1 Automata, Computability, and Complexity

Central areas of theory of computation include Automata, Computability, and Complexity, which are linked by the question:

What are the fundamental capabilities and limitations of computers?

This question is interpreted differently in each of these areas.

Complexity Theory

Some computer problems are easy, some are hard. For example sorting numbers is an easy one, whereas a scheduling problem, where you must find a schedule of classes for the entire university to satisfy some constraints, such as no two classes take place in the same room at the same time, is much harder. If you have just a thousand classes, finding the best schedule may require centuries, even with a supercomputer. The central question of complexity theory is

What makes some problems computationally hard and others easy?

Remarkably we don't know the answer to it, though it has been intensely researched for over 40 years.

Computability Theory

During the first half of the past century, mathematicians discovered that certain basic problems cannot be solved by computers (e.g. whether a mathematical statement is true or false). In complexity theory, the objective is to classify problems that are easy ones and hard ones; whereas, in computability theory, the classification of problems is by those that are solvable and those that are not. Computability theory introduces several of the concepts used in complexity theory.

Automata Theory

Automata theory deals with the definitions and properties of mathematical models of computation, which play a role in several applied areas of computer science (finite automaton used in text processing, compilers, etc., and context-free grammars used in programming languages and artificial intelligence).

Automata theory is an excellent place to start studying theory of computation. The theories of computability and complexity require a precise definition of a computer.

2 Mathematical Notions and Terminology

Sets

A **set** is a group of objects, called its **elements** or **members**, represented as a unit. Examples include $S = \{7, 21, 57\}$ and $S = \{n | n = m^2 \; for \; some \; m \in \mathcal{N}\}$, where $\mathcal{N} = \{1, 2, 3, ...\}$ is an *infinite* set of natural numbers. The set with no members is called the **empty** set, written as $\emptyset$.

A is a **subset** of B, written $A \subseteq B$, if every member of A also is a member of B. We say that A is a **proper** subset of B, written $A \subsetneq B$, if A is a subset of B, and not equal to B.

Union ($A \cup B$), **intersection** ($A \cap B$), and **complement** ($\bar{A}$) are useful operators defined on sets. The **power set** of A is the set of all subsets of A.

Sequences and Tuples

A **sequence** of objects is an ordered list of these objects such as $\{7, 21, 57\}$. The order doesn't matter in a set, but in a sequence it does. A sequence with k elements is a **k-tuple**.

If A and B are two sets, the Cartesian product or cross product of A and B, written $A \times B$, is the set of all ordered pairs wherein the first element is a member of A, and the second element is a member of B.

Functions and Relations

A **function** $f : D \to R$ defines a mapping from inputs in its domain D to outputs in its range R. If f is a function whose output value is b when the input value is a, we write $f(a) = b$.

A function that use all the elements of the range is said to be **onto** the range.

A function with k arguments is called a **k-ary** function.

A **predicate** or **property** is a function whose range is $\{TRUE, FALSE\}$. A property whose domain is a set of k-tuples $A \times \ldots \times A$ is called a **relation** or a **k-ary relation** on A.

For example:

beats	SCISSORS	PAPER	STONE
SCISSORS	F	T	F
PAPER	F	F	T
STONE	T	F	F

A binary relation R is an equivalence relation if R satisfies:

1. R is **reflexive** if for every x, xRx;
2. R is **symmetric** if for every x and y, xRy implies yRx; and
3. R is **transitive** if for every x, y, and z, xRy and yRz implies xRz.

Example:
Equivalence relation $\equiv_7$. For $i, j \in \mathcal{N}$, we say $i \equiv_7 j$ if $i - j$ is a multiple of 7.

1. Reflexive: $i \equiv_7 i$ since $i - i = 0$, which is a multiple of 7.
2. Symmetric: If $i - j$ is a multiple of 7, then $j - i$ is as well.
3. Transitive: If $i - j$ and $j - k$ are both multiples of 7, then $i - k = (i - j) + (j - k)$ is the sum of two multiples of 7, which is a multiple of 7.

Graphs

An ***undirected graph***, or simply a ***graph***, is a set of discrete objects called ***nodes*** or ***vertices***, where some of these objects are joined by links called ***edges***. Formally, a graph $G = (V, E)$ is defined by its non-empty vertex set V, and its edge set E (Figure 1).

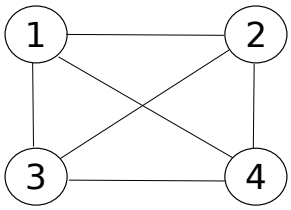

Figure 1: A sample graph $G = (V = \{1, 2, 3, 4\}, E = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\})$ represented with a diagram.

The number of edges at a particular node is the ***degree*** of that node. No more than one edge is allowed in between any two nodes. We may allow an edge from a node to itself, called a ***self-loop***, depending on the situtation.

We say that graph G is a ***subgraph*** of graph H if the nodes of G are a subset of the nodes of H, and the edges of G are the edges of H on the corresponding nodes (Figure 2).

A ***path*** in a graph is a sequence of nodes connected by edges. A ***simple path*** is a path that doesn't repeat any nodes. A graph is ***connected*** if every two nodes have a path between them. A path is a ***cycle*** if it starts

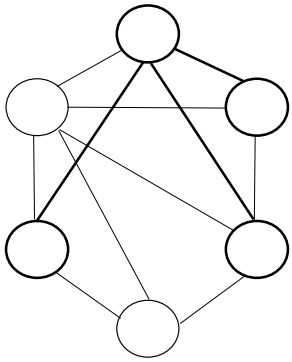

Figure 2: Graph G (shown darker) is a subgraph of H.

and ends in the same node. A ***simple cycle*** is one that contains at least three nodes and repeats only the first and last nodes.

A graph is a ***tree*** if it is connected and has no simple cycles. A tree may contain a specially designated node, called the ***root***. The nodes of degree 1 in a tree, other than the root, are called the ***leaves*** of the tree.

A ***directed graph*** has directed edges. In a directed graph, we represent an edge from i to j as a pair (i, j) (Figure 3). The number of edges coming into and going out of a node respectively define the node's ***in-*** and ***out-degrees***.

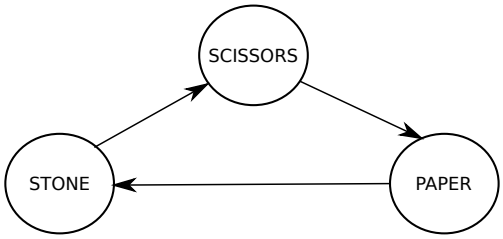

Figure 3: A directed graph $G = (V = \{S_c, P, S_t\}, E = \{(S_c, P), (P, S_t), (S_t, S_c)\})$ of the relation ***beats***.

A path in which all the edges point in the same direction as its steps is called a ***directed graph***. A directed graph is ***strongly connected*** if a directed path connects every two nodes.

Strings and Languages

An ***alphabet*** is a non-empty finite set, where the members are the ***symbols*** of the alphabet. A ***string over an alphabet*** is a finite sequence of symbols from that alphabet, usually written next to one another and not separated by commas (eg. 01001 is a string over $\Sigma = \{0, 1\}$). If w is a string over Σ, the length of w, written $|w|$, is the number of symbol it contains. The ***reverse*** of a string w, written w^R , is the string obtained by writing w in the opposite order. String z is a ***substring*** of w if z appears consecutively within w.

Let $x = x_1 \dots x_m$ and $y = y_1 \dots y_n$. The concatenation of x and y, written xy, equals $x_1 \dots x_m y_1 \dots y_n$, a string of length $m + n$. To denote the concatenation of a string with itself many times, we use x^k to mean $xx \dots x$ (k times).

The ***lexicographical order*** of strings is the same as the familiar dictionary order. We will occasionally use a modified lexicographical order, called ***shortlex order*** or simply ***string order***, that is identical to lexicographical order, except that shorter strings precede longer strings (e.g. $, 0, 1, 00, 01, 10, 11, 000, ...$).

We say that string x is a ***prefix*** of string y if a string z exists where $xz = y$, and that x is a ***proper prefix*** of y if in addition $x \neq y$. A ***language*** is a set of strings. A language is ***prefix-free*** if no member is a proper prefix of another member.

Boolean Logic

Boolean logic is a mathematical system built around the two values $TRUE$ and $FALSE$. Boolean values can be manipulated with Boolean operators to form simple statements to more complex expressions. A unary operation for Boolean values is the ***negation*** or NOT, denoted by $\neg$, whereas binary operators include ***conjunction*** or AND operation, denoted by $\wedge$, ***disjunction*** or OR operation, denoted by $\vee$, ***equality***, denoted by $\leftrightarrow$, and ***implication***, denoted by $\rightarrow$.

Distributive law comes in handy when we manipulate Boolean expressions:

- $P \wedge (Q \vee R) \leftrightarrow (P \wedge Q) \vee (P \wedge R)$
- $P \vee (Q \wedge R) \leftrightarrow (P \vee Q) \wedge (P \vee R)$

3 Definitions, Theorems, and Proofs

Definitions describe the objects and notions. After definitions are made, we usually make ***mathematical statements*** about them. Typically, a statement expresses that some object has certain property. The statement may or may not be true but it must be precise (i.e. no ambiguity). A ***proof*** is a convincing (in an absolute sense; beyond any doubt vs. beyond reasonable doubt) logical argument that a statement is true.

A ***theorem*** is a mathematical statement proved true. Occasionally we prove statements, called ***lemmas***, that are interesting only because they assist in the proof of another, more significant statement. Occasionally, a theorem or its proof may allow us to conclude that other, related statements are true. These statements are called ***corollaries*** of the theorem.

Finding Proofs

The only way to determine the truth or falsity of a mathematical statement is with a mathematical proof, which relies only on the facts (and those facts derived from the initial set of facts). One cannot base a mathematical proof on informal arguments, intuition, handwaving, and alike.

4 Types of Proof

Following types of arguments are frequently used for mathematical proofs. Note that a proof may contain more than one type of argument because the proof may contain within it several different subproofs.

Proof by Construction

Proof by construction is a technique used by starting from given facts (and rules of mathematics) to derive newer facts until we reach the mathematical statement we are given to prove.

Example: A graph is k-regular if every node in the graph has degree k.

Theorem 4.1 *For each even number $n > 2$ there exists a 3-regular graph with n nodes.*

Proof Construct graph $G = (V, E)$ with $V = (0, 1, 2, \ldots, n-1)$ and $E = \{\{i, i+1\}\ for\ 0 \leq i \leq n-2 \cup \{\{n-1, 0\}\} \cup \{\{i, i+n/2\}\}\ for\ 0 \leq i \leq n/2-1\}$. First 2 relations make the graph a circle. The last relation puts edges between nodes in opposite sides of the circle, making the degrees of all nodes 3. ∎

Proof by Contradiction

Proof by contradiction is to add the negation of the mathematical statement (our assumption) we are given to prove to the given facts, to eventually reach an obviously false consequence, also called a contradiction (for every possible case). This will let us conclude that since the given facts cannot be false, our assumption must be incorrect, making our initial mathematical statement true.

Example:

Theorem 4.2 *$\sqrt{2}$ is irrational.*

Proof Assume $\sqrt{2}$ is rational. Then:

$$\sqrt{2} = \frac{m}{n}$$

If both m and n are divisible by the same number, reduce them. Therefore at least one of them is an odd number. Continue:

$$n\sqrt{2} = m$$
$$2n^2 = m^2$$

Here, m^2 is even; therefore m is even. Replace $m = 2k$.

$$2n^2 = (2k)^2$$
$$2n^2 = 4k^2$$
$$n^2 = 2k^2$$

Now we observe that n^2 is even, thus n is even. But in the beginning we said at least one of m, n is odd. We have a contradiction in the initial assumption of $\sqrt{2}$ is rational; thus it is irrational. ∎

Proof by Induction

Proof by induction is an advanced method used to show that all elements of an infinite set have a specified property.

Every proof of induction consists of two parts, the ***basis***, and the ***induction*** step. Supposing the infinite set over which we are to prove a property $\mathcal{P}(k)$, where $k \in \mathcal{N}$ and $\mathcal{N} = \{1, 2, 3, \ldots\}$is the natural numbers, the basis is to prove that $\mathcal{P}(1)$ is true. The induction step proves that for each $i \geq 1$, if $\mathcal{P}(i)$ (strong induction uses $\mathcal{P}(j)$ is true for all $j \leq i$) is true, then so is $\mathcal{P}(i + 1)$.

Example:

Theorem 4.3 *A binary tree with n leaves has $2n - 1$ nodes.*

- *Formally, $\mathcal{P}(T)$: if T is a binary tree with n leaves, then T has $2n - 1$ nodes.*
- *Induction is on the $size$ = number of nodes of T.*

Proof *Basis:* If T has 1 leaf, it is a one-node tree. $1 = 2 \times 1 - 1$ so OK.

Induction: Assume $\mathcal{P}(U)$ holds for trees with fewer nodes than T. In particular, assume for the subtrees of T.

- T must be a root plus two subtrees U and V.
- If U and V have u and v leaves, respectively, and T has t leaves, then $u + v = t$.
- By the inductive hypothesis, U and V have $2u - 1$ and $2v - 1$ nodes, respectively.
- Then T has $1 + (2u - 1) + (2v - 1)$nodes.
  $= 2(u + v) - 1$.
  $= 2t - 1$, proving the inductive step. ∎