1. Classes of problems

1. Vectors, matrices, and sets
2. Constraint vs. optimization
3. Graph problems
4. Formal specification of problems

Inquiry

- What is a problem?
- How useful is the study of computational problems?
- Is there a precise way to specify a problem?
- Can you think of a problem that has more than one solution, using different design approaches, with different efficiencies?
1. Classes of problems

**Topic objective**
Describe, categorize, and formally specify a variety of computational problems.

**Background objectives**
1.0a Apply logical inference**
1.0b Use a quantifier**
1.0c Distinguish predicate from propositional logic**
1.0d Explain linked-list, stack, and queue operations**

See Background slides and study questions for support with these
1. Classes of problems

**Classes of problems**

- Basic functional problems
  - Strings, vectors, and collections
  - Computational geometry
  - Graphs
  - State-space search and problem reducibility
  - Evaluation and satisfiability of logical formulas
- The above classes have constraint and optimization versions
- Interactive problems

How this topic fits into the course

- In this course we will
  - discuss ways to solve these problems, using different design approaches
  - analyze efficiencies of different algorithms
- Extremely difficult problems may require use of approximate or probabilistic solutions
- Some problems may be specified using the languages of predicate and temporal logic
1. Classes of problems

Classroom exercise

• *Each group:* Choose a challenging problem to work on below, brainstorming solutions
• *Each student:* Consider a problem to stay with throughout the course, so that you can consider different solution approaches to this problem

1. Vectors, matrices, and sets

• What was your study of arrays in Data Structures?
• How are *sets* represented?
• What is a *matrix*?
1. Problem classes

Subtopic objectives

1.1a Describe an array problem**
1.1b Solve a problem in combinatorics

Numeric problems: factoring

- **Problem**: find all the factors of \( n \)
- **Application**: cryptography, which uses the difficulty of factoring multiples of large primes to provide values for secure cryptographic keys
- **Problem**: what is the greatest common divisor of two numbers?
1. Problem classes

**String manipulation**

- **Count**: $n_0(x)$, $n_1(x)$ are the number of 0s and 1s, respectively, in string $x$
- **Match**: Determine whether two strings are identical
- **Search**: Find the first occurrence of string $s_2$ in string $s_1$
- **Pattern match**: May allow wild cards (‘?’ , ‘*’)
- **Regular-expression match**

**Array manipulation**

- **Search**: Find occurrence or lack of occurrence of value $x$ in array $A$
- **Sort**: Rearrange array elements in ascending or descending order
- **Insertion**: Insert a new element into a sorted array such that the result is sorted
- **Order statistic**: find $k^{th}$ largest element of array
1. Classes of problems

Array-partition problem

• Used in Quicksort
• Choose any element, $x$, of array
• All other elements less than $x$ should go to its left
• All elements greater than $x$ should go to its right
• End result: $x$ ends up in what its position would be if the array were sorted; but Partition does not sort the array

Problems with bit vectors

For bit vector (Boolean array) $A$:

• Logical OR: $\bigvee_{k=1,|A|} A[k] = (\exists i \leq |A|) A[i] = 1$
  (OR bits in same position to form a new bit vector $C$, composed of ORs of pairs in $A$, $B$
• Logical AND and bitwise AND are similar
• Bitwise complement of $A$: sequence of bits that are negations of bits in $A$
1. Classes of problems

Challenging array problems

• *Find duplicates*: Does array A have any duplicate values?
• *Closest pair* of values: Which two values in array A are closest together?
• *Median* (element $x$ of array with equal numbers of values less than and greater than $x$)

\[
\text{Median} (A) = A\left\lfloor \frac{n + 1}{2} \right\rfloor
\]

Word segmentation

• *Given*: A string of letters, without spaces
• *Problem*: Insert spaces so that the new string is a sequence of words in English
**Priority queue problem**

- A priority queue is a collection with restricted access
- Each item in priority queue has a key value denoting its relative *priority*
- Operations: *Insert*, *Extract*
- *Extract* retrieves and deletes the item of highest priority
- *Minimum queue* treats low-valued key as high-priority; maximum queue prioritizes high-valued key

**Matrix multiplication**

- Given two 2-dimensional matrices, their *product* is a matrix whose cells are each the sum of the products of several cells chosen from both matrices
- Given square matrices $A$ and $B$, matrix $C$ is the product, where

$$ C[i, j] = A[i, 0] \times B[0, j] + \ldots \\ A[i, k] \times B[k, j] + \ldots \\ A[i, n-1] \times B[n-1, j] \text{ for } i, j \leq n - 1 $$
1. Classes of problems

### Linear programming

- **Given:** An \((m \times n)\) matrix \(A\), vectors \(b \in \mathbb{R}^m\), \(c \in \mathbb{R}^n\)
- **Problem:** Find a vector \(x \in \mathbb{R}^n\) that solves
  \[
  \min(c^t x \text{ s.t. } x \geq 0, Ax \geq b) \quad \text{[Example p. 632]}
  \]
- \(Cx\) is called the *objective* function
- \(Ax \geq b\) is called the set of *constraints*
- *Integer programming* is a more difficult instance of linear programming that looks for *integer* solutions

### Set partition

- **Set partition:** Given a set \(S \subseteq \mathbb{N}\), can \(S\) be partitioned into two subsets, \(A\) and \(B\), s.t. the sums of the elements of \(A\) and \(B\) are the same?
- **Example:** If \(S = \{1, 3, 4\}\) then *yes*; if \(S = \{1, 2, 4\}\), then *no*
Subset sum

- **Given:** Set $S$ of natural numbers
- **Problem (Sum):** For $w \in \mathbb{N}$, is there a subset $A \subseteq S$ s.t. $\sum_{t \in A} t = w$?

Set cover and set packing

- **Given:** Set $U$ and collection $S_1, \ldots, S_m$, with $(\forall i \leq m) S_i \subseteq U$
- **Problem (Set cover):** Does a collection of not more than $k$ subsets of $S$ exist whose union is $U$?
- **Example:** Can $k$ software applications meet all $m$ software requirements?
- **Problem (Set packing):** Does a collection exist of at least $k$ disjoint sets $S_i$?
- **Example:** Are resource requirements of an information system in conflict (overlapping)?
1. Problem classes

State-space search

- **State-space search**: problem involves search in an $n$-dimensional solution space for a structure that satisfies some specification
- **State** may be
  - game board configuration (**$n$-queens**)
  - way of partitioning a set (**set-partition**)
  - a set of variable assignments (**satisfiability**)
- Many optimization problems, are characterized by *combinatorial explosion*

How large is a problem?

- One measure: the number of possible solutions
- **Decision** (yes/no) problems have two possible solutions (Does this make them easy?)
- A **search** of a database of size $n$ has $(n + 1)$ possible solutions (“+1” in case no match)
- A **sorting** problem is to find a permutation $P(n, n)$ of an array (how many permutations?)
- A problem that requires generating a truth table of $n$ variables has $2^n$ rows to generate
1. Classes of problems

**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**: 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-character PIN number with 36 possibilities for each character has $36^4$ possible values

**Multiplication rule**

- Suppose an operation has $k$ steps
- Suppose steps 1 .. $k$ have $n_1$, … $n_k$ ways to be performed
- Then there are $\prod_{i=1..k} (n_i)$ ways to perform the operation
- From this it follows that there are $O(2^k)$ ways to perform it
1. Problem classes

**Pigeonhole principle**

- (Intuition) If \( n \) pigeons enter \( m \) pigeon holes, and if \( n > m \), then at least one hole must have at least two pigeons

- (Formal) *Theorem*: If \( |A| > |B| \) then \( f : A \rightarrow B \) cannot be 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**

- *Permutations*: The possible orderings of elements of a set

- *Combinations*: The unordered subsets of a set

- Our interest is to *count* permutations and combinations in order to determine *possibilities*, so as to compute *probabilities*
### 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, $|A \cup B| = |A| + |B| - |A \cap B|$

### 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
- \( C(n, k) = \frac{n!}{((n-k)!k!)} = \frac{P(n, k)}{k!} \)
- **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 \( n! = n(n-1)(n-2) \times \ldots \times 2 \)

Combinatorial explosion

- Many computational problems are subject to **combinatorial explosion**
- For most state-space-search problems, the number of possible solutions is exponential in problem size
- For some problems, typical for AI, no easy way exists to prune the number of candidate solutions that require examination
- These problems are called **intractable**
2. Constraint vs. optimization

Optimization problems are ____
constraint satisfaction problems
(a) equivalent to
(b) harder than
(c) easier than
(d) a subset of
(e) a superset of

Subtopic objectives

1.2a Distinguish constraints from optimization*
1.2b Evaluate a formula and determine its satisfiability*
Definitions

- **Constraint problem**: To find some value that satisfies a set of constraints or conditions
- **Constraint problem examples**:
  - Search
  - Pattern matching
  - Sort of a set of records
- **Optimization problem**: to find maximum or minimum valued solution to a constraint problem, among all solutions

Constraint satisfaction problems

- **Constraint problem**:
  - A set \( x \) of variables \( x_1, x_2, ..., x_n \)
  - A set \( C \) of constraints \( C_1, C_2, ..., C_m \), that each specifies acceptable combinations of values for a certain subset of \( x \)
  - A variable assignment to \( x \) that does not violate any of \( C \) is *consistent* or legal
  - A variable assignment to all of \( x \) is *complete*
- **Solution**: a complete and consistent variable assignment
General form of CSPs and solutions

- Advantage of formulating problems as CSPs: standard state representations enable generic transition functions, goal tests, heuristics
- Form of state: set of partial variable assignments
- Initial state: no assignments
- Successor function: assigns value to one variable while violating no constraint
- Goal test: variable assignments are complete
- Order-independence of variable assignments makes backtracking helpful

The function-optimization problem

- Let \( f : \mathbb{N}^k \rightarrow \mathbb{R} \) for some \( k \) (the arity of \( f \))
- Problem: Find some \( x \in \mathbb{N}^k \) s.t. \( f(x) \) is maximal
- Example: Suppose \( x \) is the set of proportions of ingredients in a fuel mixture, \( f(x) \) is fuel efficiency under this mixture
- Optimizing \( f(x) \) means finding the most efficient mixture
- For an algorithm to optimize a function we must have \( f : X \rightarrow Y \) with \( X, Y \) finite
Conversions between constraint and optimization problems

- *Constraint version:* Does a solution to problem $\pi$ exist, of size $\leq n$? ($\geq n$)
- *Optimization version:* what is the minimum-size (maximum-size) solution to constraint problem $\pi$?

1.2.1 Problems in 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 (interpretations) exist that satisfies $\phi$ (makes $\phi$ true)?
- *Validity:* does $\phi$ hold under all interpretations?
- *Contradiction:* does $\phi$ never hold?
- *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
Expression evaluation

- Given an expression in propositional logic, with Boolean variables and with operators (¬, ∧, ∨, ⇒), and
- Given an interpretation (set of variable assignments) for the expression
- Problem: Determine whether the formula is true or false
- Example: \(((p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)), (T, F)\)

Solutions to logic problems

- A solution to the formula evaluation problem is found by the conventional intuitive procedure, using order of operations and evaluating innermost parenthesized sub-formulas
- Truth table may be used
- Satisfiability may be determined by seeing if any row of truth table shows true result
Interpretations and models

- An interpretation of a set of formulas is an assignment of truth values to the symbols.
- Example: An interpretation of \((p \land \neg(p \lor q))\) is \(p = \text{true}, q = \text{false}\). Under this interpretation, \((p \land \neg(p \lor q))\) is false.
- A formula is satisfiable if it has an interpretation under which it is true.
- A model is an interpretation that satisfies a formula.

Normal forms

- Conjunctive normal form (CNF): a form of logic sentences that consist of the joining of disjunctive (OR) clauses by conjunction (AND) operators.
- Example: \((p \lor q) \land (\neg q \lor r)\)
- Disjunctive normal form (DNF): a form of logic sentences that consist of the joining of conjunctive (AND) clauses by disjunction (OR) operators.
- Example: \((p \land q) \lor (\neg q \land r)\)
SAT reduces to a circuit problem

- **Given:** A logic circuit with $n$ inputs, one output
- **Problem:** Is there an input on which this circuit outputs *true*?
- **Equivalence with SAT:** Convert variables to inputs, gates to operators
- **Thus any** decision problem with fixed-size input is reducible to SAT

---

Max-SAT

- **Given:** A set of $k$ disjoint clauses in propositional logic $C_1, \ldots, C_k$
- **Problem:** What is the set of variable assignments that satisfies the largest number of clauses?
1.2.2 Sequences and strings

- Suppose we rank music preferences
- How do we tell if two people have similar tastes, and how similar?
- How do we tell if two words, or sentences, are similar, and how similar?

Similarity of rankings

- How similar are the music tastes of two people? Compare their rankings of a list of artists
- One measure is the number of cases where the two people rank the same two films in opposite order [pic]
- One such case is called an inversion; dissimilarity of taste is measured by the number of inversions found in rankings
Counting inversions

- One inversion in an array is a pair that is out of order.
- To find the number of inversions in the preference rankings of \( n \) items by two people, compare every possible pair.

String (Levenshtein) distance

- **Definition**: the number of single-character edits required to transform one string into another.
- **Example**: \( \text{dist}("kitten", "sitting") = 3 \), because changes \((k \rightarrow s, e \rightarrow i, \text{insert } g)\) would be required.
Longest common subsequence
• *Vector, array, and sequence* may be considered to be equivalent
• Given sequences $x_1, x_2$, what is the longest subsequence $y$ s.t. $y$ is a subsequence of both $x_1$ and $x_2$?
• This is a measure of similarity
• Elements of subsequences are not necessarily contiguous, e.g., “dab” is a subsequence of “database”

Optimal string alignment
• For two strings, optimal alignment is one with smallest possible cost as defined by *gap* and *mismatch* costs
• *Gap cost*: number of positions not matching
• *Mismatch cost* applies to each pair of symbols lined up with each other when they aren’t the same
Aligning DNA sequences

- **Definition:** “a pairwise match between the characters of each sequence”
- **Significance:** An alignment corresponds to a hypothesis about the evolutionary history connecting the sequences
- **Objective:** To find the best alignments between two sequences
- **Techniques for alignment comparison of sequences are “a cornerstone of bioinformatics”**

Understanding natural language

- Most natural languages are *ambiguous*; for example, words such as “he,” “she”, “it”, or words that may be verb, noun or adjective
- Classic example: “Time flies like an arrow”
- Hence the number of possible meanings of a sentence or paragraph is exponential in the number of ambiguous words in it
- **Goal:** Find the most likely meaning
1.2.3 Problems in planning and scheduling

• Most decision problems in life require planning ahead
• How many steps ahead do you need to plan to play a board game perfectly?
• The number of possible plans is exponential in the number of steps ahead

n-queens and search trees

• Problem: place $n$ queens on a chessboard s.t. no queen threatens another
• Many search problems can be expressed as searches of a solution tree; e.g., game playing, 8-queens.
• The tree represents possible transitions from one configuration of the game board to another
• Many games entail state-space search for “good” or winning positions
### Formalizing planning

- Conditions $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$
- Operators $\{O_1, O_2, \ldots, O_n\}$; each operator has prerequisite, add, delete lists
- Configuration $\mathcal{C}' \subseteq \mathcal{C}$ is the state of the problem at any time
- *Problem:* Does some sequence of operators exist leading from initial to goal configuration?
- Graph of exponential size represents this problem as path search; some solutions are $\Theta(2^n)$ length

### Scheduling with release time and deadlines

- *Given:* $n$ jobs, to be run on a machine, each with a *release time* when it will be ready to run; a *deadline* by which it must be finished; and a *duration*
- *Problem:* Can all jobs be run by their deadlines?
Interval scheduling problem

- **Given:** Time intervals \((s_i, f_i)\) for \(i = 1\) to \(n\), each of which starts at time \(s_i\) and ends at \(f_i\).
- **Problem:** Find a sequence of non-overlapping intervals \(1 \ldots m\) s.t. \(m\) is maximal.

Weighted interval scheduling

- **Given:** Tasks \(t_1..n\) with start times \(s_1..n\), finish times \(f_1..n\), and weights \(w_1..n\).
- **Problem:** Choose a set of non-overlapping tasks with maximum total weight.
- [pic]
Load balancing

- **Given**: A set of jobs and a set of parallel or distributed processors
- **Problem**: Assign jobs to processors such that overall task completion is earliest

1.2.4 Packing problems

- **Problems**:
  - To fit items into containers, or
  - To maximize the value of contents of a container under constraints
1. Problem classes

**Bin packing**

- Given \( n \) items of sizes \( s_1, \ldots, s_n \), and \( k \) bins each of capacity < 1.0, where \( s_1, \ldots, s_n < 1 \), can all items be packed in the \( k \) bins?
- *Example*: If two bins are of capacity 0.5 and 0.3, can items of sizes 0.2, 0.2, 0.2, 0.2 fit?

**Knapsack problem**

- Find maximum-valued subset of weighted items totaling less than a given weight \( w \)
- Given \( n \) objects and a knapsack of capacity \( C \), where object \( i \) has weight \( w_i \) and value \( v_i \), find \( x[1..i] \) to maximize sum of all values \( v_i w_i \) where sum is less than \( C \)
- Two versions of problem:
  - Continuous, where \( 0 \leq x_i \leq 1 \)
  - 0/1 where \( x_i \in \{0, 1\} \)
1.2.5 Pattern detection problems

• The *clustering* and *change detection* problems ask us to find *meaning* in data
• This is a *data mining* task

---

**Change detection**

• A data-mining problem
• *Given*: a set of points on a coordinate axis
• *Problem*: Find the origins and slopes of one, two, or more line segments that approximate the data
• *[pic]*
Clustering problem

- Given a set of objects with degrees of similarity, categorize them in clusters
- To do this, distance \((a, b)\) between objects \(a, b\) should be defined
- Examples: points in space, strings, species, images
- \(k\)-clustering of a set \(S\): a set of \(k\) partitions of \(S\)
- Goal: clusterings that maximize spacing (min. distance between elements of distinct clusters)

Low-diameter clustering

- Given: A set of pairs denoting points on an \((x, y)\) axis
- Problem: Can the points be partitioned into \(k\) subsets s.t. no two points in the same cluster are greater than \(d\) distance apart?
1. Classes of problems

1.2.6 Problems in computational geometry

- Is a series of line segments connected?
- Are two segments the same length?
- Do two line segments \((ax, ay), (bx, by)\) cross?
- Find the longest of a series of segments
- Find the longest connected path in a series

Closest pair

- Given a set of ordered pairs, points on a Cartesian plane, find two points that are closest together
Convex hull

- The Convex hull of a set of points is a subset of ordered pairs (on a Cartesian plane) that define a polygon with no concave series of edges (“dents”)
- I.e., find smallest convex polygon that contains all of a set of points
- Equivalent problem: find the posts of the shortest fence that would enclose all the points
- [pic]

Defining the center selection problem

1. *Given:* Home locations and number of schools to be built
   
   *Problem:* Where should schools be built?

2. *Given:* Home locations and school locations
   
   *Problem:* What capacities should each school have?

   *Meta-problem:* What are some possible criteria for defining “good” location or size?
Center selection

- *Given:* A set $C$ of $n$ points on a coordinate axis (e.g., homes)
- *Problem:* Find optimal locations of $k$ centers (e.g., hospitals or schools) s.t. maximum distance of any home from a hospital is minimized
- Simplest version: $k = 1$

3. Graph problems

- What is a *graph*?
- Do you remember graph algorithms from Data Structures?
Subtopic objectives

1.3a Describe a graph problem*

[For slides on challenge objective 1.3b, “Describe an advanced graph problem,” see separate set of slides, “Challenge subtopics for topic 1: Classes of problems.”]

Graph coloring

- What is the minimum number of colors needed to color all vertices of a graph (or towns on a map), such that no two adjacent vertices have same color? (An optimization problem)
- Country on a map is equivalent to vertex in graph; borders between countries are equivalent to edges in graph
- The compiler-design problem of register allocation is equivalent to graph coloring
1.3.2 Paths in graphs

Concepts:
- Search
- Reachability
- Spanning tree
- Arborescence
- Routing
- Hamiltonian path

Path search

- Finding a path in a graph from a given vertex to another vertex
- Inputs: $G = (V, E)$; origin vertex; destination vertex
- Output: a path (sequence of vertices) starting at origin, ending with destination; or a value denoting failure
Reachability and cycle detection

- Given: \( G = (V, E) \)
- Problem (reachability): Which pairs of vertices in \( V \) are joined by paths?
- Problem (cycle detection): Does \( G \) have a cycle?
- Cycle detection is a subset of the reachability problem; a graph has a cycle if it has some \( u \in V \) s.t. \( \text{reachable}(u, u) \) with path of length greater than 0

Minimum spanning tree

- Suppose you are designing a subway system
- You want to join \( n \) locations by rail at minimum expense
- The locations and the possible connections you will plan are a graph
- The optimal graph must be connected (Why?)
- The graph should be acyclic (Why?)
Example subway solution

City map

Optimal train network

Hard extensions to MST

- Minimizing edges
- Minimal network that is robust to edge deletions
- MST with limits on traffic through any node
- All these extensions are much harder than MST
1. Problem classes

**Single-source shortest path**

- In any weighted graph, from any source vertex, a tree exists composed of the set of shortest paths from the source to each other vertex.
- The problem of a single shortest path reduces to building this tree (which is **not** the MST).
- *Dijkstra’s algorithm* builds this tree of shortest paths, starting from the source vertex.

**Paths through all vertices**

- *Hamiltonian path*: A path in a graph from a vertex to itself that passes through all vertices exactly once.
- *Traveling Salesperson*: In a weighted graph, find minimum-cost path through all vertices.
- TSP is the optimization version of HP.
1. Problem classes

**Hamiltonian-cycle problem**

- **Problem:** Given a graph \( G = \langle V, E \rangle \), is there a path in \( V^* \) that starts and ends with the same vertex and that passes through every vertex exactly once?

- Yes

- No

**The traveling-salesperson problem**

- For a weighted digraph \( G \), and an integer \( B \), is there a cycle that passes through all vertices and whose weight is not more than \( B \)?

- This problem is considered *intractable*, but a claimed solution can easily be checked

- At least as hard: What is the *least-cost* such path?
Applications of TSP

- Calculating the most efficient motions of a robotic arm drilling holes
- Meeting disk I/O requests
- Sequencing execution of machine-language instructions

Other path related problems

- **Traversal**, visiting all vertices
- **Cycles**: Is graph cyclic?
- **Connectivity number** (connected components)
- **Spanning tree**: a tree of edges that includes all vertices
1. Classes of problems

Topological sort of dag

- *dag:* directed acyclic graph, i.e., graph in which no cycles exist. Trees are dags but in a dag, contrary to a tree, two edges may have the same destination
- *Problem:* to arrange the vertices of a directed acyclic graph in a sequence such that for every pair \((v_i, v_{i+1})\), there is no path in the dag from \(v_{i+1}\) to \(v_i\)
- *Example:* a listing of courses in a prerequisite relation so that no course is taken before its prerequisites

4. Formal specification of algorithmic problems

- What does an algorithm compute?
- What is an *assertion*?
- What is true, when a search engine returns a positive result?
- When it returns “not found”?
- What is true, after an array is sorted?
1.4 Formally specify a computational problem**

**Subtopic objectives**

**Assertions and correctness**

- A comment that is an *assertion* tells not what *occurs*, but something about *values* of variables and expressions
- Valid assertions can help us establish that our code does what we claim
- Chief tools for showing algorithm correctness:
  - *Preconditions*
  - *Postconditions*
  - *Loop invariants*
1. Problem classes

Pre and post conditions

• One part of specification of a problem is an assertion that holds as a postcondition

• Part of specification is a set of preconditions that must hold for any input

• Examples
  – Precondition for binary search, or postcondition for any sort: \( (\forall i < |A|)(A[i] \leq A[i+1]) \)
  – Postcondition of any search:
    \( (\text{Return value} = \text{true}) \iff (\exists i \leq |A|) A[i] = \text{key}) \)

The terms of a contract

• A precondition of a method tells what must be true about parameters if the method is to work

• A postcondition asserts that the method has done its job correctly

• Example:
  ```
  int sum_of_linear_series(int n)
  // Precondition: n > 0
  {
    int sum = 0;
    for (int i = 1; i <= n; ++i)
      sum += i;
    // Postcondition: \text{sum} = 1 + 2 + \ldots + n
    return sum;
  }
  ```
Quantifiers

• For sentence \( S \),
  – \((\exists x) S\) is true iff some assignment under \( I \) has an assignment to \( x \) s.t. \( S \) is true
  – \((\forall x) S\) is true iff \( S \) is true for all assignments of values to \( x \) under \( I \)

• Correspondence between quantifiers:
  – \((\exists x) P(x) \iff (\neg \forall x) \neg P(x)\)
  – \((\forall x) P(x) \iff (\neg \exists x) \neg P(x)\)

Notation

Logical operators (propositional logic)

\( \neg \) not (negation)
\( \land \) and (conjunction)
\( \lor \) or (disjunction)
\( \Rightarrow \) entails (implication)
\( \Leftrightarrow \) iff (equivalence)

Quantifiers (predicate logic)

\( \forall \) for all (universal quantifier)
\( \exists \) there exists (existential quantifier)
1. Classes of problems

Examples of postconditions

- **Search** \((A, \text{key})\) returns *true* iff key is in array 
  \((\exists i \leq |A|) A[i] = \text{key}\)

- **Sort**: every pair of adjacent elements is in 
  ascending order 
  \((\forall i \leq |A|) A[i] \leq A[i + 1]\)

- **Max**: returns max element of collection 
  \(y = \text{max}\{x | (\exists i \leq |A|) x = A[i]\}\)

- **Is-sorted**: returns true iff collection is in 
  ascending order 
  \(y = \text{true} \iff (\forall i \leq |A|) A[i] \leq A[i + 1]\)

Hoare triples

- A way to specify a problem as an 
  (input, output) relation

- Let \(P\) be a program

- Let \(\phi, \psi\) be assertions about the state of a 
  program at a certain time, i.e., about the 
  values of variables

- Specification \(\langle \phi \rangle P \langle \psi \rangle\) asserts that if \(P\) runs 
  starting in a state that satisfies \(\phi\), then the 
  state after termination of \(P\) will satisfy \(\psi\)
Example: Factorial

**Factorial**(x)

\[ \langle x \geq 0 \rangle \]

\[ y \leftarrow 1 \]

\[ t \leftarrow 1 \]

while \( t \leq x \)

\[ y \leftarrow ty \]

\[ t \leftarrow t + 1 \]

\[ \langle y = x! \rangle \]

- Assertion: \( \langle \phi \rangle P \langle \psi \rangle \)

[For slides on challenge objective, 1.5 Formally specify a property of an interactive system, see separate set of slides, “Challenge subtopics for topic 1: Classes of problems”]
1. Classes of problems

References


