Mathematical and computer-science Background

1. Sets, functions, and logic
2. Algorithms and interaction
3. Arrangements of data
4. Probability and statistics

1. Set, functions, and logic

- What were your experiences in Statistics and Precalculus?
- What is a set?
- What is a function? Examples?
- What is inference?
**Sets**

- A *set* is an aggregation of items, real, imaginary, or abstract, taken as a whole.

- Sets may be defined by:
  - Enumeration: $A = \{1, 2, 4\}$
  - Membership predicate:
    $A = \{x \mid is\_odd(x)\}$

- Sets are defined without respect to order:
  $\{1, 2, 4\} = \{2, 1, 4\}$

- A set contains no duplicates:
  $\{1, 2, 4\} = \{1, 2, 4, 2\}$

**Union, intersection, complement**

- *Membership*: "$x \in A$" means $x$ is an element of set $A$.

- *Union* of $A$ and $B$: $(A \cup B) = \{x \mid x \in A \lor x \in B\}$

- *Intersection* of $A$ and $B$:
  $(A \cap B) = \{x \mid x \in A \land x \in B\}$

- *Relative complement* of $A$ w.r.t. $B$:
  $(A - B) = \{x \mid x \in A \land x \notin B\}$

- *Generalized union* (intersection, $\cap$):
  $\cup \{A, B, C\} = A \cup B \cup C$
Set operations

Intersection: \( A \cap B \)

Union: \( A \cup B \)

Complement: \( A - B \)

Inclusion (subsets)

- \( A \subseteq B \) (set \( A \) is a subset of set \( B \)) means that any member of \( A \) is a member of \( B \)
- Strict inclusion: \( A \subset B \) means \( A \subseteq B \land A \neq B \)
- If \( A \subseteq B \) then \( B \supseteq A \) (\( B \) is a superset of \( A \))
- For every set \( A \), \( A \subseteq A \)
- The power set of \( A \), written \( \wp(A) \) or \( 2^A \), is the set of all subsets of \( A \)
- The null set \( \emptyset \)
  - has no members
  - is a subset of all sets
### Cartesian product of two sets:

- The set of *ordered pairs* defined by selecting one member of each of two sets
- **Example:**
  
  \[
  \{1, 2\} \times \{a, b\} = \\
  \{(1,a), (1,b), (2,a), (2,b)\}
  \]

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1,a)</td>
<td>(1,b)</td>
</tr>
<tr>
<td>2</td>
<td>(2,a)</td>
<td>(2,b)</td>
</tr>
</tbody>
</table>

### Relations

- **Relation:** a subset of a Cartesian product
- **Example:**
  
  \[ R = \{(1,a), (2,b)\} \text{ is a relation} \]
  
  \[ R \subseteq \{1,2\} \times \{a,b\} \]

<table>
<thead>
<tr>
<th>R</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
**Example: > is a relation**

<table>
<thead>
<tr>
<th>&gt;</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

- Because 2 > 1; 3 > 1; 3 > 2, etc.
- The relation > is a set of ordered pairs:

\[ \{ (2,1), (3,1), (3,2), (4,1), (4,2), (4,3) \} \]

**Functions**

- *Function*: a relation of which each left-hand member of a tuple is in not more than one tuple (maps to a unique value)
- *Examples*: `EVEN` is a function; > is a relation but not a function

- Every function has a
  - *Domain*: set of values mapped from
  - *Range*: set of values mapped to

- Computation of a function takes parameter as input, return value as output
Mathematical functions

- **Function**: A set that is a mapping from one set to itself or to another set

- **Examples**:
  - \( \text{Index('B')} = 2 \)
  - \( \text{Odd (3)} = \text{true} \)
  - \( \text{Twice(1)} = 2 \)

Exponential and logarithmic functions

- Many processes of growth and decay are described by exponential and logarithmic functions

- Function \( \log_b(x) \) is the inverse of function \( b^x \)

- These functions grow extremely slowly and extremely quickly, respectively

- These functions grow proportional to the base; i.e., the big-O analysis is independent of base
Strings

- A string is a sequence of symbols
- Since the sequence is ordered, a string may also be interpreted as a function from natural number to symbols
- $\Sigma$ (sigma) usually a finite alphabet
- $\Sigma^*$ the set of all strings over $\Sigma$
- $L \subseteq \Sigma^*$ a language (set of strings)
- $\lambda$ (lambda) null string, ""

Formal languages

- Language: a set of strings over an alphabet
- Let alphabet $\Sigma$ be $\{0,1\}$, let $\lambda$ be the null string
- Then $\Sigma^0 = \{\lambda\}$, $\Sigma^1 = \{(0), (1)\}$
  $\Sigma^2 = \{00, 01, 10, 11\}$
- $\Sigma^k$ the set of strings of length $k$
- $\Sigma^* = \bigcup_{k \in \mathbb{N}} \Sigma^k$
- For any language $L$ over $\Sigma$, $L \subseteq \Sigma^*$
The binary system

- Appropriate for two-state devices
- **Binary** is the form in which all information is represented (numeric, text, graphical, sound)
- Uses two digits (1 and 0) rather than ten
- Like decimal, uses *place values*
- We distinguish *numerals* (representations) from *numbers* (abstractions)

Logic gates

- Used to manipulate binary data
- 1 or 2 bit input, 1-bit output
- Specified using truth tables
- **NOT** (negation)
- **AND** (conjunction)
- **OR** (disjunction)
- Used as components of more complex circuits: adders, etc.
Gates and truth tables

<table>
<thead>
<tr>
<th>Gate</th>
<th>Schematic</th>
<th>Truth table</th>
</tr>
</thead>
</table>
| NOT  | ![Schematic](NOT.png) | \[
\begin{array}{c|c}
 a & \text{not } a \\
\hline
 1 & 0 \\
 0 & 1 \\
\end{array}
\] |
| OR   | ![Schematic](OR.png) | \[
\begin{array}{c|c|c}
 a & b & a \text{ or } b \\
\hline
 0 & 0 & 0 \\
 0 & 1 & 1 \\
 1 & 0 & 1 \\
 1 & 1 & 1 \\
\end{array}
\] |
| AND  | ![Schematic](AND.png) | \[
\begin{array}{c|c|c}
 a & b & a \text{ and } b \\
\hline
 0 & 0 & 0 \\
 0 & 1 & 0 \\
 1 & 0 & 0 \\
 1 & 1 & 1 \\
\end{array}
\] |

Propositional logic

- Propositional logic (propositional calculus) is a language for expressing the values *True* and *False*
- It is a language of assertions that each evaluate to *true* or *false*, including
  - *true, false*
  - Symbols (*p, q, r, ...*) that are Boolean variables
  - Negation: \( \neg p \)
  - Conjunction: \( p \land q \)
  - Disjunction: \( p \lor q \)
  - Implication: \( p \rightarrow q \iff \neg q \lor p \)
Semantics of propositional logic

• Let $p$, $q$, be formulas
• Wherever $p$ is true, $\neg p$ is false
• $(p \land q)$ is true when both $p$ and $q$ are true
• $(p \lor q)$ is true when either $p$ or $q$ or both are true
• $(p \rightarrow q)$ is true when $p$ is false or $q$ is true
• $(p \iff q)$ or $(p \equiv q)$ or $(p$ iff $q)$ is true when $p$ and $q$ have the same values under all interpretations (truth assignments)

Truth tables

• A truth table lists the values of a formula under all possible interpretations

Example:

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$(p \land \neg q)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$</td>
<td>$f$</td>
<td>$f$</td>
</tr>
<tr>
<td>$f$</td>
<td>$t$</td>
<td>$f$</td>
</tr>
<tr>
<td>$t$</td>
<td>$f$</td>
<td>$t$</td>
</tr>
<tr>
<td>$t$</td>
<td>$t$</td>
<td>$f$</td>
</tr>
</tbody>
</table>

• Formulas with the same truth table are equivalent: $(\phi \iff \varphi)$ or $(\phi \equiv \varphi)$ or $(\phi$ iff $\varphi)$
Problems in propositional logic

- **Evaluate**: given a particular set of variable assignments and a formula, evaluate formula
- **Satisfiability (SAT)**: Given a formula $\phi$, does a set of variable assignments exist that satisfies $\phi$ (makes $\phi$ true)?
- **Validity**: does $\phi$ hold under all variable assignments?
- **Examples**: $(p \land \neg p)$ is not satisfiable; $(p \lor \neg p)$ is satisfiable and valid; $p \land q$ evaluates to true if $p, q$ are true

Some inference rules

- **Modus Ponens**: $(p \land (p \rightarrow q)) \rightarrow q$
- **Modus Tollens**: $((p \rightarrow q) \land \neg q) \rightarrow \neg p$
- $p \rightarrow q$ is defined as $\neg p \lor q$
- **Note**: if $p$ is false, then $p \rightarrow q$ holds vacuously for any $q$, whether true or false
Implication and equivalence

$(p \Rightarrow q)$ is true whenever

- $q$ is true or
- both $p$ and $q$ are false

Assertions $p$ and $q$ are equivalent $(p = q, p \iff q)$, whenever

- $(p \Rightarrow q) \land (q \Rightarrow p)$
- $p$ iff $q$
- both $p$ and $q$ are true
- both $p$ and $q$ are false

Quantifiers in predicate logic

$\forall$ universal for all
$\exists$ existential there exists

• Example: $(\forall x \in \mathbb{N}) \ (\exists \ x')$ s.t. $x' = x + 1$
  (Every natural number has a successor)

• A predicate is a function that returns T or F

• Predicate logic is characterized by the use of propositional operators, quantifiers, predicates, and other functions.
Predicates and logic

- **Predicate**: a function that returns a Boolean value
- **Signature** $P : A \rightarrow \{0,1\}$
- Every set has a characteristic function (predicate)
- Where $S$ is a subset of universal set $U$, $S = \{x \in U \mid P(x) \}$
- Here, $P$ is $S$’s characteristic function

Proof techniques

- **Direct**: straightforward step by step inference
- **By contradiction**: Suppose that the claim we wish to disprove is true, then show that this leads to a contradiction
- **Inductive**, using Induction Principle:
  - For a set $A \subseteq \mathbb{N}$, if $0 \in A$
  - and if $x \in A \Rightarrow (x + 1) \in A$, then $A = \mathbb{N}$
- For examples, see handout sheets
Background concepts

0.1a Recognize basic concepts of precalculus*

0.1b Write the truth table for a propositional-logic formula or logic circuit*

2. Algorithms and interaction

- What *programming* have you done?
- What is an *algorithm*?
**Algorithm:**
A precise plan to solve a *functional* problem in a finite number of steps

- Algorithms compute functions by converting pre-specified input to output
- Methods, procedures, and Excel named functions may implement algorithms

\[ x \rightarrow f \rightarrow y \]

**Algorithms compute functions**

- A function in mathematics is a passive *set of pairs* that maps from one set to another one; e.g., a table
- A computation entails *action*
- A computer program or subprogram computes a mathematical function
- A C or C++ “function” is actually a subprogram, not a mathematical function
Ways to represent an algorithm or process

- Natural language
- Design notation
  - Flowchart
  - Pseudocode
  - Unified Modeling Language (UML)
- Programming language
  (JavaScript, Java, C++…)

Control structures

- Any algorithm may be built from three basic control structures:

  **Sequence**
  - Begin
  - Cook dinner
  - Eat dinner
  - Clean up
  - End

  **Branch**
  - Begin
  - Exercises are confusing?
  - Yes
  - Review chapter
  - No
  - End

  **Loop**
  - Begin
  - Dial
  - Got answer?
  - Yes
  - End
  - No
An algorithm to add 2 numbers

1. Prompt for integers $x_1$, $x_2$
2. $y \leftarrow x_1 + x_2$
3. Display $y$

- In pseudocode and flowcharts, the symbol $\leftarrow$ stands for assignment of a value to a variable
- An algorithm is almost always general purpose: it works with variables, not only specific constant values

Finding absolute value

Input $n$
If $n \geq 0$
    display $n$
otherwise
    display $(-n)$

- With the branch control structure, one and only one of the alternatives executes
- In pseudocode, the subordinate (conditional) steps are normally indented
Multiple alternatives

• Require multiple branches in flowchart; if in pseudocode

If \( n = 1 \) display ‘a’
otherwise
  if \( n = 2 \) display ‘b’
otherwise
    if \( n = 3 \) display ‘c’
otherwise
    if \( n = 4 \) display ‘d’

Three control structures suffice

• Three control structures: sequence, branch, and loop are used in structured programming; may be combined

• Any solvable problem can be solved using only these three

• Note: Some problems are not solvable
**Problem: Display all values from 1 to 10**

- A **counted** loop; counter is $i$

```
Begin

i ← 1

If $i > 10$ then
    Display $i$
else
    Add 1 to $i$
End
```

**Decision trees**

- Diagram a branching algorithmic process
- Root is starting point; leaves are terminations
- Each branching step is a node
- Edges reflect alternative choices
- Depth of tree expresses algorithm’s running time
Defining “algorithm”

• Not algorithms:
  – Online “algorithms” (non-functionality)
  – Probabilistic “algorithms” (nondeterminism)
  – Anytime algorithms (termination on cue)
  – Event loop (interleaved input/output)

• An algorithmic process is equivalent to
  – A run of a halting C++ or Java program with all input at start, all output at end
  – Evaluation of a recursively defined function

\[
\text{Product } (x, y) \\
\text{result } \leftarrow 0 \\
\text{for } i \leftarrow 1 \text{ to } x \\
\quad \text{result } \leftarrow \text{result } + y \\
\text{return result}
\]
Problem: find rightmost zero

**Last0(A)**
// Precond: |A| > 0 and A contains a 0

\[
y \leftarrow -1
\]
\[
i \leftarrow 1
\]
while \(i \leq |A|\)
  If \(A[i] = 0\)
    \[
y \leftarrow i
\]
    \[
i \leftarrow i + 1
\]
return \(y\)
// Post: \(y\) stores location of rightmost 0 in \(A\)

---

**Largest(A)**

\[
y \leftarrow A[1]
\]
\[
i \leftarrow 2
\]
While \(i \leq |A|\)
  if \(A[i] > y\)
    \[
y \leftarrow A[i]
\]
    \[
i \leftarrow i + 1
\]
return \(y\)

**Search(A, x)**

\[
y = false
\]
for \(i \leftarrow 1\) to \(|A|\)
  if \(A[i] = x\)
    \[
y = true
\]
return \(y\)
Interactive problems

- A service (solving a sequential-interactive problem) is a series of responses to inputs
- A protocol defines rules for providing a service
- Examples:
  - Search engine
  - DBMS responses to database queries
  - Web server response to clicks
- Note: Response may reflect state of server, so that a service is not a series of algorithm executions generating output from input

Interactive computation

- Feature of computing today: Computation as an ongoing service, not assumed to terminate
- Must solve problems whose inputs cannot be completely specified a priori
- Dynamic input and output during computation
- Persistence of state between interaction steps
- Environment is an active partner in computation
Background concept

0.2 Design a looping algorithm*

3. Arrangements of data

• What data structures did you learn in CS I and CS II, or Data Structures?
• What is an array? Collection?
• Who has worked with graphs? Trees?
Data structures

- Linear structures
  - Array (random access)
  - Linked list (sequential access)
- Restricted access
  - Stack (LIFO, push/pop)
  - Queue (FIFO, enqueue/dequeue)
  - Priority queue (insert, extract-min)
- Graph $G = \langle V, E \rangle$
  - Directed, undirected
  - Weighted
- Tree: root, parent, child, leaf, subtree

Collections

- \textit{Set} (distinct, unordered)
- \textit{Multiset} (unordered)
- \textit{Array}
- \textit{Dictionary} (search, insertion, deletion)
- \textit{List} (ordered)
- \textit{Special-access} (stack, queue, priority queue)
- \textit{Graph} (set of vertices, set of edges)
- Possible implementations of collections: arrays, linked lists, trees
Graph: a set of vertices $V$ and a set of edges $E$ connecting the vertices. ($E$ is a relation on $V$.)

$V = \{a, b, c, d, e\}$

$E = \{(a, b), (b, e), (b, d), (c, d), (d, c)\}$

Graphs

- A graph is a set of vertices and a set of directed or undirected edges
- Cycles; connectivity
- Graphs model communication relations, state transitions, precedence in time
- Algorithms exist to find paths, minimum-weight paths, minimal spanning trees
- Implementations: 2D array or array of linked lists
Applications of graphs

- Describing network topologies: star, linear bus, ring
- Network routing algorithms search graphs
- Scheduling (e.g., a prerequisites diagram)
- Transition-system automata (vertices are states; edges are transitions between states)
- Trees to represent hierarchies of all kinds

Paths

- Path: A sequence of pairwise adjacent vertices whose edges connect two vertices in a graph
- Path from $a$ to $d$: $(a,b,d)$
A tree is a hierarchic structure

- **Examples:**
  - Family tree
  - Organizational hierarchy
  - Module hierarchy chart
  - Taxonomy
  - Disk directory
- A rooted tree has a root at top, leaves at bottom
- A nonroot node has one parent node
- A nonleaf has one or more child nodes
- Every tree node is the root of a subtree

Tree: a connected acyclic graph

- Exactly one path joins any pair of vertices
- Removing an edge disconnects the tree
- Adding any edge creates a cycle
- Cardinalities of edges and vertices:
  \[ |E| = |V| - 1 \]
Height of a complete binary tree with \( n \) nodes is \( O(\log_2 n) \)

- \( \ldots \)because size \( n \) doubles with each level added
- Or, \( 2^{height-1} \leq n \leq 2^{height} - 1 \)

Background concepts

0.3a Find a path in a graph*
0.3b Explain the relation between the logarithm function and the heights of trees*
4. Probability and statistics

- What might happen?
- What is likely to happen?
- How likely?
- What do you expect?
- When is it hardest to plan precisely?

Possibility trees

- A series of events that each has a finite number $n$ of alternative outcomes may be diagrammed by a possibility tree, which is $n$-ary
- *Theorem* (instance of the Multiplication Rule): a series of $k$ events, each with $n$ possible outcomes, has $n^k$ distinct paths from root to leaf of its possibility tree
- *Example*: a four-digit PIN number with 36 possibilities for each character has $36^4$ possible values
**Counting elements of disjoint sets**

- For a finite set $A$ partitioned as $A_1, A_2, \ldots, A_k$,
  $|A| = \sum_{i=1}^{k} |A_i|$

- **Theorem**: for finite disjoint sets $A, B$, with $B \subseteq A$, $|A - B| = |A| - |B|$

- **Proof**:
  - if $B \subseteq A$, then $B \cap A - B = \emptyset$ (partition of $B$)
  - so $|B| + |A - B| = |A|$
  - hence $|A - B| = |A| - |B|$

- Also, $|A \cup B| = |A| + |B| - |A \cap B|$

**Pigeonhole principle**

- (Intuition) If $n$ pigeons enter $m$ pigeon holes, and if $n > m$, then at $\geq 1$ hole must have $\geq 2$ pigeons

- (Formal) **Theorem**: If $|A| > |B|$ then no $f : A \rightarrow B$ is injective; i.e., $(\exists a, b \in A, a \neq b) f(a) = f(b)$

- **Example**: at least two people in Framingham have the same last-four, because there are 10K last-4s and more than 10K persons in Framingham

- **Corollary**: Any function from an infinite set to a finite one is non-injective
Permutations and combinations

- **Set**: A non-duplicating collection of items, not defined by ordering
- **Sequence**: An aggregate defined by ordering; possibly with duplication
- **Permutations**: The possible orderings of elements of a set
- **Combinations**: The unordered subsets of a set
- Our interest is to **count** permutations and combinations so as to determine probabilities

---

Permutations

- **Definition**: Orderings of $n$ objects taken $k$ at a time, without repetition; there are $(n!)$ permutations for $n$ objects
- **Example**: There are $5! = 120$ ways to order the letters A, B, C, D, E
- **$k$-permutations ($P(n,k)$)**: Orderings of $n$ objects taken $k$ at a time; there are $(n! / (n – k)!) k$-permutations of $n$ objects
- **Example**: there are $P(6, 3) = 120$ different ways to throw a die such that only 1, 2, or 3 show
Combinations

- **Definition**: the number of ways to select from \(k\) objects at a time, taken from a set of \(n\) objects, without order or repetition
- \(C(n, k) = n! / ((n - k)! k!)
- **Example**: There are \(C(36, 6)\) ways to play the lottery where 6 numbers are chosen out of 36
- \(C(n, k)\) is also written \(\binom{n}{k}\) (“\(n\) choose \(k\)”)  
- Note that \(n! = n (n - 1) (n - 2) \times \ldots \times 2\)

Binomial coefficients

- \(C(n, k)\) is also called a **binomial coefficient**, is computed by Pascal’s Triangle, and is defined by the following recurrence, called *Pascal’s formula*:

\[
C(n, k) =
\begin{cases} 
1 & \text{if } k = 0 \text{ or } k = n \\
C(n - 1, k - 1) + C(n - 1, k) & \text{otherwise}
\end{cases}
\]

- **Binomial theorem**: \((a + b)^n = \sum_{k=0}^{n} C(n, k) a^{n-k} b^k\)
Background concepts

0.4 Explain basic notions of combinatorics*

References

