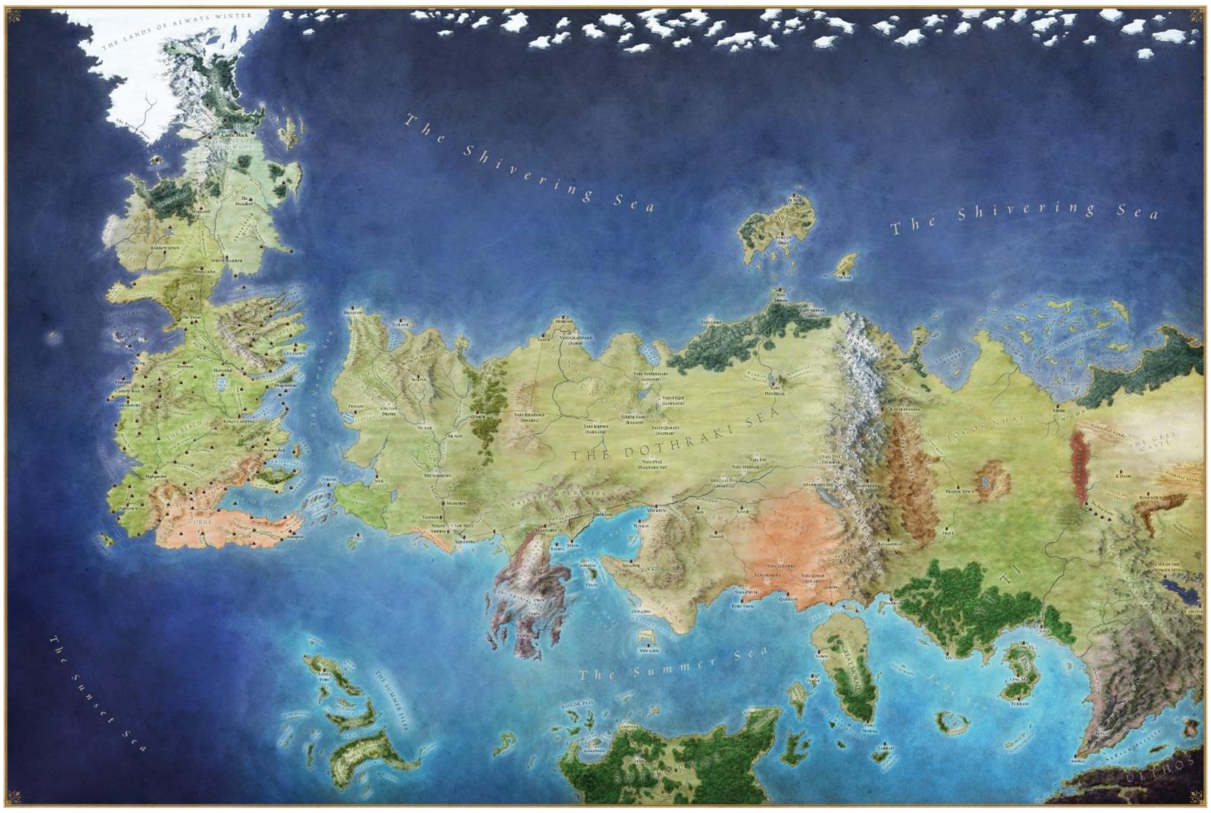
Data Structures and Algorithms 2

CA Exercise 2 – A Song of Ice and Fire (Game of Thrones) Route Finder

---

"Create a route finder for the known world of A Song of Ice and Fire (Game of Thrones) series."

The objective of this team CA exercise is to create an interactive application that can search for and retrieve various routes between specified towns/cities in the "known world" as depicted in George R R Martin's *A Song of Ice and Fire* book series, which is the basis for the TV series *Game of Thrones*.



This nicely detailed version of the map of the known world can be found here. Note that you can find and use any version of this map that you like for this CA exercise. For instance, you might prefer to find/use a "blank" map and then just show the recommended route between the two specified towns/cities.

The application should display the map of the known world in the first instance, and then allow the user to specify the starting point and a destination in some appropriate way (e.g. clicking on the map with the mouse, or selecting from a drop down list, or using a dialog box, or similar). The route (or routes) should then be suitably reported to the user, and graphically illustrated on the map itself (e.g. using coloured route lines). Note that you are not expected to include every town/city in the known world in your application, but you should include at least 30-40 locations/towns/cities/ports and the appropriate route links (by land or sea) between them. Information regarding distance, ease of passage, and safety should be stored (or should be calculable) for each route link.

The application should allow for the identification of various routes between two locations according to some given criteria. It should be able to generate:

- Multiple valid route permutations between a starting point and a destination (can limit it to a maximum user-specified number of routes where there are too many permutations).
    - First things first: identify at least one valid route to start with.
- The shortest route (in terms of distance, as the crow flies).
- The easiest route (in terms of overall/average ease). For instance, it is more difficult to traverse a mountain than flatlands.
- The safest route (in terms of maximising safety/minimising danger). Some areas/routes are more dangerous than others!

The key thing is that different routes can be identified based on different criteria: any valid route, multiple valid routes, shortest route, easiest route, and safest route.

The application should also allow users to specify routes or towns/cities to avoid on the generated route(s), and to also specify waypoint towns/cities that must be visited on the route to the final destination.

Implementation Notes

- This is a <u>team CA exercise</u>. Teams should consist of 2 students only (ideally both from the same lab group). Students may be permitted to work by themselves in very limited circumstances (but this requires the prior approval of your lab lecturer).
- This CA exercise is worth <u>approximately 35%</u> of your overall module mark. You must submit it on Moodle and demonstrate it (as a team) to your assigned lab lecturer for it to be assessed and included in your overall marks. The demo/interview is mandatory!
- The key point of this CA exercise is in the use of <u>custom graphs</u> (i.e. nodes/vertices connected by links/edges), along with some <u>algorithms to search/traverse graphs</u>.
    - Think of towns/cities as nodes/vertices and route links (roads, sea crossings, etc.) as links/edges.
        - It might be useful to support links that have one or more “elbow” points too (i.e. non-straight links).
    - You should <u>develop your own custom graph-based data structure</u> in the first instance as the main data structure for representing towns/cities and their interconnecting route links. However, beyond the required custom graph, you are free to also use any other JCF collections/classes you wish to implement the required features and algorithms, etc.
- The system should be <u>comprehensive</u>, and use a suitable <u>database file</u> of many towns, cities and routes/links across the known world of *A Song of Ice and Fire* (Game of Thrones) (enough to be useful; at least 30-40 locations/towns/cities/ports and the appropriate route links (by land or sea) between them).
    - The database can be stored or represented in any appropriate file format (e.g. CSV, XML, plaintext, etc.). There is no need to save back to the database file in the

    application, but it should be easy to update the database file to manually add new towns/cities, route links, distances, ease, safety, etc.
    - You don't have to be familiar with *A Song of Ice and Fire* (Game of Thrones) for this CA exercise. It is essentially just an arbitrary "map" for you to work to. It is easy to find information online regarding city/town names, locations, etc. This wiki is a good resource to start with, but there are many others you could use too.
- The application should provide a suitable user-friendly JavaFX graphical user interface.
    - Some level of "graphical feedback" should be provided on the map as part of the user interface (e.g. marking/naming the towns/cities along an identified route and the links between them). However, what the user interface looks like is ultimately up to you, and the key thing is the data graph representation and usage.
    - You could also use a TreeView control to show the route(s) as a collapsible hierarchy.
    - Other JavaFX controls might also be used as appropriate to provide for a nice, intuitive and coherent user experience.
- You should use Dijkstra's algorithm to identify the shortest/easiest/safest route between two towns/cities.
- A route may have specified waypoints (one or more) between the starting point and final destination that the route(s) has to go through. Waypoints should ideally be supported in some way in all route finding operations (single route, shortest route, easiest route, etc.).
- Similarly, it should also be possible to specify towns/cities/roads/etc. to avoid on the route.

Indicative Marking Scheme

- Custom graph data structure/classes = 15%
- Generate a single valid route between two towns/cities = 10%
- Generate multiple valid route permutations = 10%
- Shortest route algorithm/identification = 10%
- Easiest route algorithm/identification = 5%
- Safest route algorithm/identification = 5%
- Graphical feedback/illustration of route(s) on the map = 5%
- Waypoint support = 10%
- Avoiding specified towns/cities/routes = 5%
- JavaFX GUI = 10%
- JUnit testing = 5%
- JMH benchmarking of key route retrieval methods = 5%
- General (overall completeness, structure, commenting, logic, etc.) = 5%

Note that it is not expected that all students will attempt all aspects of this assignment. Use the above marking scheme to guide your efforts as this is what you will be marked against.