CSC3113
Theory of Computation

Introduction

- **Automata, Computability, and Complexity**
- **Basic Mathematical Concepts**
  - Sets, Graphs, Relations, and Languages
  - Definitions, Theorems, and Proofs

Mashiour Rahman
Assistant Professor
Computer Science Department
American International University-Bangladesh

Introduction

- There are many sophisticated methods for managing computer resources, enabling communications translating programs, designing chips and database, creating computers and programs that are faster, cheaper, easier to use, more secure.
- These ideas and models are powerful and beautiful, excellent examples of mathematical modeling that is elegant and lasting value.
- Theory of computation (TOC) is based on analysis of the fundamental capabilities and limitations of computers/computing machines.
- Three areas are explored for this purpose –
  - Automata
  - Computability
  - Complexity
- Though these ideas and models are mathematical in nature, each of these three areas has different interpretation of the analysis. And the solution also very according to the interpretation.
- Next we will go through about these three areas in brief.

Complexity Theory

- Deals with the method and ideas to decide if problems are computationally hard and easy.
- There are elegant schemes for classifying problems according to their computational difficulties. For example –
  - Try to alter the aspects of the problem which is at the root of the difficulty so that the problem is more easily solvable.
  - Settle for less than a perfect solution to the problem. In certain cases finding solutions that only approximate the perfect one is relatively easy.
  - Some problems are hard only in the worst case situation, but easy most of the time.
- Cryptography is one of the examples which requires computational problems that are hard, rather than easy, because secret codes should be hard to break without the secret key or password.

Computability Theory

- Deals with certain basic problems that cannot be solved by computers.
- For example – Problem of determining whether a mathematical statement is true or false. No computer algorithm can perform this task, at least, no algorithm is known till now.
- Development of ideas concerning theoretical models of computers that eventually would help to lead to the construction of actual computers.
- In computability theory the classification of problems is by those that are solvable and those that are not.

Automata Theory

- Deals with definition and properties of mathematical models of computation.
- Allows to practice with formal definition of computation as it introduces concepts relevant to other nonmathematical area of Computer Science.
- Such models include –
  - *finite automaton*, used in text processing, compilers, and hardware design.
  - *Context-free grammar*, used in programming languages and artificial intelligence.

Assumed Background

- Sets / Sequences
- Functions / Relations
- Equivalence relations / Partitions
- Graphs
- Types of proof
  - Proof by construction
  - Proof by contradiction
  - Proof by induction
- Next we will go through the basic knowledge on the above topics.

Sets

- The symbols $\in$ and $\notin$ denote set membership and non membership, respectively. example: $7 \in \{7, 21, 57\}$ and $8 \notin \{7, 21, 57\}$
- *Subset*: $A \subseteq B$, Every element of *A* is an element of *B*.
  - *Proper Subset*: If A is a subset of B and not equal to B.
- *Multiset*: {7} and {7,7} are different as mltisets but identical as sets.
- *Infinite set*: natural numbers $\mathrm{N} = \{1,2,3, \ldots\}$ and integers $\mathrm{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$, contains infinitely many elements.
- *Empty set*: Set with 0 members, written as Ø.
- $\{n \mid \text{rule about } n\}$: A set containing elements according to some rule.
  - $\{n \mid n = m^2 \text{ for some } m \in \mathrm{N}\}$ means the set of perfect squares.
- *Cardinality* of a set: the number of elements in it.
- Set Operations:
  - *Compliment*: $\overline{A}$, is the set of all elements under consideration that are not in *A*.
  - *Union*:$A \cup B = \{x : x \in A \text{ or } x \in B\}$, the set we get by combining all the elements of in A and B. example:$\{7, 21\} \cup \{9, 5, 7\} = \{7, 21, 9, 5\}$.
    $\bigcup$S=$\{x : x \in P \text{ for some set } P \in S\}$ is the set whose elements are the elements of all the sets in S. example, $\bigcup$S={a,b,c,d}if S={{a, b},{b,c},{c,d}}.

Sets

- *Intersection*: $A \cup B = \{x : x \in A \text{ and } x \in B\}$, the set of elements that are in both A and B. example: $\{7, 21\} \cap \{9, 5, 7\} = \{7\}$.
  $\bigcap$S=$\{x : x \in P \text{ for each set } P \in S\}$ is the set whose elements are the elements of all the sets in S. example, $\bigcap$S={c,d} if S={{a,c,d},{c,d},{b,c,d}}.
- Two sets *A* and *B* are equal, written as $A = B$, if $A \subseteq B$ and $B \subseteq A$.
- *Difference* of two sets *A* and *B*, written *A*-*B*, is the set of all elements of *A* that are not elements of *B*. That is, $A - B = \{x : x \in A \text{ and } x \notin B\}$.
- Two sets are *disjoint* if they have no element in common. That is, $A \cap B = \phi$.

- *Power Set*: Power set of a set *A* is the set of all subsets of *A*.
  if $A = \{0, 1\}$, then the power set of $A = \{\phi, \{0\}, \{1\}, \{0,1\}\}$.
- A partition of a nonempty set *A* is a subset Π of 2^A such that,
  - Each element of Π is empty.
  - Distinct numbers of Π are disjoint.
  - $\bigcup \Pi = A$.
  - Example, $\{\{a, b\}\{c\}\{d\}\}$ is a partition of $\{a, b, c, d\}$.

Sequences

- *Sequence*: a list of object in some order.
  - $(7, 21, 57)$ is a sequence of 7, 21, and 57.
- Order matters, so $(7, 21, 57)$ is not the same as $(21, 7, 57)$.
- Repetition is allowed, so $(7, 21, 57)$ is not the same as $(7, 21, 7, 57)$.
- *Tuple*: Finite sequence.
- *K-Tuple*: A sequence with *k* elements.
- *Pair*: A 2-tuple is called a *pair*.
- *Cartesian product*/*cross product* of A and B, written $A \times B$, is the set of all pairs wherein the first element is a member of A and the second element is a member of B.
  If $A = \{1,2\}$ and $B = \{x, y, z\}$,
  $A \times B = \{(1, x), (1, y), (1, z), (2, x), (2, y), (2, z)\}$
  $A \times A = A^2 = \{(1,1), (1,2), (2,1), (2,2)\}$

Functions

- A *function* maps an input to an output.
- Also called *mapping*, written as $f(a) = b$, meaning, if f is a function whose output value is *b* when the input value is *a*.
- *Domain*: the set of possible inputs.
- *Range*: the set of outputs.
- The notation for saying that f is a function with domain D and range R is $f : D \rightarrow R$.
- *k-ary function*: a function with *k* arguments (*arity* of a function).
  - Input: $(a_1, a_2, \ldots a_k)$, a *k*-tuple (*argument*).
  - unary function if $k = 1$
  - binary function if $k = 2$

Relations

- *Predicate* (*property*) : a function whose range is {TRUE, FALSE}.
- *Relation*: a property whose domain is a set of k tuples, A^k for a set A.
- Relation, k-ary relation or k-ary relation on A is written as $R(a_1, a_2, \ldots, a_k)$.
- *Binary relation*: 2-ary relation. Customary infix notation aRb, where R is the relation between the elements a and b.
- *Inverse* of a binary relation $R \subseteq A \times B$, denoted $R^{-1} \subseteq B \times A$ is simply the relation $\{(b, a) : (a, b) \in R\}$.
- *Equivalence relation*: two objects being equal
  - reflexive: $\forall x, xRx$.
  - symmetric: $\forall xy, xRy$ iff yRx
  - transitive: $\forall xyz, xRy$ and $yRz \Rightarrow xRz$

Graphs

- A graph consists of a finite set of vertices (nodes) with lines connecting some of them (edges). `G = (V, E)`.
- *Undirected graph*:
  - *degree* of a node: the number of edges at a particular node.
  - *path*: a sequence of nodes connected by edges.
    - *simple path*: a path that doesn't repeat any nodes.
  - *cycle*: a path starts and ends in the same node
  - *tree*: no cycle
    - *leaves*: nodes of degree 1 in a tree.
    - *root*: special designated node.
- *Directed graph*:
  - *in-degree* and *out-degree*
  - *directed path*
  - *directed acyclic graph* (DAG)
- *Sub Graph*: Graph *G* is a subgraph of graph *H*, if the nodes of *G* are a subset of the nodes of *H* (i.e. $G.V \subseteq H.V$).
- *connected*: every two nodes of a graph have a path between them.
  - *strongly connected*: every 2 nodes of a di-graph have a path between them.

Strings

- Strings of characters.
- *Alphabet*: any finite set, Σ and Γ designate alphabets and a typewriter font for *symbols* from an alphabet. Example: $\Sigma_1 = \{0,1\}$, $\Sigma_2 = \{a, x, y, z\}$, $\Gamma = \{0,1, x, z\}$.
- A *string over an alphabet*: a finite sequence of symbols from the alphabet. If $\Sigma_1 = \{0,1\}$, then `01001` is a string over Σ_1.
- *Length* of a string *w*: $|w|$.
- *Empty* string: ε.
- String *z* is a *substring* of string *w* if *z* appears consecutively within *w*. Example: *z*=`cad`, *w*=`abracadabra`.
- If *w*=*xv* for some *x*, then *v* is a *suffix* of *w*; If *w*=*vy* for some *y*, then *v* is a *prefix* of *w*.
- If *w* has length *n*, we can write $w = w_1 w_2 \ldots w_n$ where each $w \in \Sigma$. *Reverse* of *w*, written w^R, is the string obtained by writing *w* in the opposite order (i.e. $w_n w_{n-1} \ldots w_1$).
- *Concatenation* of two strings *x* and *y*, written *xy*, is the string obtained by appending *y* to the end of *x*, as in $x_1 \ldots x_n y_1 \ldots y_n$. To concatenate a string with itself many times we use the superscript notation $\overbrace{xx \cdots x}^{k} = x^k$.

Languages

- A *language* is a set of strings.
- The set of all strings of all lengths, including the empty string, over an alphabet Σ is denoted by Σ^*.
- *Lexicographic ordering* of strings is the same as the familiar dictionary ordering, expect that shorter strings precede longer strings. Example: Lexicographic ordering of all strings over the alphabet $\Sigma = \{0,1\}$ is $(\varepsilon,0,1,00,01,10,11,000, \ldots)$.
- A language *L* over the alphabet *A* is a subset of A^*. $L \subseteq A^*$.

Proofs

- Proof: a convincing logical argument that a statement is true.
  - convincing in an absolute sense
- Methods of proof
  - *The pigeonhole principle*: there are n pigeonholes, $n + 1$ or more pigeons, and every pigeon occupies a hole, then some hole must have at least two pigeons.
  - *Proof by construction*: Prove a particular type of objects exists by constructing the object.
  - *Proof by contradiction*: Assume a theorem is false and then show that this assumption leads to a false consequence.
  - *Proof by induction*: A proof by induction has –
    - A predicate: P,
    - A basis: $\exists k, P(k)$ is true,
    - An induction hypothesis: for some $n \geq k$, $P(k), P(k+1), \cdots, P(n)$ are true.
    - An inductive step: $P(n+1)$ is true given the induction hypothesis.