Assignment 1

Programming Assignment - Search

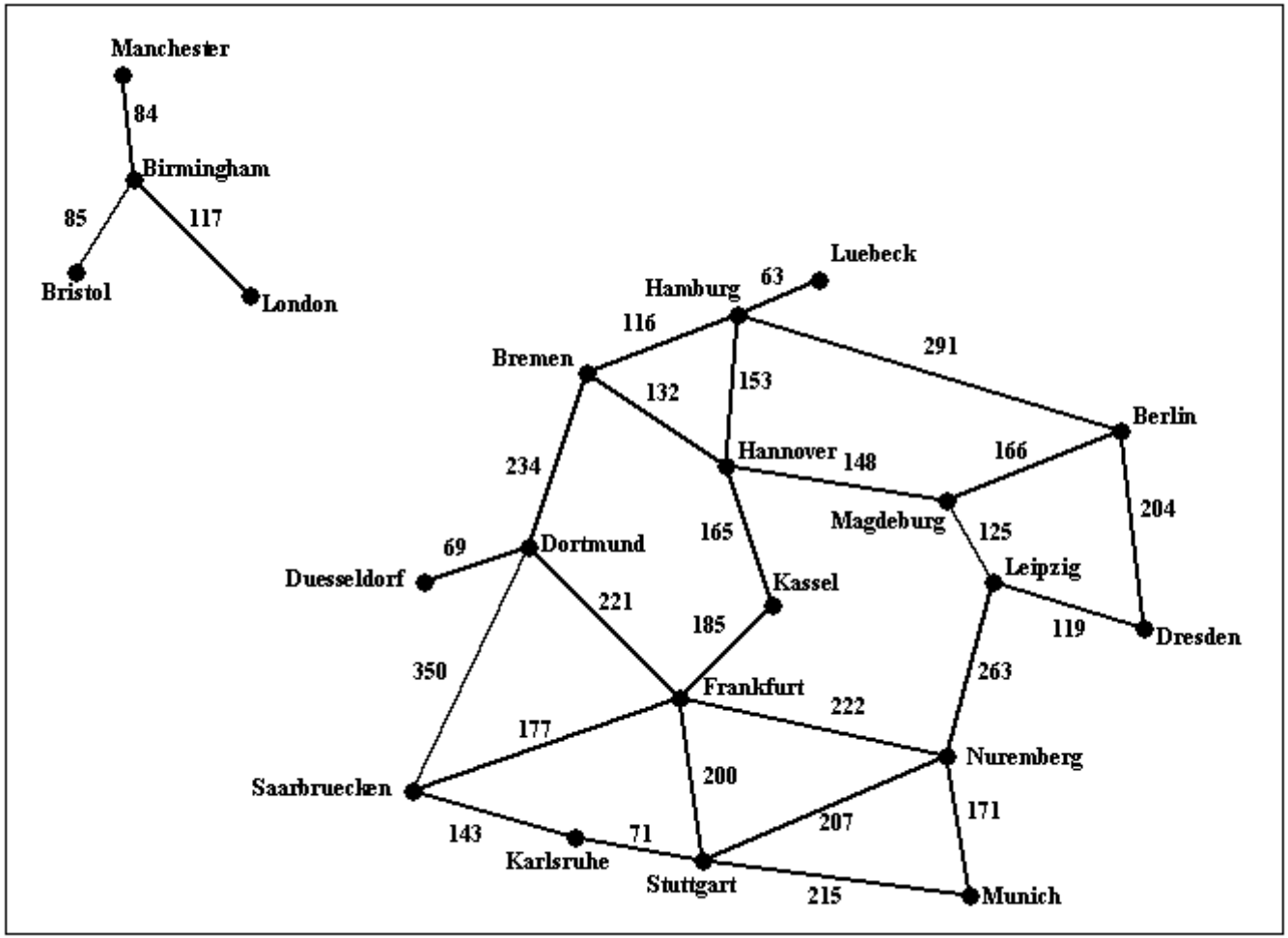

Figure 1: Visual representation of input1.txt

Implement a search algorithm that can find a route between any two cities. Your program will be called find_route, and will take exactly commandline arguments as follows:

find_route input_filename origin_city destination_city heuristic_filename

An example command line is:

find_route input1.txt Munich Berlin (For doing Uninformed search)
or
find_route input1.txt Munich Berlin h_kassel.txt (For doing Informed search)

If heuristic is not provided then program must do uninformed search. Argument input_filename is the name of a text file such as input1.txt, that describes road connections between cities in some part of the world. For example, the road system described by file input1.txt can be visualized in Figure 1 shown above. You can assume that the input file is formatted in the same way as input1.txt: each line contains three items. The last line contains the items "END OF INPUT", and that is how the program can detect that it has reached the end of the file. The other lines of the file contain, in this order, a source city, a destination city, and the length in kilometers of the road connecting directly those two cities. Each city name will be a single word (for example, we will use New_York instead of New York), consisting of upper and lowercase letters and possibly underscores.

IMPORTANT NOTE: MULTIPLE INPUT FILES WILL BE USED TO GRADE THE ASSIGNMENT, FILE input1.txt IS JUST AN EXAMPLE. YOUR CODE SHOULD WORK WITH ANY INPUT FILE FORMATTED AS SPECIFIED ABOVE.

The program will compute a route between the origin city and the destination city, and will print out both the length of the route and the list of all cities that lie on that route. It should also display the number of nodes expanded. For example,

find_route input1.txt Bremen Kassel

should have the following output:

nodes expanded: 12
distance: 297 km
route:
Bremen to Hannover, 132 km
Hannover to Kassel, 165 km

and

find_route input1.txt London Kassel

should have the following output:

nodes expanded: 7
distance: infinity
route:
none

For full credit, you should produce outputs identical in format to the above two examples.

If a heuristic file is provided then program must perform Informed search. The heuristic file gives the estimate of what the cost could be to get to the given destination from any start state (note this is just an estimate). In this case the command line would look like

find_route inf input1.txt Munich Kassel h_kassel.txt

Here the last argument contains a text file what has the heuristic values for every state wrt the given destination city (note different destinations will need different heuristic values). For example, you have been provided a sample file h_kassel.txt which gives the heuristic value for every state (assuming kassel is the goal). Your program should use this information to reduce the number of nodes it ends up expanding. Other than that, the solution returned by the program should be the same as the uninformed version. For example,

find_route input1.txt Bremen Kassel h_kassel.txt

should have the following output:

nodes expanded: 3
distance: 297 km
route:
Bremen to Hannover, 132 km
Hannover to Kassel, 165 km

Suggestions

The code needs to run on omega. If you have not even tried logging in on omega until the last day, there is a high probability that something will go wrong. You may find it convenient to do the code development and testing on your own laptop or home machine, but it is highly recommended that you log in to omega and compile a toy program ASAP, and that you compile and run an intermediate version of your code well before the deadline. Notify the instructor for any problems you may have.

Pay close attention to all specifications on this page, including specifications about output format, submission format. Even in cases where the program works correctly, points will be taken off for non-compliance with the instructions given on this page (such as a different format for the program output, wrong

compression format for the submitted code, and so on). The reason is that non-compliance with the instructions makes the grading process significantly (and unnecessarily) more time consuming.

Grading

The assignments will be graded out of 100 points.

- 40 points: The program always finds a route between the origin and the destination, as long as such a route exists.
- 20 points: The program terminates and reports that no route can be found when indeed no route exists that connects source and destination (e.g., if source is London and destination is Berlin, in the above example).
- 20 points: In addition to the above requirements, the program always returns optimal routes. In other words, no shorter route exists than the one reported by the program.
- 20 points: Correct implementation of any informed search method.
- Negative points: penalty points will be awarded by the instructor and TA generously and at will, for issues such as: code not running on omega, submission not including precise and accurate instructions for how to run the code, wrong compression format for the submission, or other failures to comply with the instructions given for this assignment. Partial credit for incorrect solutions will be given ONLY for code that is well designed and well documented. Code that is badly designed and badly documented can still get full credit as long as it accomplishes the required tasks.

How to submit

Implementations in C, C++, Java and Python will be accepted. If you want to you can also use CLISP. If you would like to use another language, make sure it will compile on omega and clear it with the instructor beforehand. Points will be taken off for failure to comply with this requirement.
The assignment should be submitted via Blackboard. Submit a ZIPPED directory called assignment1_<net-id>.zip (no other forms of compression accepted, contact the instructor or TA if you do not know how to produce .zip files). The directory should contain source code. Including binaries is not necessary as your code will be recompiled by the TA. The submission should also contain a file called readme.txt, which should specify precisely:

- Name and UTA ID of the student.
- What programming language is used.
- How the code is structured.
- How to run the code, including very specific compilation instructions, if compilation is needed. Instructions such as "compile using g++" are NOT considered specific.
- Insufficient or unclear instructions will be penalized by up to 10 points.
- **Code that does not run on omega machines gets AT MOST 75 points**.

Submission checklist

Is the code running on omega?
Does the submission include a readme.txt file, as specified?