Theory of Computation

Adam Campbell

June 15, 2008

Brief history of computation

- computational devices and methods date back thousands of years
  - abacus; over 4000 years old, Sumer
  - Euclid's *Elements*; around 300 B.C.
    - one of the first algorithms
    - Greatest Common Divisor of integers *a* and *b*
- algorithm: step-by-step, unambiguously defined process for solving a problem in a finite number of steps
- computation in the last 100 years
  - Is there a mathematical formalism that allows for any computable function to be solved algorithmically?

The Decision Problem

- David Hilbert's Entscheidungsproblem, 1928
- led to formalization of algorithm, 1936
  - Alonzo Church and Stephen Cole Kleene, λ-calculus
    - recursive functions
    - basis of functional programming
  - Alan Turing, Turing machines

Church-Turing Thesis

- each person thought their models defined the "effectively computable" functions
  - that is, any computable function can be computed on a Turing machine (or represented in λ-calculus)
- λ-calculus and Turing machines were found to define an equivalent set of functions
- not proven, but thought to be true due to large number of equivalent models
  - general recursion, counter machines, register machines, inhibitor Petri nets, cellular automata, ...

The Turing machine

- currently the simplest model of computation
- a tape of symbols, a read/write head, an internal state, and a transition table
- 7-tuple, $M = \langle Q, \Gamma, B, \Sigma, \delta, q_0, F \rangle$
  - Q: finite set of states
  - Γ: finite set of tape symbols
  - $B \in \Gamma$: blank symbol, used to delimit end of input
  - $\Sigma \subseteq \Gamma - \{B\}$: set of input symbols
  - $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, N\}$: transition function
  - $q_0 \in Q$: initial state
  - $F \subseteq Q$: accepting states

An example

- Given a string of zeroes, is its length even?
  - $Q = \{ODD, EVEN, YES, NO\}$
  - $\Gamma = \{0, B\}$
  - $B = B$
  - $\Sigma = \{0\}$
  - $\delta =$ see below
  - $q_0 = EVEN$
  - $F = \{YES\}$

Tape Symbol	**Current State *EVEN***	**Current State *ODD***
0	$ODD, 0, R$	$EVEN, 0, R$
B	YES, B, N	NO, B, N

Categorizing problems

- decision problems
  - the problems with *yes*/*no*, *true*/*false*, 1/0, etc. answers
- decidable problems
  - if an algorithm exists that halts for both *yes* and *no* instances
- semi-decidable problems
  - if an algorithm exists that halts for *yes* instances and doesn't necessarily halt for *no* instances
- undecidable problems
  - if there exists no algorithm that can determine all *yes* instances

The Halting Problem is undecidable

- INPUT: a program P and its data D
- QUESTION: Will P halt when given D?
- proof by contradiction

$$H(P,D) = \begin{cases} HALT\text{, if } P \text{ halts on } D \\ LOOP\text{, if } P \text{ does not halt on } D \end{cases}$$

$$Q(P) = \begin{cases} HALT\text{, if } H(P,P) = LOOP \\ loop\ forever\text{, if } H(P,P) = HALT \end{cases}$$

- contradiction when...

$$Q(Q) = \begin{cases} HALT\text{, if } H(Q,Q) = LOOP \\ loop\ forever\text{, if } H(Q,Q) = HALT \end{cases}$$

Classifying decidable problems

- classifying problems by time and space requirements relative to size of input
- P, NP, NP-Complete, NP-Hard, Co-NP, ...
- Non-deterministic Turing Machines
  - Q: finite set of states
  - Γ: finite set of tape symbols
  - $B \in \Gamma$: blank symbol, used to delimit end of input
  - $\Sigma \subseteq \Gamma - \{B\}$: set of input symbols
  - $\delta : Q \times \Gamma \to \{Q \times \Gamma \times \{L, R, N\}\}$: transition function
  - $q_0 \in Q$: initial state
  - $F \subseteq Q$: accepting states

P

- those decision problems that can be solved on a deterministic TM in polynomial time
  - primality testing
  - minimal spanning tree
  - 2-coloring a graph, ...
- n^{5000} is polynomial, but not efficient

NP

- those decision problems that can be solved on a non-deterministic TM in polynomial time
- *P* is a subset of *NP*
- to show a problem is in NP
  - decision problem
  - a given solution should be verifiable in deterministic polynomial time

Graph coloring

- INPUT: Graph $G = (V, E)$, integer k
- QUESTION: Can G be colored with $\leq k$ colors?
- $GRAPH - COLORING \in NP$
  - decision problem
  - given a coloring of G, you can verify it in polynomial time

NP-Complete

- a problem X is complete for a class C if
  - $X \in C$
  - for every $S \in C$, there is a polynomial reduction from S to X
- a reduction of S to X takes each instance of $s \in S$ and maps it to an instance of $x \in X$ such that $x = YES$ iff $s = YES$
- basically, X is more difficult (or at least as difficult) as every other problem in C
- so, if X can be solved in polynomial time, so can every other problem in C

K-SAT

- INPUT: Boolean expression using *AND*, *OR*, and *NOT*, and a list of variables
- QUESTION: Is there an assignment of *TRUE*/*FALSE* values to the variables that satisfies the boolean expression?
  - $(v_{11} \vee v_{12} \vee \neg v_{13}) \wedge$
  - $(v_{21} \vee \neg v_{22} \vee v_{23}) \wedge$
  - $(v_{31} \vee \neg v_{32} \vee \neg v_{33}) \wedge$
  - ...
- Cook's theorem (1971) shows that $K - SAT$ is NP-Complete; first known NP-Complete problem
- the proof is beyond the scope of this tutorial
- essentially, any NTM can be transformed in polynomial time to a Boolean satisfiability problem

Showing other problems are in NP-Complete

- need only to show that an existing problem in NP-Complete can be reduced in polynomial time to your problem
- Richard Karp (1972) showed 21 more NP-Complete problems
- Garey and Johnson (1974) is an excellent resource for NP-Complete problems
- thousands of known NP-Complete problems
- currently, none of them have a polynomial time algorithm
- if one goes down, they all go down

Interesting problems

- MAX-CLIQUE
- INPUT: Graph $G = (V, E)$, integer k
- QUESTION: Does there exist a clique in G containing at least k vertices?
- is an element of NP-Complete

Interesting problems

- MAX-CLIQUE-5
- INPUT: Graph $G = (V, E)$
- QUESTION: Does there exist a clique in G containing at least 5 vertices?
- is an element of P
- n^5 algorithm

Summary

- equivalent models of computation
  - λ-calculus, Turing Machines, general recursive functions, ...
- decision problems
  - decidable, semi-decidable, undecidable
- classes of problems
  - P, NP, NP-Complete

Further topics

- PSPACE, NEXP, ...
- P = NP?
- Information theoretic proof that $P \neq NP$?
  - Can we prove that an exponential number of steps is required on a deterministic Turing machine in order to obtain enough information to find a solution to NP-Complete problems?