EECS 477: Introduction to algorithms.

Lecture 1

TTh 8:30-10am

Prof. Igor Guskov
guskov@eecs.umich.edu
office: 126 ATL building (AI lab)

September 3, 2002

What's covered

- Basics: relevant math, proof techniques, asymptotic notation
- Design and analysis of algorithms
  - Study existing algorithms, find common (fundamental) patterns
  - Distiguish types of algorithms: greedy, D&C, etc.
  - Learn to analyze new algorithms and implementations
- Computational complexity basics

Why study algorithms?

- to get rich, famous, satisfy hidden aspirations?

- Multiple algorithms for the same problem

- Realize trade-offs

- Understand algorithm efficiency issues: 2^n grows much much faster than n^3.

- Even for the simplest problems: multiplication example

Example: integer multiplication: board

- $235 * 14$
  - American: $235 * 14 = 235 * 10 + 235 * 4 = (235 * 1) * 10 + 235 * 4$
  - *à la russe*: $235 * 14 = 235 * 1110_b = 235 * (8 + 4 + 2) = 235 * 2 + (235 * 2) * 2 + (235 * 2 * 2) * 2$
- a longer example $2356 * 1472$
  - D&C: $2356 * 1472 = (23 * 14) * 10^4 + (23 * 72 + 56 * 14) * 10^2 + 56 * 72$

Example: integer multiplication

- Which algorithm is the best? Can you prove it?

- How do you count the operations?

- Empirically? Multiplications only? Additions only?

- Need some basic tools and definitions before proceeding rigorously: a few lectures to cover those

Notation

- Propositional calculus: Boolean variables, constants, connectives $\vee, \wedge, \neg, \Rightarrow, \Leftrightarrow$

- Quantifiers: $(\forall k \in \mathbf{N})(\exists n \in \mathbf{N})[n > k]$. That is, for any natural number k there is another integer greater than k.

- Duality (good for formal manipulations):

  $\neg(\forall x \in X)[P(x)]$ is equivalent to $(\exists x \in X)[\neg P(x)]$

  $\neg(\exists x \in X)[P(x)]$ is equivalent to $(\forall x \in X)[\neg P(x)]$

Sets

- Set A is well-defined when for any x one can tell if $x \in A$

- *Theorem:* $A = B$ if and only if $A \subset B$ and $B \subset A$

  printing all the primes below 100: check that every printed number is a prime, check that every prime is printed

- Many different objects are defined as sets: e.g. graphs with vertex set V are subsets of $V \times V$.

  Of course, a general set has no structure upon it.

Functions and relations

- Cartesian product: $X \times Y = \{(x, y)|x \in X, y \in Y\}$. So that $\mathbf{R} \times \mathbf{R} = \mathbf{R}^2$ which is a plane.

- Relation: a subset ρ of $X \times Y$, e.g. $\leq$. There are *partial* orders and *total* orders.

- Function $f : X \to Y$: for any $x \in X$ there is just one $y \in Y$: there are $|Y|^{|X|}$ such functions

- When $Y = \{true, false\}$ then $f : X \to Y$ is a predicate

- Injections, surjections, bijections, inverse

Functions and computation: I

- Algorithms implement functions: map the set of possible inputs to the set of outputs

- Boolean functions test properties output $\{0,1\}$. What is the number of Boolean functions on N inputs?

- Reversible computation: do not erase computed bits (erasure dissipates energy) – low power electronic circuits design: (can we find inputs looking at outputs?)

- Another example – lossless compression.

Functions and computation: II

- Program testing: black-box (oracle) model for testing
  - test inputs to verify function properties that must hold: triangle area
  - how to test invertibility?
- Monotone functions: strictly monotone functions have inverses perhaps on a restriction of their image. (injectivity!)
- Restriction on subdomain of inputs

Functions and computation: III

Define functions (= specifying an algorithm?)

- table
- arithmetic formula, composition ($f \circ g$).
- case analysis (if-then)
- set-theoretic: cartesian product, extensions

What else can we do with formal approach?

- Prove general theorems
  - Modus ponens: $a \wedge (a \Rightarrow b)$ implies b
  - $\forall n \in \mathbf{N} : n(n+1)$ is even (no need to consider odd case in your algorithm perhaps).

- reduce our wish list

  The halting theorem: there is no algorithm that, given the text of a program, always finishes and correctly says whether the program is going to finish for all inputs

Proofs

- Exhaustive search, enumeration
  - prove that $a \Rightarrow b$ is the same as $\neg a \vee b$
  - easy but useless for large/infinite domains
- Direct formal proofs
  - Given a theorem $H_1 \wedge H_2 \ldots \Rightarrow C$ establish all hypotheses and make the conclusion, e.g. $1234567895 \mod 11 = 0$
- Deduction: from general statements to special cases

Proofs by contradiction

- *Theorem:* There are infinitely many prime numbers.

- *Proof:*

  Suppose that the set P of prime numbers is finite.

  Form $x = \prod_{k \in P} k$.

  Let d be the smallest integer greater than 1 that divides $x + 1$. Note that $d \in P$.

  Hence, $(x + 1) \mod d = 0$ and $x \mod d = 0$ which is impossible. We conclude that P is infinite.

Proofs by contradiction

- The preceding proof had a constructive component

- $new\_prime(P) := 1 + \prod_{k \in P} k$.

- It would seem that $P_0 = \{2\}, P_{k+1} = P_k \cup \{new\_prime(P_k)\}$ gives an ever growing sets of primes. Does it?

Let's check

- $\{2\}$

- $\{2, 2+1\} = \{2, 3\}$

- $\{2, 3, 2*3+1\} = \{2, 3, 7\}$,

- $\{2, 3, 7, 2*3*7+1\} = \{2, 3, 7, 43\}$

- it seems to be working so far

Induction and deduction

- Wrong.
- 42*43+1 = 139*13
- this induction does not seem to work
- Induction: from particular instances to general laws
- Deduction: from general to particular

Mathematical induction

- Mathematical induction: a rigorous proof procedure
- Need to prove $P(n), \forall n \in \mathbf{N}$
- Two stages
  1. Basis: $P(a)$
  2. Induction step: $(\forall n > a)[P(n-1) \Rightarrow P(n)]$
- Example: $\sum_{i=1}^{n}(2i-1) = n^2$

Mathematical induction: a la russe

```
int russe(int m, int n) {
  if(m==1)
    return n;
  else {
    if(m%2==0)
      return russe(m/2,n*2);
    else
      return n + russe(m/2, n*2);
}
```

We can prove the correctness of the algorithm using induction.