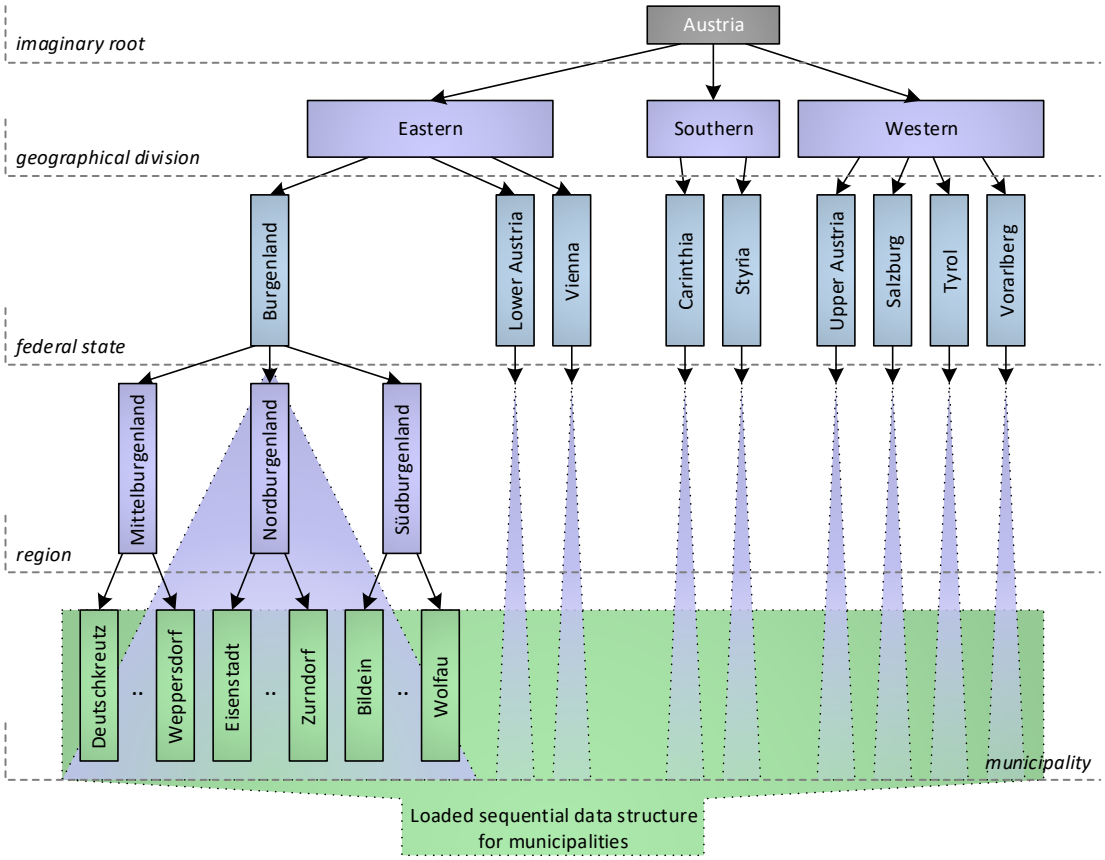
SP from Algorithms and Data Structures 1 - Second level

Assignment

Complete and/or modify the loaded data on the municipalities of Austria to **create a hierarchy of territorial units of Austria** as follows:

- The root is imaginary and represents Austria (let's call such a node $n^{Austria}$),
- the sons of the node representing the root represent the geographical division (let's call one such node n) into eastern, western and southern Austria,
- the sons of each such node n^{geo} are the nodes representing federal states (let's call one such node n^{state}),
- the sons of each such node n^{state} are nodes representing regions (let us call one such node n^{reg}),
- the sons of each such node n^{reg} are nodes representing municipalities (let us call one such node n^{muni}).

An example of part of such a hierarchy for Burgenland is shown in the following figure (similar hierarchies are also found for the other federal states - the "narrow" purple sub-hierarchies):



Pay attention to an ***efficient way of loading the hierarchy***. Information about higher territorial units can be found in the file **country.csv** and the assignment of municipalities to regions can be found in the file **municipalities.csv**. The files are encoded in UTF-8. You can edit the input file as you wish, but the required data must not be lost (i.e. no municipality or relevant column can "disappear", but you can convert the files to another format or remove unnecessary data in advance). If you modify the input data, describe this modification in the documentation.

Implement, that:

- the node n^{muni} contains data from the loaded sequential structure from SP level 1 - ensure that there is no duplication of data.
- hierarchy nodes $n^{Austria}$, n^{geo}, n^{state}, n^{reg} store cumulative population data (record both male and female population) in each year from the nodes that represent their sons.
- it is possible to apply predicates from level 1 to the hierarchy nodes representing any territorial units. Consider another predicate `hasType` that returns `true` if the type of the territorial unit (municipality/region/federal state/geographical division) is the same as the specified type. Thus, there will be four predicates in total.

Implement the iterator of the hierarchy of territorial units. The iterator must be able to be moved manually (by entering an option from the terminal or by pressing a button in the GUI application) to:

- the superior territorial unit or to
- the chosen son of the current territorial unit.

Consider a hierarchy with a root identical to the current node of the iterator. ***Allow the algorithm from level 1 to run over this hierarchy*** with one of the four predicates - the choice is specified by the user.

Example: from the level Austria navigate to Eastern Austria and the Federal State of Burgenland and then list all municipalities containing the string "kreutz", then navigate back to Eastern Austria and list all its Federal States.

Tips

- As a hierarchy, you can use either the multipath hierarchy tree implementation from the labs, or the hierarchy nodes can be represented by the territorial units themselves.
- Think about the organization of the data and its codes before implementing loading.
- Use the same object you loaded into the sequential structure in SP level 1 (e.g., `Municipality`) to hold the cumulative data at the hierarchy nodes - just modify its name appropriately (e.g., `TerritorialUnit`) so that it still makes sense in level 1.
- Implement the territorial unit hierarchy iterator as a forward iterator (if you use the hierarchy from the labs, you can use the forward iterator implementation from the labs). If you have correctly implemented the object from level 1, then it will be sufficient to call this algorithm with the parameters: a copy of the current iterator and an iterator on the "last element" of the traversal (`nullptr`) of the hierarchy.

Evaluation

Demonstration of functionality (optional)

- 5 points.
- The heap must be provably clean at the end of the demonstration of functionality.
- Takes place in the 9th week of the semester.
- Points are not dependent on producing documentation.
- Due to the originality check, points are conditional on uploading the SP to Moodle.

During defense

- You must implement level one functionality (score at least a 1p for level one) to get a level two score.
- The complexity of data loading is O(n) and it is assumed that during loading each row of each file is required to be processed at most once and elements are inserted into the data structure efficiently (complexity O(1)) – max. 5p.
- The iterator over the hierarchy of territorial units is functional – max. 3p.
- The algorithm for processing the territorial units (defined in level 1) is at this level:
    - implemented without change, existing code is reused, iterators from this level are sent to it as parameters – max. 7p;
    - implemented directly for use in this level. However, it uses a traversal (in the form of a recursive method) that takes as parameters a predicate in the sense of level 1 – max. 4p;
    - implemented directly for use in this level, inflexible (code duplicity) – max. 2p.

Documentation

The documentation is submitted with the final version of the SP at the end of the semester. The documentation must be prepared according to the published requirements (document SP rules).

In particular, make sure that the documentation thoroughly explains how you retrieve your data. Also include the complexity of the loading. If you have modified input files, describe your motivation and also these modifications.