Computational Thinking:
What is the science in computer science?

Prof. Brigitte Pientka

School of Computer Science
McGill University

Computer science is no more about computers than astronomy is about telescopes.
– E. Dijkstra

A grand vision

"Computational Thinking will be a fundamental skill used by everyone in the world in the middle of the 21st century"

J.M. Wing, "Computational Thinking," Communications of the ACM Viewpoint, Mar 2006, pp. 33-35.

A grand vision

"Computational Thinking will be a fundamental skill used by everyone in the world in the middle of the 21st century"

J.M. Wing, "Computational Thinking," Communications of the ACM Viewpoint, Mar 2006, pp. 33-35.

- In many areas, it is already a reality: engineers, biologists, chemists, physicist. . ., anthropologist, psychologists, economist, criminologist

A grand vision

"Computational Thinking will be a fundamental skill used by everyone in the world in the middle of the 21st century"

J.M. Wing, "Computational Thinking," Communications of the ACM Viewpoint, Mar 2006, pp. 33-35.

- In many areas, it is already a reality: engineers, biologists, chemists, physicist. . ., anthropologist, psychologists, economist, criminologist
- Google "Computational X" – replace *X* with your favorite discipline

A grand vision

"Computational Thinking will be a fundamental skill used by everyone in the world in the middle of the 21st century"

J.M. Wing, "Computational Thinking," Communications of the ACM Viewpoint, Mar 2006, pp. 33-35.

- In many areas, it is already a reality: engineers, biologists, chemists, physicist. . ., anthropologist, psychologists, economist, criminologist
- Google "Computational X" – replace *X* with your favorite discipline

What is computational thinking?

What is computational thinking?

Computational thinking is the key

- for solving problems

What is computational thinking?

Computational thinking is the key

- for solving problems
- for achieving what one human alone cannot do

What is computational thinking?

Computational thinking is the key

- for solving problems
- for achieving what one human alone cannot do
- for understanding the power and limits of human intelligence and capabilities of machines

What is computational thinking?

Computational thinking is the key

- for solving problems
- for achieving what one human alone cannot do
- for understanding the power and limits of human intelligence and capabilities of machines

What it is and what it is not

- Conceptual – not just about writing programs
- A way humans think – not computers
- Fundamental – not rote, mechanical skill

What is computational thinking?

Computational thinking is the key

- for solving problems
- for achieving what one human alone cannot do
- for understanding the power and limits of human intelligence and capabilities of machines

What it is and what it is not

- Conceptual – not just about writing programs
- A way humans think – not computers
- Fundamental – not rote, mechanical skill

Computational thinking is drawing fundamentally on concepts from computer science!

The two A's of computational thinking

- Abstraction
  - Helps solve problems
  - Transfer and solve multiple problems
  - Multiple layers of abstraction - Different view points
- Automation
  - Goal is to execute, simulate, observe behavior of abstract problem descriptions

The two A's of computational thinking

- Abstraction
  - Helps solve problems
  - Transfer and solve multiple problems
  - Multiple layers of abstraction - Different view points
- Automation
  - Goal is to execute, simulate, observe behavior of abstract problem descriptions

> Computer Science is a science of abstraction – creating the right model for a problem and devising the appropriate mechanizable techniques to solve it.
>
> - A. Aho and J. Ullman

The two A's of computational thinking

- Abstraction
  - Helps solve problems
  - Transfer and solve multiple problems
  - Multiple layers of abstraction - Different view points
- Automation
  - Goal is to execute, simulate, observe behavior of abstract problem descriptions

Computer Science is a science of abstraction – creating the right model for a problem and devising the appropriate mechanizable techniques to solve it.

- A. Aho and J. Ullman

Example, please !

Example

Search for me in the phone book

Search for me in the phone book

Lessons learned

- We need a way to describe what we did
- We can apply our solution to other search problems
  Search for an artist on your ipod; search for a participant in a list
- There are different solutions; which is the best one?
- What are the requirements we exploited?

Four axes of computational thinking

1. Document solutions (to be able to revisit later)
2. Communicate solutions (to your friend, co-worker, etc.)
3. Analyze and study solutions (Which one is better? Why? Equally powerful?)
4. Implement solutions

Four axes of computational thinking

1. Document solutions (to be able to revisit later)
2. Communicate solutions (to your friend, co-worker, etc.)
3. Analyze and study solutions (Which one is better? Why? Equally powerful?)
4. Implement solutions

Four axes of computational thinking

1. Document solutions (to be able to revisit later)
2. Communicate solutions (to your friend, co-worker, etc.)
3. Analyze and study solutions (Which one is better? Why? Equally powerful?)
4. Implement solutions

This talk: Computer science has diverse roots

- From flow charts to abstract machines (Engineering)
  Focus on 1 and 2
- Reasoning in ancient and modern times (Philosophy, Mathematics)
  Focus on 2, 3 and 4
- Thinking about the limits and power of computation

From flow charts to abstract machines

Flow charts : where they come from – what they are today

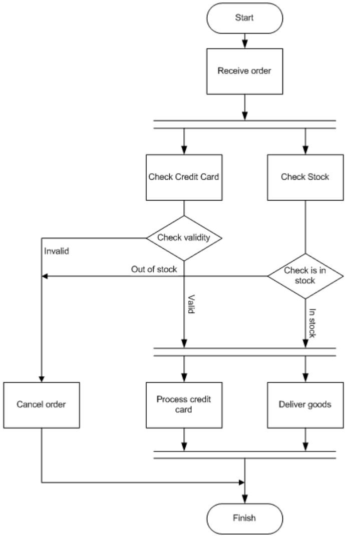
When did flow charts originate?

- 1921: F. Gilberth introduces it to the American Society of Mechanical Engineers (ASME)
- 1930s: Industrial engineering curricula
- 1940s: Reaches industry
- 1950s: Model computer programs

What is their use today?

- Unified Modelling Language (UML)
- Used pervasively in software engineering.

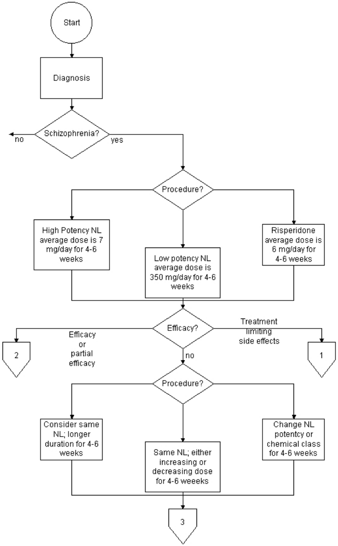
Online order system



Used commonly in other science

- Medicine
- Chemistry
- Biology
- Economics
- ...

Treatment for Schizophrenia



Flow charts can be complex!

Advantages

- Organize the thoughts
- High-level view; abstract over details;

Flow charts can be complex!

Advantages

- Organize the thoughts
- High-level view; abstract over details;

Disadvantages

- Interactions can be complex!

Flow charts can be complex!

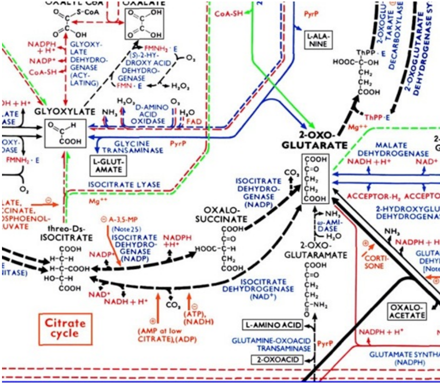
Advantages

- Organize the thoughts
- High-level view; abstract over details;

Disadvantages

- Interactions can be complex!
- No rigorous analysis possible
  What is the shortest path? What components are connected?
- Hard to communicate the meaning.
  What is the meaning of each line?
  What is the meaning of the colors?
- Deriving an implementation not always obvious How can we implement and mechanize the process?

System biology



Flow charts can be complex!

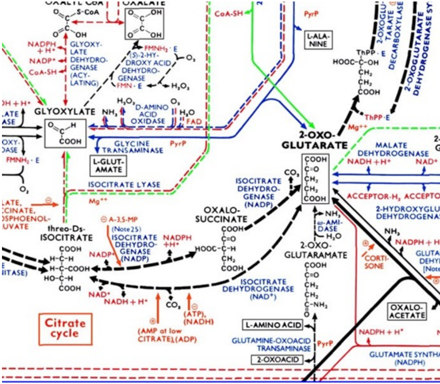
Advantages

- Organize the thoughts
- High-level view; abstract over details;

Disadvantages

- Interactions can be complex!
- No rigorous analysis possible
  What is the shortest path? What components are connected?
- Hard to communicate the meaning.
  What is the meaning of each line?
  What is the meaning of the colors?
- Deriving an implementation not always obvious How can we implement and mechanize the process?

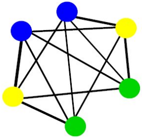
System biology



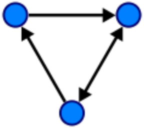
Analyzing flow charts = Study graphs

- Abstract representation of a set of objects where some pairs of the objects are connected by links.

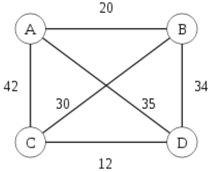
Undirected graph



Directed graph



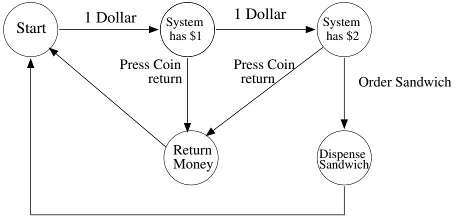
Weighted graph



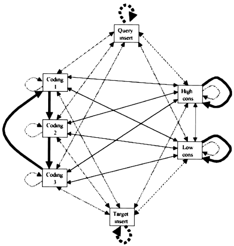
- Some questions about graphs:
  - What is the shortest path?
  - Can we partition graphs in strongly connected components?
  - Optimal route through a graph connecting two nodes?
  - Can we color the nodes such that directly connected nodes will have different colors?

Implementing flow charts = Finite state machine

- Composed of a finite number of states, transitions between those states, and actions - Special case of a graph
- More precise, computational model than flow charts
- Allows us to execute, simulate and observe behavior



Finite state machine depiction of seven state aligner.



Kent W J , Zahler A M Genome Res. 2000;10:1115-1125

Abstract machines and computability

- Abstract machines are a theoretical model of a computer and allow us to describe our problem computationally
  - Finite State Machines / Automata
  - Turing Machines
  - Post Machines
  - Register machines
  - ...

  } (theoretical) languages for computations!

⇒ Amazing fact: These are all formally equivalent!

⇒ Whether you have a Mac or a PC they can compute the same things (theoretically) and are subject to the same limitations.

Church-Turing thesis:

Abstract machines capture the informal notion of computability.

Reasoning in ancient and modern times

The language of logic

Long before there were computers, people were interested in describing computations and thinking computationally.

- Gottfried Wilhelm Leibniz (1646–1716) : Philosopher, mathematician (inventor of calculus), built calculating machine, binary arithmetic
- Leibniz's dream: Compile an *encyclopedia* of all human knowledge.

> 'The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say: Let us calculate [calculemus], without further ado, to see who is right.' (The Art of Discovery 1685, W 51)

Propositional and predicate logic

- George Boole (1815–1864) : Propositional logic
  Logical reasoning $\leftrightarrow$ algebraic reasoning! Formulate the laws of reasoning about classes (concepts) in analogy to the rules of algebra
- Gottlob Frege (1848–1925) : Predicate logic
  - Boole's logic is not expressive enough.
  - We need to express (quantifiers):
    '*For all* x, ...' and '*There exists* an x, such that ...'
- Notation that allows to make proofs gap-free: mechanical, without recourse to intuition

E. g., 'All amounts a, if $a > 2$ and I buy a sandwich then the remaining amount is $a - 2$.

Modern notation:
$\forall a.(\mathit{amount}(a) \wedge a > 2 \wedge \mathit{buy\_sandwich} \rightarrow \mathit{amount}(a-2))$.

- Modus ponens with axioms of arithmetic

Modelling Computation using Logic

- Unifying foundational framework
- Powerful tool for modeling and reasoning about aspects of computation i.e. correctness
- Computation = Constructing a proof
  $\Longrightarrow$ Logic programming
- Translate state transition systems into logic!

"I expect that digital computing machines will eventually stimulate a considerable interest in symbolic logic ... The language in which one communicates with these machines ... forms a sort of symbolic logic."

A. Turing

Modelling Computation using Functions

- μ-recursive functions (Gödel 1930s): precisely describe what is computable
- The λ-calculus (Church 1936): a calculus of anonymous functions. e. g., $(\lambda\, x.\ 2x)$ instead of $f(x) = 2x$
- Modelling Vending Machine:

$$
\begin{array}{lcl}
\mathit{buy\_sandwich}(a) & = & \begin{cases} a-2 & a \geq 2 \\ \perp & \mathit{otherwise} \end{cases} \\
\mathit{buy\_coke}(a) & = & \begin{cases} a-1 & a \geq 1 \\ \perp & \mathit{otherwise} \end{cases}
\end{array}
$$

- Computation = Evaluating functions
  $\Longrightarrow$ Functional Programming

Programs = Proofs

- Gentzen (1935): Calculus of Natural Deduction
  Calculus to capture reasoning practice
- Curry (1958): Observed a connection between logic and functions
- Howard (1969): Observed there is an isomorphism between

proofs	$\Longleftrightarrow$	functions
proposition	$\Longleftrightarrow$	types
proof transformations	$\Longleftrightarrow$	function evaluation

- Highly influential to programming and language design

"For my money, Gentzens natural deduction and Churchs lambda calculus are on a par with Einsteins relativity and Diracs quantum physics for elegance and insight. And the maths are a lot simpler. "

P. Wadler

Programming language paradigms

- Abstract machines $\rightarrow$ imperative programming
  - Series of instructions and commands; loops
  - Maintain state implicitly or explicitely
  - Examples: Assembler, C, Basic, C++, Java
- Logic $\rightarrow$ logic programming
  - No state
  - Driving force: Subset of first-order logic
  - Computation = proof search
  - Examples: Prolog, Datalog, logical frameworks
- λ-calculus $\rightarrow$ functional programming
  - Avoid state
  - Driving force: Recursion, functions, data-types
  - Computation = application of functions to arguments
  - Examples: Lisp, ML, Haskell, F#

"A language that doesn't affect the way you think about programming, is not worth knowing."

- Alan Perlis

Thinking about the limits and power of computation

Thinking about the limits and power of computation

- What are the limitations of computations?
- Can we solve any problem?
- Can we solve any problem efficiently, i.e. in practice?

Limitations of computing

- Does there exist a yes-or-no answer for every problem?

  $\Rightarrow$ Turing (1936) Halting problem

  $\Rightarrow$ Church (1936) Decision problem

- Are there problems which cannot be computed?

Limitations of computing

- Does there exist a yes-or-no answer for every problem?

  $\Rightarrow$ Turing (1936) Halting problem

  $\Rightarrow$ Church (1936) Decision problem

- Are there problems which cannot be computed? Yes!

Limitations of computing

- Does there exist a yes-or-no answer for every problem?

  $\Rightarrow$ Turing (1936) Halting problem

  $\Rightarrow$ Church (1936) Decision problem

- Are there problems which cannot be computed? Yes!

- These are formal *limitations* of *all* models of computability (even your PC or MacBook!)

Limitations of computing

- Does there exist a yes-or-no answer for every problem?

  $\Rightarrow$ Turing (1936) Halting problem

  $\Rightarrow$ Church (1936) Decision problem

- Are there problems which cannot be computed? Yes!

- These are formal *limitations* of *all* models of computability (even your PC or MacBook!)

- Suppose that solutions to a problem can be verified quickly. Then, can the solutions themselves also be computed quickly?

Limitations of computing

- Does there exist a yes-or-no answer for every problem?

  $\Rightarrow$ Turing (1936) Halting problem

  $\Rightarrow$ Church (1936) Decision problem

- Are there problems which cannot be computed? Yes!

- These are formal *limitations* of *all* models of computability (even your PC or MacBook!)

- Suppose that solutions to a problem can be verified quickly. Then, can the solutions themselves also be computed quickly?
  - Stipulated by S. Cook in 1971 – as of today: unsolved
  - 1 Million Dollar prize for a solution offered by the Clay Institute of Mathematics!

\$1 M Question: P = NP?

Example NP-Problem:

> Suppose that you are organizing housing accommodations for a group of 400 students. Space is limited and only 100 students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice.

- Easy to check if solution is satisfactory
- Generating a solution is hard – it is completely impractical!

\$1 M Question: P = NP?

Example NP-Problem:

> Suppose that you are organizing housing accommodations for a group of 400 students. Space is limited and only 100 students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice.

- Easy to check if solution is satisfactory
- Generating a solution is hard – it is completely impractical!
  N = combinations of choosing 100 students from 400 applicants ;
  N > number of atoms in the universe

$1 M Question: P = NP?

Example NP-Problem:

> Suppose that you are organizing housing accommodations for a group of 400 students. Space is limited and only 100 students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice.

- Easy to check if solution is satisfactory
- Generating a solution is hard – it is completely impractical!
  N = combinations of choosing 100 students from 400 applicants ;
  N > number of atoms in the universe
- The problem cannot be solved by brute force

\$1 M Question: P = NP?

Example NP-Problem:

> Suppose that you are organizing housing accommodations for a group of 400 students. Space is limited and only 100 students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice.

- Easy to check if solution is satisfactory
- Generating a solution is hard – it is completely impractical!
  N = combinations of choosing 100 students from 400 applicants ;
  N > number of atoms in the universe
- The problem cannot be solved by brute force – but can we compute a solution quickly by some procedure?

Consequences of P = NP

Negative consequences:

- Cryptography: We rely on certain problems being difficult; A constructive, efficient solution could break many existing cryptosystems such as Public-key cryptography which is used in transactions with banks, online shopping sites, etc.

Positive consequences (enormous!):

- Rendering tractable many currently mathematically intractable problems.
- Some NP problems: Travelling salesman problem, logistics, protein structure prediction, ...

> " ...it would transform mathematics by allowing a computer to find a formal proof of any theorem which has a proof of a reasonable length, since formal proofs can easily be recognized in polynomial time. Example problems may well include all of the CMI prize problems."
>
> Stephen Cook

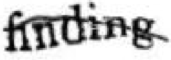
Beyond the computer as a machine

What is computable when we consider the computer as the combination of human and machine? How can we exploit the limitations of computing?

What is computable when we consider the computer as the combination of human and machine? How can we exploit the limitations of computing?

Captcha : Completely Automated Public Turing test to tell Computers and Humans Apart

- Challenge-response test used in computing to ensure that the response is not generated by a computer.



Take home message

- Computational thinking is a fundamental skill – it is learning to think at different layers of abstractions. It is as fundamental as knowing some basics about probability theory or discrete algebra.
- It has fascinated human beings in the past – it continues to fascinate us today.
- Computer science has deep philosophical, mathematical and engineering challenges.
- Computers (machine and human!) allow us
  - to go beyond solving problems on paper
  - to go beyond what one human being could achieve
  - to explore and understand our surrounding world
- Computer science is about computational thinking – it is challenging and tests the limits of our creativity and intelligence.

That's it!

That's it!

Thank you.