Computer Science as a Liberal Art

the Convergence of Technology and Reason

Natural or Artificial Science?

"Computer Science is no more about computers than astronomy is about telescopes."

-- E. W. Dijkstra

- *Studying a particular computing technology makes it an artificial science.*
- *Building a computer to solve a specific task is engineering.*
- *Studying computation as a natural phenomenon (mechanism or mentalism) is a natural science.*

Reasoning & Reckoning

"REASON ... is nothing but Reckoning"

-- Thomas Hobbes, Leviathan, 1651

Mechanization of Thought:

- Can human thinking be mechanized? (routine, automatic calculations)

  Is all reckoning reasonable?
- Rationalization of Mechanical Processes:
- Can machine calculation be explained by logical reasoning?

The Liberal Arts

Trivium (Three Roads)

Language

/ | \

Grammar **Rhetoric** **Logic**

Quadrivium (Four Roads)

Mathematics

/ \

The discrete The continued

/ \ / \

Absolute Relative Stable Moving

| | | |

Arithmetic **Music** **Geometry** **Astronomy**

Historical Methods of Computing

- fingers; abacus
- *geometry*: rules for solving measurement problems
- *algebra*: rules for solving arithmetical problems
- adding machines (± × ÷)
- modern digital computers

We are in the *Dark Ages* of Computing:

- Even recent technology seems arcane!
- Are there limits to the potential of computers?

Information

- The science of transmitting data:

  text: **100...10** ; or *pictures*: **bits in 2-D**

data	transmits	paper is:
• **Storage:**	*now* to *then*	*permanent* in time
• **Communication:**	*here* to *there*	*portable* in space

main issues

- *Efficiency:* data compression
- *Accuracy:* coding theory

Technological Growth

There has been an *exponential* increase: information capacity doubles every 8-9 months!

Current technology:

- **Storage density** 1 gigabyte/gm (microdrive)
  ~*Encyclopedia Britannica*

- **Communications bandwidth**
  100 terabyte/sec (fiber optic)
  ~*Library of Congress*

Models of Computation

{<u>control processor</u>}
read: ↑ *data* ↓ :write
[…<u>storage memory</u>…]

- Information can be *destroyed*, but never *created*.
- Its capability depends on organization:

Turing Machine: {boolean operations} on *bits* in a
[…<u>linear storage tape</u>…]
RAM model: {arithmetical operations} on *numbers* in an
[…<u>addressible memory</u>…]

Potentials & Limits of Computing

Potentials: The practical aspect of increased processing *speed* will yield the ability to engineer more powerful computers with:

- higher performance hardware
- more complex software

Limits: But the *kinds* of problems which can be solved will be constrained by the fundamental theoretical issues of:

- *Accuracy* (Qualitative)
- *Efficiency* (Quantitative)

Problems and Solutions

- A **problem** is a *specification* which indicates correctness of the input-output relation.
- An **algorithm** is a *process* which transforms data in a step-by-step methodical fashion.

An algorithm *A solves* (or *computes*) problem *P* if:

(*instance*) *P* (*answer*)
encode ↓ ↑ decode
input → [*A*] → output

Euclidean Algorithm

Specify the *greatest common divisor*:
GCD(*i*, *j*) divides *i* and *j*, and is maximal
Solve in an accurate and efficient manner:
(*a* <u>mod</u> *b* = remainder when *a* is divided by *b*)

_ mod _ = → _ mod _ = → _
↑ ↑ ↓ ↑ ↓
30 **21** 9 *3* 0
↓ ↓ ↑
_ mod _ = → _

Designing Algorithms

An *art* whose current state is a wide variety of clever and useful techniques, but no systematic method which works in general, except for...

The Feynman Problem-Solving Method

1. Write down the problem.
2. Think very hard.
3. Write down the solution.

With oracular knowledge one can *get it right*, but what about for mere mortals?

Analyzing Algorithms

Empirical testing of code does not suffice (all possible cases cannot be considered), especially for safety-critical applications:

- air traffic control; medical life support

Absolute certainty requires line-by-line analysis to verify a program for:

- accuracy (*correctness* theory).
- efficiency(*complexity* theory).

Are there any limitations to this?

Universal Computation

"anything which is computable in principle"

- *Intuitively* **computable** (mentalism)

$$\text{input} \rightarrow \left\{ \begin{array}{c} [\text{ idea }] \\ \downarrow \\ [\text{ computer program }] \end{array} \right\} \rightarrow \text{output}$$

- *Effectively* **computable** (mechanism)

This 'Church-Turing' thesis leads to a problem in proving the correctness of arbitrary programs.

Unsolvability Phenomenon

- **Halting problem**: consider "arbitrary" program P as *data*. Does P always halt?

$$P \rightarrow [\ H\ ] \rightarrow \text{yes/no}$$

No procedure for H can exist! Hence,

- There can be no general method for verifying the accuracy of an *arbitrary* program.
- However, it might still be possible to prove the correctness of a specific (class of) program(s).

Computational Complexity

Let *size* of data = |input| + |output| $\rightarrow \infty$

E.g. take 100 digits: GCD (781…270, 69…378)

- Brute force method (exhaustive search) is to take largest of all possible divisors: 10^100 steps ~ 10^80 years
- Euclidean algorithm (repeated remaindering): # steps $\propto$ # digits

Say an algorithm is *feasible* if it requires time proportional to the size of the data.

Intractability Phenomenon

A problem is *intractable* if it does not admit a feasible algorithmic solution.

- Is there a winning strategy from a given chess position on an $n \times n$ board? (This cannot even be *checked* efficiently.)
- Factor a given number into primes (answer can be efficiently checked)

Appears intractable on conventional computer, but has feasible algorithm on quantum computer!

Explicit Definability

Graph: set of nodes related by edges ($a \to b$)

Graph Simplicity (local property): •–––→–––•

- $(x \to y) \Rightarrow (y \to x)$
- $\neg (x \leftrightarrow x)$

Ordering (global property): • < • < • < •

- $(x \neq y) \Rightarrow [(x \to y) \Leftrightarrow \neg (y \to x)]$
- $(x \to y)$ & $(y \to z) \Rightarrow (x \to z)$

Recursive Definability

Reachability in a simple graph ($s \sim t$):

- Is there a *path* of edges from s to t ?

Not explicit, but still tractable. Idea (linear-time):

- Mark all nodes reachable from s, then see if t got marked

Define path in terms of edge and *itself*:

- $(x - y) \Rightarrow (x \sim y)$
- $(x \sim y \sim z) \Rightarrow (x \sim z)$

Implicit Definability

A definition in which one **conjectures** *any* (some / every) result which **substantiates** *the* (unique) answer.

GCD(a, b): guess any integers x and y which make a positive $d = a*x + b*y$ that divides both a and b.

Reach(s, t): guess any bunch of nodes which form *either* a <u>finite</u> chain between s and t; *or* a <u>closed</u> set including s and excluding t.

Factor(n): guess any collection (will be unique) of prime factors whose product is n.

Logic as a Language

The grammatical **perspicacity** of a problem specification (*syntactic*) corresponds closely with (*captures*) the computational **efficiency** of an algorithmic solution (*semantic*).

This fits with Quine's epistemology that we only know the world (of computing) through (the formal) language (of logic).

Frontier of Knowledge: Is there a (mathematical) logic which captures the notion of (physical) computation that is both accurate and efficient?

Speculative Remarks

'**natural**' limitations to computing:

- *mass:* bounded information per node
- *energy:* self-powered <u>automatus</u>

universal data assumption: graphs of <u>all</u> shapes and sizes can be in memory

- data structure I/O: converting arbitrary graph to string
- many hard problems run in linear-time on tree-like graphs

Latest Research: on trees, a method to transform problem specifications into accurate & efficient programs:

logic = implicit definitions; *device* = self-powered automata

Conclusion

The "**Holy Grail**" of Computer Science would be some way to validate *Feynman's Method*, turning algorithmic design from an art into a *science*.

• to transform <u>feasible</u> specifications into correct and efficient algorithms (*automatic programming*)

How closely will abstract thinking (human reasoning) and machine computation (technological devices) converge?