Hacettepe University – Computer Engineering
BBM204 Software Practicum II – Spring 2023

PROGRAMMING ASSIGNMENT 4

Topics: Graphs - Shortest Path, Regular Expressions, Greedy Programming

Course Instructors: Assoc. Prof. Dr. Erkut Erdem, Prof. Dr. Suat Özdemir, Asst. Prof. Dr. Adnan Özsoy
TAs: Ali Burak Erdoğan, Alperen Çakın, Dr. Selma Dilek
Programming Language: Java (OpenJDK 11)
Due Date: **Thursday, 01.06.2023 (23:59:59)**

İMECE

Türkiye's First Sub-Meter High-Resolution Observation Satellite

İMECE, Türkiye's first sub-meter high-resolution observation satellite, has recently been launched with the goal of capturing detailed images for a variety of applications. The satellite is equipped with a powerful camera that could capture images with unprecedented detail, allowing researchers to monitor everything from natural disasters to urban development.

With the launch of **İMECE**, Türkiye made its space history by launching for the first time an electro-optical satellite camera with a sub-meter resolution that was developed using only domestic and national resources. **İMECE** will simultaneously orbit the sun at an altitude of 680 kilometers, meeting Türkiye's need for high-resolution satellite imagery. The satellite, capable of capturing images from any location in the world without any geographical limitations, will benefit Türkiye in various ways, including target detection and identification, natural disasters, mapping, and agricultural practices.

As the students of **HUBBM** and **HUAIN**, two of the best computer engineering departments in the country, you have been selected to implement some of the most important algorithms that **İMECE** will use for a number of applications. Your algorithms will be used to update the existing algorithms onboard **İMECE** to optimize several of its missions.

The success of this mission depends on your algorithms - are you up to the challenge?

1 Reading the Input Data Using Regular Expressions

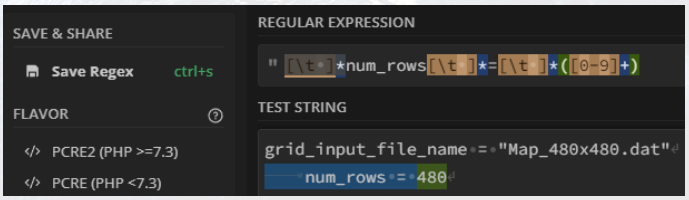
The necessary input parameters for this assignment are given in a `.dat` file as the *first command line argument*, and it will be structured as follows:

```
grid_input_file_name = "Map_480x480.dat"
    num_rows = 480
num_cols= 480
mission_0_source = ( 300,100)
mission_0_destination= (400, 400 )
mission_1_source =( 150 ,300 )
max_flying_height= 4000
fuel_cost_per_unit= 0.1
climbing_cost_per_unit=0.5
```

Note that the **order of variables** can change along with the **order** and the **placement** of the whitespace characters (**tabs** or **spaces**) around the **variable names**, **equal sign**, **comma**, **parentheses** and **outside of quotes**. To compensate for those changes easily, you are supposed to implement this part using several **regular expressions**. To read the necessary parameters for this mission, you must complete four methods in *DatReader* class from the starter code. One of those methods is given below as an example and guidance on how to implement this part.

```
1       public int getIntVar(String varName) {
2           Pattern p = Pattern.compile("[\\t ]*" + varName + "[\\t ]*=[\\t ]*([0-9]+)");
3           Matcher m = p.matcher(fileContent);
4           m.find();
5           return Integer.parseInt(m.group(1));
6       }
```

The first line of the method contains the necessary regular expression for matching an integer variable with an optional number and placement of whitespace characters. Using the variable `num_rows` as an example, the figure below illustrates the match and the groups (blue, encompassing the whole match, is **Group 0**, while green, encompassing only the relevant part in parentheses, is **Group 1**).



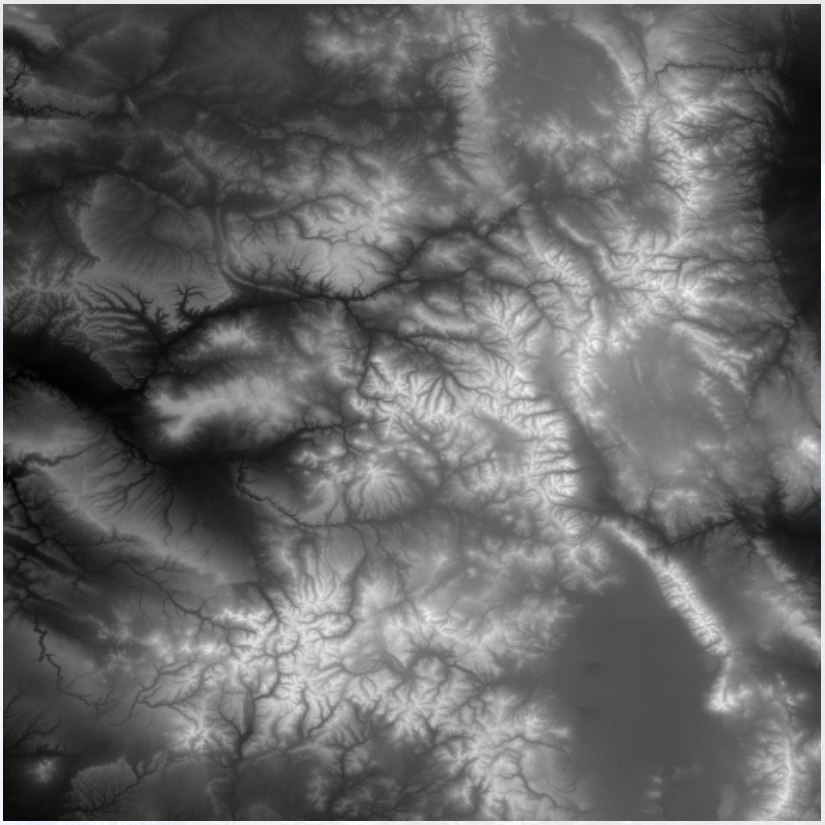
A logic similar to the example *getIntVar()* method should be used for other methods while parsing the input file.

In the *getStringVar()* method, you should alter the given regular expression such that it matches the value of the variable given by *varName* as **Group 1** and returns it.

In the *getDoubleVar()* method, you should alter the given regular expression such that it matches the value of the variable given by *varName* as **Group 1**, parses it as a *Double* and returns it. Your regular expression should support floating point numbers with an arbitrary number of decimals or without any (e.g. 5, 5.2, 5.02, 5.0002, etc.).

In the *getPointVar()* method, you should alter the given regular expression such that it matches the value of the variable given by *varName* as two integers between parentheses separated by a comma as **Group 1** and **Group 2** respectively. Your method should then create an instance of the *Point* class by two integers parsed using **Group 1** and **Group 2**, then return the *Point* instance.

2 Mission 0 - Finding the Most Cost-Efficient Path



You have been selected by the **İMECE** developers team to contribute to the implementation of some very important functionalities of the satellite. In this first mission, you are expected to implement methods that calculate and draw **the most cost-efficient path to be taken by a surveillance drone over a geographical area**.

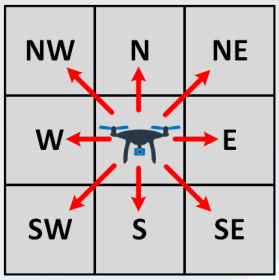
2.1 Background Information and Mission Objectives

You will be given topographic data of a geographical area as a 2D array of land elevation values (integers). The unit of the elevation data is not relevant to this mission, but you may think of it as the land height in meters. Your objective is to find the most cost-efficient path between two coordinate points (**source** and **destination**), which will be used by a surveillance drone. The drone has a limited power supply needed for extending the maximum flight time before the next refueling, so the satellite will perform the calculations given the topographic data, and feed the calculated path data to the drone.

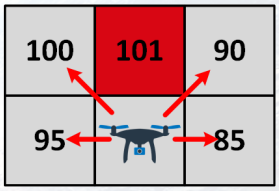
There are some important constraints that need to be considered while calculating the path:

- A drone can move only in a specific way: if we assume that the current position of the drone is a specific pixel on the given map, the drone's next move can only be performed towards one of the neighboring pixels.

E.g., if we consider the map to be positioned such that the top is **North**, the bottom is **South**, the left is **West**, and the right is **East**, a drone can only move towards eight directions at most (even less if the current position is at an edge of the map): **E, W, N, S, SE, SW, NE, NW**.



- A drone has a maximum height it can fly over and it will be given as an input. If the land height at any point is higher than that maximum height, then the drone cannot move in that direction. This is illustrated in the figure below. Here, the drone is at the bottom edge of the map (cannot move to directions **SW, S, SE**), and its maximum flying height is **100**. Therefore, the only possible moves for the drone are **W, NW, NE, E**.



- Drones use fuel to fly. Flying in a straight line causes uniform fuel consumption. However, if a drone needs to go upward as well when the next point on the map is higher than the current position, the fuel consumption increases. Our goal is to find a path for a drone such that it will be the most cost-effective based on the formula given below, which calculates the total cost of moving from a **source** point (x_s, y_s) to an **end** point (x_e, y_e):

$$cost_{s,e} = (Dist_{s,e} \times fuelCostPerUnit) + (climbingCostPerUnit \times heightImpact)$$

Distance between a **start** point (x_s, y_s) to an **end** point (x_e, y_e), denoted as $Dist_{s,e}$, is calculated as the Euclidean distance between two neighboring pixels given their x and y coordinates. The cost of fuel per distance unit and the cost of elevation (climbing) will be given as inputs. Finally, the impact of the point height $heightImpact$ will be calculated as follows:

$$heightImpact = \begin{cases} 0, & \text{if } height(s) \geq height(e) \\ height(e) - height(s), & \text{otherwise} \end{cases}$$

Mission Objective:

Find and draw the most cost-efficient path from a source to a destination point (pixel), which will be taken by a surveillance drone over a geographical area.

2.2 Input File Format

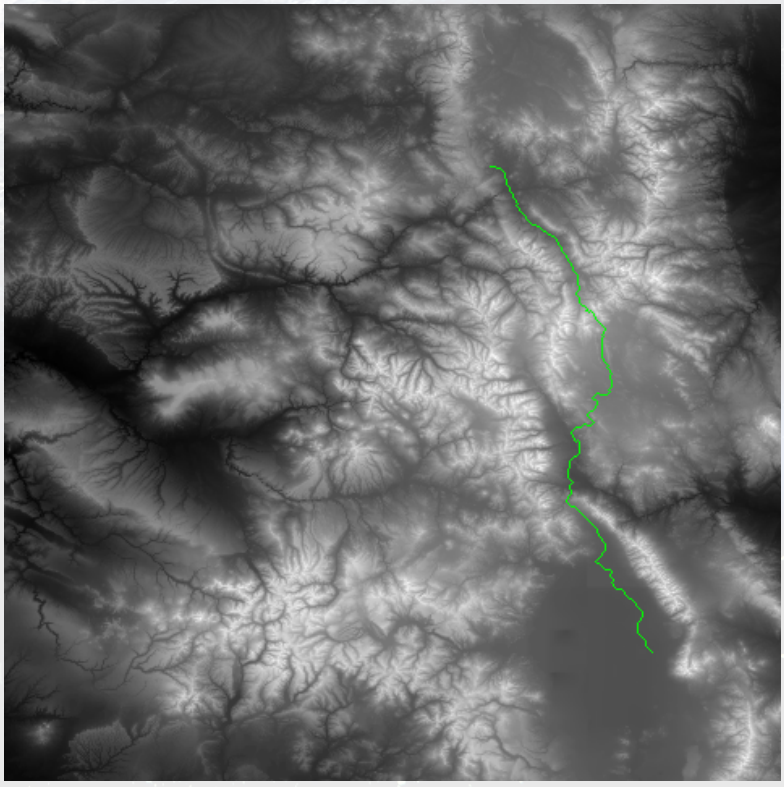
The input file with the 2D geographic elevation data will be given in the `.dat` format and its name (as well as all other input parameters) will be extracted from the input file given as *first command line argument*. Your program should parse the elevation values for each pixel given in the file and use it to initialize the *grid* instance variable of the *IMECEPathFinder* class. A sample input file is given as *Map_480x480.dat*. An excerpt from the sample input file is illustrated below (only numbers).

```
x coordinates:    0     1     2     3     4     5     6     7     8     9   . . .
-------------------------------------------------------------------------------------
y coordinates: |
       0       | 2537  2483  2475  2480  2518  2532  2480  2478  2431  2428 . . .
       1       | 2541  2549  2614  2700  2647  2746  2690  2621  2550  2487 . . .
       2       | 2525  2525  2640  2769  2802  2883  2856  2694  2631  2574 . . .
       3       | 2514  2505  2526  2614  2717  2715  2867  2836  2771  2644 . . .
       4       | 2506  2482  2480  2528  2518  2561  2586  2662  2654  2593 . . .
       .          .     .     .     .     .     .     .     .     .     .
       .          .     .     .     .     .     .     .     .     .     .
       .          .     .     .     .     .     .     .     .     .     .
```

Note that the pixel coordinates are to be considered as follows: $(0, 0)$ is the top leftmost point with an elevation of 2537. Its reachable neighbors are pixels $(1, 0)$ to the **E** with an elevation of 2483, $(1, 1)$ to the **SE** with an elevation of 2549, and $(0, 1)$ to the **S** with an elevation of 2541.

2.3 Expected Solution and Output Format

For the given sample input map file, the maximum flying height of 4000 meters, the source point with coordinates $(300, 100)$, and the destination point with coordinates $(400, 400)$, the expected most cost-efficient path (green pixels) is depicted in the figure on the right.



If you inspect the solution, you may observe that the height of the selected points (pixels) along the path never exceeds the maximum flying height of the drone. Moreover, while calculating the best path, the ability of a drone to move into only adjacent pixels (points) at each step is taken into consideration. A drone never takes a step from its current position to a point (pixel) that is not a direct neighbor of its current point. This explains why we don't see a straighter path.

You are expected to complete:

- ***drawGrayscaleMap()***, ***getMostEfficientPath()***, ***getMostEfficientPathCost()***, *and* ***drawMostEfficientPath()*** *methods of the* ***IMECEPathFinder*** *class.*

You are expected to produce two outcomes for this part of the assignment: an output file and a STDOUT output. The **DrawingPanel** class will be used for drawing the map in this assignment (a very useful Java class from Building Java Programs by Reges and Stepp). To draw a grayscale map of the given land area as illustrated in the first figure of this mission, you need to draw a grid using the **Graphics** object obtained by **getGraphics()** method from the **DrawingPanel** class. The map colors should be grayscale values in the range $[0, 255]$, scaled based on minimum and maximum elevation values in the given grid.

The output file must be named *grayscaleMap.dat* and it should contain a grayscale value for each pixel (**do not mark the path pixels in any special way**!) in a 2D format just like the input file. These greyscale values are the values used to draw the map. An excerpt from the expected output file is illustrated below:

```
102 98 97 97 101 102 97 97 93 93 94 ...
103 103 109 116 111 120 115 109 103 ...
101 101 111 122 125 131 129 115 110 ...
100 99 101 109 117 117 130 128 122 ...
100 97 97 101 101 104 106 113 112 ...
 .   .   .   .   .   .   .   .   .
 .   .   .   .   .   .   .   .   .
 .   .   .   .   .   .   .   .   .
```

Note that for the given input file, the minimum elevation value is 1326, while the maximum elevation value is 4334. Therefore, the grayscale value for the pixel $(0, 0)$ becomes 102 when scaled from 2537 to the range $[0, 255]$.

The expected STDOUT output format for the sample input is given below. **Note that your output must match the given output format for full credit**.

```
########## Mission 0 ##########
The most cost-efficient path's size: 346
The most cost-efficient path has a cost of: 1176.4943217537957
```

Sometimes there may be a case where it is impossible to reach the **destination** from the **source** point. If we assume the maximum flying height of 2000 instead of 4000 meters, as in our previous example, then there is no suitable path such that the drone can fly, as the elevations are too high and the drone gets stuck. In such a case, when there is no possible path, the following output should be printed instead:

```
########## Mission 0 ##########
ERROR PathNotFound: There is no most cost efficient path that meets all criteria!
```

3 Mission 1 - Finding the Path with the Least Elevation Change

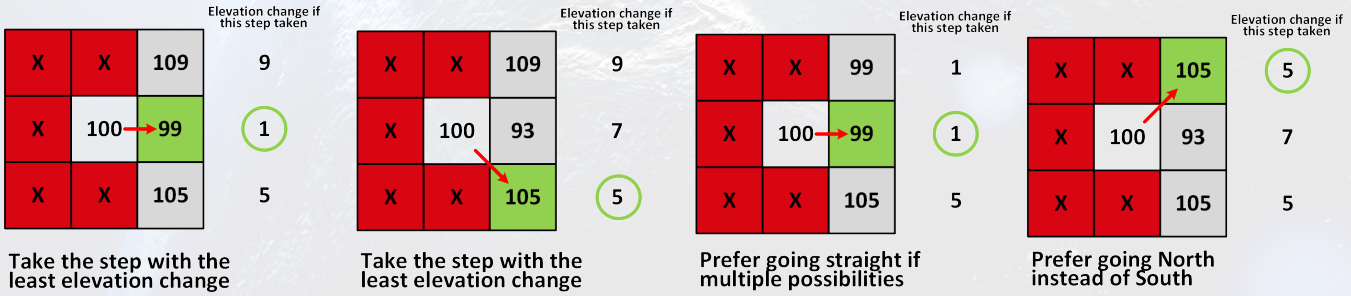
In certain scenarios, it can be beneficial to determine the most optimal route for traveling on land. For example, when traversing mountainous terrain on foot, you may want to identify the path that results in the least overall change in elevation with each step taken. This can be referred to as the path of least resistance and can be useful in a variety of applications where efficient land travel is necessary.

3.1 Background Information and Mission Objectives

In situations where there are many options to choose from, a "greedy" strategy involves selecting the option that appears to be the best choice in the current moment. An example of such a path is illustrated on the right. For this mission, you are expected to apply the following greedy approach to look for the path that results in the least change in elevation overall.

3011	2900	2852	2808	2791	2818
2972	2937	2886	2860	2830	2748
2937	2959	2913	2864	2791	2742
2999	2888	2986	2910	2821	2754
2909	2816	2893	2997	2962	2798

In the case of our 2D grid map, you can assume that an asset (e.g., a soldier, a drone, etc.) is positioned at a **source** point on the grid and that this asset needs to escape from the grid towards the **East** (right). At each step, a possible move is only into one of the three adjacent cells in the next column. During this greedy escape, the assumption is that the cell with an elevation closest to the current cell's elevation will be chosen as the next step, regardless of whether this involves moving uphill or downhill. This is illustrated in the scenario examples given below.



The figure above shows a few possible cases for choosing where to take the next step assuming the asset is at the center point with an elevation of 100 meters. Assume that in the case of a tie, we would always prefer to just go straight forward, and in the case of a tie between moving **NE** or **SE**, we would always prefer to move towards **NE** (up instead of down).

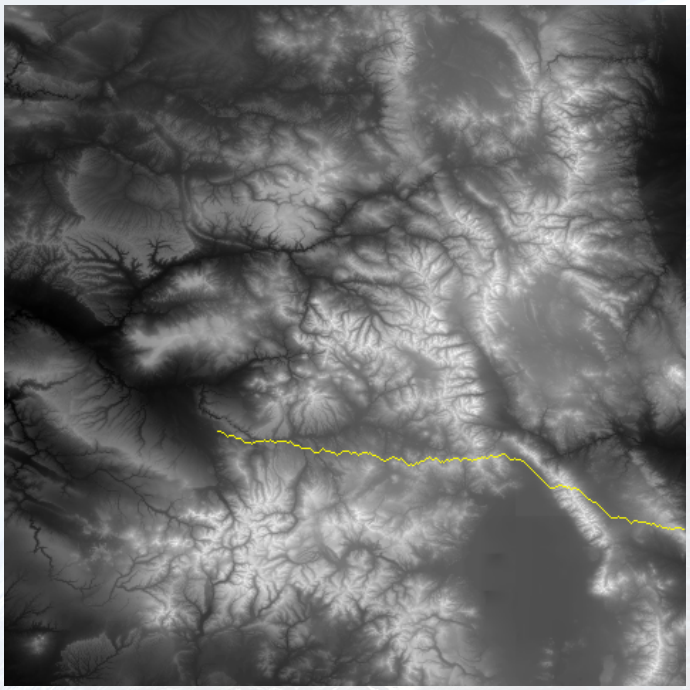
Mission Objective: ***Find and draw the path with the least change in elevation from the given source point towards the East.***

3.2 Input File Format

Your program should use the same *grid* obtained in the previous mission as the input.

3.3 Expected Solution and Output Format

For the given sample input map file and the source point with coordinates $(150, 300)$, the expected escape path with the least elevation change towards the **East** (yellow pixels) is depicted in the figure below.



You are expected to complete:

- ***getLowestElevationEscapePath()***, ***getLowestElevationEscapePathCost()***, *and* ***drawLowestElevationEscapePath()*** *methods of the* ***IMECEPathFinder*** *class.*

The expected STDOUT format for the sample input is given below. **Note that your output must match the given output format for full credit.** Also, note that the size of the path means the total number of points that are visited, including the source and destination.

```
########## Mission 1 ##########
The size of the escape path with the least elevation cost: 330
The escape path has the least elevation cost of: 12747
```

Must-Use Starter Codes

You MUST use **this starter code**. All classes should be placed directly inside your **zip** archive. Feel free to create other additional classes if necessary, but they should also be directly inside **zip**.

Grading Policy

- Submission: 1%
- Implementation of the methods: 90%
  - Regular Expressions for Input Data: 10%
  - Mission 0: 50%
  - Mission 1: 30%
- Output tests: 9%

Important Notes

- Do not miss the deadline: **Thursday, 01.06.2023 (23:59:59)**.
- Save all your work until the assignment is graded.
- The assignment solution you submit must be your original, individual work. Duplicate or similar assignments are both going to be considered as cheating.
- You can ask your questions via Piazza (https://piazza.com/hacettepe.edu.tr/spring2023/bbm204), and you are supposed to be aware of everything discussed on Piazza.
- You must test your code via **Tur^{3}Bo Grader** https://test-grader.cs.hacettepe.edu.tr/ (**does not count as submission!**).
- You must submit your work via https://submit.cs.hacettepe.edu.tr/ with the file hierarchy given below:
  - **b<studentID>.zip**
    - DatReader.java <FILE>
    - DrawingPanel.java <FILE>
    - IMECEPathFinder.java <FILE>
    - Main.java <FILE>
    - Point.java <FILE>
- The name of the main class that contains the main method should be **Main.java**. **You MUST use this starter code**. The main class and all other classes should be placed directly in your **zip** archive. Feel free to create other additional classes if necessary, but they should also be inside the **zip**.
- This file hierarchy must be zipped before submitted (not .rar, only .zip files are supported).
- **Usage of any external libraries is forbidden.**
- Do not submit any .jar or .class files.

Run Configuration

Your code will be compiled and run as follows:

Linux

```
javac -cp *.jar *.java -d .
java -cp .:* Main input_parameters.dat
```

Windows Powershell

```
javac -cp *.jar *.java -d .
java -cp '.;*' Main input_parameters.dat
```

Windows CMD

```
javac -cp *.jar *.java -d .
java -cp .;* Main input_parameters.dat
```

Academic Integrity Policy

All work on assignments **must be done individually**. You are encouraged to discuss the given assignments with your classmates, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in pseudocode) **will not be tolerated**. In short, turning in someone else's work (including work available on the internet), in whole or in part, as your own will be considered as **a violation of academic integrity**. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.

The submissions will be subjected to a similarity check. Any submissions that fail the similarity check will not be graded and will be reported to the ethics committee as a case of academic integrity violation, which may result in the suspension of the involved students.

References

[1] TÜBTAK Uzay, "İMECE", https://uzay.tubitak.gov.tr/tr/uydu-uzay/imece, Last Accessed: 17/04/2023.

[2] Wikipedia, "İMECE", https://en.wikipedia.org/wiki/%C4%B0MECE, Last Accessed: 17/04/2023.

[3] Directorate of Communications (DoC), "İMECE satellite launches into space", https://www.iletisim.gov.tr/english/haberler/detay/imece-satellite-launches-into-space, Last Accessed: 17/04/2023.

[4] Anadolu Ajansı, "Türkiye to launch new observation satellite on Tuesday", https://www.aa.com.tr/en/science-technology/turkiye-to-launch-new-observation-satellite-on-tuesday/2867773, Last Accessed: 17/04/2023.

[5] Stanford Nifty Assignments, "Nifty Assignment: Mountain Paths", http://nifty.stanford.edu/2016/franke-mountain-paths/, Last Accessed: 16/05/2023.