COSC 201
Algorithms and Data Structures
Lecture 1 (24/2/2025)
Introduction

Brendan McCane
brendan.mccane@otago.ac.nz
and
Michael Albert

Who we are

Lecturer, Paper Coordinator
Brendan McCane
brendan.mccane@otago.ac.nz

Teaching fellow, Lab Coordinator
Chris Edwards
chris.edwards@otago.ac.nz

Lectures

- We are paperless – all information is provided via email, (occasionally) Blackboard, or the course webpage.
- No lecture handouts will be produced.
- In principle, lecture slides will be available on the course webpage by Thursday p.m. at the latest for the following week.
- Also available there will be any relevant supplementary material (including code samples from code discussed during lectures) and all the information that would normally be included in a course outline.
- Lectures are recorded and I may occasionally add some supplementary material to the recordings.

Lab expectations

- The labs for COSC201 are supposed to be like a workshop where you can go and try things out (with some limited guidance) and explore the concepts we're talking about.
- One of the great things about programming is that making a mistake (in the lab environment) can't result in an explosion, a plague, or somebody losing a limb. So feel free to make mistakes and try stuff!
- Many people find that the best way to improve their programming skills is simply by writing little pieces of code and seeing what they do, then writing a bit more, and so on. That's the behaviour we're trying to encourage in the labs.
- Lab assessment will be quite informal, generally you'll have a chat with a demonstrator (possibly virtually) to show that you've done something.

Keep in touch!

If problems arise or something is going particularly well then we want to hear about it as quickly as possible.

- Regarding lab work get in touch with Chris.
- Regarding lecture materials get in touch with Brendan.
- If you wish to remain anonymous then contact a class representative who will forward the message (contact details for class representatives will be on the course web pages, Blackboard, and can be looked up via OUSA).

Assessment

Internal

- Quizzes (10%). There will be one quiz question in each lecture, each worth 0.5%.
- Two assignments. Due dates:
  - Friday, April 4 (10%)
  - Thursday, May 15 (10%)
- Assessed labs (10% total)

External

- Three hour final exam (60%). You must get at least 40% in the final exam to pass the paper. That is, if there are 100 marks in the final exam, then you must get 40/100 to pass the paper.

Workload expectations

- The generic equivalence for workload in an Otago paper is that “1 point = 10 hours”.
- This is an 18 point paper.
- Allowing a generous 50 hours for final exam preparation that means that in an average week during semester you should be spending 10 hours on COSC201.
- Lectures are two hours, labs are four hours (if you attend the full length) and the tutorial session is one hour, so that still leaves three hours for review, preparation, extra lab work etc.
- The load is not completely uniform and you will be spending a bit more time in weeks leading up to the assignments, and a bit less in others.

Academic integrity

- For COSC201 and indeed most COSC papers, the most basic principle of academic integrity is write your own code and do not share it.
- We encourage you to discuss your work with your peers, ask questions, and help out where you can but just don’t ever give anyone a copy of code that you’ve written (at least not until after the due date!)
- Submissions will be checked for similarity, and we have unfortunately needed to pursue cases of academic misconduct in the past. You really don’t want to get tied up in that procedure (and neither do I).
- Please read the material on academic integrity on the paper’s web pages.
- If you’re at all in doubt about whether something is acceptable – just ask.

Learning objectives

When you successfully complete this paper, you will be able to demonstrate:

- An ability to analyse algorithmic complexity using big-O and related notations
- An understanding of the fundamental structural properties of algorithms including: greedy algorithms, dynamic programming, divide and conquer, depth and breadth first search, and space-time trade-offs
- A knowledge of, and ability to use, fundamental data structures used in sorting and searching, graph, tree and network representations
- An ability to solve practical computational problems including the choice of an appropriate algorithm and/or data structure for the context
- An understanding of how the computational overheads and scalability of algorithms or data structures affect their suitability in applications
- An increased proficiency in programming

Topics

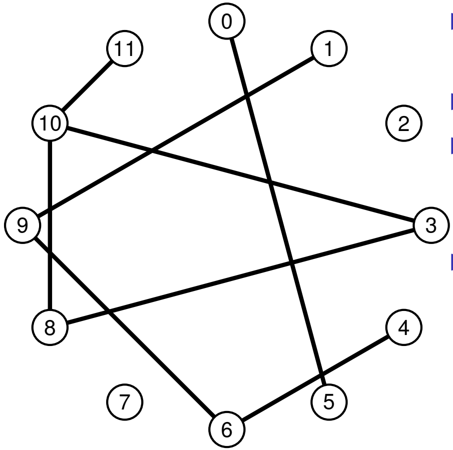
- Extension and refinement of Big-O notation for algorithmic analysis.
- The union-find data structure (a case study)
- Merge sort
- Priority queues and heap sort
- Search trees and balancing
- Hashing and hash maps
- Graphs and graph algorithms (shortest paths, minimal spanning trees)
- Dynamic programming

Themes

- Thinking about scalability
- Using experiments and simulations to reinforce theory
- Reading and learning from code
- Understanding the different axes that are important in computer science:
    - Theory to practice,
    - Problem-solving to development,
    - Design to engineering,
    - Experiment to production,
    - COSC 201 to COSC 202,

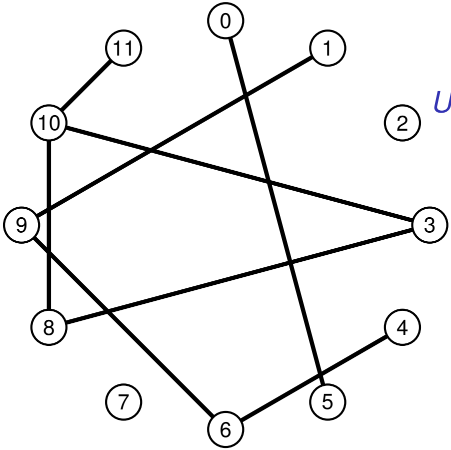
  and beginning to learn how to move smoothly, productively, and appropriately along them.

Disjoint set example



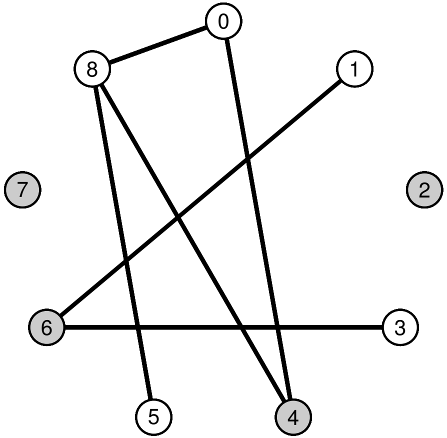
- There are 12 (in general, n) 'objects' (vertices, nodes, circles, dots . . .)
- Some pairs have been connected (by an edge).
- They form groups where two objects are in the same group if there is a sequence of edges between them.
- What are the groups in this example?

Dynamic disjoint set data type



Make(n)	Make a set of n vertices with no edges between them.
Union(x, y)	Connect x and y by an edge (do nothing if they are already connected by an edge)
Find(x)	Find (and return) a representative of the group that x belongs to. If x and y are in the same group then we require that $\mathit{Find}(x) = \mathit{Find}(y)$.

Example



- $\mathit{Make}(9)$
- $\mathit{Union}(8, 0)$
- $\mathit{Find}(8) \rightarrow 0$
- $\mathit{Union}(3, 6)$
- $\mathit{Union}(8, 5)$
- $\mathit{Union}(0, 4)$
- $\mathit{Union}(4, 8)$
- $\mathit{Union}(1, 6)$

x	0	1	2	3	4	5	6	7	8
$\mathit{Find}(x)$	4	6	2	6	4	4	6	7	4

Next time(s)

- Implementing union-find (several ways)
- Analysis of efficiency
- Introducing some relatives of big-O: big-Θ, little-o, and $\sim$
- Experiments!