Algorithms and Imperative Programming

COMP26120

Todd Davies

May 2, 2015

Introduction

This is a two-semester practical introduction to algorithms and data structures, concentrating on devising and using algorithms, including algorithm design and performance issues as well as 'algorithmic literacy' - knowing what algorithms are available and how and when to use them.

To reflect the emphasis on practical issues, there are two practical (laboratory) hours to each lectured hour. Lectures serve to motivate the subject, orient students, reflect on practical exercises and impart some basic information. A range of practical applications of algorithms will also be presented in the lectures. Other information resources will be important, including a set textbook, which will provide essential support.

The course-unit starts with a 5-week primer on the C programming language, enabling students to become competent programmers in this language as well as in Java (and, possibly, in other languages). This teaching is supported by an on- line C course and extensive laboratory exercises.

There is a follow-up course unit on Advanced Algorithms in the Third Year. This presents the foundational areas of the subject, including (1) measures of algorithmic performance and the classification of computational tasks by the performance of algorithms, (2) formulating and presenting correctness arguments, as well as (3) a range of advanced algorithms, their structure and applications.

Aims

- To make best use of available learning time by encouraging active learning and by transmitting information in the most effective ways.
- To give students a genuine experience of C.
- To make students aware of the importance of algorithmic concerns in real-life Computer Science situations.
- To emphasise practical concerns, rather than mathematical analysis.
- To become confident with a range of data structures and algorithms and able to apply them in realistic tasks.

Additional reading

Algorithm design: foundations, analysis and internet examples - Goodrich, Michael T. and Roberto Tamassia

Contents

1 Trees

1.1 Definition

A tree is an abstract data type for hierarchical storage of information. Each element in a tree has a parent element, and zero or more children elements. The node at the top of the tree is called the root.

A sub-tree is the tree consisting of all the descendents of a child of a tree, including the child itself.

A tree is said to be *ordered* if a linear ordering relation is defined for the children of each node, that is to say that if we wanted to, we could apply this relation to sort the children into an ordered list.

A binary tree is one where each node can have a maximum of two children. A binary tree is *proper* if each node as two (or zero) children.

The depth of a node is the number of ancestors of the node excluding the node itself.

1.2 Tree algorithms

1.2.1 Depth of a node

The depth of a node in a tree is the number of ancestors of the node, excluding the node itself.

```
1  int depth() {
2    if(parent == null) {
3      // We've got no parent; we are the root!
4      return 0;
5    } else {
6      return parent.depth() + 1;
7    }
8  }
```

1.2.2 Tree traversal

Pre-order, in-order and post-order traversal.