CS3236: Project

Pierre Senellart ⟨pierre.senellart@nus.edu.sg⟩

Due Friday, November 14, 23:59

Submission: Submission can be **by upload to the IVLE Workbin only**. Submit either a report in PDF format (for the theory project) or a ZIP file containing your implementation and a short report (for programming projects). Late answers are still accepted by November 15, 23:59 on the IVLE Workbin, but receive only 50% of the possible marks. Any later submissions receive no marks.

Choose only one of the projects described here, either a theory project or a programming project. Projects are individual; if you have a grand project in mind that may require collaboration between several students, contact me to check if it is feasible; such a joint project would need to be significantly broader or deeper than an individual project.

1 Theory projects

Pick one of the theory topics below. Read the material and summarize the content *in your own words*. You may (and in some case should) consult additional literature to fully understand a topic if needed. Your report should consist of max. 8 pages in 12pt font. What is expected:

1. a clear description of the problem;
2. short motivation: why is the problem important? what is the impact?
3. at least one fully detailed example;
4. a summary of results, and their consequences.

Plagiarism is absolutely forbidden. Reference any used material, and explicitly mention if some content is not yours.

Here is a list of suggested topics. You can also propose another topic, but in that case contact me by September 15 to agree on a topic.

1. Any of the topics covered in chapters 47, 48, 49, or 50 of the textbook. These chapters are fairly short, so it is expected that you also look for accompanying literature, duly referenced.
2. List decodable codes (see PDF on IVLE).
3. Zero-error communication (see PDF on IVLE).

2 Programming projects

Choose one of the topic below (or come up with your own, but in that case check with me by September 15 whether it is acceptable). Implement what is requested, and provide both (readable and properly commented) code and a short report on what has been done and on how to operate your mplementation. Your implementation can be in any programming language for which an open and easy-to-obtain compiler or interpreter exists. This includes Java, C, C++, Python, Perl, Ruby, PHP, JavaScript, Objective C (no proprietary MacOS/iOS calls), C# (no proprietary .NET calls), etc. If in doubt, ask.

2.1 Hamming code

1. Implement the $(7, 4)$ Hamming code. Your encoding program should take as an input a file (ASCII characters or other specified character set), and output an encoded bit string. Your decoding program should take a bit string, and convert it back into a text file.

2. Test your program on several kinds of inputs to see that encoding and decoding works correctly. I will test your program on other input.

3. Write a small program that implements a binary symmetric channel. For each bit, flip it with probability $\frac{1}{5}$, $\frac{1}{4}$, $\frac{1}{3}$, and $\frac{1}{2}$. Describe how well you expect to correct errors in each of these cases. Now, run your noisy channel simulation on your encoded test data. Now run your decoder program and describe how well your code actually worked for each of these cases, i.e., how well you can decode. Does it match your expectations?

4. In the first homework exercise you came up with a code of your own. Implement it and compare its performance to the $(7, 4)$ Hamming code. If you like to explore you can also invent another code, but you need to explain how it works in words in addition to your program.

5. Can you think of a way to combine your own code with the Hamming code to get a new code that allows you to tolerate more errors? Implement your combination of codes and analyze the theoretical performance of your new code: what is its rate? what is the expected error probability?

2.2 Arithmetic coding

Implement an arithmetic coder and decoder with a simple prior (e.g., Laplace prior). Run it on a variety of files and compare its performance to at least two existing compressors and decompressors (e.g., gzip, bzip2, PPMd, etc.) in terms of compression rates and running time of compression and decompression. When comparing to existing software, be sure to both specify the algorithm used by the software (e.g., a ZIP archive may use bzip2, LZMA, DEFLATE, or other algorithms) as well as the implementation used (e.g., WinZIP 18.5 or InfoZIP 3.0).

2.3 Lempel-Ziv

Implement a Lempel-Ziv coder and decoder, choosing a method to specify message termination. Run it on a variety of files and compare its performance to at least 2 existing compressors

and decompressors (e.g., gzip, bzip2, PPMd, etc.) in terms of compression rates and running time of compression and decompression. When comparing to existing software, be sure to both specify the algorithm used by the software (e.g., a ZIP archive may use bzip2, LZMA, DEFLATE, or other algorithms) as well as the implementation used (e.g., WinZIP 18.5 or InfoZIP 3.0).