

Data Structures and Algorithms II

Lecture 1

1999

Alison Cawsey
email: alison
room: G36

- About the course
- Motivation: Why learn about algorithms.
- Intro to graph, string and geometric algorithms.
- Revision of ADTs and C++ Classes.

Note: Course Web Site:
<http://~alison/alg/details.html>

1

Books?

- Material on graphs is covered in almost any data structures book, e.g.,

“Data Structures and Algorithms with Object Oriented Design Patterns in C++”, Bruno Preiss, John Wiley, 1999.

Data Structures, Algorithms and Software Principles, Standish, Addison Wesley.

Data Structures Algorithms and Software Principles, Heilman, McGraw Hill, 1996.

- Material on string processing and geometric algorithms is best covered in:

“Algorithms in C++” Sedgewick, (FIRST OR SECOND EDITION, NOT THIRD) pub Addison Wesley, 1992.

Relatively advanced text; useful general reference.

- .. but fairly detailed course notes provided.
- Course notes should cover lecture material - use web site if you want copies of lecture handouts as well.

3

About the Course

- Focuses on useful *algorithms* for range of practical applications.
- Mainly independent of programming language (but C++ examples given).
- Specific topics include:
 - Graph data structure and algorithms.
 - String processing algorithms
 - ✦ String search
 - ✦ Pattern matching
 - ✦ Parsing
 - ✦ Compression
 - ✦ Cryptography
 - Geometric Algorithms
- DOESN'T cover complexity analysis or sorting algorithms. Assume covered elsewhere!

2

Lectures and Labs

Generally:

- Tuesday and Thursday lectures will be on general algorithms and data structures.
- Friday's session will be either a tutorial, or include revision of useful C++ material (OOP, pointers etc).
- Labs Monday and Wednesday. You must get some sections of coursework ticked off in lab.

4

Coursework

- Two main assessed programming exercises, each preceded by two small related lab exercises to be checked in lab.
- Marks: 40% for each main exercise, 20% for lab exercises.
- Exercises:
 - File Compression. Due beginning week 5.
 - Graph search. Due end of week 8.

5

Why learn about algorithms

If you know about available algorithms you can:

- avoid re-inventing the wheel.
- select best algorithm based on information on complexity for different tasks.
- develop own algorithms more easily.
- understand tools which use particular algorithms (e.g., gzip).

7

Motivation: Why learn about algorithms

- Algorithms provide standard methods for doing common programming tasks (e.g., sorting algorithms).
- Experts have:
 - proved these algorithms work correctly.
 - analysed their complexity (efficiency).
e.g., merge sort correctly sorts items and has complexity $O(n \log n)$.
 - Any programmer needs to be aware of standard algorithms, and where to find algorithms to meet new task.

6

Intro to Algorithms: Graph Algorithms

- Graph = network of nodes and links
- Numerous applications use graph data structures (e.g., to represent rail links, circuits, relationships between modules..).
- Standard graph datastructure can be used for all of them.
- Then standard algorithms can be used for these very different applications... to:
 - Search for a route between two cities.
 - Check paths in circuit.
 - Find possible sequence of modules to take.
- We will look at some of the ways of implementing a graph data structure, and some of the commonly used algorithms.

8

String Processing Algorithms

= algorithms for processing sequences of characters (e.g., data in text or graphics files).

Algorithms include:

- Search and pattern matching algorithms, used in all text editors etc.
- Parsers, used in compilers to parse computer programs etc.
- Compression algorithms.
- Encryption algorithms.

Consider WWW applications. All these types of algorithms involved.

9

Revision: Abstract Data Types

- All programming languages have some existing datatypes (e.g., integer, character).
- Each has a set of allowed operations on them (e.g., +, -).
- Notion of abstract data type allows programmer to define new types (e.g., stack).
- Each user-defined datatype must be implemented in terms of built-in datatypes of the language (e.g., class, int).
- BUT: *user* of that datatype (another programmer) shouldn't need to know the underlying implementation.
- A fixed set of operations is defined for the ADT (e.g., push, pop). User only access/modifies datatype via these operations, not by low-level operations on underlying datastructure.
- C++ classes provide a convenient way to implement ADTs.

11

Geometric Algorithms

- Operate on representations of geometric objects (points, lines, polygons etc).
- Useful both for graphics applications, and certain database tasks, where constraints on data can be viewed in terms of geometric objects.

10

ADTs and C++ Classes

- C++ Classes provide a convenient way to represent ADTs.
- The operations can be defined as *public* methods, and the underlying data can be defined as *private*.
- "User" can only use public methods to manipulate data.

e.g.,:

```
Class IntQueue
{
public:
    IntQueue();
    void Clear();
    logical IsEmpty();
    int GetValue();
    logical Add(int NewValue);
    void Remove();
private:
    ListNode *Front, *Rear;
}
```

12

Writing Methods for Classes

- The class definition contains function prototypes for the object's *methods* or operations.
- We define the methods like normal functions, but they are prefixed by `ClassName::`, e.g:

```
logical IntQueue::IsEmpty ()
{
    return logical (Front == NULL)
}
```

- Within the function definition, can access all the private and public data items and functions, without having to specify that they are to operate on the current object.

```
void IntQueue::Clear ()
{
    while (! IsEmpty()) Remove();
}
```

13

Constructors and Destructors

- Constructors are automatically invoked when object is created (e.g., when we declare variable of type `IntQueue`; or use *new*).
- Used to allocate memory and/or initialise data values.

```
IntQueue::IntQueue ()
{
    Front = Rear = NULL;
}
```

- Destructors are called when object is deleted or passes out of scope (e.g., local vars in function, when fn exits).
- Used mainly to de-allocate memory. e.g., could have `IntQueue` destructor:

```
IntQueue::~IntQueue ()
{
    while (! IsEmpty())
        Remove ();
}
```

15

Creating Objects

We can create variables of this new type in normal way:

```
IntQueue myqueue;
myqueue.add(3);
```

But it is common to create *pointers* to an object, either:

```
IntQueue * myqueue; // Create pointer
...                // to IntQueue type.
myqueue = new queue; // Allocate memory
...                // for queue
myqueue->add(3);
...
delete myqueue;     // de-allocate memory
```

This means user has to handle memory allocation/de-allocation using `new/delete`.. but using pointers is more flexible.

14

Summary

- Course focuses on practical algorithms, particularly graph, string and geometric algorithms.
- Important to know standard algorithms so can choose right algorithm for given task.
- Most of course independent of programming language used, but Friday sessions will revise relevant C++ material.

16