

# Advanced Programming and Data Structure

Second Semester Project – Non-linear Data Structures

***Knights of the Hash Table***

Engineering Department  
La Salle - Universitat Ramon Llull  
7 May, 2023

*But first, some words of wisdom...*

*“Generally, the craft of programming is the factoring of  
a set of requirements into a set of functions and data structures.”*

---

DOUGLAS CROCKFORD, JavaScript: The Good Parts, 2008

*“The difference between a bad programmer and a good one is  
Whether he considers his code or his data structures more important.”*

---

LINUS TORVALDS, Re: Licensing and the library version of git, 2006

*“Data dominates. If you’ve chosen the right data structures  
And organized things well, the algorithms will almost always be self-evident.”*

---

ROB PIKE, Notes on programming in C, 1989

*“It is better to have 100 functions operate  
on one data structure than 10 functions on 10 data structures.”*

---

ALAN PERLIS, Epigrams on Programming, 1982

*And now for something completely different...*

## Index

<b>1</b>	<b>Introduction.....</b>	<b>1</b>
<b>2</b>	<b>Requirements .....</b>	<b>2</b>
2.1	Concerning swallows and coconuts (Graphs) .....	2
2.1.1	Data .....	2
2.1.2	Functionalities .....	3
2.2	The ways of science (Binary search trees) .....	4
2.2.1	Data .....	4
2.2.2	Functionalities .....	5
2.3	Shrubberies (R-Trees) .....	6
2.3.1	Data .....	6
2.3.2	Functionalities .....	7
2.4	Of heretics and blasphemers (Tables) .....	8
2.4.1	Data .....	8
2.4.2	Functionalities .....	8
<b>3</b>	<b>Execution .....</b>	<b>10</b>
3.1	Start menu .....	10
3.2	“Concerning swallows and coconuts” menu .....	10
3.3	“The ways of science” menu.....	12
3.4	“Shrubberies” menu .....	14
3.5	“Of heretics and blasphemers” menu.....	15
<b>4</b>	<b>General considerations .....</b>	<b>18</b>
4.1	Groups .....	18
4.2	Language.....	18
4.3	Third-party implementations.....	18
4.4	Development .....	18
4.5	Plagiarism detection .....	18
<b>5</b>	<b>Evaluation.....</b>	<b>19</b>
5.1	Deadline .....	19
5.2	Delivery format .....	19
5.3	Report contents .....	19
5.4	Validation exam .....	20
5.5	Retake .....	20

## 1 Introduction

Finding a good work team is as important in the engineering world as it is anywhere else, as it has the potential of critically impacting the results we end up obtaining.

We nowadays have many different tools at our disposal that make it much easier to form teams, and even working remotely with them. This reality helps us overcome challenges that many others – including historical figures – have previously faced.

One of these figures was Arthur, son of Uther Pendragon, King of the Britons and sovereign of all England, who decided to form the Round Table<sup>1</sup> in the court of Camelot. This table, which was designed to avoid placing a figure as its head, was formed by the kingdom's most formidable knights.

Both the process of forming the Table and its day-to-day operations were rather complex, partly due to the time's lack of modern technological advances. Thus, we want to leverage said advances to develop **The Hashy Grail**, a software system that should allow us to figure out how work processes in the Middle Ages could have been optimised.

This document, which will be updated throughout the semester, defines the project's requirements, gives execution examples, provides relevant considerations and describes the evaluation system.

---

<sup>1</sup> Some secondary fonts dispute the table's shape.

## 2 Requirements

Life during the Middle Ages was rather difficult. Like in all stages in the long history of humanity, the intersection of different social constructs ended up generating more complex dynamics than just the sum of their parts. Still, we chose to split the requirements into different blocks, so as to reduce their size.

The following sections explain the functionalities to develop during this project, alongside descriptions of the data we will process.

### 2.1 Concerning swallows and coconuts (Graphs)

One of the most used communication systems during the Medieval Ages were homing birds, swallows being the most popular when it came to carrying secrets. Despite this, there is one fact regarding this aviary protocol that even in those times wasn't widely known: The difference between European and African swallows.

These two species have vastly different flight capacities, especially if we take situations into account where they must carry long messages or even relatively heavy items, such as coconuts to feed themselves with. Therefore, we are going to try and find ways of optimizing their journeys between points of interest.

#### 2.1.1 Data

For the previously described reasons we'll have a `.paed` file, containing the following set of information in plain text:

- **Number of points of interest:** The file's first line will contain an integer  $N$ , stating the number of points of interest we'll find in the following lines.
- **List of points of interest:**  $N$  text lines each containing the following fields, separated by semicolons:
  - **Id:** Integer identifying the point of interest. It will be unique but not necessarily consecutive. They will not be sorted either.
  - **Name:** The name of the point of interest. It can include more than one word. Uniqueness isn't guaranteed.
  - **Kingdom:** Name of the kingdom the point belongs to. It can include more than one word. Uniqueness isn't guaranteed.
  - **Climate:** The type of climate that can be found in the point of interest. The value can be `POLAR`, `CONTINENTAL`, or `TROPICAL`.
- **Number of known journeys:** The next line in the file will contain an integer  $J$ , stating the number of journeys we know of. Each journey is a measure of time and distance between two points of interest.
- **List of known journeys:**  $J$  lines of text each containing the following fields, separated by semicolons:

- **Point A:** Identifier corresponding to the first point of interest.
- **Point B:** Identifier corresponding to the second point of interest.
- **Time E:** Time in minutes a European swallow takes to fly from point A to point B (or vice-versa).
- **Time A:** Time in minutes an African swallow takes to fly from point A to point B (or vice-versa).
- **Distance:** Distance in kilometers between point A and point B, measured in a straight line.

### 2.1.2 Functionalities

This section's implementation must offer the following functionalities:

#### 2.1.2.1 Representation as a graph

Your code must define and implement the representation of a graph, while trying to **efficiently** store the given data.

The undertaking of this task is essential to develop the rest of this section.

#### 2.1.2.2 Kingdom exploration

To help us understand the geopolitical situation of our locations, this functionality will prompt the user to enter the identifier of a point of interest. Then, the program will show all the system's points of interest that belong to the same kingdom and can be reached from the entered one.

Regarding the list order, the program should prioritize those points of interest that can be reached without leaving the kingdom.

#### 2.1.2.3 Detecting common journeys

Next, we want to try and detect the most common journeys in aviary messaging, regardless of bird type.

Assuming that birds tend to fly in a straight line, find the set of journeys that connect all points of interest and minimize the total distance to traverse.

#### 2.1.2.4 Premium messaging

Lastly, we also want to offer a functionality to figure out the fastest way (i.e., requiring less time) to send a message from a given point of interest to another one, visiting intermediate points of interest if needed.

This feature must prompt the user to enter:

- The identifier of the origin point of interest.
- The identifier of the destination point of interest.
- Whether the swallow will fly while carrying a coconut or not.

The expected output is a list of points of interest to visit to reach the destination from the origin, in the order to follow. The program should also tell the user whether it would be more efficient to send a European or African swallow, as well as the total time and distance to fly.

The following list of considerations must be taken into account when implementing this functionality.

- European swallows can only visit polar or continental climates.
- African swallows can only visit continental or tropical climates.
- Swallows can't travel more than 50 kilometers without stopping at a point of interest when they're carrying a coconut. We can ignore the time they spend resting.
- If we don't have information about the journey between two points of interest, we'll assume that swallows can't directly fly between them.
- The existence of a solution isn't guaranteed.

## 2.2 The ways of science (Binary search trees)

One of the issues that most agitated the popular masses during medieval times was the very real possibility of witches choosing to hide amongst the population. However, succumbing to this fear would lead to making irrational decisions. Thus, every time a rumour spread about the existence of witches, knights would be put in charge of conducting a set of scientific experiments to validate or disprove them.

In this section we'll simulate said experiments.

### 2.2.1 Data

For the previously described reasons we'll have a `.paed` file, containing the following set of information in plain text:

- **Number of residents:** The file's first line will contain an integer  $N$ , stating the number of residents we'll find in the following lines.
- **List of residents:**  $N$  text lines each containing the following fields, separated by semicolons:
  - **Id:** Integer identifying the resident. It will be unique but not necessarily consecutive. They will not be sorted either.
  - **Name:** The name of the resident. It can include more than one word. Uniqueness isn't guaranteed.
  - **Weight:** Real number representing the resident's weight, in kilograms.
  - **Kingdom:** Name of the kingdom the resident lives in. It can include more than one word. Uniqueness isn't guaranteed.

### 2.2.2 Functionalities

This section's implementation must offer the following functionalities:

#### 2.2.2.1 Representation as a binary search tree

Your code must define and implement the representation of a binary search tree, while trying to **efficiently** store the given data.

The undertaking of this task is essential to develop the rest of this section.

#### 2.2.2.2 Basic functionalities

After representing the dataset information as a binary search tree, the program should be able to modify it during runtime. Persisting the changes to the file is NOT required.

Your solution must offer the options to add new residents (by prompting the user for all their information), as well as to remove them (by asking for their id).

#### 2.2.2.3 Visual representation

The program must provide a way to visualize the binary search tree. This visualization can be in the form of text, following the format presented in the execution examples. Optionally, you can implement a 2D graphical representation.

#### 2.2.2.4 Scientific identification

Thanks to the medieval era's advanced technology and wide set of scientific knowledge, we know that witches can be identified by comparing their weight to certain objects (yes, this is scientifically proven).

This functionality will prompt the user for the information of the object they want to compare the system's residents to. Specifically, they will enter:

- The object's name.
- The object's weight (real number, in kilograms).
- The object's category. It must be one of the following values: DUCK, WOOD, or STONE.

This functionality's behaviour will depend on the object's category.

- For DUCK objects, this functionality will show those residents that weigh exactly the same as the entered object.
- For WOOD objects, this functionality will show the first resident that weighs less than the object.
- For STONE objects, this functionality will show the first resident that weighs more than the object.



#### 2.2.2.5 Witch-hunt

Once witches were detected amongst the population, scientifically-versed knights would round them up in special raids.

This functionality will prompt the user to enter two real numbers: A minimum weight and a maximum weight. Next, the program will show the information about all residents with a weight comprised between these values (both included).

### 2.3 Shrubberies (R-Trees)

In current times we're used to seeing shrub borders, also known as shrubberies, in the labyrinthic gardens of luxurious castles. This is because they originated in medieval times, where they had a different use. Specifically, knights appreciated the work of master shrubbers so much that shrubberies were often used as rewards for the most dangerous quests.

Knights liked shrub borders so much, that shrubs in their natural state became harder to find. Thus, we've decided to develop a tool to simulate how these processes could have been efficiently carried out.

#### 2.3.1 Data

For the previously described reasons we'll have a `.paed` file, containing the following set of information in plain text:

- **Number of shrubs:** The file's first line will contain an integer `N`, stating the number of shrubs we'll find in the following lines.
- **List of shrubs:** `N` text lines each containing the following fields, separated by semicolons:
  - **Type:** Text indicating the shrub's geometrical shape. The value can be `CIRCLE` or `SQUARE`.
  - **Size:** Real number representing the length of the shrub's main dimension, in meters. This corresponds to the radius for `CIRCLE` shrubs, and to the side for `SQUARE` shrubs.
  - **Latitude:** Real number representing the first coordinate of the shrub's geometric centre, in degrees.
  - **Longitude:** Real number representing the second coordinate of the shrub's geometric centre, in degrees.
  - **Colour:** Hexadecimal RGB code representing the colour of the shrub's leaves. It will always be a shade of green.

### 2.3.2 Functionalities

This section's implementation must offer the following functionalities:

#### 2.3.2.1 Representation as an R-tree

Your code must define and implement the representation of an R-tree, while trying to **efficiently** store the given data.

The undertaking of this task is essential to develop the rest of this section.

**Note:** It's not mandatory to take the shrubs' shape and size into account when indexing the data structure, meaning that you can limit yourselves to working with their position. That being said, considering them will be positively graded.

#### 2.3.2.2 Basic functionalities

After representing the dataset information as an R-tree, the program should be able to modify it during runtime. Persisting the changes to the file is NOT required.

Your solution must offer the options to add new shrubs (by prompting the user for all their information), as well as to remove them (by asking for their position).

#### 2.3.2.3 Visualization

The program must provide a way to visualize the shrubs' indexation.

Specifically, we want to be able to see all the tree's internal divisions, as well as their contents. You're highly encouraged to implement a graphical representation, even if it's not mandatory.

#### 2.3.2.4 Search by area

To simulate local contexts, we'll want to be able to focus on a single region in the planet.

Thus, the user must be able to find the shrubs contained in a specific area, determined by two points (that is, two pairs of coordinates).

#### 2.3.2.5 Aesthetic optimization

To create beautiful shrub borders, shrubbers have to consider both the shape and colour of the shrubs they worked with. Our program will be able to check the ideal design for a given location.

First, the user will be prompted to enter the location they want to check (as a pair of coordinates), and an integer value  $K$ . Next, the program will find the  $K$  nearest shrubs and calculate:

- The most common type among them, `CIRCLE` or `SQUARE`.
- Their average colour.

## 2.4 Of heretics and blasphemers (Tables)

As we've seen, Knights of the Round Table held many different responsibilities. Occasionally, this made it harder for them to fulfil their duties with satisfactory results. It's because of this reason that the Episodic Royal Decree number XV was drafted, with the goal of reducing their workload.

What nobody expected was the ruthless efficiency of some of their successors. This may be because nobody expects the Spanish Inquisition, the organism that picked up the cruel responsibility of identifying and combating the kingdom's heretics.

To better understand its behaviour, we want to create a tool to efficiently simulate the process of heretic identification.

### 2.4.1 Data

For the previously described reasons we'll have a `.paed` file, containing the following set of information in plain text:

- **Number of suspects:** The file's first line will contain an integer  $N$ , stating the number of suspects we'll find in the following lines.
- **List of suspects:**  $N$  text lines each containing the following fields, separated by semicolons:
  - **Name:** Text containing the name of the suspect. It can include more than one word. Uniqueness is guaranteed.
  - **Number of rabbits:** Positive integer indicating the number of rabbits the suspect has seen in their lifetime.
  - **Occupation:** Text indicating the occupation of the suspect. Its value can be MINSTREL, KNIGHT, KING, QUEEN, PEASANT, SHRUBBER, CLERGYMAN or ENCHANTER.

### 2.4.2 Functionalities

This section's implementation must offer the following functionalities:

#### 2.4.2.1 Representation as Tables

Your code must define and implement the representation of tables/dictionaries/maps, while trying to **efficiently** store the given data.

The undertaking of this task is essential to develop the rest of this section.

#### 2.4.2.2 Basic functionalities

After representing the dataset information as tables, the program should be able to modify them during runtime. Persisting the changes to the file is NOT required.

Your solution must offer the options to add new suspects (by prompting the user for all their information), as well as to remove them (by asking for their name).

#### 2.4.2.3 Edict of Grace

During the process of suspect investigation, inquisitors would offer whole towns the possibility of confessing in exchange of reduced sentences.

Similarly, the program must allow the user to mark or unmark an existing suspect as a heretic. Suspects whose occupation is `KING`, `QUEEN` or `CLERGYMAN` can never be heretics.

#### 2.4.2.4 Final judgement

To finalize their decisions regarding suspects, inquisitors considered many factors and statistics. Our program must be able to perform some related lookups.

First, the user must be able to retrieve a suspect's information by entering their name.

Next, the program must have a functionality that allows the user to get a list of suspects in a range. Specifically, it must provide the information of the suspects that have seen a number of rabbits in the range entered by the user.

Finally, the user must be able to see a histogram with the distribution of heretics by occupation.

You can choose the format of the histogram, but it must show at least the number of heretics for each occupation as text. You're highly encouraged to implement a graphical representation, even if it's not mandatory.

#### 2.4.2.5 Considerations

By default, a suspect is a heretic if they have seen more than 1975 rabbits, except if their occupation is `KING`, `QUEEN` or `CLERGYMAN`.

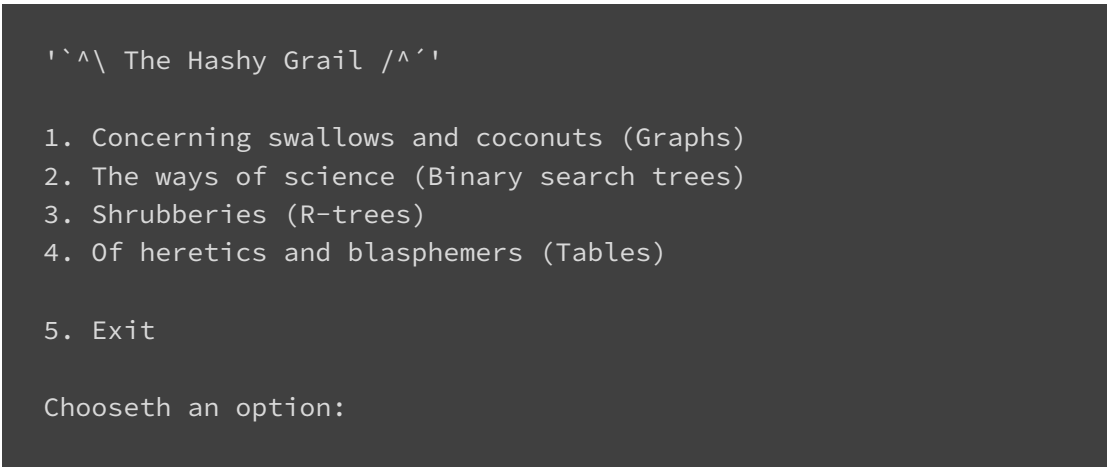
### 3 Execution

To standardize the more formal aspects of the project, we're providing you with the following execution examples that represent a user interacting with your code in the terminal.

**Note:** Datasets will be provided as a testing resource and to help in analysing results. These examples are not representative of the results you should see, as they only show the program's execution flow. Naturally, no projects with *hardcoded* data will be accepted.

#### 3.1 Start menu

Figure 1 shows the menu that will be presented to the user upon system start-up. It will allow them to choose a section of the project and to finish the execution.

A terminal window showing the start menu of 'The Hashy Grail'. The title bar reads '``^\\ The Hashy Grail /^\''. The menu lists five options: 1. Concerning swallows and coconuts (Graphs), 2. The ways of science (Binary search trees), 3. Shrubberies (R-trees), 4. Of heretics and blasphemers (Tables), and 5. Exit. Below the list, it prompts 'Chooseth an option:'.

```
``^\\ The Hashy Grail /^\'

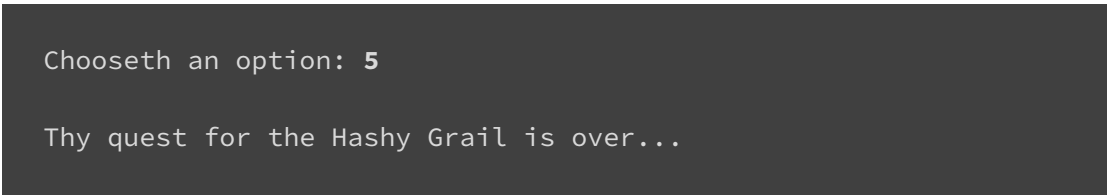
1. Concerning swallows and coconuts (Graphs)
2. The ways of science (Binary search trees)
3. Shrubberies (R-trees)
4. Of heretics and blasphemers (Tables)

5. Exit

Chooseth an option:
```

Figure 1: The program's start menu.

A farewell message should be displayed whenever the user chooses to exist the program, as seen in Figure 2.

A terminal window showing the exit message. It displays 'Chooseth an option: 5' followed by a blank line and then 'Thy quest for the Hashy Grail is over...'.

```
Chooseth an option: 5

Thy quest for the Hashy Grail is over...
```

Figure 2: The program's exit message.

Naturally, each one of the sections will have its own submenu. All of them will be detailed in the following pages.

#### 3.2 "Concerning swallows and coconuts" menu

Once the user chooses the first option in the start menu (Concerning swallows and coconuts), we'll show the submenu that can be seen in Figure 3.

```
Chooseth an option: 1

A. Kingdom exploration
B. Detecting common journeys
C. Premium messaging

D. Go back

Which functionality doth thee want to run?
```

Figure 3. "Concerning swallows and coconuts" menu.

As you can imagine, once the user chooses option D the program will return to the start menu, the one in Figure 1.

The behaviour of each one of the other functionalities is exemplified in Figure 4, ...

Figure 5 and ...

Figure 6.

```
Which functionality doth thee want to run? A
What place is to be explored? 42

42 - Castle of Camelot, Kingdom of Mercia (Continental Climate)

Its reachable places in the Kingdom of Mercia are the following:

7 - Castle of Aaargh, Kingdom of Mercia (Continental Climate)
13 - Bridge of Death, Kingdom of Mercia (Polar Climate)
12 - Gorge of Eternal Peril, Kingdom of Mercia (Polar Climate)
```

Figure 4. Interaction where the user runs the "Kingdom exploration" option.

```
Which functionality doth thee want to run? B

These are the most common journeys:
...
```

Figure 5. Interaction where the user runs the "Detecting common journeys" option. The rest of the format is left unspecified.

```
Which functionality doth thee want to run? C
What is thy origin? 13
What is thy destination? 7
Does the swallow carry a coconut? YES

The most efficient option would be to send an African swallow.
Time: 4213 minutes
Distance: 3742 kilometers
Path:
...
```

Figure 6. Interaction where the user runs the “Premium messaging” option. The rest of the format is left unspecified.

### 3.3 “The ways of science” menu

Once the user chooses the second option in the start menu (The ways of science), we’ll show the submenu that can be seen in Figure 7.

```
Chooseth an option: 2

A. Add resident
B. Remove resident
C. Visual representation
D. Scientific identification
E. Witch-hunt

F. Go back

Which functionality doth thee want to run?
```

Figure 7. “The ways of science” menu.

As you can imagine, once the user chooses option F the program will return to the start menu, the one in Figure 1.

The behaviour of each one of the other functionalities is exemplified in Figure 8, Figure 9, Figure 10, Figure 11 and Figure 12.

```
Which functionality doth thee want to run? A

Resident identifier: 42
Resident name: Roger the Shrubber
Resident weight: 75.19
Resident kingdom: Nni

Roger the Shrubber now accompanies us.
```

*Figure 8. Interaction where the user runs the "Add resident" option.*

```
Which functionality doth thee want to run? B

Resident identifier: 3

Sir Robin the Not-Quite-So-Brave was turned into a newt.
```

*Figure 9. Interaction where the user runs the "Remove resident" option.*

```
Which functionality doth thee want to run? C

    |--- Rabbit of Caerbannog (666, Kingdom of Aaargh): 1.5kg
    |
|--- Dingo (736, Kingdom of Anthrax): 43.85kg
|
|    |--- Brother Maynard (7, Kingdom of Antioch): 49.9kg
|
* Tim the Enchanter (1, Kingdom of Aaargh): 69.42kg
|
|--- The Old Man from Scene 24 (24, Kingdom of Mercia): 72.4kg
    |
    |    |--- Roger the Shrubber (42, Kingdom of Nni): 75.19kg
    |    |
    |    |--- A Famous Historian (1, Kingdom of UK): 77.7kg
    |    |
    |--- King Arthur (5, Kingdom of Mercia): 78.1kg
```

*Figure 10. Interaction where the user runs the "Visual representation" option.*

```
Which functionality doth thee want to run? D

Object name: Shrubbery
Object weight: 44.4
Object category: WOOD

1 witch was found!
    * Dingo (736, Kingdom of Anthrax): 43.85kg
```

*Figure 11. Interaction where the user runs the "Scientific identification" option.*



```
Which functionality doth thee want to run? E

Minimum weight: 44.4
Maximum weight: 73.3

3 witches were rounded up!
  * Brother Maynard (7, Kingdom of Antioch): 49.9kg
  * Tim the Enchanter (1, Kingdom of Aaargh): 69.42kg
  * The Old Man from Scene 24 (24, Kingdom of Mercia): 72.4kg
```

Figure 12. Interaction where the user runs the “Witch-hunt” option.

### 3.4 “Shrubberies” menu

Once the user chooses the third option in the start menu (Shrubberies), we’ll show the submenu that can be seen in Figure 13.

```
Chooseth an option: 3

A. Add shrub
B. Remove shrub
C. Visualization
D. Search by area
E. Aesthetic optimization

F. Go back

Which functionality doth thee want to run?
```

Figure 13. “Shrubberies” menu.

As you can imagine, once the user chooses option F the program will return to the start menu, the one in Figure 1.

The behaviour of each one of the other functionalities is exemplified in Figure 14, Figure 15, Figure 16, Figure 17 and Figure 18.

```
Which functionality doth thee want to run? A

Shrub type: CIRCLE
Shrub size: 1.742
Shrub latitude: 52.898379
Shrub longitude: -3.888177
Shrub colour: #388E3C

A new shrub appeared in Britain.
```

Figure 14. Interaction where the user runs the “Add shrub” option.

```
Which functionality doth thee want to run? B
```

```
Shrub latitude: 52.898379
```

```
Shrub longitude: -3.888177
```

```
The shrub was removed to be merged into a border.
```

Figure 15. Interaction where the user runs the "Remove shrub" option.

```
Which functionality doth thee want to run? C
```

```
Let there be a visualization...
```

Figure 16. Interaction where the user runs the "Visualization" option. The rest of the format is left unspecified.

```
Which functionality doth thee want to run? D
```

```
Enter the area's first point (lat,long): 52.905,-3.905
```

```
Enter the area's second point (lat,long): 52.725,-3.875
```

```
3 shrubs were found in the area:
```

```
* 52.898981, -3.889005: CIRCLE (r=1.171m) #43A047
```

```
* 52.894735, -3.895871: SQUARE (s=3.514m) #1B5E20
```

```
* 52.886559, -3.883433: CIRCLE (r=2.222m) #4CAF50
```

Figure 17. Interaction where the user runs the "Search by area" option.

```
Which functionality doth thee want to run? E
```

```
Enter the point to search (lat,long): 52.9,-3.9
```

```
Enter the amount of shrubs to consider (K): 7
```

```
Most common type: CIRCLE
```

```
Average colour: #689F38
```

Figure 18. Interaction where the user runs the "Aesthetic optimization" option.

### 3.5 "Of heretics and blasphemers" menu

Once the user chooses the fourth option in the start menu (Of heretics and blasphemers), we'll show the submenu that can be seen in Figure 19.

```
Chooseth an option: 4
```

- A. Add suspect
- B. Remove suspect
- C. Edict of Grace
- D. Final judgement (one suspect)
- E. Final judgement (range)
- F. Histogram by occupation
  
- G. Go back

```
Which functionality doth thee want to run?
```

Figure 19. "Of heretics and blasphemers" menu.

As you can imagine, once the user chooses option G the program will return to the start menu, the one in Figure 1.

The behaviour of each one of the other functionalities is exemplified in Figure 20, Figure 21, Figure 22, Figure 23, Figure 24 and Figure 25.

```
Which functionality doth thee want to run? A
```

```
Suspect's name: Tim
```

```
Number of rabbits seen: 1478
```

```
Suspect's occupation: ENCHANTER
```

```
A new suspect has been registered.
```

Figure 20. Interaction where the user runs the "Add suspect" option.

```
Which functionality doth thee want to run? B
```

```
Suspect's name: Tim
```

```
Tim has been publicly executed.
```

Figure 21. Interaction where the user runs the "Remove suspect" option.

```
Which functionality doth thee want to run? C
```

```
Suspect's name: Tim
```

```
Mark as heretic (Y/N)? Y
```

```
The Spanish Inquisition concluded that Tim is a heretic.
```

Figure 22. Interaction where the user runs the "Edict of Grace" option.

Which functionality doth thee want to run? **D**

Suspect's name: **Uther**

Register for "Uther":

- \* Number of rabbits seen: 12533
- \* Occupation: KING
- \* Heretic? No

Figure 23. Interaction where the user runs the "Final judgement (one suspect)" option.

Which functionality doth thee want to run? **E**

Minimum number of rabbits: **27**

Maximum number of rabbits: **154**

The following suspects have been found:

Sir Bedevere:

- \* Number of rabbits seen: 73
- \* Occupation: KNIGHT
- \* Heretic? No

Zeut:

- \* Number of rabbits seen: 27
- \* Occupation: MINSTREL
- \* Heretic? Yes

Brother Maynard:

- \* Number of rabbits seen: 100
- \* Occupation: CLERGYMAN
- \* Heretic? No

Figure 24. Interaction where the user runs the "Final judgement (range)" option.

Which functionality doth thee want to run? **F**

Generating histogram...

Figure 25. Interaction where the user runs the "Histogram by occupation" option.

## 4 General considerations

### 4.1 Groups

The project will be carried out in groups of 3 or 4. You must form them via the corresponding eStudy tool in the timeframes established by the subject's team.

Remember that teamwork isn't a Divide and Conquer strategy, but rather a way to learn by sharing perspectives and making important decisions as a group.

### 4.2 Language

The project can be implemented in any programming language, to be chosen by the group. If you decide to use C, it must compile, execute and work properly on the university servers.

### 4.3 Third-party implementations

The main data structures for the project (graphs, trees, tables...) must be implemented by the students. The use of third-party libraries or default data structures that implement important functionalities is strictly disallowed.

When it comes to auxiliary data structures (lists, stacks, queues...) it's highly recommended for students to implement them. If you decide to use already implemented ones you must justify your choices in the report to pass, **as well as analyse their asymptotic costs**.

Similarly, it's your responsibility to **disclose the use of any AI-powered tools** during the development of this project. Be aware of the potential downsides of relying on them as a source of information and include the required appendix in your report if you decide to do so. Using such tools without disclosing it falls under the university's definition of plagiarism (see 4.5).

### 4.4 Development

Once a group has been established, a project will be assigned to you in the university's Atlassian tools. You must use them in order to develop the project, regardless of the chosen language. This is especially important when it comes to the Bitbucket git versioning system.

**Note:** Being responsible when using the tools at our disposal is an essential part of engineering. Thus, we ask you to NOT include datasets in the version control system. Remember that you can use a `.gitignore` file to configure exceptions.

### 4.5 Plagiarism detection

Cheating makes learning harder for the student and denotes a lack of respect towards classmates who have spent time and efforts developing the code.

A project is classified as a **highly important** academic activity. Plagiarism, whether partial or total, from a classmate **or from the internet** will be considered a **premeditated action**. Therefore, applying [the university's copies regulation](#), it will be considered a **very serious misconduct**.

As is specified in the subject syllabus, partially or totally copying from previous submissions will also be considered as plagiarism, including in the case of students retaking the subject.

## 5 Evaluation

The project is split into four phases, each corresponding to a content unit. Each phase will start with the release of the corresponding statement and end in two evaluation activities: A checkpoint consisting of a meeting with the subject's team to validate the project's status, and a P2P self-evaluation questionnaire. For indicative purposes, a grade will be calculated in each phase based on these two elements.

The definitive evaluation will take place at the end of the semester and will consist of two separate marks: **Code** and **report**. The project grade will be equal to their weighted average, but each group member will get an individual grade by multiplying it by two factors in the [0.5, 1.5] range: **Mentoring** and **P2P**.

$$S2\_Project\_Grade = 0.6 \cdot Code\_Mark + 0.4 \cdot Report\_Mark$$

$$S2\_Grade = S2\_Project\_Grade \cdot Mentoring\_Factor \cdot P2P\_Factor$$

The mentoring factor will be determined by the subject's team depending on the supervision during project sessions, as well as the checkpoint results.

The P2P factor will be calculated from the results of the four self-evaluation questionnaires. In case a student doesn't answer one of them, the factor will be calculated from the remaining three. If two or more are left unanswered, the project grade will be **NP** and the student will have to retake the semester.

All phases of the code will have to satisfy their minimum requirements independently. Otherwise, the code mark will be at most **4**.

Both the code and report mark must be greater than or equal to **5** in order to pass. The project grade will be the lower of the two otherwise, and no individual factors will be applied to any group members.

### 5.1 Deadline

The deadlines for delivering each one of the parts are **May 21st 2023 at 23:55h** and **May 28th 2023 at 23:55h**, respectively. No submissions will be accepted after their deadlines.

### 5.2 Delivery format

The code must be delivered as a **ZIP** file containing the project folder alongside a README file in **TXT** or **MD** format, explaining with all the necessary details how to run it (programming language, IDE, versions, instructions...).

The Bitbucket repository should be in an equivalent state to the project you deliver in the eStudy. The report must be delivered in **PDF** format and should include the contents described in the next section.

### 5.3 Report contents

Your report should include the following sections:

- Cover page (with the group number and its members' full names and logins).
- Numbered table of contents.
- Justification for the chosen programming language, with advantages and disadvantages.
- For each of the project's data structures:
  - Structure design and justification for your technical decisions.
  - Explanation of the implemented algorithms.
  - Performance and result analysis of each algorithm.
  - Explanation of the used testing method.
  - Observed problems.
- Conclusions (from a personal perspective, but most importantly from a technical one).
- Bibliography (following ISO 690 or APA 7 rules).
- If you used any AI-powered tools, an appendix disclosing how you used them. It must include a description of your approach and a **complete list of tools and prompts**.

While the report is to be delivered at a later date than the code, this doesn't mean that you should start working on it at the very end. Take notes of your decisions, results...

Similarly, analysing the performance of your implementation throughout the project's development is highly recommended, so as to have time to fix any mistakes that you may detect.

The use of LaTeX to write the report is suggested. Its contents must be written in a **formal tone**, and they will be evaluated on their **quality, much more than their quantity**.

#### 5.4 Validation exam

To validate that the semester's contents have been understood even when working as a group, any student that delivers the project will have to take a final exam.

If the exam is passed with a **5** or more, it won't have any effect over the final grade. Otherwise, it will substitute it, meaning the student will have to retake the semester.

#### 5.5 Retake

If the project is failed (including not delivering it or failing the validation exam), a retake exam will take place during July's extraordinary call, where a minimum mark of **5** is required to pass.

Depending on the mentoring and the state of the project, groups that are close to passing may exceptionally be allowed a second delivery (with a maximum grade of **8**). The subject team has the power to allow only part of the members of a group to re-deliver the project, meaning that the rest would have to retake the semester with the corresponding exam.

Re-delivering the project and the retake exam are mutually exclusive mechanisms. If a student is allowed a second delivery but decides to take exam the latter grade will prevail.