Diploma and Part II(General)

Introduction to Algorithms

by

Martin Richards

mr@cl.cam.ac.uk

http://www.cl.cam.ac.uk/users/mr/

University Computer Laboratory
New Museum Site
Pembroke Street
Cambridge, CB2 3QG

The Course

- This course covers some of the material that the Part 1b students were given in their Discrete Mathematics course of last year.
- These student will be joining you for the course Data Structures and Algorithms that I will be giving later this term.
- The notes were originally written by Arthur Norman and slightly modified by Alan Mycroft.
- The course is not directly examinable, but the material it contains is fundamental to many other courses in Computer Science, particularly Data Structures and Algorithms.

Content

- Proof by induction
- Sets, functions
- Relations, graphs
- Reasoning about programs
- $O(f)$ and $\Theta(f)$ notation
- Solution of recurrence formulae

Induction Proofs

To prove a proposition about integer n

1) Prove it true for the base case, eg for $n = 0$

2) On the assumption it is true for $n - 1$ prove it true for n

Simple Example

To prove (by induction) that

$$\sum_{i=0}^{n} i = n(n+1)/2$$

1) Prove, for $n = 0$

$$\sum_{i=0}^{0} i = 0(0+1)/2$$

2) Prove, on the assumption

$$\sum_{i=0}^{n-1} i = (n-1)n/2$$

that

$$\sum_{i=0}^{n} i = n(n+1)/2$$

More General Induction Proofs

To prove a proposition about integer n

1) Prove it true for the base case, eg for $n = 0$

2) On the assumption it is true for all k such that $0 \leq k < n$ prove it true for n

Structural Induction

Well Founded Induction

Orderings

No infinite downward chains

Examples

More Example Proofs

Prove Ackermann's function is total

```
    ack(0, y) = y+1

    ack(x, 0) = ack(x-1, 1)

    ack(x, y) = ack(x-1, ack(x, y-1))
```

Defined in ML

```
fun ack(0, y) = y+1

  | ack(x, 0) = ack(x-1, 1)

  | ack(x, y) = ack(x-1, ack(x, y-1));
```

Lexicographic Ordering

Treat the two arguments of ack as a 2-tuple.

Use lexicographic ordering

```
  (0,0) < (0,1) < (0,2) < ...
< (1,0) < (1,1) < (1,2) < ...
< (2,0) < ...
< ...
```

Proof

To prove ack(x,y) terminates

Base case: `x=0, y=0`

```
ack(x,y) = ack(0,0) = 1
```

Induction:

Prove ack(x,y) terminates assuming

ack(p,q) terminates for all `(p,q) < (x,y)`

case: `x=0`

```
ack(x,y) = ack(0, y) = y+1
```

case: `y=0`

```
ack(x,y) = ack(x, 0) = ack(x-1,1)
```

general case:

```
ack(x,y) = ack(x-1, ack(x, y-1))
```

So `ack(x,y)` terminates for all positive `(x,y)`

Another Example

Consider expressions composed of only

- Even integers
- The operators + and *

Prove that the value of any such expression is even.

Proof

Induction on n, the number of operators in the expression

Base case: $n = 0$

The expression is an even number

Induction: $n > 0$

Prove for n, assuming true for smaller values of n

case 1: The leading operator is +

The operands have fewer operator so can be assumed to yield even integer. The sum of two even numbers is even.

case 2: The leading operator is *

The product of two even numbers is even.

So all such expressions yield even numbers

Eval in ML

```
datatype E = Num of int
           | Add of E * E
           | Mul of E * E;

val e = Add(Num 10, Mul(Num 4, Num 6));

fun eval (Num k) = k
  | eval (Add(x,y)) = eval x + eval y
  | eval (Mul(x,y)) = eval x * eval y;


eval e; (* gives the answer: 34 *)
```

Prove `eval e` always terminates.

Sets

A set is a collection of zero or more distinct elements.

Examples

$\{1, 2, 3\}$

$\{1, \texttt{"string"}, \{\{\}, \{2\}\}, x\}$

$\{x^2 | x \epsilon \{0, 1, \ldots\}\}$

Sets Operations

- Intersection
- Union
- Cartesian Product
- Power Sets
- Infinite Sets
- Set Construction
- Cardinality

Relations

A binary relation is some property that may or may not hold between elements of two sets A and B, say.

Notation

xRy where x is an element of A, y is and element of B, and R is the name of the relation.

Examples

Relations

Kinds of relation

Reflexive

xRx

E.g. $=$

Symmetric

$xRy \Rightarrow yRx$

E.g. $\neq$ or "married to"

Transitive

$xRy \wedge yRz \Rightarrow xRz$

E.g. $<$

Relations

Equivalence Relations

Reflexive, Symmetric and Transitive

E.g. "same colour as" or "related to"

Partial Order

Reflexive, Anti-symmetric and Transitive

E.g. $\leq$ or "subset of"

Closures

Reflexive Closure

Symmetric Closure

Transitive Closure

Relations as Graphs

Adjacency List

Boolean Matrix

Warshall's Algorithm

Transitive Closure on a Boolean Matrix

Cost of Algorithms

What does it cost in time/space to solve a problem of size n by a given algorithm.

Examples

- Sort n integers
- Find the shortest path between 2 vertices of a graph with n vertices
- Determine whether a propositional expression of length n is true for all settings of its variables
- Factorise an n-digit decimal number
- Given x, calculate x^n

Example

Cost of computing x^n

```
LET exp(x, n) = VALOF
{ LET res = 1
  FOR i = 1 TO n DO res := res * x
  RESULTIS res
}
```

Cost = a + f + (m+a+f)n + r = $K_1 + K_2 n$

where

a = cost of assignment

f = cost of FOR loop test

m = cost of multiply

r = cost of returning from a function

Cost functions

$C_{max}(n)$ = maximum cost for problem size n

$C_{mean}(n)$ = mean cost for problem size n

$C_{min}(n)$ = mimimum cost for problem size n

$O(f(n))$ Notation

$C_{max}(n)$ = maximum cost for problem size n

$C_{mean}(n)$ = mean cost for problem size n

$C_{min}(n)$ = mimimum cost for problem size n

Cost $= O(f(n))$ means

$$\text{Cost} \leq kf(n), \text{ for all } n > N$$

i.e. except for a finite number of exceptions

Why the exceptions?

$\Theta(f(n))$ Notation

Cost $= \Theta(f(n))$ means

$$k_1 f(n) \leq \text{Cost} \leq k_2 f(n), \text{ for all } n > N$$

i.e. except for a finite number of exceptions

More formal notation:

$\exists k_1 \quad \exists k_2 \quad \exists K \quad \forall n$

$(n > K \wedge k_1 > 0 \wedge k_2 > 0) \Rightarrow$

$(k_1 f(n) \leq C_{min}(n) \wedge (C_{max}(n) \leq k_2 f(n))$

or

$\exists k_1 > 0 \quad \exists k_2 > 0 \quad \exists K \quad \forall n > K$

$(k_1 f(n) \leq C_{min}(n) \wedge (C_{max}(n) \leq k_2 f(n))$

Logarithms

$\log_2 x = \lg x$

$2^y = x$

$2^{\lg x} = x$

$\lg 1024 = 10$

$\lg 1000000 \simeq 20$

$\lg 1000000000 \simeq 30$

The base does not matter (much)!

$$\begin{aligned} a^y &= x & y &= \log_a x \\ a &= b^z & z &= \log_b a \\ b^{zy} &= x & zy &= \log_b x \\ & & y &= \frac{\log_b x}{\log_b a} \end{aligned}$$