C.H. Robinson Interview Project Design

Purpose/Problem:

*Create an application that receives a three-letter code for a North American Country and returns a list of all countries a driver must travel through to go from the United State of America to the destination.*

*Create an application where the user can enter a country code into an input field, and the application should display the list of countries (formatted or not) somewhere on the view.*

Abstract Solution:

* Essentially a “shortest path” problem
* Use a “Graph” data structure to represent North America
  + Each vertex will be a country
  + Edges will be added between countries that border each other
  + In this case, countries and borders can be hard-coded in as countries in North America won’t change

Diagram

Description automatically generated

* A shortest path algorithm will be designed to find shortest path from USA to destination country
  + BFS
    - Start at root and explore all neighbors before moving to next node level
* Simple Web-App with input box for destination country and submit button
  + Hitting submit will display the path from USA to destination
  + Error message if destination doesn’t exist
* Backend code written in Java, front end using HTML/JSP

Backend Design:

* **NorthAmericaGraph**
  + No built-in graph class in java – instead used adjacency list
  + Each country represented by a number 0-9
    - Number represents index of respective country in *COUNTRIES* array that hold all countries
  + Constructor creates adjacency list and adds the edges (border) between each country where necessary
  + Only method is getter for the adjacency list
* **GraphController**
  + printShortestPath(adj, src, dest, v)
    - make *previous[]* of length # of vertices (10)
    - call BFS(adj, src, dest, v, previous[])
      * if this returns false, source and dest aren’t connected
    - make new LinkedList *path*
    - int *next* = *dest*
    - loop while *previous[next]* != -1
      * add *previous[next]* to path
      * set *next* to *previous[next]*
    - return *path*
  + BFS(adj, src, dest, v, previous[]) – ***BFS is shortest path in unweighted graph***
    - New LinkedList *queue* and Boolean *visited[]*
    - Set all visited elements to “false” and all previous elements to “-1”
    - Set *visited[src]=*true and add *src* to *queue*
    - Loop while *queue* is not empty
      * *Int c = queue.remove()*
      * Loop through *c’s* neighbors
        + If a neighbor isn’t visited

Set it to visited

*Previous[*neighbor*]* = *c*

Add neighbor to *queue*

If neighbor is *dest* -> return true

* + - If *dest* is never reached -> return false

Frontend Design:

* **Index.jsp**
  + A form with one input box and a submit button
  + Displays “Enter destination Country”
  + Submitting the form will submit the destination country
* **printAction.jsp**
  + create session variable *destination*
    - make sure destination exists in the graph – if not, redirect to index and show error message
  + create a NorthAmericaGraph
  + use GraphController class to find shortest path, and create another session variable *path* to be used by results page to print
* **results.jsp**
  + take *path* from printAction and display the list to the screen
  + will also have a link to return to the main menu (index)

Questions/Comments/Alternate Methods

* Researched 3rd Party *Graph* software – couldn’t find any that made sense to me
* Also experimented with adjacency matrix instead of adjacency list
* A lot of time spent on trying to get the form to print results on the index page
  + Figured results page with back button would suffice for now, but is something I would work to change
* Another thing I would work to change is make it not case sensitive as my application is now