Course proposal for Inf2-IADS

Informatics 2: Introduction to Algorithms and Data Structures

Sharon Goldwater

28 Nov 2018

1 Inspiration for the design of this course

IADS is a 20pt course running long and thin (both semesters). This will shift some algorithms content earlier in year 2 than at present (as often requested by students), cover the material over a longer period (good for retention), and allow better use of time in December/January (e.g., for a courswork).

The *learning outcomes* are based on a detailed list in the ACM guidelines, and on the rather brief/vague outcomes of our current Inf2b and DMMR courses. This course aims to integrate theory and practice, introducing some aspects that are more empirical than either Inf2b or DMMR (see L.O. 3) while retaining the importance of theoretical analysis. This integrated approach aligns with the aims of the revised pre-honours curriculum to better promote both practical skills and critical judgement (e.g., when is it appropriate to do X vs Y?)

The *suggested topics* are based on Inf2b, DMMR, discussion arising from the curriculum review/consultation, and discussion with JLongley (who has offered to deliver half the course.) In particular, a few key topics currently covered in Inf2a (dynamic programming, CFGs) are included in the syllabus here, to compensate for the closure of Inf2a.

The course proposal/descriptor does not depend on a particular choice of programming language. However, as a pragmatic matter, it is suggested that *Python* is used for the practical work in this course (see Sec A).

2 Learning outcomes

The learning outcomes cover five basic areas: (1) algorithmic analysis, (2) data structures, (3) implementation and empirical issues, (4) specific algorithms and general strategies, and (5) limitations.

Specifically, on completion of the course, students should be able to:

1) Explain both formally and informally the difference between "best", "expected", and "worst" case behavior of an algorithm, and use asymptotic notation to analyse the time and space complexity of algorithms. Use recurrence relations to determine the time complexity of recursively defined algorithms.
2) Describe the properties, typical implementations, and example application use cases of abstract data types (e.g., stacks, queues, sets, dictionaries, priority queues) and discuss the costs and benefits of dynamic and static data structure implementations; use the above knowledge to justify the selection of appropriate data types in a range of settings.

3) Work with a range of data structures to implement basic algorithms given pseudocode or a task specification; perform empirical studies to compare the performance of different implementations of the same algorithm or data type on various input and explain what can be learned from empirical analysis that cannot be learned from asymptotic analysis (and vice versa).
4) Describe various algorithmic strategies (e.g., brute-force, greedy, divide-and-conquer, recursive backtracking, dynamic programming) and give examples of each from a range of application areas including language processing and information retrieval. Hand-simulate a range of algorithms, including algorithms for searching, sorting, hashing, solving graph problems, and examples of dynamic programming. Give example applications that would use each algorithm and choose appropriate algorithms to use for example problems.
5) Define informally the classes P and NP and give examples of problems in NP. Explain the halting problem and its significance.

3 Assessment

Learning outcomes 1, 2, 4, 5 (and aspects of 3) can be assessed by exam. Outcome 3 requires a practical assignment.

Due to this balance, a 20% coursework/80% exam weighting is proposed.

4 Learning and teaching activities

4.1 Breakdown of activities

The number of lectures, labs, and tutorials will be similar to the current 20-credit courses Inf2a and Inf2b. Specifically:

- 30 lectures (2a/2b have 32/33)
- 10 tutorials (2a/2b have 9)
- 8 labs (2a/2b have 6/7)

The course runs over two semesters, so there will be three lectures every two weeks. Tutorials and labs would probably be in alternate weeks. Some of the unused lecture slots could potentially be used for in-class learning activities or feed-forward/feed-back as needed.

4.2 Design of practical work and feedback

This proposal as stated is language-agnostic. My personal feeling is that Python would be a good choice (see Appendix A), but other languages may be possible.

The specific design of labs, coursework, and feedback is not yet determined, but here is one example of a possibile structure:

- Some labs and/or a small first assessed assignment could use automarking. This would only check whether solutions give the correct answer. Discussion of style and efficiency could be in a feedback lecture or during tutorial groups.

- Design two larger assignments with similar style but different content, where students may need to write a short report (or other non-automarked content). The first is *formative*, with students working in pairs—this both promote collaborative learning and reduces the number of submissions needing feedback. This assignment could be due in late January, and a similar but *summative* individual assignment could be due in March/April.

5 Course content

5.1 Overview and fit with other courses

The first 20-ish lectures follow content of current Inf2b and DMMR, in particular algorithmic analysis, searching/sorting, various data structures, graphs and graph algorithms. (However the presentation of this content might need to change somewhat to ensure integration of the topics and the desired integration of theory and practice.)

The remaining 10 or so lectures are partly key bits of Inf2a and partly new. Mainly:

- dynamic programming (examples to include CFGs and CKY)
- limitations: P vs NP and the halting problem

Under the planned update to pre-honours curriculum, content would be arranged as follows across the different courses. (Only changes directly relevant to this course are listed; see the Transition Plan for a more complete picture.)

2019-20 (transition year):

- Inf2a is closed. Inf2b is reduced to 10 points, covering the Learning half only.
- No major updates to DMMR, but may reduce graphs content slightly.
- IADS to spend slightly less time on graphs than in the final version (because still covered in DMMR). If using Python in IADS (previously introduced in Inf2a), this will allow students to use the time to better learn it.

2020-21 (transition completed)

- DMMR is updated, replacing graphs content with full coverage of discrete probability.
- *Inf2: Foundations of Data Science* is introduced, which will use Python. So if IADS is using Python, it will no longer be entirely responsible for introducing it, and time spent on that can be reduced.
- IADS is now fully responsible for teaching graph content, so can replace the time on Python with time on graphs, as per the suggested syllabus.

Ideally, the lecturers of IADS and DMMR will liaise regarding exactly how to split up and coordinate the material in the two courses. (J Longley has expressed willingness to do so.)

5.2 Indicative syllabus/lecture schedule

Lectures below are labelled with the course and lecture number where the content is currently taught: 13 content lectures from Inf2b plus 5 lectures on graphs/trees from DMMR.

Semester 1: 14-15 lectures

- (2B-01) Introduction to Algorithms.
- (2B-02) Asymptotic Notation.
- (2B-03) Asymptotic Notation and Algorithms.
- May need to insert 1-2 lectures here on induction, or delay L07-09 until after DMMR lec 10-11 (week 4) where they do induction.
- (2B-04) Sequential Data Structures.
- (2B-05) Hashing.
- (2B-06) MergeSort and Divide-and-Conquer
- (2B-07) AVL trees.
- (2B-08) Priority Queues and Heaps
- (2B-09) HeapSort and QuickSort.
- (2B-12) Large-scale Indexing and Sorting.
- (DMMR-19) Graphs: basic definitions and examples
- (2B-10) Graphs I: graph data structures, BFS [also has some defn's; need to remove redundancy with previous lecture]
- (2B-11) Graphs II: DFS, connected components, TopSort

Semester 2: 14-15 lectures

- (DMMR-20) Bipartite Graphs and Matching
- (DMMR-21) Graph Isomorphism; Paths and Connectivity; Euler paths/circuits
- (DMMR-22) Euler and Hamiltonian paths/circuits (continued); shortest paths;
- (DMMR-24) Trees; minimum spanning tree
- (2B-13) Ranking Queries for the WWW. [requires knowledge of graph defns and data structures]
- Approx 5 lectures covering dynamic programming with examples and applications.
  - Definitely include CKY parsing, preceded by an introduction to CFGs.
  - Additional examples could include (e.g.) edit distance, matrix multiplication, seam carving in images.
  - It could also be possible to discuss other algorithmic strategies for special cases, e.g., greedy parsing algorithms.
- Approx 4 lectures on limitations of algorithms/computation (hard problems, P vs NP, and the halting problem):
  - Introduce problems such as SAT (they may have seen it already in Inf1a) or TSP and discuss that there are no known ways to solve with polynomial algorithm. But it is possible to check a provided solution in polynomial time. Say that if they are interested in these ideas, take ITCS or ADS.
  - Introduce the concept of an undecidable problem, and sketch the proof of the undecidability of the halting problem.

6 Resources and personnel

Some materials for this course are available from existing courses, but they may need some adaptation, and additional materials will need to be developed.[1] This is also a very large course. Therefore, at least two staff should be allocated in the first year, and three might make sense (e.g., one for each semester mainly focusing on lectures and tutorials, with the third focusing mainly on the labs and assignments.).

JLongley has expressed enthusiasm for helping to develop and deliver part of this course (probably mainly the second half). Other potential staff have not yet been approached.

The contents of this course are very standard, and courses similar to it in flavour (e.g., Inf2a) have not had trouble recruiting enough tutors and demonstrators. Markers are often more difficult to recruit, but the example coursework structure given here is designed to minimize the need for markers as much as possible.

A Discussion of programming language

Although not baked in to the proposal, my personal view is that *Python* should be used for the practical work in this course, for several reasons:

- Python is concise but high-level, making it easy to show example code and to design practicals that engage with a range of application areas.
- During the curriculum transition year (see Sec 5.1), no other pre-honours course will be using Python, but students are expected to know it for third year courses. After the transition year, the new Data Science course will be using Python. Using it in both courses will mean less time carved out of either course individually and better reinforcement of Python skills. (See Sec 5.1 for more on managing this transition.)
- Python is supported by auto-marking frameworks such as NBGrader (a tool that integrates with Jupyter notebooks) and CodeRunner.

Java also has wide support and students will know it from first year, but is perhaps less easy to show in examples.

Most of our students will be simultaneously learning C as part of Inf2-Computer Systems, but not all of them, because that course is not required for all degrees. Also, C is likely harder to use for some desired application areas (eg, language processing).

[1] Inf2b has a detailed set of course notes already, but it is also probably worth identifying a good supporting textbook, which would help with the second half. Many options are available but these have not been examined in detail as the course lecturer(s) may want to choose.