Assignment 1

Uninformed & Informed Search

Task 1

Max: [4308: 100 Points, 5360: 100 Points]

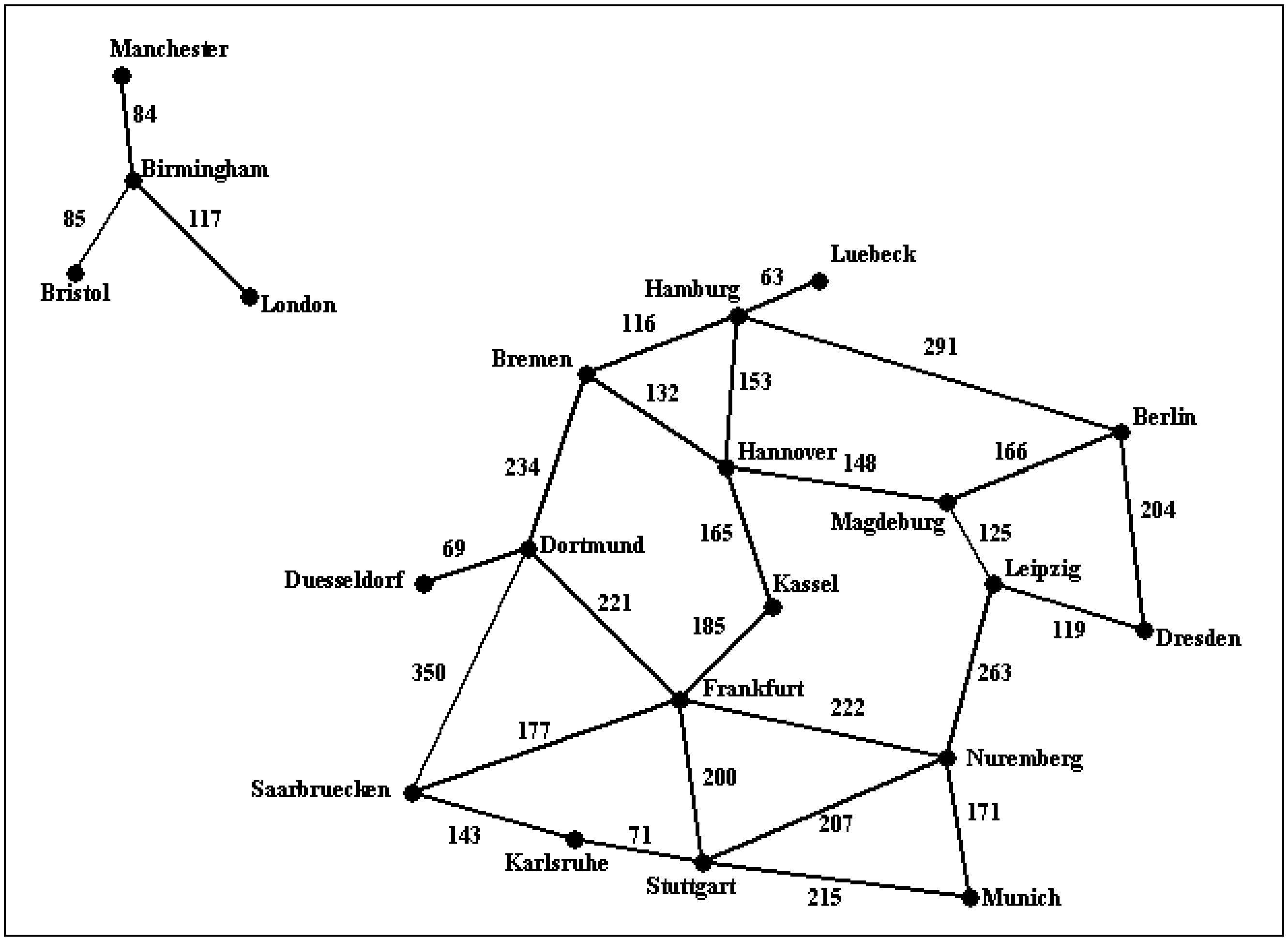

Figure 1: Graphical representation of information in input1.txt

Implement a program that can find a route between any two cities using state space search. 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 Bremen Kassel (For doing Uninformed search)
or
find_route input1.txt Bremen Kassel 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, 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 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 and nodes generated. For example,

```
find_route input1.txt Bremen Kassel
```

should have the following output:

```
Nodes Popped: 12
Nodes Expanded: 6
Nodes Generated: 20
Distance: 297.0 km
Route:
Bremen to Hannover, 132.0 km
Hannover to Kassel, 165.0 km
```

and

```
find_route input1.txt London Kassel
```

should have the following output:

```
Nodes Popped: 7
Nodes Expanded: 4
Nodes Generated: 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 input1.txt Bremen 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 Popped: 3
Nodes Expanded: 2
Nodes Generated: 8
Distance: 297.0 km
Route:
Bremen to Hannover, 132.0 km
Hannover to Kassel, 165.0 km
```

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.
- 15 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).
- 25 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. Please note that you only need to make one submission that meets this and all previous requirements to get credit for all the parts.
- Negative points: penalty points will be awarded by the instructor and TA generously and at will, for issues such as: 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 credit only as long as it accomplishes the required tasks.

Supplementary Information

Traces of sample runs are provided here. These are provided to show how the nodes are visited and how counters are updated for sample cases. You DO NOT need to generate such output.

- Bremen to Kassel (Uninformed)
- Bremen to Kassel (Informed)
- London to Kassel (Uninformed)
- London to Kassel (Informed)

How to submit

For each part: Implementations in C, C++, Java, and Python will be accepted. Points will be taken off for failure to comply with this requirement.

Create a ZIPPED folder called <net-id>_assmt1.zip (no other forms of compression accepted, contact the instructor or TA if you do not know how to produce .zip files).

The folder should contain just the source code (No need to submit compiled binaries or any test files). It should also contain a file called readme.txt, which should specify precisely:

- Name and UTA ID of the student.
- What programming language is used for this task. (also mention if the code is omega compatible)
- 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 if the TA needs to do additional steps to get your code to run.
- Insufficient or unclear instructions will be penalized.
- Code that the TA cannot run gets AT MOST 75% credit.