Introduction à l'algorithmique **Winter 2024**

IFT2125-6001 Assessment (7.5%)

Homework 3 : An aMAZEing homework!

General instructions: To be done in teams of 1 to 2 people. Submit a single pdf and the python and C++ code files. Indicate your name and registration number (matricule) on all files (pdf and code).

Your code must pass the tests provided with the assignment, otherwise a grade of 0 will be assigned to exercises whose basic tests are not passed. Other tests will be added during the correction. Only standard Python and C++ libraries are allowed, unless otherwise stated.

It goes without saying that the code must be clear, well indented and well commented.

The submission date is Thursday March 28 at 10:30 p.m. The structure of the submission in studium must be as follows:

```
StudiUM
├── devoir3_NAME1_NAME2.pdf
├── ClimbingDifficultyCalculator.cpp
├── ClimbingDifficultyCalculator.h
├── vitre.py
└── labyrinth_generator_creator.py
```

1 Find the error (10 points)

Each of the following proofs or statements contains at least one error. Clearly identify where all these errors are, and explain appropriately why you say they are errors (e.g., say why such a step is not correct, give what is missing or should be replaced in a definition, give counterexamples invalidating a statement, etc.)

Question 1: (2 points) Proof that $2^n \in \Omega(4^n)$:

By definition of Ω, we are looking for c and n_0 such that $2^n \geq c4^n, \forall n \geq n_0$.

$$2^n \geq c4^n \tag{1}$$
$$\frac{2^n}{4^n} \geq c \tag{2}$$
$$\left(\frac{1}{2}\right)^n \geq c \tag{3}$$
$$\log_{\frac{1}{2}} \left(\frac{1}{2}\right)^n \geq \log_{\frac{1}{2}} c \tag{4}$$
$$n \geq \log_{\frac{1}{2}} c \tag{5}$$

Since $\log_{\frac{1}{2}} c$ is constant, by taking $c = \frac{1}{2}$, we will have $n \geq \log_{\frac{1}{2}} \frac{1}{2} = 1$, which is true by taking $n_0 = 1$.

Therefore, we have found $c = \frac{1}{2}$ and $n_0 = 1$ such that $2^n \geq c4^n, \forall n \geq n_0$.

Thus, we have $2^n \in \Omega(4^n)$. ■

Question 2: (2 points) Proof that $2^n \in \Omega(4^n)$:

By using the limit rule,

$$\lim_{n\to\infty} \frac{2^n}{4^n} = \lim_{n\to\infty} \frac{2^n}{2^{2n}} \tag{1}$$
$$= \lim_{n\to\infty} \frac{\lg(2^n)}{\lg(2^{2n})} \tag{2}$$
$$= \lim_{n\to\infty} \frac{n}{2n} \tag{3}$$
$$= \lim_{n\to\infty} \frac{1}{2} \tag{4}$$
$$= \frac{1}{2} \tag{5}$$
$$\tag{6}$$

Since $\frac{1}{2} \in \mathbb{R}^{>0}$, the limit rule allows us to conclude that $2^n \in \Theta(4^n)$.

Thus, by the definition of $\Theta(4^n)$, we have $2^n \in \Omega(4^n)$. ■

Question 3: (2 points) Here's the formal definition of $O(f(n))$:

$$\Omega(f(n)) = \left\{t(n) : \mathbb{N} \to \mathbb{R}^{\geq 0} \middle| (\exists c \in \mathbb{R}^{\geq 0})(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)\ f(n) \geq cf(n)\right\}$$

Question 4: (2 points) Proof that $(\exists x \in \mathbb{N})(\forall y \in \mathbb{N})\ x \geq y$ is true :

$$(\exists x \in \mathbb{N})(\forall y \in \mathbb{N})\ x \geq y \iff \quad (1)$$
$$(\forall y \in \mathbb{N})(\exists x \in \mathbb{N})\ x \geq y \iff \quad (2)$$

The last line is true because for a given y, we take $x = y + 1 \geq y$ therefore $x \geq y$. ■

Question 5: (2 points)

Let f and g be two functions $\mathbb{N} \to \mathbb{R}^{\geq 0}$ such that $f(n) < g(n)\ \forall n \geq n_0$.

Therefore $lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

2 Rock climbing - C++ code (20 points)

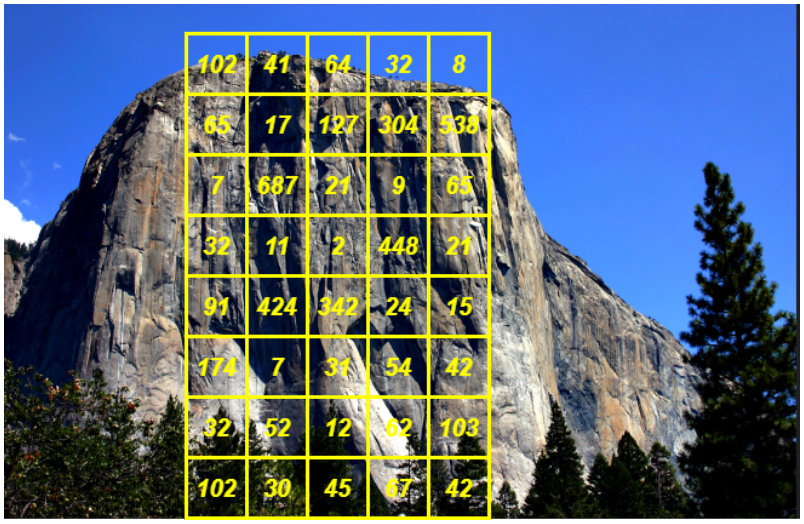
In this problem, you are asked to find the optimal way for a person to climb a wall using the C++ language.

Your algorithm must use dynamic programming. Your algorithm should be efficient to get all the points.

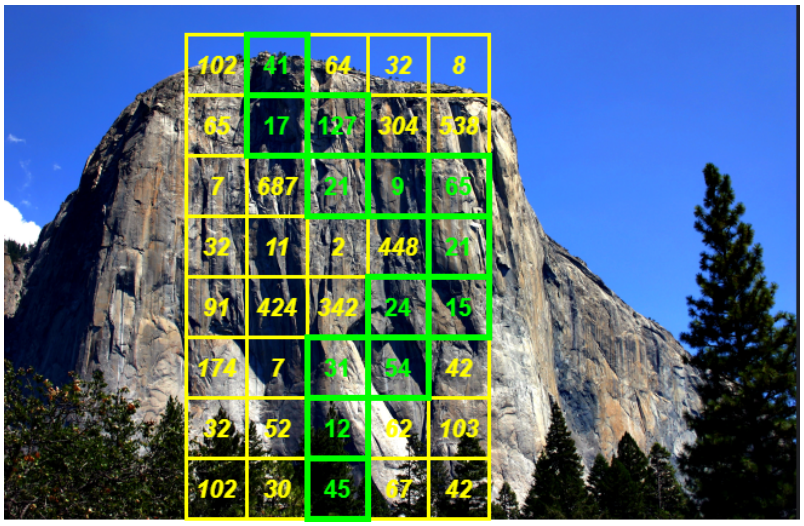
The climbing wall will be represented as a rectangle divided into $m \times n$ regions. Each region will have a corresponding degree of difficulty, represented by an integer. Your algorithm therefore receives as input a matrix of integers $m \times n$.

The climber starts at the bottom of the wall, that is to say on the m^{th} row of the matrix. They are free to choose in which of the n regions at the bottom of the wall they will position themself initially to begin their ascent. Their objective is to reach any of the regions at the top of the wall, that is to say the first row of the matrix. When climbing, the climber cannot move down the wall. The only movement options they have is to either go to the area immediately to the right, left, or up from where they are. The climber is looking for the easiest path allowing them to scale the wall, that is to say the path whose sum of degrees of difficulty of the regions visited is minimal.

For example, on the following wall:



The optimal ascent would be :



Inputs :

The input is provided as a .txt file which contains the elements of the matrix. For the output, you are only asked to return the total sum corresponding to the optimal path.

```
Example :
102,41,64,32,8
65,17,127,304,538
7,687,21,9,65
32,11,2,448,21
91,424,342,24,15
174,7,31,54,42
32,52,12,62,103
102,30,45,67,42
--> must return the total length of the path = 482
```

Code
Call example :

```
Compiling :
g++ -o climbing_difficulty.exe climbing_difficulty.cpp
ClimbingDifficultyCalculator.cpp

Execution :
.\climbing_difficulty.exe wall1.txt
```

Submission
Complete the *ClimbingDifficultyCalculator.cpp* and *ClimbingDifficultyCalculator.h* files provided, and submit **only** these files. Do **not** resubmit the *climbing_difficulty.cpp* file.

3 Windows tester - Python code (20 points)

In this problem, you are asked to find the minimum number of tests that need to be done in the worst case to determine the strength of a window, using the Python language.

Your algorithm must use dynamic programming. Your algorithm should be efficient to get all the points.

The ministère de l'éducation, des loisirs et du sport du Québec (MELS) whishes to install gourgeous large windows in some of Québec's schools. As children will play ball games near these windows, the ministry first wishes to evaluate the resistance of a window to ball throws hitting it, by determining the threshold $s \in \mathbb{N}$ in Newtons at which a window breaks when hit by a ball thrown with a force of s Newtons.

To do these tests, you only receive k identical windows from the manufacturer. You also have a machine allowing you to throw a ball with a force of x Newtons, for x an integer that you can modify between each throw and which can go from 1 to N (N is a parameter supplied with the machine, you can assume $1 \leq s \leq N$).

If a ball is thrown at a window with a force greater than or equal to s, the window will break and will no longer be usable for testing. On the other hand, if the ball is thrown with a force strictly less than s, the window will not break and can be reused.

The goal of your algorithm is to calculate, given k and N, the minimum number of tests that would need to be done in the worst case to find s. In other words, calculate the value of

$$\min_{S \in \text{Strategies}_{N,k}} \left\{ \max_{s} \{\text{number of throws made by strategy } S \text{ when the threshold is } s\} \right\}$$

where $\text{Strategies}_{N,k}$ is the set of valid strategies (which allow finding s without running out of windows, regardless of the value of s) with a force up to N and k windows.

For example, if we only have one window ($k = 1$) and $N = 4$, the only solution is to try with $x = 1$, then $x = 2$ if the window was not broken before, then $x = 3$ if the window was not broken before (no need to try for $x = 4$, because if $s = 4$, the throw at $x = 3$ will detect it since the window would not have broken). In the worst case, it will therefore be necessary to do 3 throws, and therefore the algorithm should return 3.

If we have two windows ($k = 2$) and $N = 4$, the optimal solution would be to first try with $x = 2$, then try with $x = 1$ (if the window did not break) or $x = 3$ (if the window broke). In all cases, it will take 2 throws, so the algorithm would return 2.

Hints: To help you out, here are some ideas for you to think about: Should your dynamic programming array be 2D or 3D? What should the axes of the table be? number of windows remaining, number of broken windows, number of tests performed, smallest force tested that broke the window, largest force tested that did not break the window, number of force levels to test, etc.

Examples :

```
Example 1 :
4 1
--> should return 3

Example 2 :
4 2
--> should return 2
```

Code
Your code will directly take N and k as parameters in the main function. A function named *vitre(N,k)* will then be called and will return the desired value as an int. There is no file reading/writing for this question.

Call example :

```
python3 vitre.py 4 2
```

A file *test_vitre.py* and input files are provided to help you test your code.

Call example :

```
python3 test_vitre.py
```

Submission
Complete the *vitre.py* file provided, and **only** submit this file. Do **not** resubmit the *test_vitre.py* file, or any other test file.

4 Maze - Code Python (50 points)

In this problem, you are asked to implement 2 maze generation algorithms, using the Python language.

You are free to choose the algorithms. You can invent them yourself, or implement algorithms that already exist (in such a case, cite your sources!). The Wikipedia page Maze generation algorithm might be a good place to start.

Then, you will need to transform your maze into a 3D model in .scad format which can be opened with OpenSCAD.

Here are two examples of final results:

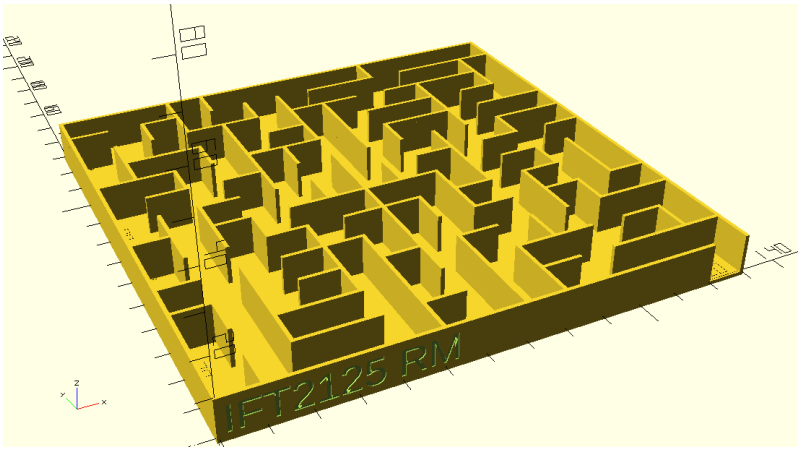


Figure 1: A maze

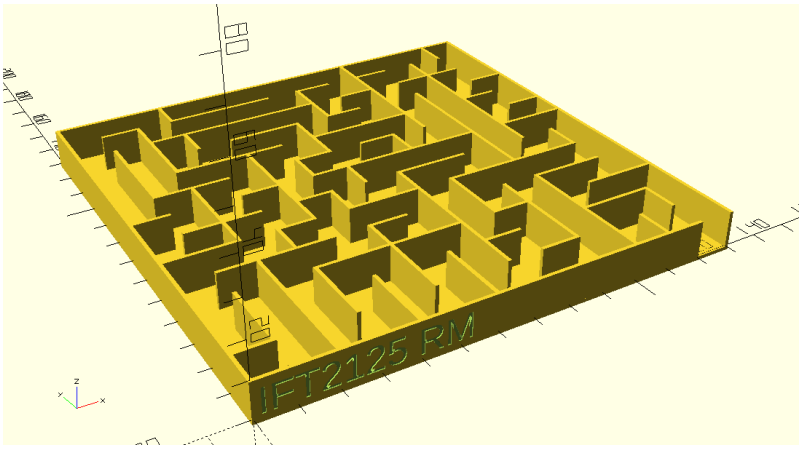


Figure 2: Another maze

To get full points, your mazes must look "interesting". They should not be too small, there should be a good density of walls and corridors, no inaccessible areas, not too trivial to solve, etc.

Code

Your code must take as an argument a number which will be either 1 or 2, and which allows to select which of your two algorithms we use. It must then write the 3D labyrinth in a .scad file that can be opened with OpenSCAD, and the file name must contain the number of the algorithm used (example: labyrinth_algo1.scad and labyrinth_algo2.scad).

Call examples :

```
python3 labyrinth_generator_creator.py 1
```

```
python3 labyrinth_generator_creator.py 2
```

For this question, there are no test files provided to you. We give you two examples of .scad files as a reference.

Submission
Complete the *labyrinth_generator_creator.py* file provided, and only submit **only** this file. Do **not** submit other files, including your own .scad files (we will generate them ourselves).

Bonus 10%
We will grant a maximum bonus of 10% (for the assignment) to teams who 3D print a copy of a maze created with their own code (free printing at the math and computer science library, see Indiana Delsart. An introductory workshop will be given specifically for those in the course at a time to be decided). The model should be approximately 8-12cm by 8-12cm with a height of 0.8-1.2cm with 10x10 or more. It must also bear the course number (IFT2125) and your initials, as is the case on the figures above.