Computational Complexity

Last time we studied sorting algorithms, and measured their performance by the number of comparisons.

In the next two lectures, we will study general "decision problems", measured by the number of computations.

Decision problems are those with yes/no answers, like:

- are these two graphs isomorphic?
- is this flow maximum?
- does this graph have a Hamiltonian cycle?

Turing Machines

In order to make these questions precise, we need to specify a "model of computation".

A Turing machine, informally, is a finite machine equipped with an infinite "tape" where it can read and write symbols from some finite "alphabet."

Formally, a Turing machine is a quadruple $M = (K, \Sigma, \delta, s)$ where:

- K is a finite set of "states" (think of K as the set of all possible states of RAM on a computer)
- Σ is a finite set of symbols, called the "alphabet" (think of Σ as either $\{0,1\}$ or all letters and digits or all unicode characters)
- δ is the transition function which tells the machine what to do once it has read a symbol, given what state it is in. Possible actions are:
  - halt
  - return "yes" or "no"
  - change states
  - write a symbol on the tape at the current position
  - Move the tape.

  (Think of δ as the program.)
  Technically, δ is a function from
  $$K \times \Sigma$$
  to
  $$(K \cup \{\text{halt}, \text{yes}, \text{no}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}.$$
- s is the state to start in.

Goal

Encode a problem as a string with letters from Σ.

Feed this string to the Turing machine.

If the Turing machine returns "yes", then the problem has a solution. If the Turing machine returns "no", then it does not.

This gives a correspondance:

algorithms $\longleftrightarrow$ Turing machines
problems $\longleftrightarrow$ strings (words)

The Church-Turing Thesis

Every function that may be calculated can be computed by a Turing machine.

At some level, this is a theorem that has been proved. However, there is no clearcut definition of "calculable" functions, so this remains a "thesis."

Philosophical Implications

Are the laws of physics computable?

If so, then our universe is equivalent to a Turing machine. Free will is impossible. Etc.

If not, why can't we harness one of these incomputable physical events to build a "hypercomputer"?

Open: are all quantum mechanical events (Turing-)computable?

Roger Penrose: the human mind uses quantum-mechanical "non-algorithmic" computation.

Boolean functions

An n-ary Boolean function is a map

$$f: \{\text{true}, \text{false}\}^n \to \{\text{true}, \text{false}\}.$$

For example, $\vee$ (or), $\wedge$ (and), $\Rightarrow$ (implies), and $\Leftrightarrow$ (iff) are four of the sixteen binary Boolean functions.

Disjunctive Normal Form

A Boolean expression is in disjunctive normal form (DNF) if it is a disjunction of conjunctions, e.g.,

$(x_1 \wedge x_3) \vee (\neg x_1 \wedge x_2) \vee (\neg x_2 \vee x_3)$.

Fact: Every n-ary Boolean function can be expressed as a DNF Boolean expression involving variables $x_1, \ldots, x_n$.

Proof: Let $T \subseteq \{\text{true}, \text{false}\}^n$ that make f true. For each $\vec{t} \in T$, let $\delta_{\vec{t}}$ be the conjunction of all variables x_i with $t_i = \text{true}$ with negations of all x_i with $t_i = \text{false}$. Then $f \equiv \bigvee_{\vec{t} \in T} \delta_{\vec{t}}$. ■

Note: this expression may have exponential length.

Conjunctive Normal Form (CNF)

A Boolean expression is in conjunctive normal form (CNF) if it is a conjunction of disjunctions, e.g.,

$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$.

(Question: is there any assignment of true/false to x_1, x_2, x_3 that makes this true? I.e., is this expression satisfiable?)

Fact: Every n-ary Boolean function can be expressed as a CNF Boolean expression in the variables $x_1, \ldots, x_n$.

Proof: Let $F \subseteq \{\text{true}, \text{false}\}^n$ be the set of assignments that make f false. For each $\vec{f} \in F$, let $K_{\vec{f}}$ be the disjunction of all variables x_i with $f_i = \text{false}$, and with negations of all variables x_i with $f_i = \text{true}$. Then $f \equiv \bigwedge_{\vec{f} \in F} K_{\vec{f}}$. ■

SATisfaction

The SAT problem is simply: given a Boolean expression in CNF form, determine if it is satisfiable.

Upperbound: SAT can be solved by exhaustive search.

But: This requires exponential time.

SAT is not too easy

It is commonly believed that SAT requires an exponential amount of time, or at least, super-polynomial.

Yet, the best we know today is:

Theorem (Williams 2006): There is no Turing Machine that can solve SAT in less than $n^{1.8}$ time.

K-SAT

Every CNF expression is the conjunction of a number of clauses.

The k-SAT problem is: given a CNF Boolean expression where every clause has k variables (also called "literals"), determine if it is satisfiable.

3-SAT is no easier than SAT

Consider an arbitrary CNF expression,

$$c_1 \wedge c_2 \wedge \cdots \wedge c_m,$$

where the c_i's are disjunctions containing variables and their negations.

We will produce an equivalent 3-SAT expression.

Case 1: $c_i = x_1$ (without loss)

Replace c_i by

$$(x_1 \vee y \vee z) \wedge (x_1 \vee y \vee \neg z) \wedge (x_1 \vee \neg y \vee z) \wedge (x_1 \vee \neg y \vee \neg z),$$

where y and z are new variables.

Case 2: $c_i = x_1 \vee x_2$ (without loss)

Replace c_i by

$$(x_1 \vee x_2 \vee y) \wedge (x_1 \vee x_2 \vee \neg y),$$

where y is a new variable.

Case 3: Do nothing.

Case ≥ 4: $c_i = x_1 \vee x_2 \vee \cdots \vee x_m$ (without loss).

Replace c_i by

$$(x_1 \vee x_2 \vee y_1) \wedge (\neg y_1 \vee x_3 \vee y_2) \wedge (\neg y_2 \vee x_4 \vee y_3) \wedge \cdots \wedge (\neg y_{m-3} \vee x_{m-1} \vee x_m).$$