Knowledge Technologies, Semester 2 2012

Project 2: Approximate String Search

Due:	5pm, Wed October 23, 2013
Submission mechanism:	To be described on the LMS.
Submission materials:	Source code that compiles on the CSSE servers (see below).
Assessment criteria:	Source code: Clarity; documentation; correctness of method Report: Clarity and written expression; quality of evidence; strength of arguments. Each is worth 50% of the total.
Marks:	10% (undergraduate), 15% (masters)

Overview

The aim of the project is to develop software that, given a target string (in this case, a text) and a set of queries, finds the best approximate match to each query in the text.

Technical merit and good programming practice form a component of the assessment. However, the particular focus of this project is a critical analysis of methodologies — which is to say, what you have learned about the problem and can pass along to another person attempting it. Impeccably-coded software with little insight will not receive a strong mark.

Resources

The directory `/home/subjects/comp90049/local/data/` contains the principal materials required for the project. These include the query file `surnames.txt` and the text `turgenev.txt`, which is a translation of the Russian novel Fathers and Sons.

Task 1: Software

Your code should perform the following broad task.

1. Preprocess the text to remove all non-alphabetic characters, other than spaces, and convert all letters to lower-case. The resulting text should have an alphabet of size 27.
2. Load the text into a suitable data structure (see below).
3. Process each query in turn (see below).
4. For each query, return the substring in the text that most closely matches the query, together with a match score.

You need to implement two different methods for doing the matching. In each case, carefully *instrument* your code: roughly count the volume of data structures created, and the number of character comparisons used during execution.

1. A trie. Be *extremely* careful with this. A simple implementation is all that is required, but memory requirements may grow much faster than you expect. Test thoroughly on small volumes of text! You will then need to implement search procedures allowing 1 or 2 mismatches. (Or 3 mismatches, if your tests show that this will complete in a reasonable length of time.)

2. An edit distance with local alignment. Again, be careful to test on small volumes of data; you may find that you only have time to run a relatively small number of queries.

   (C code for a simple edit distance, was given in lectures. But this code does not show how to *trace-back* to find the alignment. Source code examples for trace-back can be readily found on the web; feel free to make use of them.)

Task 2: Report

The report will consist of about 600 words (undergraduate) or 1200 words (masters) and should aim to discuss:

1. A basic description of the task;

2. Whether the methods worked – use example queries to demonstrate how they contrast with each other.

3. Considerations for scalability (that is, performance and behaviour as the text size or query length grows), and what effect this might have on the results.

Note that we will be looking for evidence that you have thought about the task and have determined reasons for the relative performance of the two strategies.

Undergraduates: A limit of 600 words means that your report is brief: being concise will be invaluable, and overlong reports will be penalised. You will not have space to discuss the technical elements of your implementation. Spending more time on your report will allow you to use the word limit more effectively.

Submission

Submission will entail two files:

1. An archive which contains your code and a `README` file which briefly explains how to compile your submission on the CSSE servers (preferably as a Makefile or `run.sh`) and the location and format of the output

2. Your written report, as a single file in Portable Document Format (PDF)

Your software can be implemented in any programming language or combination of programming languages, again with the proviso that it compiles and runs on the CSSE Unix servers.

If your report is submitted in any format other than PDF, we reserve the right to return it with a mark of 0.

Changes/Updates to the Project Specifications

If we require any (hopefully small-scale) changes or clarifications to the project specifications, they will be posted on the LMS. Any addendums will supersede information included in this document.

Academic Misconduct

For most people, collaboration will form a natural part of the undertaking of this project. However, it is still an individual task, and so reuse of code or excessive influence in algorithm choice and development will be considered cheating. We will be checking submissions for originality and will invoke the University's Academic Misconduct policy (`http://academichonesty.unimelb.edu.au/policy.html`) where inappropriate levels of collusion or plagiarism are deemed to have taken place.