Programming Assignment
2004 Comprehensive Exam

Department of Computer Science
Brown University

Out: 9am, Monday, January 26 2004
Due: 9pm, Thursday, January 29 2004
Presentations: Friday, January 30 2004

1 Overview

Your task is to build a transportation assistant that provides a user with driving directions from a source to a destination, much like `mapquest.com` and `maps.yahoo.com`. Your system should take as input two street addresses—source A and destination B—and it should output driving directions in some easy-to-read format. These driving directions should be geared towards satisfying user preferences: for example, given a user profile that indicates a preference for saving time, the system should output the fastest route; but given a user profile that indicates a preference for simplicity, the system should output the least complex route. We will judge the quality of your system along three dimensions:

- *search engine*: how does your system select a route, or set of routes, to exhibit: i.e., how close to optimal is your selection mechanism, given some specification of user preferences; and how efficiently does it make its selections?
- *user interface*: how cleanly does your system accept input and display output?
- *documentation*: both user documentation and programmer documentation.

2 Functionality: Search Engine

The primary task of the search engine is to find an optimal route from source to destination, given some specification of a single user's preferences. We assume each user's preferences can be modeled via a linear cost function. Given a set of attributes relevant to travel (e.g., time, distance, complexity, "scenicness", etc.), each user associates weight w_i with the ith attribute. Now, if a given route r has attribute values x_i, then the user's cost of taking route r is given by $c(r) = \langle \vec{w}, \vec{x} \rangle = \sum_i w_i x_i$. The attributes of interest in this assignment are: distance, time, complexity (as measured by the number of different road segments taken), number of stop signs, and number of stop lights.

The files `*.dat` in the directory `/home/amygreen/comps/` contain various sample user profiles. It is a simple matter, and your basic task, to find an optimal route for the individual user's with profiles `shortest.dat`, `fastest.dat`, and `uniform.dat`. The files `population*.dat` contain sets of user profiles. Search engines such as `google.com` that attempt to find solutions of interest to the entire user population might optimize with respect to sets of user profiles. Given such a set, and given a source-destination pair, your system should present not necessarily one, but perhaps several routes from source to destination, selected so that as many user profiles as possible are as nearly optimally satisfied as possible.

3 Road Data Files

In addition to the various profile files, you will be given the following four files that store information about Rhode Island roads:

- a *main file* (*.shp) that describes the entities (i.e., road segments) and their geometrical attributes.
- an *index file* (*.shx) that consists of fixed length records, each containing the offset of the corresponding main file record from the beginning of the main file.
- a *data* (*.asc) file that contains the attribute values of each record in the main file.
- a *meta-data* (*.htm) file that interprets the attribute values.

These four files can be found in the directory `/home/amygreen/comps`. The directory also includes `shapefile.pdf` that precisely describes the format of the main file and the index file.

The main file is a binary file that consists of variable-length records in which each record describes an entity. Two important points about the data contents and format of the main file are:

- *Endianness issue*: In the main file, both *big-endian* and *small-endian* formats are used to store the data attribute values (see page 3 of `shapefile.pdf`). As you read the main file, you need to perform a conversion from the non-native format to the native format of your (virtual) machine. For example, Intel's 80x86 processors and their clones are little endian; Sun's SPARC and the PowerPC families are all big endian; and the Java Virtual Machine is big endian as well.

  In order to help you with the testing of your modules that load the binary file and perform endian-conversion, we provide a sample text file, `sample100.txt`, that includes the text representation of the first 100 records of the main file (refer to the first line of `sample100.txt` for a textual description of the format of each record). You should compare the information contained in this file with the output of your modules to make sure that you can correctly read and interpret the binary main file. The sample file is also included in the directory `/home/amygreen/comps`.

- *Main file shapes*: The main file that you will work with contains shapes of type `PolyLine` only (see page 4 of `shapefile.pdf`). Therefore, you do not have to deal with and parse other shape types such as `Point` or `Polygon`.

The index file is also a binary file and is provided to enable fast random access to the records in the main file. This file is redundant and is provided for convenience only—it can be recreated from the main file. You are free to decide whether to use it or not. In fact, we encourage you to consider creating your custom index file, one that is better suited to the requirements of your task.

The data file is an ASCII file that consists of one attribute record per entity. The first line of this file lists the road attributes separated by TABs. Each subsequent line is a record, consisting of attribute values separated by TABs. The records are stored in the same order in the main file and the data file.

Note that the road attributes do not coincide with the attributes in the user preference profiles. On the contrary, there are many more road attributes, including such things as speed limit, stop signs, and traffic lights. Use these road attributes to compute user attribute values: e.g., compute the time of a route based on speed limit, stop signs, and traffic lights.

The meta-data file describes the attributes and their values: e.g.,

- sign $\in \{-1, 0, 1\}$:
  sign $= 0$ means no stop sign;
  sign $= -1$ means a stop sign at the left end of the street;
  sign $= +1$ means a stop sign at the right end of the street.

- light $\in \{-30, -15, 0, +15, +30\}$:
  light $= 0$ means no traffic light;
  light $= -n$ means an n second traffic light at the left end of the street;
  light $= +n$ means an n second traffic light at the right end of the street.

4 User Interface

The interface to your system should take as input two road addresses (in Rhode Island), one source and one destination, and it should output one or more sets of driving directions from the source to the destination, *along with the corresponding cost values for each route*. The road address should also contain the *road type*; e.g., whether the road is an `avenue`, `street`, etc. You can assume that `(road, road type)` is unique, given a town.

Because we are looking for creative solutions to the problem of presenting multiple routes, this part of the assignment is deliberately ill-specified. The only requirement is that your interface should be *usable*. First, it should be easy for a new user to operate your system effectively. Second, it should be robust: it should be tolerant of user errors and it should easily recover from such errors. Third, make sure it is easy for us to select each of the user profiles for testing purposes: e.g., create a drop down menu that loads the corresponding profile for each element in the list.

5 Documentation

You code should be well commented. This does not mean that you should insert a comment per each line of your code. Rather, you should use your comments to ease the understanding of the high-level functionality of your components as well as the relatively complicated pieces. In addition to commenting your code, you should prepare the following documents for submission:

- **User's Guide**: This document should first provide a functional description of your application. It should then provide instructions for "normal" usage; it should describe how to set the application up and how end-users would use the application. It should also describe error conditions/messages and how to recover from them (if at all possible).

- **Programmer's Guide**: This guide should contain the basic design and structure of your implementation. In particular, you should discuss the components of your system and how they interact (i.e., how they share data and how they interface with each other). All the external libraries and tools used should be clearly identified. You should discuss the primary data structures and the algorithms employed and their asymptotic running times. The testing strategy adopted should be described. Sufficient details should be provided so that a programmer unfamiliar with your code can easily understand and extend your code. You should also discuss the limitations of your code (e.g., additional assumptions about the input format, errors you do not handle, etc.).

Both documents should be typeset. The User's Guide should be approximately 2 pages, and the Programmer's Guide should be approximately 4 pages. These numbers are merely guidelines and not strict requirements—you are welcome to use fewer or more pages, as you deem necessary.

6 Design and Implementation

The application can be realized either as a standalone program or as a web-based application. In either case, the route databases should be stored in a format that supports fast searches. You can use either internal-memory data structures (e.g., a hash table) or external-memory storage (e.g., a relational database). For Web-based applications, you should ensure that the search structure persists across successive HTTP requests so that you do not create it from scratch for each request. Some platforms (e.g., Java servlets and the PLT Scheme Web server) provide a transparent mechanism for data persistency. Alternatively, you can explicitly manage external-memory storage.

You may write your application in any programming language (or combination of programming languages). You can use any libraries that are considered part of the language (for example, the STL is considered part of C++; all classes in java.* or javax.* are considered part of Java). Furthermore, you can use any generic software tools, packages, and systems (such as parsers, libraries of data structures and algorithms, Perl modules, DBMSs, Web servers, JSP/ASP, CGI, etc.).

You are also allowed to download and install programming languages and support software if they are not readily available on our system, provided that these are designed to solve the problem or replace the application that you are asked to implement. You are allowed to use a free/open-source Geographic Information System (GIS) visualizer (see Appendix for relevant pointers), only to graphically display the results of your searches. You are not allowed to use any GIS software to implement other parts of your assignment, such as loading the main file or parsing the meta-data file. You are allowed, however, to use tools that would help you to perform endian-conversion. You should consult the Programming Exam Committee if you are not sure about whether a specific software is allowed or not.

For developing your application, you may use any machines that you normally have access to at Brown or you may use your own personal machine. However, your application should run on a standard departmental Maxbuilt system, under either Linux or Windows, from the submission directory (see Section 7).

Except for the allowed generic libraries and tools discussed above, all work that you hand in must be your own. You should not discuss your work or ideas with anyone else. You should not share code or ask others for advice on your design or code. If you have questions, you should contact a member of the Programming Exam Committee. Corrections, clarifications and answers to questions of general interest will be posted to the comps news group (`brown.cs.comps`).

7 Submission

You should submit the following materials by **9:00pm EST on January 29, 2004**:

1. **Documentation**:
   - Printout of the User's Guide
   - Printout of the Programmer's Guide
2. **Project files**:
   - Source files
   - Executables
   - Documentation

The documentation and the project files should be placed in the submission directory:

```
/pro/comps/name
```

where `name` is your login name. Please do not modify the permissions of this directory, which should remain accessible only to you. The submission directories will be locked at 9:00pm EST on January 29 and made available to the Programming Exam Committee. You will be graded on what is there at the time. The documentation files should also be handed to Fran Palazzo (before 4:30pm) or e-mailed to her (`fp@cs.brown.edu`) between 4:30pm and 9pm on January 29.

8 Presentation

You will be asked to give a presentation of your application to the Programming Exam Committee. The format of the presentation will be informal: you will explain your design choices, conduct a walkthrough of your application by running it from the submission directory, outline your testing strategy, and answer questions from the committee members. The committee will set up the schedule of the presentations, which will be held on January 30.

You should not prepare slides or bring any other material besides what you have already submitted.

9 Grading

Your grade will be approximately computed as follows:

40%	Functionality
25%	Efficiency
15%	Documentation
10%	Design
10%	Usability

Good luck and have fun!

A Resources

You may find the following resources useful for your assignment:

- Free GIS viewers
  `http://www.lbszone.com/viewers/`
- Open source and free GIS software repository
  `http://opensourcegis.org/`
- The Common Gateway Interface (CGI):
  `http://hoohoo.ncsa.uiuc.edu/cgi/overview.html`
- Java Server Pages (JSP):
  `http://java.sun.com/products/jsp/`
- MS Active Server Pages (ASP):
  `http://www.asp.net/`
- MySQL (open source database):
  `http://www.mysql.com/`
- Berkeley DB (open source database):
  `http://www.sleepycat.com/`