Cartography

source: Wikimedia Commons, user: Themightyquill

Jan Rochel <jan@rochel.info>

deadline: around the end of the term

> In this assignment you are asked to define your own graph data-type and derive a graph representing country adjacencies from a textual description. Then you are to implement a graph algorithms, such as minimum spanning tree and a shortest path algorithm.

1 (Map description)**.** Using parser combinators you will define a parser for the natural language description of the countries. From the description your program should extract a number of facts. Each fact describes an adjacency of two countries. Consider as an example:

> The following five countries are part of a region known as Central Asia. Kazakhstan shares in the south a border with Turkmenistan, Uzbekistan, and Kyrgyzstan. Uzbekistan borders on Turkmenistan and Kyrgyzstan as well Tajikistan. The biggest lake in the area (which is actually considered a sea by some) is the Caspian Sea to which both Kazakhstan and Turkmenistan have access. The Aral sea is located on the border of Uzbekistan and Kazakhstan. Lake Balkhash is completely enclosed by Kazakhstan.

You are to write a parser to extract from such a description a list of facts representing country adjacencies:

- Kazakhstan shares a border with Turkmenistan
- Kyrgyzstan shares a border with Kazakhstan
- Uzbekistan shares a border with Turkmenistan

The parser you write should account for a number of different ways of expressing adjacencies and should also be able to parse but ignore irrelevant (irrecognisable) sentences and additional comments like 'This country is beautiful.' It is compulsory to implement the parser using a parser combinator library, such as the one provided with this assignment.
In the end you will get more points if your parser is versatile and accepts a larger language, but it might be good advice to first implement it for a minimal subset of English and extending it only after solving the rest of the assignment.
Give at least two parsable .txt-files with a description of any region of your choice (fictional or real) demonstrating the versatility of your parser.

2 (Graph library)**.** In this exercise you are asked to implement a module in which you define your own adjacency-list based graph representation using *Data.IntMap*. An *IntMap* is a data structure that represents a partial mapping from *Int*s to some value *a*. Define your graph data-structure such that

- every node in the graph is represented by a unique *Int*
- every node is mapped to a list of nodes representing adjacencies (the nodes it is connected to)
- to every node is associated with a value of some type (like for instance a *String* for the country it represents). You need to parametrise your graph data-type over that type.

Define functions for graph creation/manipulation as needed for the following two exercises. In the end there should be at least functions for adding/removing nodes, and edges and a constant representing the empty graph. **Hint:** As suggested by the API documentation you might want to import *Data.IntMap* using the keywords *qualified* and *as*.

Bonus: Use *Data.IntSet* for the adjacency lists.
Bonus: Implement all of graph-related exercises for weighted graphs. You might want to consider using the heap-package for the algorithms.

3 (Country adjacency graph)**.** Write a function which generates a graph from descriptions obtained in exercise 1.

4 (Shortest path)**.** Outside of the EU, crossing borders between two countries may cost time and money. Find for a given graph and two countries within the graph a path that minimises this cost. Algorithmically: implement a shortest path algorithm. This can be achieved by a simple breadth-first search (or Dijstra's algorithm for weighted graphs) which may proceed as follows:

1. We start with a list of paths of a single path consisting only of starting point itself.
2. For each of the nodes adjacent to the current endpoints of the paths you can derive a list of new paths with the respective node added as the new endpoint.
3. Remove the old endpoints from the graph.
4. As soon as any of the endpoints is the target point a shortest path has been found.

Remark: This description describes an implementation that uses accumulating parameters for the paths. Try to find an implementation that does not rely on accumulating parameters. You might want to consider starting from the destination point.
Bonus: Make a more efficient variant of the breadth-first search by starting the search from the starting point and the destination simultaneously.

5 (Minimum spanning tree)**.** Imagine a telecommunication company wants to build a network such that all countries are connected but connect as few countries as possible in order to save costs. Algorithmically speaking: you need to find a minimum spanning tree for a given graph. More explanations to come soon.