Middle East Technical University | Department of Computer Engineering

CENG 315

Algorithms

Fall 2024-2025

Take-Home Exam 4

Due date: 17 November 2024, Sunday, 23.59

1 Problem Definition

While running from the Queen of Hearts, Alice finds herself at the Mad Hatter's tea party! Mad Hatter promises not to give her away if she cuts and serves the special patterned cake according to his requirements.

Mad Hatter does not care about the total count of slices, but he wants each slice to be *tasty*. However, his idea of tastiness is more about the shape and cutting of the slices. A slice of cake is tasty if all of the following holds:

- The dimensions of the created slice is one of the `perfect_cuts`,
- The cutting action is either done horizontally or vertically at each step,
- Each cutting action cuts completely through that piece of the cake

Since the cake has a pattern on it, Mad Hatter forbids Alice to rotate it. He wants Alice to minimize the amount of cake that he deems "not tasty enough" after cutting the tasty slices.

```
unsigned int alice_cutting_cake(const unsigned short cake_width,
                                const unsigned short cake_height,
                                bool **perfect_cuts);
```

In this exam, you are tasked with implementing the method `alice_cutting_cake` to return the minimum amount of cake left that is not *tasty* enough. The cake is represented as a 2D rectangle with width `cake_width` and height `cake_height` in centimetres. The dimensions of the *tasty* slices are given to you with the `perfect_cuts` 2D array of booleans, where each value stands for the existence of a perfect cut with the indices of the array cell as dimensions. For example, if `perfect_cuts[4][5] == true`, then a slice with width 4 and height 5 is a *tasty* slice (if cut properly). However, this does not mean that a slice with width 5 and height 4 is *tasty*, since the orientation of the cake and the perfect cuts are important.

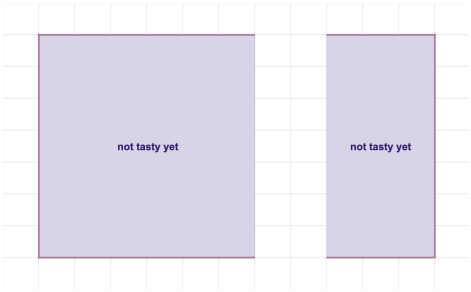
Your implementation should return the amount of cake that is deemed "not tasty enough" by the Mad Hatter as the remaining area in the 2D representation of the cake. You are expected to solve this task using a dynamic programming approach. Test cases and VPL limits will cause recursive solutions to not terminate on most inputs.

1.1 Example Run

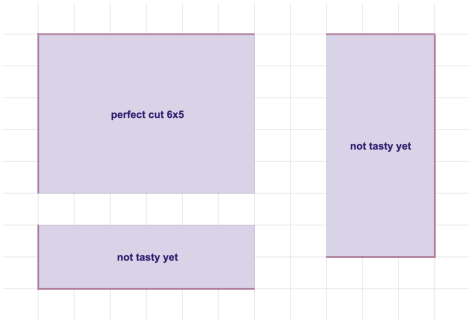
```
cake_width: 9
cake_height: 7
perfect_cuts:
perfect_cuts[6][5] == true
perfect_cuts[3][2] == true

wasted cake area that is not tasty enough: 3
```

Step by step, Alice can cut the cake as follows:

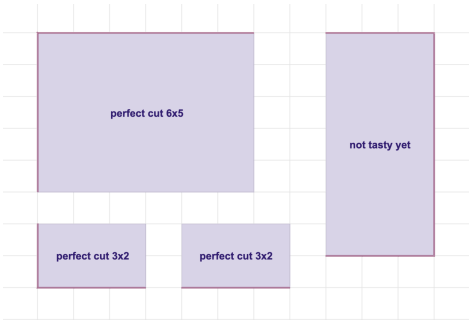
With the vertical first cut, the cake is divided into two pieces:



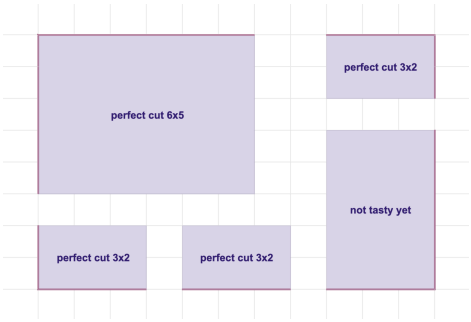
6x7 sized piece is divided into two pieces by cutting horizontally, achieving one *tasty* slice of size 6x5:



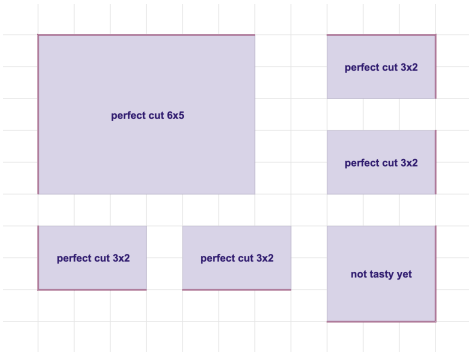
6x2 sized piece is then cut vertically to achieve two *tasty* slices of size 3x2:



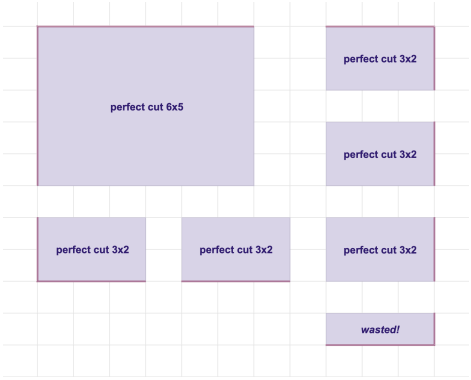
3x7 sized piece is then cut horizontally to achieve one *tasty* slice of size 3x2:



Remaining 3x5 sized piece is then again cut horizontally to achieve one more *tasty* slice of size 3x2:



Lastly, remaining 3x3 sized piece is then cut horizontally to achieve the last *tasty* slice of size 3x2:



2 Limits and Specifications

In this exam, the complexity of your implementations will be checked by the run of your implementations staying within the VPL limits. Hence the system and variable limitations are as follows:

- `cake_width`, `cake_height` <= 600,
- maximum dimensions $601x601$ for `perfect_cuts` of size `[cake_width+1][cake_height+1]`,
- 200 as the maximum number of `true` cells in `perfect_cuts`,
- 16 seconds as the maximum execution time,
- a maximum memory limit of 1 GB,
- and a stack size of 4 MB for function calls (ie. recursive solutions)

Additionally:

- You will implement your solutions in the `the4.cpp` file.
- 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 functions in the file `the4.cpp`, but you are free to add other functions to `the4.cpp`.
- Do not include any other library or write include anywhere in your `the4.cpp` file (not even in comments).
- You are given a `test.cpp` file to test your work on ODTUCLASS or your locale. You can and you are encouraged to modify this file to add different test cases.
- You can test your `the4.cpp` on the virtual lab 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 grade you see in VPL is your official grade since the test cases are generated randomly.
- If you want to test your work and see your outputs on your locale you can use the following commands:

```
> g++ test.cpp the4.cpp -Wall -std=c++11 -o test
> ./test
```

3 Regulations

- **Implementation and Submission:** The template files are available in the Virtual Programming Lab (VPL) activity called "THE4" on ODTUCLASS. At this point, you have two options:
  - You can download the template files, complete the implementation, and test it with the given sample I/O on your local machine. Then submit the same file through this activity.
  - You can directly use the editor of the VPL environment by using the auto-evaluation feature of this activity interactively. Saving the code is equivalent to submitting a file.

  Please make sure that your code runs on ODTUCLASS. There is no limitation in running your code online. The last save/submission will determine your final grade.

- **Programming Language:** You must code your program in C++. Your submission will be tested on the VPL environment in ODTUCLASS, hence you are expected to make sure your code runs successfully there.
- **Cheating: This assignment is designed to be worked on individually.** Additionally, the use of any LLMs (chatgpt, copilot, the other one that you are thinking about...) and copying code directly from the internet for implementations is strictly forbidden. Your work will be evaluated for cheating, and disciplinary action may be taken if necessary.
- **Evaluation:** Your program will be evaluated automatically using "black-box" testing, so make sure to obey the specifications. No erroneous input will be used. Therefore, you don't have to worry about invalid cases.