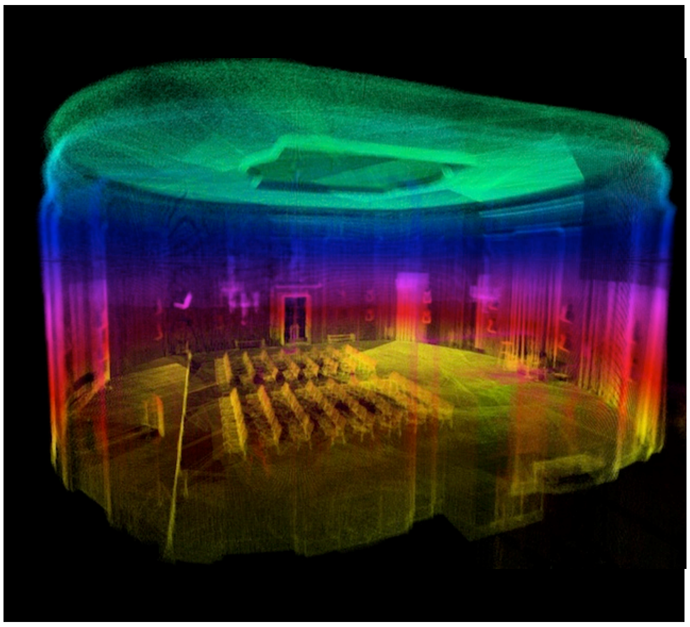
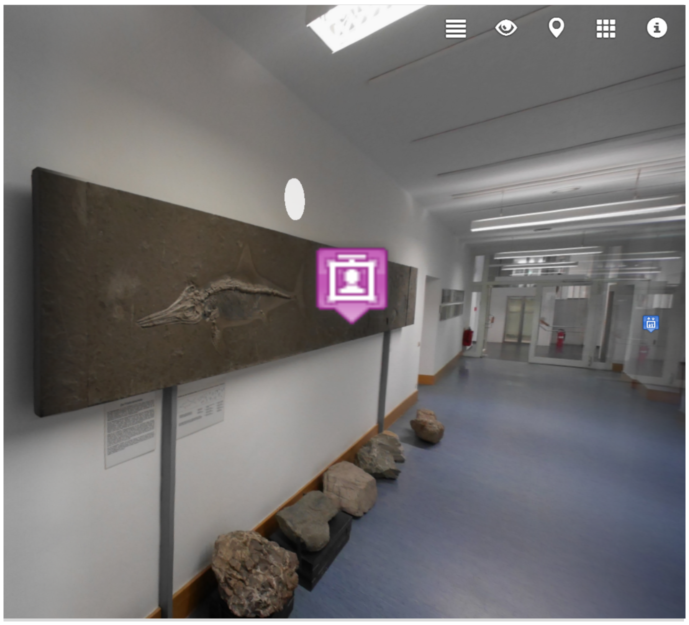
NAVVIS





Code Challenge

Code Matters

Beautiful is better than ugly.
Explicit is better than implicit.

Simple is better than complex.
Complex is better than complicated.

Flat is better than nested.
Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.
Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one - and preferably only one - obvious way to do it.

Now is better than never.

Although never is often better than *right* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

+

Task 1: Coding in JavaScript

Write a JavaScript / HTML program that lets the user open a CSV file from disk, displays it in a text box, processes it, and displays the output in another text box.

To read the input: The CSV file basically contains a 2D matrix of numbers, where each line holds a single row, e.g.: "`2<delim>4<delim>99<delim>\n`". The delimiter can be either space or a single comma (‘,’). Write the output in the same format as the input.

Once the input data is read, your application should perform filtering of “bad” values. Any entry of the matrix is “bad” when it has a value of 0 (zero). The application should now replace these bad values and compute the true value by interpolating it from the surrounding values, i.e., from the spatial neighbors of the entry in the matrix.

Write the matrix with the replaced values to the output text field.

Remarks:

- You may not use any 3rd party code or libraries. Use of the standard libraries is allowed and encouraged.

+

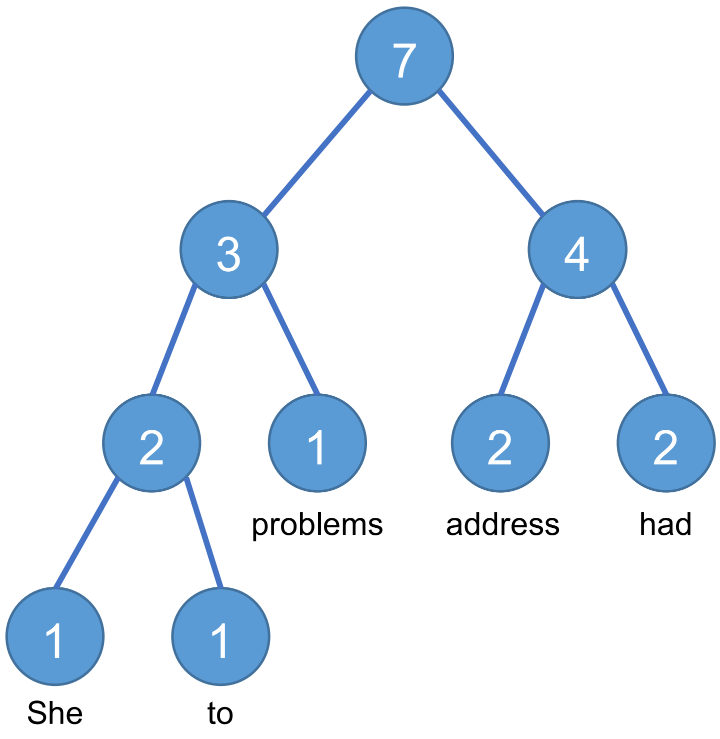
Task 2: Coding in Java

Write a Java-based program (command line) that reads text from a file, splits it into words at spaces and newline characters and constructs an (unbalanced) binary tree where each leaf node represents a unique word.

The tree construction shall start by creating a node for each unique word, where a node has a field to keep track of the occurrence count. The algorithm starts with the two least occurring nodes and creates a parent node. The parent node gets assigned an occurrence count that is the sum of the word occurrences. The process then repeats, i.e., it locates the two nodes with the least occurrence count, creates a parent node, and so on, until all nodes are part of the tree.

Finally, print the tree to the console (very basic output is sufficient).

For example, the text “She had had to address address problems” results in this tree (note that there are multiple variants):



Remarks:

- You may not use any 3rd party code or libraries. Use of the standard libraries (java.*) is allowed and encouraged.

}]); //End

When done

Please send us your solution and the code packaged into a single .zip archive.