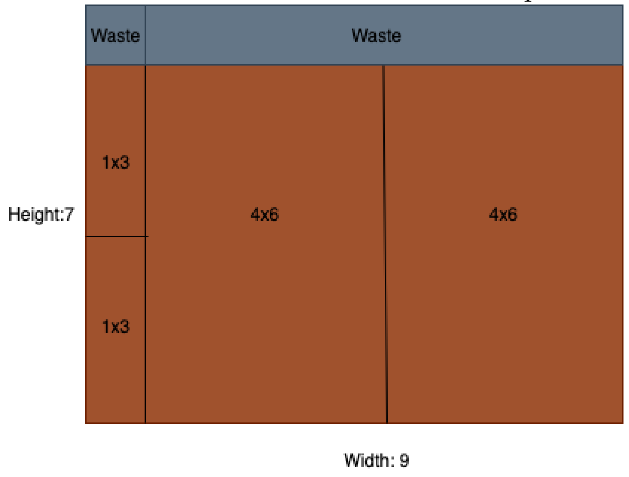
With another vertical cut, the right slice is divided into two:



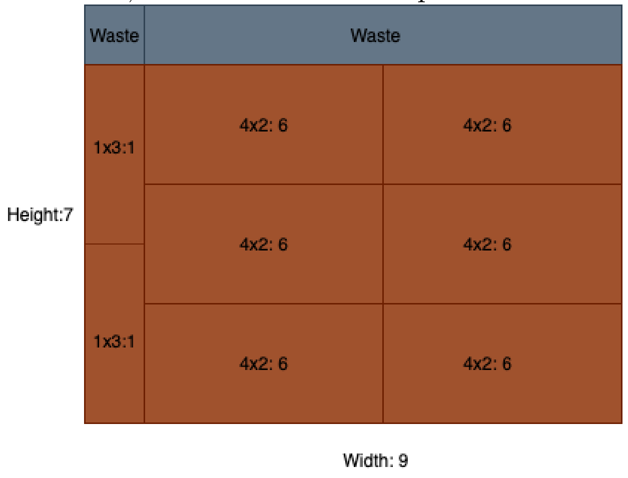
The piece of size $1x9$ is cut vertically two times and gets the final values.



Slices of size $4x7$ are cut horizontally at height 1 from the top two to create wastes. After that, 2 horizontal cuts create the final form of the pieces.



In the end, the final value of the pieces is 38 and the waste area is 9.



Note: The auto-grader will use many randomly generated `values` tables. You should not hard-code any behavior specific to this example.

2 Limits and Specifications

In this exam, the complexity of your implementations will be checked by running them under VPL limits. The system and variable limitations are as follows:

- `chocolate_width`, `chocolate_height` ≤ 600,
- maximum dimensions for the `values` table: $[chocolate\_width + 1][chocolate\_height + 1]$ (i.e., up to 601×601),
- `values[w][h]` $= -1$ or a non-negative integer not exceeding 10^9,
- at most 1000 entries of `values[w][h]` will be non-negative,
- 16 seconds as the maximum execution time,
- a maximum memory limit of 1 GB,
- a stack size of 4 MB for function calls (so **recursive solutions are strongly discouraged**).

Additionally:

- You will implement your solution in the file `the4.cpp`.
- Do not change the first line of `the4.cpp`, which is `#include "the4.h"`.
- Do not change the arguments and the return value of the given function `chocolate_value` in `the4.cpp`, but you are free to add helper functions.
- Do not include any other library or write additional `#include` lines in `the4.cpp` (not even in comments).

- You are given a `test.cpp` file to test your work on ODTUCLASS or locally. You can, and are encouraged to, modify this file to add different test cases.
- In the VPL environment, if you click *Run*, your function will be compiled and executed with `test.cpp`. If you click *Evaluate*, you will get feedback for your current work and your work will be graded with randomly generated inputs.
- The first three test cases are provided as examples. They display the parameters, the expected output, and your program's output to help guide you. The remaining cases are hidden.
- After the submission period, additional randomly generated test cases will be used to determine your final grade.

3 Regulations

- **Implementation and Submission:** The template files are available in the VPL activity `\THE4"` on ODTUCLASS. You can either:
  - download the template files, complete the implementation locally, and submit, or
  - directly edit and evaluate your code in the VPL environment.

  Make sure that your code compiles and runs correctly on ODTUCLASS. The last saved/submitted version will determine your final grade.
- **Programming Language:** You must write your program in C++. Your submission will be tested in the VPL environment on ODTUCLASS.
- **Cheating:** This assignment is individual work. Sharing code with others, using someone else's implementation, or copying from the internet is strictly forbidden. Your code may be checked against current and previous semesters as well as online sources. Any detected similarity may result in disciplinary action and a score of 0.
- **Evaluation:** Your program will be evaluated automatically using black-box testing. All inputs will obey the constraints; you do not need to handle invalid inputs.