

Topic 02

University Computer Science

What this course is, and how to use it

This is the second course in University Computer Science: **discrete mathematics** – the mathematics of objects you can list: statements, sets, graphs, counts. It is the proof toolkit behind algorithms, databases and every correct-by-construction claim in CS.

Scope limit, stated once: a study course, not a degree; every problem is original and self-contained, and every proof names its strategy.

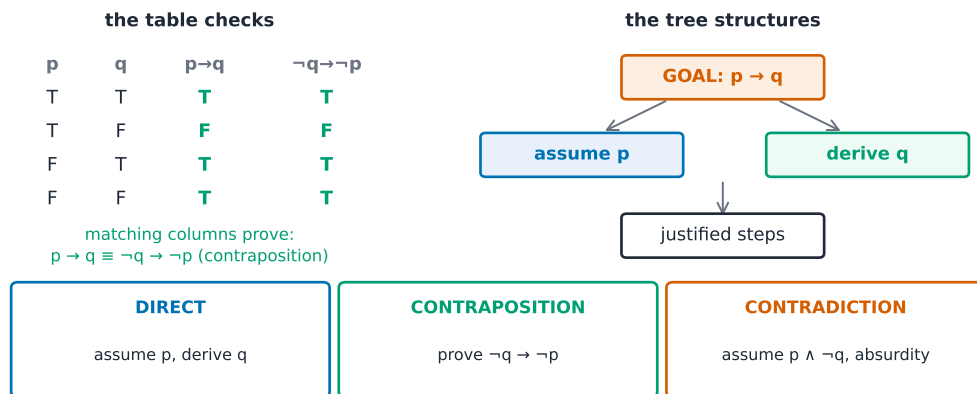
- Units 2.1–2.3 are the proof engine: logic, sets, induction.
- Units 2.4–2.6 are the combinatorial engine: counting, graphs, discrete probability.

The discrete habit: **name the strategy before the proof** – direct, contraposition, contradiction, induction – and every identity is something you can verify on a small example first.

Logic and proofs

Why it matters. Logic is the type system of proof: translate the claim, check the argument, and choose the proof shape before writing a line.

The idea, with a picture. Propositions connect with $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$; a **truth table** settles any formula over its variables $\checkmark$. The key equivalences: $p \rightarrow q \equiv \neg p \vee q$; contraposition $p \rightarrow q \equiv \neg q \rightarrow \neg p$; De Morgan $\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\checkmark$. **Predicate logic** adds the quantifiers $\forall x P(x)$ and $\exists x P(x)$, with negation flipping each.



The table checks; the tree structures – a proof is a data structure of justified steps

The derivation (why three proof shapes). To prove $p \rightarrow q$: **direct proof**

(assume p , derive q); **contraposition** (prove $\neg q \rightarrow \neg p$ – useful when negatives are friendlier); **contradiction** (assume $p \wedge \neg q$, derive the impossible) $\checkmark$. Each shape converts the goal into an assumption you can compute with – choosing the shape is choosing your data.

Worked example 1 – the translation. Claim: every email has exactly one address: $\forall e \exists! a \text{ Has}(e, a)$. Its negation pushes $\forall$ to $\exists$ and negates what follows: some email has no address, or has two or more $\checkmark$. One such email breaks the claim. Translate, negate, re-read: the three-step habit.

Worked example 2 – the strategy named. Claim: if n^2 is even then n is even. Strategy: contraposition (the negative is friendlier). Proof: suppose n odd, so $n = 2k + 1$; then $n^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$, odd $\checkmark$ – contrapositive proved, hence the claim.

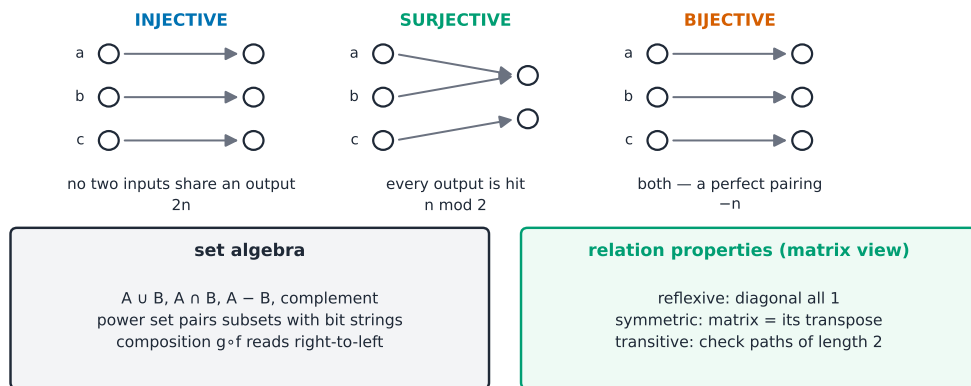
Common mistakes. Proving the converse of what was asked (direction matters: $p \rightarrow q$ is not $q \rightarrow p$). Distributing negation without flipping connectives (De Morgan flips $\wedge$ to $\vee$). Writing examples where a universal proof is demanded – an example is a check, not a proof.

Summary. Translate $\rightarrow$ choose shape (direct / contraposition / contradiction) $\rightarrow$ write justified steps; $p \rightarrow q \equiv \neg p \vee q$; De Morgan flips; negation swaps $\forall/\exists$; truth tables verify, proof trees structure.

Sets, functions and relations

Why it matters. Sets are the data model of discrete maths; functions are the typed pipes between them; relations carry the structure databases and graphs are built from.

The idea, with a picture. Operations: union, intersection, difference, complement; **power set** $\mathcal{P}(A)$ = all subsets, $|\mathcal{P}(A)| = 2^{|A|}$ $\checkmark$. Functions classified by arrows: **injective** (no two inputs share an output), **surjective** (every output is hit), **bijective** (both – a perfect pairing) $\checkmark$. Composition $g \circ f$ applies f then g ; bijections invert.



Arrows classify the function; the matrix reads the relation's properties at a glance

The derivation (why bijections count). A bijection pairs every element of A with exactly one of B , so $|A| = |B|$ when one exists – and finding a bijection IS a counting proof $\checkmark$. That is why the number of subsets of an n -set pairs with bit strings of length n (each element in or out – one bit each), proving 2^n without listing $\checkmark$.

Worked example 1 – classify three arrow diagrams. $f: \mathbb{Z} \rightarrow \mathbb{Z}$ by $f(n) = 2n$: injective (different inputs $\rightarrow$ different doubles) not surjective (odd numbers unhit) $\checkmark$; $g(n) = \lfloor n/2 \rfloor$: surjective not injective $\checkmark$; $h(n) = -n$: bijective – its own inverse $\checkmark$.

Worked example 2 – the relation test. On $\{1, 2, 3\}$: $R = \{(1, 1), (2, 2), (3, 3), (1, 2), (2, 1)\}$ – reflexive (all loops), symmetric (edges come in pairs) and transitive – the chain $(1, 2), (2, 1)$ demands $(1, 1)$ and the chain $(2, 1), (1, 2)$ demands $(2, 2)$, both present $\checkmark$. So R is an **equivalence relation** with classes $\{1, 2\}$ and $\{3\}$ – check each property against the definition, not intuition.

Common mistakes. Confusing $\in$ (member) with $\subseteq$ (subset). Writing $f \circ g$ for $g \circ f$ (composition reads right-to-left). Testing symmetry by looking at one pair instead of all.

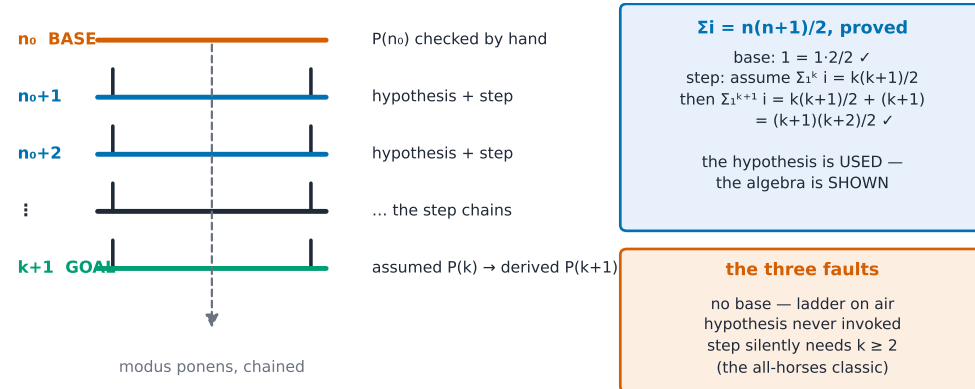
Summary. Union/intersection/difference/complement; $|\mathcal{P}(A)| = 2^{|A|}$ (bijection with bit strings); injective/surjective/bijective by arrows; composition right-to-left; relation properties – reflexive, symmetric, antisymmetric, transitive – tested against definitions.

Induction and recursion

Why it matters. Induction is the correctness proof of every loop and every recursive data structure; recursion is induction run backwards.

The idea, with a picture. To prove $\forall n \geq n_0 P(n)$: **base case** ($P(n_0)$)
+ **inductive step** (assume $P(k)$ – the **induction hypothesis**)

– derive $P(k+1)$ $\checkmark$. Recursion defines F by smaller calls: factorial $0! = 1$, $n! = n \cdot (n-1)!$ – the same ladder, built instead of climbed.



Base plants the ladder; the step climbs one rung; every rung below is the hypothesis

The derivation (why induction is valid). The step proves that if rung k holds then rung $k+1$ holds; the base proves rung n_0 ; chaining gives every rung up to any n – induction is repeated modus ponens $\checkmark$. Fault detection: no base (a ladder on air), hypothesis not invoked (a step that proves nothing), or step from k assuming more than $P(k)$.

Worked example 1 – the sum identity. Claim: $\sum_{i=1}^n i = \frac{n(n+1)}{2}$. Strategy: induction. Base $n = 1$: $1 = \frac{1 \cdot 2}{2} \checkmark$. Step: assume $\sum_{i=1}^k i = \frac{k(k+1)}{2}$; then $\sum_{i=1}^{k+1} i = \frac{k(k+1)}{2} + (k+1) = \frac{(k+1)(k+2)}{2} \checkmark$ – the hypothesis is used, the algebra is shown, the claim closes.

Worked example 2 – the faulty proof caught. The claim that all horses are the same colour: base trivially true; the step – remove one horse from each side – fails at $k+1 = 2$ because the two overlapping sets share no horse $\checkmark$ – the step must hold for EVERY k , including the smallest; a step that silently needs $k \geq 2$ without base $P(2)$ is the classic fault.

Common mistakes. Omitting the base – hypothesis plus step prove nothing without a floor. Using $P(k+1)$ to prove $P(k+1)$ (circular). Strong induction declared but ordinary step written – say which and do which.

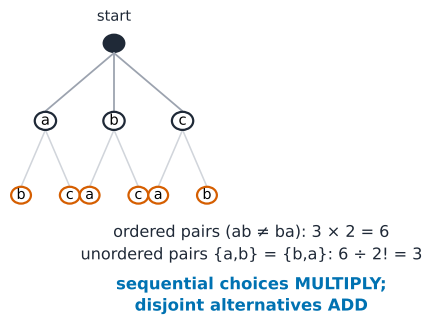
Summary. Base + hypothesis + step, hypothesis explicitly invoked; strong induction assumes all rungs below; recursion = induction's mirror (defined downwards); faults: missing base, unused hypothesis, step valid only for large k .

Counting and combinatorics

Why it matters. Counting is complexity analysis's arithmetic: sizes of state spaces, branch counts, the how-many behind every loop bound.

The idea, with a picture. **Addition rule** : disjoint choices add; **multiplication rule** : sequential choices multiply ✓. **Permutations** (ordered): $P(n, r) = \frac{n!}{(n-r)!}$; **combinations** (unordered): $\binom{n}{r} = \frac{n!}{r!(n-r)!}$ ✓. **Inclusion-exclusion** : $|A \cup B| = |A| + |B| - |A \cap B|$ – add, then subtract the double count.

the tree: ordered vs unordered



the two formulas

$$P(n, r) = n! / (n-r)! \quad \text{ordered}$$
$$C(n, r) = n! / (r!(n-r)!) \quad \text{unordered}$$

same choices; combinations divide by the $r!$ arrangements of each set

inclusion-exclusion, three sets

$$|A \cup B \cup C| = |A| + |B| + |C| - \text{pairs} + \text{triple}$$
$$45 + 35 + 30 - 12 - 10 - 8 + 5 = 85; \text{none: } 15$$

add singles, subtract pairs, add the triple

The tree makes the rules visible: branches multiply, alternatives add, overcounts subtract

The derivation (why the formulas divide). Permutations choose and arrange: n options, then $n-1, \dots, r$ factors – multiply. Combinations identify arrangements of the same r objects: every set of r was counted $r!$ times as an arrangement – divide by $r!$ ✓. The division is the entire difference between ordered and unordered.

Worked example 1 – the committee. A team of 3 from 8 people, with one designated leader: choose the team $\binom{8}{3} = 56$, pick the leader among them 3 ways $\rightarrow 168$ ✓; or choose the leader first (8) then 2 members from 7 ($\binom{7}{2} = 21$) $\rightarrow 168$ – two counting routes, one answer, and the agreement is the check.

Worked example 2 – inclusion-exclusion with three sets. Among 100 students: 45 like tea, 35 coffee, 30 cocoa; 12 tea $\wedge$ coffee, 10 tea $\wedge$ cocoa, 8 coffee $\wedge$ cocoa; 5 all three. At least one: $45 + 35 + 30 - 12 - 10 - 8 + 5 = 85$ ✓ – none: $100 - 85 = 15$. Add singles, subtract pairs, add the triple: each element counted once.

Common mistakes. Adding where choices are sequential (routes $\times$ vehicles multiply). Using permutations where order is irrelevant (committees are combinations). Applying two-set inclusion-exclusion to three sets (the triple comes back with a plus).

Summary. Disjoint $\rightarrow$ add; sequential $\rightarrow$ multiply; $P(n, r)$ ordered, $\binom{n}{r} = P/r!$

unordered; inclusion-exclusion: + singles – pairs

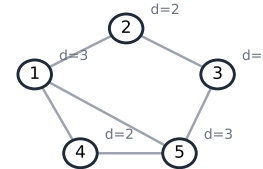
- triples; count two ways and compare – the checksum of combinatorics.

Graphs and trees

Why it matters. Graphs are the universal model – networks, states, dependencies; trees are the workhorse of search and storage; and the **handshake lemma** is the first must-hold identity of the course.

The idea, with a picture. $G = (V, E)$; representations: **adjacency matrix** (n^2 space, $O(1)$ edge test), **adjacency list** ($O(|V| + |E|)$) ✓. **Tree** : connected, acyclic, $|E| = |V| - 1$ ✓. Traversals: **BFS** (queue, level order, shortest paths in unweighted graphs), **DFS** (stack/recursion, component discovery).

the picture (5 vertices, 6 edges)



$$\Sigma \text{deg} = 3 + 2 + 2 + 2 + 3 = 12 = 2 \cdot 6 \quad \checkmark$$

adjacency matrix

0	1	0	1	1
1	0	1	0	0
0	1	0	0	1
1	0	0	0	1
1	0	1	1	0

n^2 space, $O(1)$ edge test

adjacency list

1: 2 4 5
2: 1 3
3: 2 5
4: 1 5
5: 1 3 4

$O(|V| + |E|)$ – for sparse graphs

**tree: connected + acyclic $\rightarrow |E| = |V| - 1$; a 12-vertex tree with 5 leaves:
7 internal vertices carry degree sum $22 - 5 = 17$**

BFS: queue, level order, unweighted shortest · DFS: stack, components · model by symmetry, represent by density

Three representations of the same graph – matrix, list, picture – and the degree sum checked by handshake

The derivation (why handshaking must hold). Each edge contributes 1 to the degree of each endpoint, so every edge adds exactly 2 to the degree sum: $\sum_v \text{deg}(v) = 2|E|$ ✓ – immediate corollaries: the number of odd-degree vertices is even; a graph where all $n \geq 2$ vertices have distinct degrees cannot exist (degrees run $0..n-1$ but 0 and $n-1$ cannot coexist) ✓.

Worked example 1 – the model choice. (Design question) social friendships (undirected, adjacency list – sparse), task dependencies (directed acyclic, matrix when small – cycle tests), city metro (weighted, Dijkstra's queue – beyond scope, named as such) ✓ – the problem's symmetry dictates the graph, the density dictates the representation.

Worked example 2 – tree arithmetic. A tree has 12 vertices of which 5 are leaves (degree 1): with $|E| = 11$ and handshake, the internal 7 vertices carry degree sum $22 - 5 = 17$ ✓ – every tree question is a walk over $|E| = |V| - 1$ plus $\sum \deg = 2|E|$.

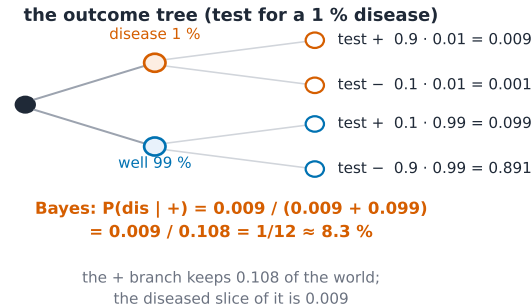
Common mistakes. Forgetting degree 0 counts (isolated vertices are vertices). Confusing connected (one piece) with complete (all edges). Assuming BFS gives shortest paths in weighted graphs (it does not – that needs weighted methods).

Summary. $G = (V, E)$; matrix vs list by density; tree: connected + acyclic + $|E| = |V| - 1$; BFS level order / DFS component order; $\sum \deg = 2|E|$, odd-degree count even; choose the model from the problem’s symmetry.

Discrete probability

Why it matters. Randomised algorithms, average-case analysis and hashing all run on exact discrete probability – the same counting from 2.4, now over equally likely outcomes.

The idea, with a picture. A sample space Ω of equally likely outcomes; events are subsets; $P(E) = |E|/|\Omega|$ ✓. **Conditional probability** $P(A \mid B) = \frac{P(A \cap B)}{P(B)}$; **Bayes’ theorem** $P(A \mid B) = \frac{P(B|A)P(A)}{P(B)}$ ✓; independence $P(A \cap B) = P(A)P(B)$.



the rules
$$P(E) = |E| / |\Omega|$$
(equal outcomes — counting!)
$$P(A \mid B) = P(A \cap B) / P(B)$$
— the B-branch, renormalised
Bayes: $P(A|B) = P(B|A)P(A) / P(B)$
slices rescaled between worlds
independence: $P(A \cap B) = P(A)P(B)$
two dice: $P(\text{sum } 7) = 6/36 = 1/6$

The tree shows what the conditioning removes: $P(A|B)$ lives on the B-branch, renormalised

The derivation (why conditioning shrinks the world). Conditioning on B keeps only the B -outcomes and renormalises their weights to sum to 1 – the fraction of B -world where A also holds ✓. That is the entire numerator/denominator of Bayes: $P(B \mid A)P(A)$ sizes the $A \cap B$ slice; $P(B)$ rescales to the B -world.

Worked example 1 – dice by counting. Two fair dice: Ω has 36 pairs. $P(\text{sum} = 7) = 6/36 = 1/6$ (six pairs); $P(\text{sum} = 11) = 2/36$ ✓; $P(\text{sum} = 7 \text{ or } 11) = 8/36 = 2/9$ – counting from 2.4 does all the work.

Worked example 2 – Bayes on a test. A disease affects 1 %; a test has sensitivity 90 % and specificity 90 %. $P(\text{dis} \mid +) = \frac{0.9 \times 0.01}{0.9 \times 0.01 + 0.1 \times 0.99} = \frac{0.009}{0.108} = 1/12 \approx 8.3 \%$ ✓ – the same arithmetic as MED12 §4, derived here from the tree: rare conditions stay improbable after imperfect evidence.

Common mistakes. Adding conditional probabilities as if independent (the conditioning world must be one world). Confusing $P(A \mid B)$ with $P(B \mid A)$ (the prosecutor’s fallacy). Counting outcomes of different likelihood as equal (only equally likely outcomes give $|E|/|\Omega|$).

Summary. $P = |E|/|\Omega|$ on equally likely outcomes; $P(A \mid B) = P(A \cap B)/P(B)$ – the B-branch renormalised; Bayes rescales slices between worlds; independence multiplies; dice, cards and hash tables are all counting problems underneath.