Progetto di

Models and Algorithms for Data Visualization

Corso di Laurea in Ingegneria Informatica e Robotica – A.A. 2023-2024

Dipartimento di Ingegneria

docente

Prof. Giuseppe Liotta

Greek Myth Genealogy Project

studenti

361600	**Aurora**	**Traversini**	aurora.traversini@studenti.unipg.it
361596	**Omar**	**Criacci**	omar.criacci@studenti.unipg.it
362533	**Nicola**	**Cucina**	nicola.cucina@studenti.unipg.it

0. Contents

1. Introduction and domain analysis

This project aims to investigate the family relationships among the major Greek deities. This specific theme was selected since it was one of the subject of the Graph Drawing Contest (2016 edition) [2] and therefore it was particularly stimulating to attempt to offer an alternative representation to those presented in this particular contest, especially because it was thought challenging to try to represent the intricate genealogical relationships among the Greek deities through a family tree, given the sometimes *incestuous* nature of some of these ties between them.

The *target users* of this project are all those who, while not experts in the field, are passionate about Greek mythology and therefore interested in learning more about the complex relationships among the deities and learning more about some of the less famous characters. It was chosen to develop a web application so that it would be as accessible as possible to everyone and given the support found online for this kind of technology.

2. Requirements gathering

2.1 Data Modeling

2.1.1 Data Extraction

The input file is `greek-gods.csv` as provided by the GD Contest [2], which is organized as shown in the table 2.1:

NAME	FATHER	MOTHER	SEX	POPULARITY
Acis	Pan	Symaethis	M	1520000
Acrisius			M	28400
Aether	Erebus	Nyx	U	1850000
...	...	...	...	...

Table 2.1: First items from *greek-gods.csv*

By looking at the winner of the contest [3], it seemed appropriate to integrate some additional data, such as the *type* of each of the deities or a brief summary of who they are, as shown in table 2.2:

NAME-EN	NAME-EL	MAIN-TYPE	SUB-TYPE	DESCRIPTION
Aphrodite	$A\phi\rho o\delta\iota\tau\eta$	god	olympian	goddess of beauty, love, ...
Apollo	$A\pi o\lambda\lambda\omega\nu$	god	olympian	god of music, arts, ...
Ares	$A\rho\eta\varsigma$	god	olympian	god of courage, war, ...
...	...	...	...	...

Table 2.2: First items from *greek-gods-enrichment.csv*[4]

2.1.2 Data Cleaning and Transformation

A new attribute was created by transforming the *popularity* attribute from *numerical* to *categorical* using the four 25% percentiles of the distribution. In the table 2.3 are shown the various intervals.

POPULARITY	CATEGORY
$[0 - 1.080.000]$	0
$[1.080.000 - 2.740.000]$	1
$[2.740.000 - 8.690.000]$	2
$[8.690.000 - 196.000.000]$	3

Table 2.3: Quantization of the POPULARITY attribute

This is done to try to imitate the network visualization presented by the winner of the contest and because some of the values are in totally different scales, which would interfere with an effective visualization.

Then, a new dataset has been created by joining the two previously cited datasets, in particular a *LEFT JOIN* operation was performed, considering as the *left* table the dataset of the GD Contest and as the *right* table the other dataset, ending up with a result shown in Listing 1.
Since some of the entries in the dataset are not present in the additional dataset, it was necessary to manually add some information [5].

2.1.3 Output

Dataset type: Network

Nodes are greek deities described by the following attributes:

- *Name* (key): the greek name of the deity.
  - Cardinality: this is used as a key, therefore the cardinality of this attribute is the same as the number of items (117)
- *Sex*: categorical attribute with 3 values.
  - Male
  - Female
  - Unknown

Listing 1 Final dataset obtained

```
    {
      "id": "Acis",
      "sex": "M",
      "popularity": 1,
      "popularity-value": 1520000,
      "main-type": "god",
      "sub-type": "rustic",
      "description": "river god"
    },
    {
      "id": "Acrisius",
      "sex": "M",
      "popularity": 0,
      "popularity-value": 28400,
      "main-type": "human",
      "sub-type": "king",
      "description": "king of argos, father of Danae and
      grandfather of the famous demi-god perseus"
    },
    {
      "id": "Aether",
      "sex": "U",
      "popularity": 1,
      "popularity-value": 1850000,
      "main-type": "god",
      "sub-type": "primordial",
      "description": " god of light and the upper atmosphere"
    },
    ...
```

- *Popularity-value*: numerical value representing the number of Google results.
- *Popularity*: ordinal attribute, quantization of the *popularity-value* attribute, divided in 4 categories.
  - $[0 - 1.080.000]$
  - $[1.080.000 - 2.740.000]$
  - $[2.740.000 - 8.690.000]$
  - $[8.690.000 - 196.000.000]$
- *Main-type*: categorical value that represents the class of the deity.
  - God
  - Titan
  - Human
  - Personification
- *Sub-type*: categorical value that specifies further the class of the deities. There are a total of 19 different subtypes.
  - Twelve different Gods
  - Four different Humans
  - Two different Titans

Links represent parental relationship, it is described by:

- *Source*, using the name of the deities, it represents the parent node in the relationship.
- *Target*, using the name of the deities, it represents the child node in the relationship.

Dataset type: Table

By ignoring the parental relationship between the various deities, we can create a *Table* where all of the *items* are the deities with all the same attribute as before. This is done so that we can better describe some of the attributes in the dataset that do not rely on the information provided by the links in the network.

2.2 Task Modeling

The tasks have been modeled by imagining what could be of use to users interested in researching and exploring differents aspects and/or relationships present in the Greek mythology.

2.2.1 Input tasks

- Discover more background information about the given deity.
- Determine whether two deities have a parental relationship.
- Find the close relatives of a given deity, defined as its parent and children nodes.
- Find the common ancestor between two given deities.
- Find the different types and hierarchical relationship between the deities.
- Find the most popular deity.
- Find the sex distribution of the deities.

2.2.2 Output tasks

Each input task is defined as an {action, target} pair:

- Discover more background information about the given deity.
  - *Locate* the node <Name>
  - *Identify* the attributes associated with the node <Name>
- Determine whether two deities have a parental relationship.
  - *Locate* the link between the two nodes <NameA> and <NameB>
  - *Identify* the link between node <NameA> and node <NameB>
- Find the close relatives of a given deity.
  - *Locate* the nodes adjent to <Name>
  - *Compare* all the links incident on <Name> and return only the nodes adjent to <Name>
- Find the common ancestor between two given deities.

  - *Locate* the node <Ancestor> that has two distinct paths leading towards <NameA> and <NameB>
  - *Identify* the paths with minimal lengths that start in <NameA> or <NameB> and end in <Ancestor>

- Find the different types to which the various deities belong to.
  - *Explore* the various paths in the *Types* tree, explained more in detailn in section 3.1.2

- Find the most popular deities.
  - *Locate* the deities with higher popularity values
  - *Compare* the values of the popularity attribute

- Find the sex distribution of the deities.
  - *Summarize* the distribution of the *sex* attribute

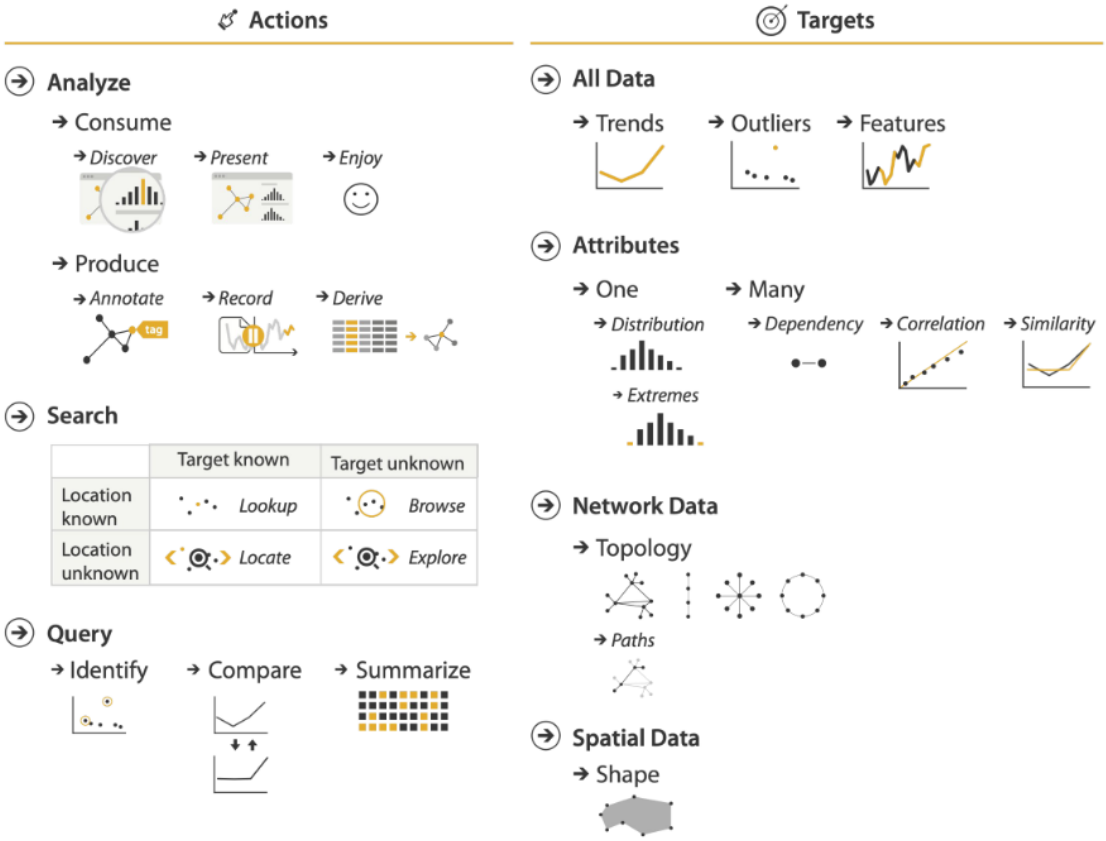


Figure 2.1: Task modeling *actions* and *targets*

3. Design

3.1 Visualization and Interaction Design

It was decided to create a *visual language* composed of different *visual idioms*, each one specialized on a different aspect of the dataset and lending itself to implement different kinds of tasks. The idioms are organized in a page in a sequential fashion so that they do not interfere with one another.

3.1.1 Visual idiom - Family Tree

This is a node-link representation of the dataset.

Marks

- Nodes: Circle with a label showing the deity name
- Links: Connection between nodes

Channels

- Nodes:
  - Categorical attribute popularity → Circle size
  - Categorical attribute main-type → Circle color hue
- Links: Categorical attribute sex → Segment color hue

Semantics

- Nodes: The size of the circle encodes the *popularity* of the deity, the bigger the size the higher the popularity, while the color represents the *main-type* of the deity

- Links: The color of the segment represents the *sex* of the descendant

Interaction paradigm: Full view

Interaction operations

- Selection
- Filtering
- Abstract Zooming via Tooltip

3.1.2 Visual idiom - Sunburst

Data: This data structure is a *Tree* where the deities are organized in a heirchical structure such that:

- in the first layer the *main-type* is used to separate the various classes,
- the *sub-type* is used in the second layer to further specify each of the categories,
- finally, each deities' name is placed in its corresponding place.

Marks

- Nodes: Interlocking areas
- Links: Implicit in the adjacency of the areas along the radial direction

Channels

- Nodes:
  - Color hue → Different categories of deities
  - Angle → Sum of the deities in a given category
  - Angular proximity → Siblings
  - Distance from the center → Depth in the tree
  - Color saturation → Depth in the tree

Semantics: The size of the circular sectors is proportional to the number of deities contained in each subtree.

Interaction paradigm: Top-down

Interaction operations

- Abstract zooming
- Reconfiguring

3.1.3 Visual idiom - Barchart

Data: The name of the deities is used as *key*, while the *popularity* is the numerical attribute that we want to represent.

Marks: horizontal lines with a label showing the popularity of the deities.

Channel

- Position → Name of the deity
- Length → Popularity of the deity
- Color hue → Highlights Zeus

Semantics: The longer the bar, the higher the popularity of the given deity.

3.1.4 Visual idiom - Piechart

Data: Only the *sex* attribute is used, therefore the categorical value used as *key* is the sex of the deities, either male, female or unknown, while the numerical value represented in percentage is the sum of all the deities of the given sex.

Marks: Interlocking areas with a label showing the associated sex.

Channel

- Angle → Number of deities of the given sex
- Color hue → Different sexes

Semantics: The bigger the area, the greater the number of deities with that particular sex.

3.2 Architectural and Technological choices

Here are explained the criteria that led to the selection of some technologies over others, taking into account the trade off between *Expressiveness* (i.e., the degree of customization) and *Effort* (i.e., the amount of code to be written).

3.2.1 HTML5, CSS3, JavaScript

The combination of HTML5, CSS3, and JavaScript was selected for the front-end to ensure a responsive, interactive, and accessible web application. HTML5 provides the structural framework, CSS3 for styling, and JavaScript handles the interactive elements. This choice also allows for the *separation of concerns*, ensuring mantainability and scalabily of the project.

The HTML page is stored in a file named *main.html*, in which each visual idiom is encapsulated within designated *div* elements, ensuring that they do not interfere with one another, and various JS scripts are used to draw the various idioms.

3.2.2 D3.js

Referring to the trade-off between *Expressiveness* and *Effort*, it was decided to use a library that fell into the category of *Low-level Building Blocks*, in order to make more sophisticated and appealing visualizations, but paying the price at the level of the amount of code that needs to be written in comparison to the category of the *Low-level Building Block* or a *Chart Template*. However, the advantage of such libraries over a *Graphic Library* lies in the ability to have access to some utilities (i.e. a building block) which serve a particular purpose and can be used in combination with other utilities from the same library. The most famous library in this category is D3.js, which is why it was decided to use this one.

D3 (or D3.js) is a "free, open-source JavaScript library for visualizing data. Its low-level approach built on web standards offers unparalleled flexibility in authoring dynamic, data-driven graphics" [6].
D3.js facilitates the creation and manipulation of dynamic and interactive data visualizations in web browsers heavily relying on SVG for rendering visual elements. This library provides a declarative approach to building visualizations, allowing developers to bind data to the Document Object Model (DOM) and apply data-driven transformations to the document.

It provides granular control over the final visual output, making it possible to create custom, intricate visualizations like family trees, sunburst charts, and bar charts that are tailored to the complex nature of Greek mythological data.

3.2.3 D3-dag

D3-dag, an extension of D3.js, is a JavaScript library designed for creating and visualizing directed acyclic graphs (DAGs), maintained by Erik Brinkman [7].

A family tree can be represented as a DAG where each person is a node, and edges represent parent-child relationships. D3-dag simplifies the process of creating such visualizations by providing a set of tools and methods tailored for working with DAGs.

Some key features of D3-dag are:

- **Layout Algorithms**: D3-dag provides layout algorithms that automatically position nodes in a way that reflects the structure of the DAG. This is crucial for visualizing family trees, where the arrangement of nodes often follows a hierarchical pattern.
- **Integration with D3 Ecosystem**: since D3-dag is built on D3.js, it seamlessly integrates with the broader D3 ecosystem.
- **Customization**: D3-dag allows for customization of the visualization, enabling the user to control the appearance of nodes, edges, and the overall layout. This flexibility is valuable to create a family tree with a specific design or styling.

In conclusion, the decision to choose D3-dag was influenced by the library's ability to handle hierarchical structures and its ease of use for creating DAG-based visualizations. It provides a solid foundation for building an interactive and aesthetically pleasing family tree visualization on the web.

To use D3-dag, as shown on the READ.me file in the github repository[7], it's only necessary to load it using *unpkg* by including it in the body of the HTML.

3.3 Algorithm engineering

Here are explained more in detail algorithmic techniques to compute the representation of the visual idiom defined in the previous phases with the goal of optimising the efficiency of the perception process of the user. All the algorithmic techniques that will be briefly illustrated below refer to the two idioms that use a Graph data structure since both the Bar Chart and the Pie Chart idioms do not require any in depth analysis to be drawn.

3.3.1 Finding a common ancestor through LCA algorithm

In the *Family Tree*, given the previously established task, we wanted to give the user the opportunity to select two representative nodes of two Greek deities in

order to be able to effectively identify the path to their common ancestor. To achieve this, the following algorithm was therefore used.

- *LCA Computation* : used to find the lowest common ancestor of two selected nodes [8].
- *Path Retrieval* : it invokes the *getPath()* function twice, each time with the LCA node and one of the selected nodes, to retrieve the paths from the LCA to each selected node.
- *Path Concatenation* : it concatenates the obtained paths into a single path.

Algorithm 1 Lowest Common Ancestor

Input: A graph $G = (V, E)$ and two vertices v and u.

Output: A List of nodes w that are the lowest common ancestor between v and u.

1: Color the ancestors of v in red by visiting the graph with a DFS(v) only using inward edges
2: Color the ancestors of u in black by visiting the graph with a DFS(u) only using inward edges
3: Find the intersection V' between V_{red} and V_{black}
4: Consider the graph G' induced by the nodes in V'
5: Every node in V' with $outdeg = 0$ is a LCA candidate

The overall time complexity of the algorithm is dominated by the DFS operations in the first two steps, making it $O(V + E)$. The subsequent steps involve the set intersection which costs $O(V')$ and the creation of the induced subgraph which takes $O(V' + E')$. The identification of LCA candidates has a time complexity of $O(V')$. Overall, the algorithm's efficiency depends on the structure of the graph, with the worst-case scenario being $O(V + E)$.

After this, the path retrieval is done through a *getPath()* function, which is a recursive function thagt determines the path from a given ancestor node to a specified node in a DAG.

Algorithm 2 getPath(ancestor, node)

Input: A graph $G = (V, E)$ a vertex v and its ancestor u.
Output: a path p starting from v that ends in u.

```
1: currentNode = v;
2: if currentNode.name == u.name then
3:     return nodes and links
4: end if
5: for parent in currentNode.getParentNodes() do
6:     path = getPath(u, parent)
7:     if path != null then
8:         add currentNode and the link to parent to the path
9:         return path
10:    end if
11: end for
12: return null
```

The time complexity of the *getPath()* function depends on the depth of the node in the graph. If the data structure was a *Tree*, the time complexity would be $O(log(V))$, but in a *DAG*, since each node has up to two parents, the complexity increases to $O(V+E)$(*we are performing a DFS from the leaf towards the LCA*).

The overall time complexity of this procedure involves invoking the LCA algorithm which has s time complexity of $O(V + E)$, two *getPath()* functions that cost $O(V+E)$ and the concatanation of the two paths which is a $O(1)$ operation, where V is the number of vertices and E is the number of edges in the graph. Therefore, the total time complexity is $\approx O(V + E)$.

3.3.2 Sugiyama for Layered Graph Drawing

Usually, family trees do not form a *tree* as in the data structure used in graph theory. However, since a parent must be born before their child, an individual cannot be their own ancestor, and thus there are no loops. Therefore, family trees form a *directed acyclic graph*.
Since this particular graph is a DAG, it was decided to use the *Sugiyama framework* in order to obtain a Layered Drawing.

- **Cycle removal**: No need to apply any algorithm during this first step as the family tree structure is inherently already a DAG, so it doesn't contain cycles.

- **Level assignment**: The heuristic that has been selected is the *Longest-path Method*. This approach crafts an aesthetically pleasing visualization, with the least number of layers, although it doesn't assure the minimal possible width. Additionally, since the heuristic's complexity is $O(V + E)$, it is also very efficient given the graph size.
- **Crossing reduction**: In this step it was chosen to use the *Decross Two Layer* heuristic instead of the optimal decrossing method. This was done because the layout was too large, so fully minimizing decrossings optimally was prohibitively expensive while the selected heuristic is very fast and performs an efficient minimization.
- **x-coordinates assignment**: The simplex method at this stage has been choosen due to its favorable balance between the aesthetics of the visualization and computational speed.

3.3.3 Drawing a Sunburst

As previously explained in section 3.1.2, organizing the deities by the combination of their *main-type* and *sub-type* creates a Tree data structure, and as we have seen in class we can use a modified version of the Radial Tree Algorithm.

Algorithm 3 Radial-Tree-Draw-Rec

Input: a vertex v, an angle β_v and a level L.

if v has children **then**
    for each u child of v **do**
        Compute the angle β_u using $l(v)$ and $l(u)$
        Radial-Tree-Draw-Rec(u, β_u, $L + 1$)
    end for
end if
draw v on the level L in the circular sector defined by β_v

Algorithm 4 Sunburst Draw

Input: a graph $G = (V, E)$
Output: a representation of G

Post-order visit of the graph to assign to each vertex the value $l(v)$, the number of nodes contained in the subtree rooted in v
Radial-Tree-Draw-Rec(root, 2π, 0)

4. Greek Mythic Genealogy

The web page that was developed can be found at the following link Github link. What follows are some snapshots taken from the page to show the various visual idioms and the interaction that has been implemented.
The first visual idiom encountered is the **Family Tree** where all the deities are represented as nodes of various sizes according to their *popolarity*.

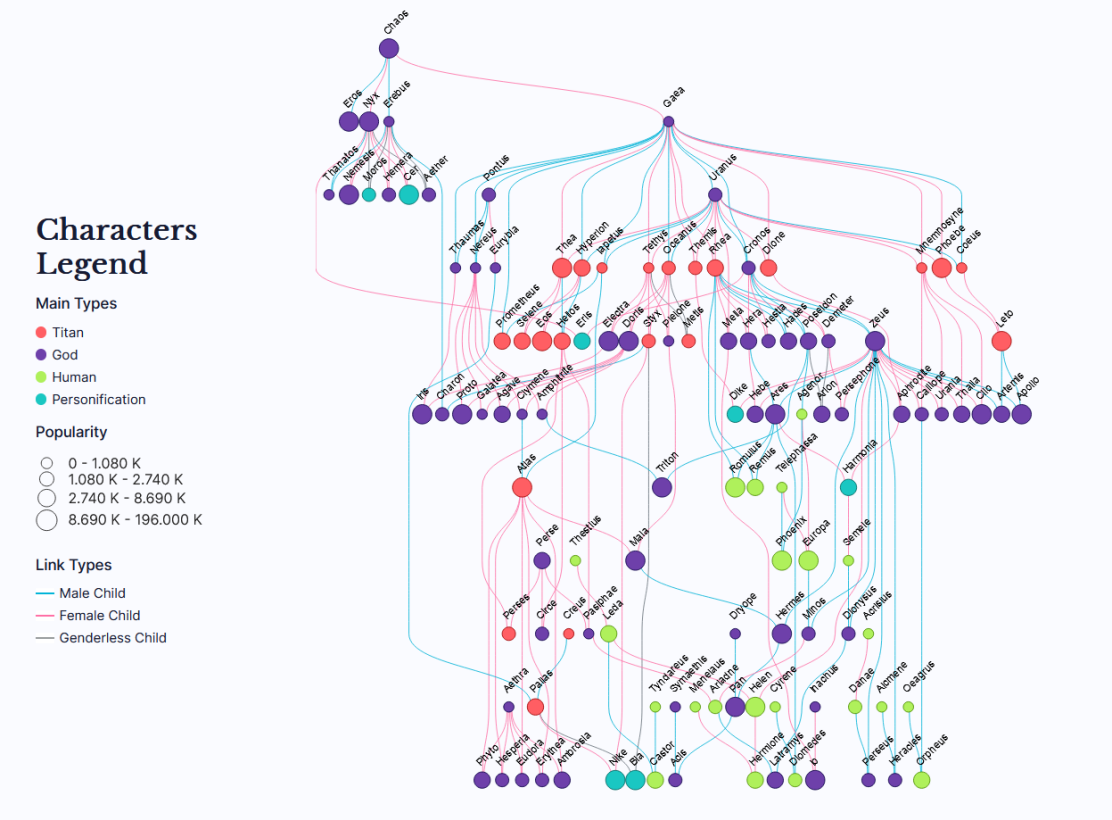


Figure 4.1: Family tree representation of the deities

By hovering the cursor over a node, a tooltip is opened which shows in detail the information relative to the deity.

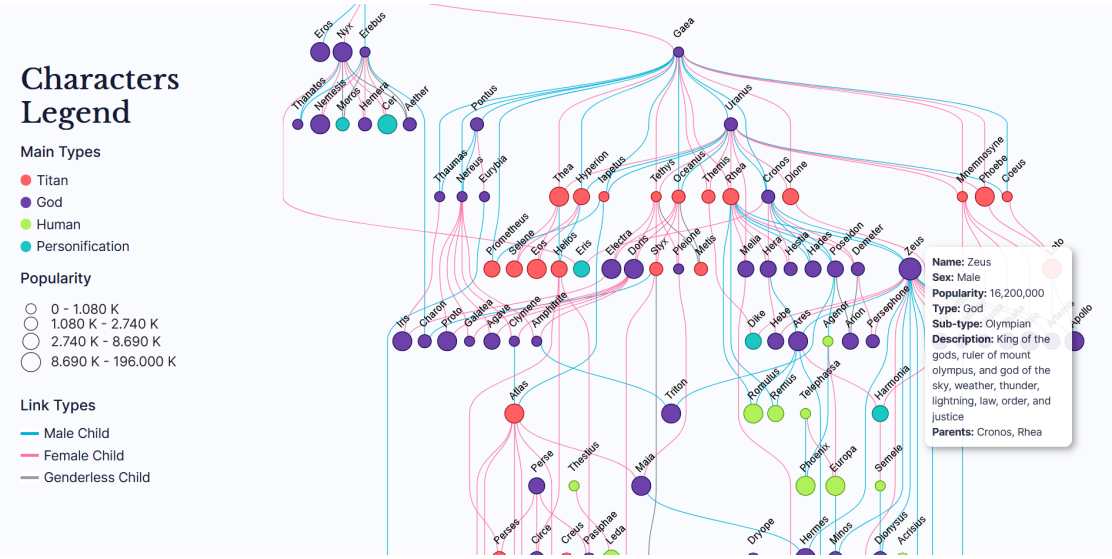


Figure 4.2: Tooltip shown while hovering over a node

Clicking a node filters the graph by desaturating the colors of all the nodes expect the ones that are adjacent to the selected node.

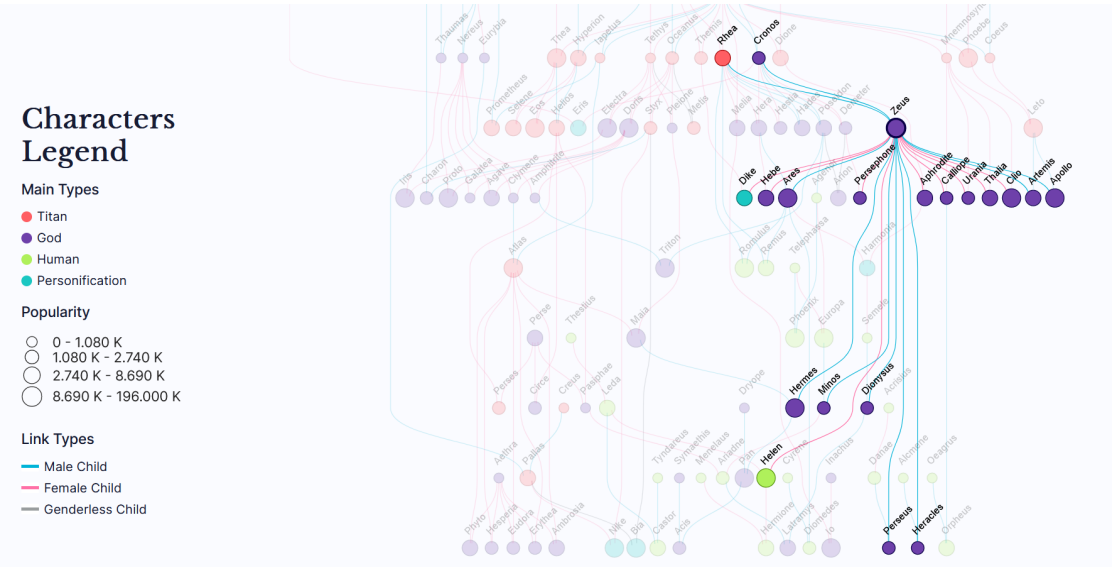


Figure 4.3: The effect of the node selection

If two nodes are selected, the LCA is computed and the path that join the nodes to the LCA is highligthed.

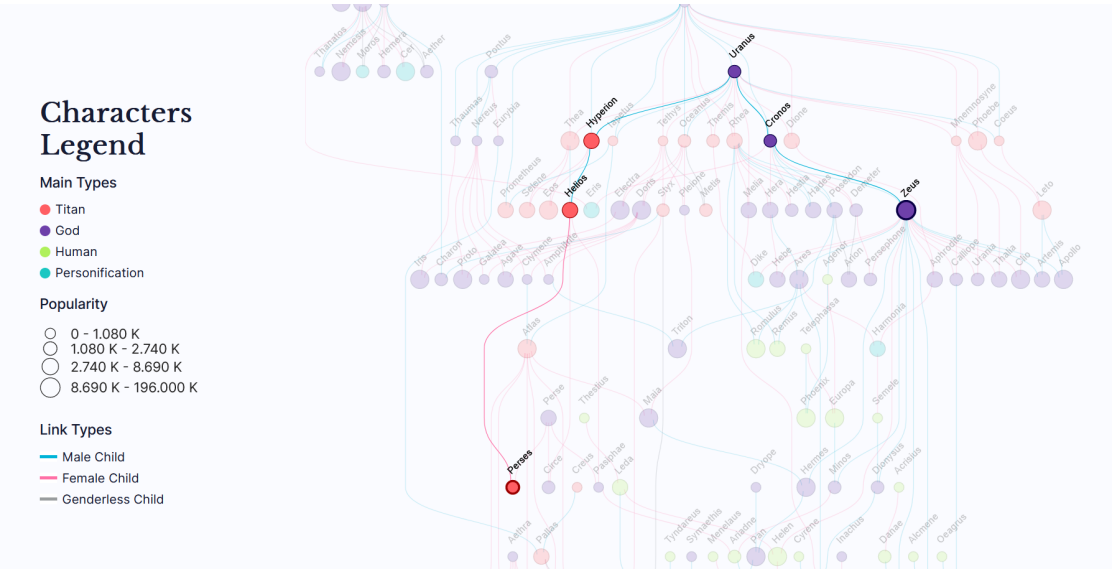


Figure 4.4: The paths connecting the nodes to their LCA

The second visual idiom encountered is the **Sunburst** diagram where all the deities are organized into *main types* and *subtypes*, with each category portrayed as a circular sector.

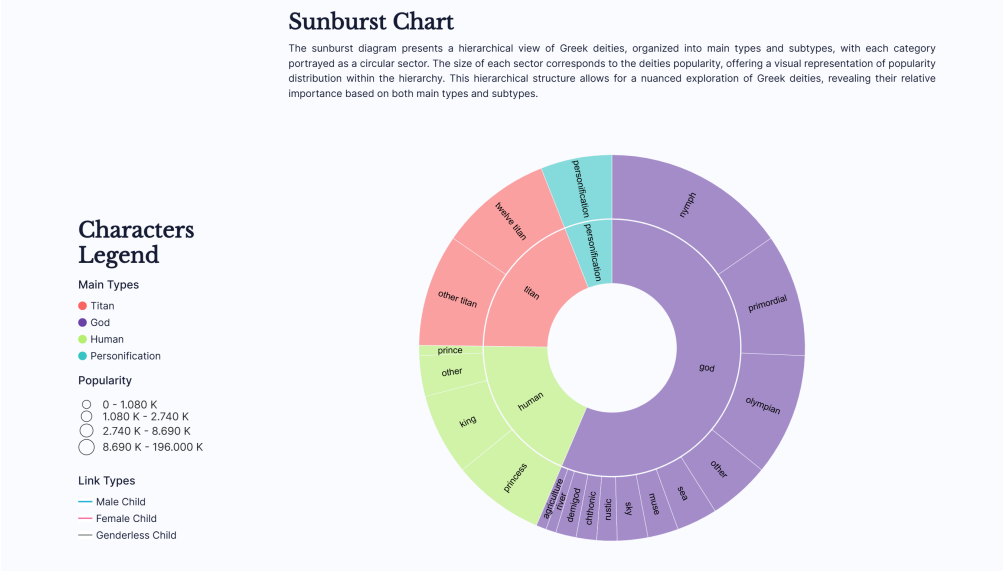


Figure 4.5: Sunburst diagram presenting a hierarchical view of Greek deities

If a category is clicked, a different configuration of the tree is shown going deeper and deeper into the hierarchy.

(a) (b)

Figure 4.6: Two different configurations of the Sunburst

At the end of the page we find the **Bar Chart** and **Pie Chart** which show the top 30 most popular deities and the *sex* distribution among all the deities.

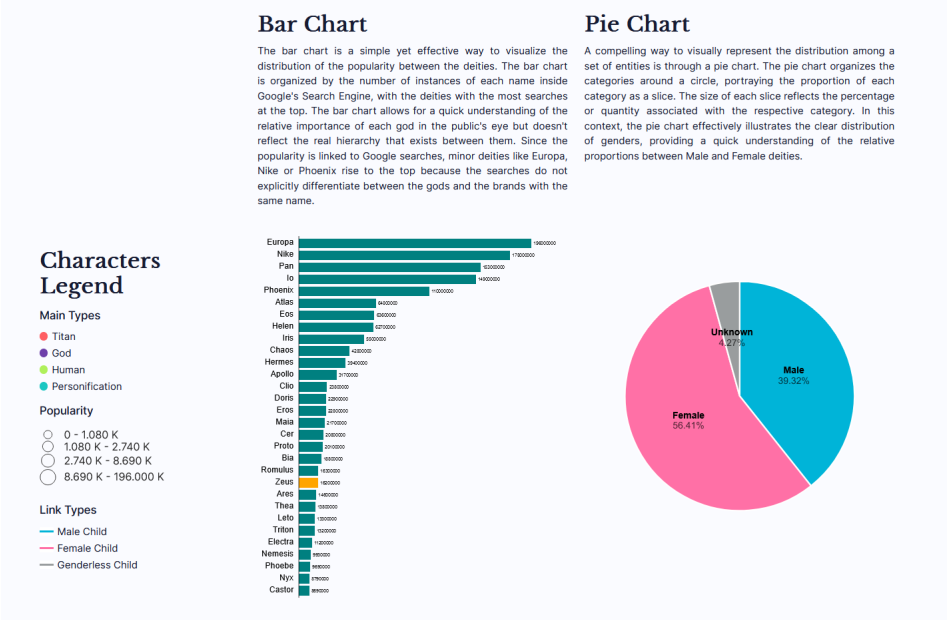


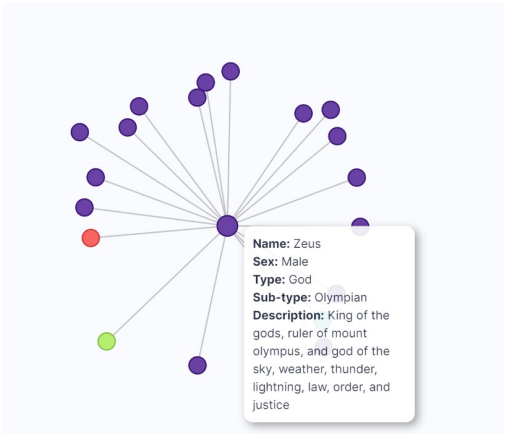
Figure 4.7: Bar Chart and Pie Chart

5. Future Developments and Complications

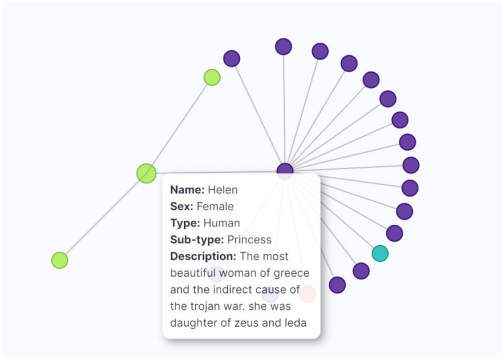
In the initial design stages, it was planned to implement another visual idiom, namely a *Force Directed Graph*, to represent direct kinship relationships between nodes.
In such an idiom, the initial configuration would have Zeus as the initial node, connected via arcs to its parents and children, as shown in 5.1a.

The idea is that with a *Direct Walk* interecation paradigm, it would be possible to explore the graph with a *bottom-up* approach: in fact, by clicking on a node, its child nodes and parent nodes (if there are any) would be added to the visualization, connected via appropriate arcs, as shown in the snippets 5.1.

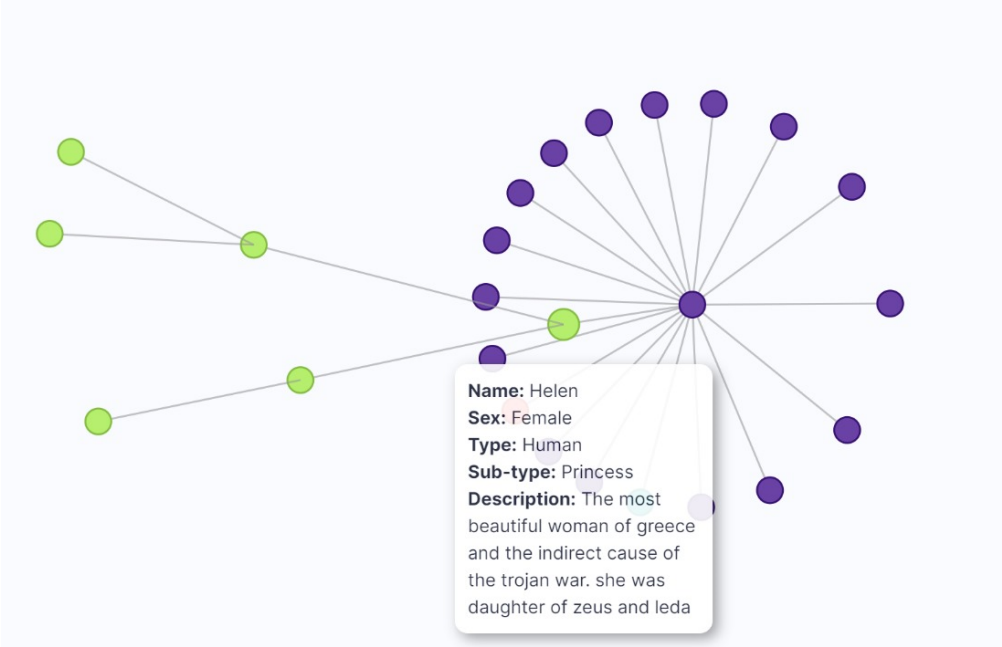
However, during the implementation of such an idiom, a preliminary User Study was carried out, bringing together people in the age range of 20 to 30 years, with no prior knowledge of the nature of the project nor of Greek mythology.
The result of this User Study showed that, due to the nature of the predetermined tasks, the *Force-Directed* idiom was not as functional as the main idiom of this project, namely *Family Tree*.



(a) Start configuration centered in Zeus



(b) Graph after clicking a node



(c) Graph after clicking a few nodes

Figure 5.1: Various screens from the *Direct Walk* representation

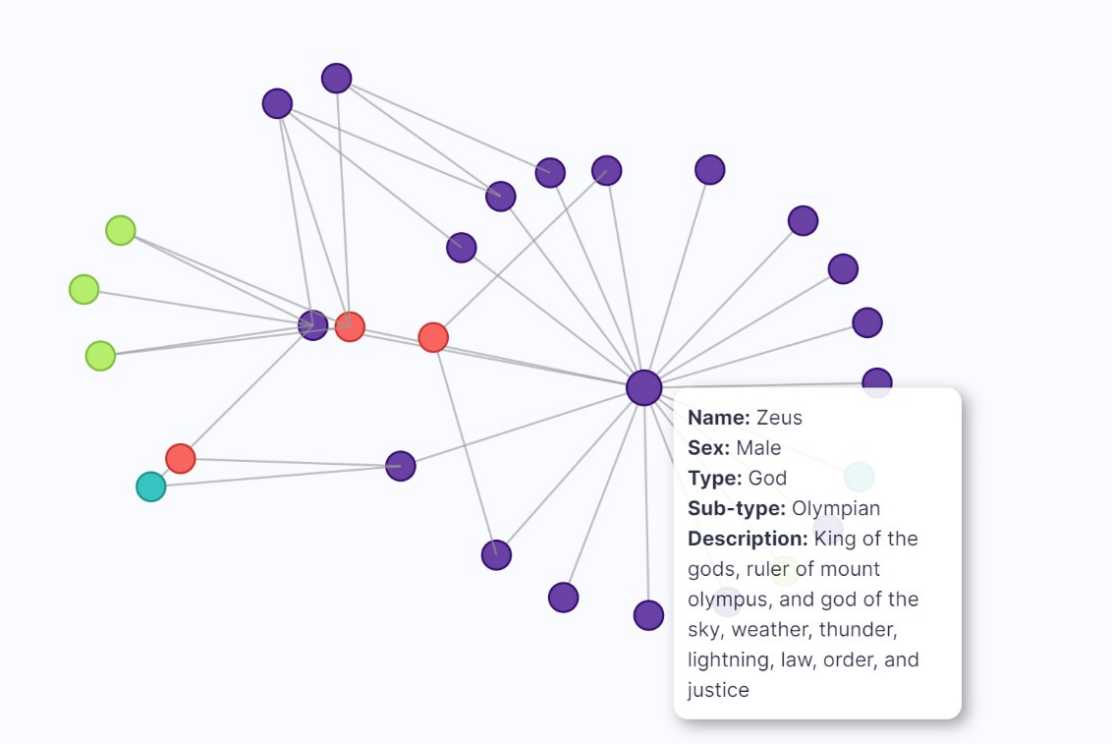


Figure 5.2: Graph with many crossings after clicking some nodes

As shown in picture 5.2, the *Direct Walk* is not suited for handling a Family Tree since the numerous crossings that we could end up with.

Among the potential future developments, one possibility could also be the implementation of some form of interaction with the *BarChart* idiom, which was designed to display the popularity of the top 30 deities.

The reason for limiting the representation to only the top 30 is purely a matter of convenience and effectiveness in visualization, considering that representing the entire original dataset would be too chaotic. Therefore, one could consider enhancing this visualization by adding a way to display the popularity of the other deities not represented.

6. Bibliography

[1] Twelve Olympians

[2] 23rd Graph Drawing Contest - Graph Drawing 2016

[3] Jonathan Klawitter and Tamara Mchedlidze (Karlsruhe Institute of Technology) winning submission

[4] Greek mythology data

[5] Names of the entries that dont match between the two datasets

[6] d3.js

[7] erikbrinkman/d3-dag: Layout algorithms for visualizing directed acyclic graphs

[8] LCA algorithm