CS2516
Algorithms and Data Structures II

CS2515 studied Algorithms and Data Structures:

- array-based and linked structure implementations
- data structures: binary search trees, AVL trees, binary heaps, hash tables (bucket arrays and open addressing)
- ADTs: stack, queue, list, priority queue, map
- algorithms for searching and updating BSTs, AVL Trees, Binary Heaps and Hash Tables
- worst case complexity for space and runtime
- implementation in Python

So what is CS2516?

CS2516

Course info

Lecturer: Ken Brown

Lectures: Tuesday, 11am – 12pm, WGB G03
Wednesday, 12pm – 1pm, WGB G03

Lab class: Wednesday, 4pm – 6pm, WGB G24
Labs start in week 2 – 24/01/2024

Assessment: 80% by final written exam (90 minutes)
20% by continuous assessment and in-class tests

Module Objective

"Students should gain expertise in the use and implementation of fundamental data structures and algorithms, and their application in the creation of efficient software."

Outline Syllabus

Revision of CS2515 & exam (= this lecture)
Recursion, Complexity and Analysis
Sorting and Selecting
- including the fundamental limits of sorting

Graphs
- implementing them, finding shortest paths, finding shortest paths between all pairs of nodes, finding trees, etc, with applications in route planning on real road networks

Sample algorithms
- text processing, matching, ??

All programming examples and assignments will be using Python 3

Learning Outcomes

"On successful completion of this module, students should be able to:

- Apply data structures and algorithms appropriately in formulating solutions of meaningful computational problems;
- Implement data structures and algorithms in a modern programming language;
- Analyse simple algorithms;
- Evaluate algorithms on the basis of performance."

Assessment

90 minute written exam, worth 80%
Continuous assessment, worth 20%

You must pass the combined exam and CA

The continuous assessment will consist of one assignment to be submitted.

There will be a repeat exam in August.

Continuous Assessment

Provisional: full details to be confirmed later

1. Experimental evaluation of graph algorithms
    - run experiments to observe the performance of different implementations of graph algorithms
    - submit written report on runtimes, working code for running the experiments, and working code for the algorithms
    - the algorithm code does not need to be your own (but it is better if it is), but all code sources must be fully cited
    - early labs will cover performance evaluation methods for other algorithms
    - mid-semester labs will cover graph implementations
    - expected submission deadline end of week 8

Plagiarism

1. Plagiarism is presenting someone else’s work as your own. It is a violation of UCC Policy and there are strict and severe penalties.

2. You must read and comply with the UCC Policy on Plagiarism www.ucc.ie/en/exams/procedures-regulations/ (and note the new guidance on the use of AI in paragraph 1.2)

3. The Policy applies to *all* work submitted, including software.

4. You can expect that your work will be checked for evidence of plagiarism or collusion.

5. In some circumstances it may be acceptable to reuse a small amount of work by others, but *only* if you provide explicit acknowledgement and justification.

6. If in doubt ask your module lecturer *prior* to submission. Better safe than sorry!

Lab Classes

Lab classes will be run on the same basis as for CS2515
- a supervised 2-hour session each week

These lab classes are important:
- you need to develop your programming skills
- programming the algorithms and data structures will help you understand the processes and the theory
- there will be more exercises than you can do in a single lab session
- the lab sessions will develop code and skills you need for the CA
- you need to work steadily at the exercises each week
- you need to speak up, at the time, in the lab class, if you are struggling
    - the demonstrators are there to help

Self-directed Learning

The university assumes you will be spending at least 8 hours per week on a 5 credit module, not counting revision at the end of the year.

There are 4 hours timetabled for CS2516.

That means you will need to spend at least 4 hours per week working on the material outside of scheduled classes. Most of this will be re-reading the lecture notes, and implementing solutions.

The module is a mix of theory and practice – you will need to practice, or you will not survive.

Textbook

There is no required textbook.

Recommendations – the same as for CS2515:

1. Data Structures and Algorithms in Python,
   Goodrich, Tamassia & Goldwasser
   Wiley

2. Problem Solving with Algorithms and Data Structures
   Miller & Ranum

3. Data Structures and Algorithms with Python
   Lee & Hubbard
   Springer

Other Textbooks

The following three textbooks are also highly recommended for more advanced algorithms material:

1. *Introduction to Algorithms (3e)*,
   Cormen, Leiserson, Rivest and Stein
   MIT Press

2. *Algorithm Design*,
   Kleinberg and Tardos
   Pearson Education

3. *The Algorithm Design Manual*,
   Skiena
   Springer

Revision of CS2515

CS2515

almost everything in Python is an object, stored in some location in memory

2 basic techniques for implementing data structures:

array-based

linked structures

array-based implementations

an array is a sequence of items, each occupying the same memory size, laid out one after another with no gaps in a fixed-size chunk of memory

- ***python lists are implemented using arrays***
- ***a python list can contain items of different types since the array actually stores references (which are same size)***

because we know the size of each item, and the location of the array start, we can compute the location of any item in the array using its index, and jump straight to it – constant time

- ***python does this automatically – e.g. mylist[3]***

to add items beyond the fixed size, create a bigger array (more than you need) and copy everything across

- ***python does this automatically if needed when you call mylist.append(x) ...***

linked structure implementations

a node is an object containing references to an item and to other nodes

a collection of nodes referring to each other creates a data structure; each node can be placed anywhere in memory

to access a particular item, we must iterate through a chain of links (references)

there is no limit on the size (apart from the total memory in the system)

main linked structures in CS2515

linked list – a linear sequence

  if we are given a location, easy to update

  ... but slow to search, and slow to reach an index

binary search tree

  reasonably fast to update and fast to search

  ... assuming the tree is balanced

AVL tree - a balanced binary search tree

  reasonably fast to update and fast to search

  because it is always balanced

the two main types of tree in CS2515

binary search tree – use it for *search*

for each node, all left descendants have smaller items, and all right descendants have bigger items

binary heap – use it for *priority queues*

for each node, all descendants have lower priority (and tree is full, except for maybe lowest level, which is full from left up to a point and then empty)

- **the most efficient implementation uses an array-based list, not a linked tree**

maps and hashing in CS2515

use maps for lookup and storage
- each entry is stored as a key and a value
- keys are unique, but the value stored with a key can change

hash tables allow for efficient maps
- start looking for key in index = compressed hashed key
- bucket arrays have other lists in each cell
- open addressing uses a flat array

(almost always) fast lookups by spreading items around the array, and controlling the size of the underlying array, but occasionally can be slow (long chains, large buckets, etc)

implementation in CS2515

every data structure we considered in CS2515 can be implemented in Python 3

- some of them were specified in full code in lecture notes
- others were the focus of the scheduled lab sessions
- all of them are implemented using Python’s object-oriented features
  - a class defines the data structure
  - objects or instances are created from the class and then used in applications

full understanding of the subtleties of the data structures and algorithms, and the impact of the complexity, only comes when you implement and experiment

The CS2515 exam

Three questions:

Q1: linked lists, stacks, queues, ...

Q2: binary tree structures and priority queues

Q3: maps and hashing

Overall, the marks are good.
Q2 was done best
– more standard routines to apply?

But there are issues in the way you are answering the questions (and, presumably, in the way you are studying the material) ...

Check your answers after you have written them

when adding to or removing from an AVL tree, the result must be an AVL tree

- **binary tree**
- **obeys the BST property**
- **is balanced**

when adding to or removing from a binary heap, the result must be a binary heap

- **parent has higher priority than children**
- **tree is full**

Read the question carefully; make sure you are answering what is asked.

e.g. 'Explain the implementation of array-based lists" – not asking you to define Python lists, or to state what methods they have in Python.

'Explain ... how we get constant time access' means you have to explain it – not just state that it is possible.

'Explain ... how an append method can be implemented for array-based lists so that ... O(1)' – you cannot just say you will use Python's append(x) method; you have to explain how the O(1) on average is obtained by your method

'Show the sequence of returned elements and the final state of ...' – you have to show both of these things

Don't waffle

unless the question explicitly asks for it, don't write an essay – just write down the description / process / sketch / complexity statement

if the question says "explain how an array-based list list can be used to implement a Queue", don't start by writing "I will set out how an array-based list can implement a Queue" - it is just wasting time.

Pay attention to complexity

managing complexity so that we can handle large problems is the whole point of the course. The O(.) statements cannot just be plucked out of thin air.

e.g. in Python, list.pop(i) is **not** O(1)
list.insert(i,x) is **not** O(1)

When implementing a queue, even if you use a head index and avoid pop(0) for dequeue, if you use Python's append(x) for enqueue then you do **not** get O(n) space – you get O(total number of enqueues), which could be exponentially worse.

When implementing a binary heap using a python list, you cannot use sort on the list – it destroys the heap.
You cannot use pop(0) – it is O(n).

Be precise!

You are describing complex processes – vague language is not enough (even if you think it make sense)

If you are asked to explain a process, you have to explain the process - you can’t just describe the final state after the process has run.

The words in a definition <u>matter</u> – you can't mix them up or re-use them for other things

Pay attention to what you write down, and make sure it is precise, correct and complete. That was the theme of CS1112/3.

Descriptions of complex structures, processes and designs

A collection of statements which might be true is not enough for an explanation.

When you write down an explanation you need to ask yourself

- **Have you explained how something works, or how something is structured, rather than describe higher level phenomena that happen to be true?**
- **Could somebody who knows Python rebuild the data structure or algorithm or design just from your description?**
- **Could somebody who passed CS1112/3 understand the structure?**

Some of you, as a group, are inventing your own definitions, or taking them from some other source.

For the definition of Binary Search Tree, many answers said, almost word-for-word:

“A binary search tree is an ADT where for each node its left child is less than its value and its right child is greater than its value ...”

A binary search tree is **not** an ADT.

left child is less ... is true, but it is not enough to define a BST. All left **descendants** must have a lower value, etc.

the ideas are too complex to cram into your memory in a couple of weeks

you need to work steadily during the term, and you need to work steadily at the programming exercises

if you try the exercises and implement the programs as we step through the module,

- demonstrators can explain things to you
- you will understand more deeply
- it will be easier to understand later lectures
- the exam Qs assume you can do the programming ...

attend all the labs and do the exercises

do not ignore the complexity results – the difference in worst-case complexity for different structures and algorithms is one of the most important aspects of these modules.

Next Lecture

Recursion, Complexity and Analysis

Sorting and Selecting

- including the fundamental limits of sorting

Graphs

- implementing them, finding shortest paths, finding shortest paths between all pairs of nodes, finding trees, etc, with applications in route planning on real road networks

Sample algorithms

- text processing, matching, ??