Discrete Mathematics Essentials

A comprehensive cheat sheet designed for undergraduate students covering core concepts in discrete mathematics, including logic, set theory, relations, algorithms, graph theory, combinatorics, and proof techniques. Simplifying abstract topics with clear definitions, formulas, examples, and common pitfalls.

Logic & Set Theory Foundations

LOGIC & PROPOSITIONAL CALCULUS

Operators

- `¬P` (NOT): Negation. True if P is false.
- `P ∧ Q` (AND): Conjunction. True if P and Q are true.
- `P ∨ Q` (OR): Disjunction. True if P or Q (or both) are true.

Operators (Cont.)

- `P → Q` (IMPLIES): Implication. False only if P is true and Q is false.
  - *If P, then Q*
- `P ↔ Q` (IFF): Biconditional. True if P and Q have same truth value.
  - *P if and only if Q*

Truth Tables

Shows all possible truth values.

```
P | Q | P → Q
--|---|-------
T | T |   T
T | F |   F
F | T |   T
F | F |   T
```

Logical Equivalences

`P ≡ Q` if same truth table.

- **De Morgan's Laws:**
  - `¬(P ∧ Q) ≡ ¬P ∨ ¬Q`
  - `¬(P ∨ Q) ≡ ¬P ∧ ¬Q`
- **Distributive Laws:**
  - `P ∧ (Q ∨ R) ≡ (P ∧ Q) ∨ (P ∧ R)`
  - `P ∨ (Q ∧ R) ≡ (P ∨ Q) ∧ (P ∨ R)`

Rules of Inference

Valid arguments to derive conclusions.

- **Modus Ponens:**
  `((P → Q) ∧ P) → Q`
- **Modus Tollens:**
  `((P → Q) ∧ ¬Q) → ¬P`

Rules of Inference (Cont.)

- **Hypothetical Syllogism:**
  `((P → Q) ∧ (Q → R)) → (P → R)`
- **Disjunctive Syllogism:**
  `((P ∨ Q) ∧ ¬P) → Q`

Tautology: Always true (e.g., `P ∨ ¬P`)
Contradiction: Always false (e.g., `P ∧ ¬P`)
Contingency: Neither (e.g., `P → Q`)

Example: Prove `P → Q ≡ ¬P ∨ Q`
Construct truth table for `¬P ∨ Q` and compare to `P → Q`.

Key Insight: The implication `P → Q` is often the trickiest. Remember it's only false when `P` is true and `Q` is false. Think of it as a promise: if the premise is true, the conclusion must follow. If the premise is false, the promise is not broken, so the implication is true regardless of the conclusion.

SETS & FUNCTIONS

Set Notation

- `A = {1, 2, 3}`: Roster method
- `B = {x | x is even}`: Set-builder
- `∅` or `{}`: Empty set
- `U`: Universal set
- `|A|`: Cardinality (number of elements)
- `A ⊆ B`: A is a subset of B
- `A ⊂ B`: A is a proper subset of B

Set Operations

- `A ∪ B` (Union): Elements in A or B or both.
- `A ∩ B` (Intersection): Elements common to A and B.
- `A \ B` (Difference): Elements in A but not in B.
- A^c (Complement): Elements in U but not in A.

Example: `A={1,2,3}, B={3,4,5}`
`A ∪ B = {1,2,3,4,5}`
`A ∩ B = {3}`
`A \ B = {1,2}`

Power Set

`P(A)` or `2^A`: The set of all subsets of A.
`|P(A)| = 2^|A|`

Example: `A = {a, b}`
`P(A) = {∅, {a}, {b}, {a, b}}`
`|P(A)| = 2^2 = 4`

Cartesian Product

`A × B = {(a, b) | a ∈ A, b ∈ B}`. Ordered pairs.
`|A × B| = |A| × |B|`

Example: `A={1,2}, B={x,y}`
`A × B = {(1,x), (1,y), (2,x), (2,y)}`
`|A × B| = 2 × 2 = 4`

Functions `f: A → B`

A relation where each input from A has exactly one output in B.

- **Domain (A):** Input values.
- **Codomain (B):** Possible output values.
- **Range (f(A)):** Actual output values (`f(A) ⊆ B`).

Types of Functions

- **Injective (One-to-one):** If $f(a_1) = f(a_2)$ then $a_1 = a_2$. No two elements map to the same image.
- **Surjective (Onto):** For every `b ∈ B`, there exists an `a ∈ A` such that `f(a) = b`. Every element in B is mapped to.
- **Bijective (One-to-one correspondence):** Both injective and surjective. Implies an inverse function exists.

Common Mistake: Confusing the codomain with the range of a function. The range is a subset of the codomain, containing only the values actually produced by the function.

Relations & Algorithms

RELATIONS

Definition: A relation `R` from set `A` to `B` is a subset of `A × B`. If `A=B`, it's a relation on `A`.

Properties of Relations (on set A)

- **Reflexive:** `(a, a) ∈ R` for all `a ∈ A`. (Loops on all vertices in digraph).
- **Symmetric:** If `(a, b) ∈ R` then `(b, a) ∈ R`. (If edge `a→b`, then `b→a`).
- **Transitive:** If `(a, b) ∈ R` and `(b, c) ∈ R` then `(a, c) ∈ R`. (If `a→b` and `b→c`, then `a→c`).
- **Antisymmetric:** If `(a, b) ∈ R` and `(b, a) ∈ R` then `a = b`. (Only `(a,a)` pairs can be symmetric, no two-way distinct edges).

Equivalence Relation: Reflexive, Symmetric, AND Transitive.

- Partitions a set into disjoint **equivalence classes**.
- **Example:** `R = {(a,b) | a ≡ b (mod n)}` on integers. Each class `[a]` contains integers congruent to `a` modulo `n`.

Partial Order Relation: Reflexive, Antisymmetric, AND Transitive.

- Elements may not be comparable.
- **Example:** `(A, ≤)` for integers, `(P(S), ⊆)` for power sets.
- A set with a partial order is called a **Poset** (Partially Ordered Set).

Representations

- **Matrix (Boolean):** `M_R[i][j] = 1` if `(a_i, a_j) ∈ R`, else `0`.
- **Digraph (Directed Graph):** Vertices for elements, directed edges for related pairs.

Example: `A={1,2,3}, R={(1,1), (1,2), (2,3)}`
`M_R = [[1,1,0],[0,0,1],[0,0,0]]`

Key Insight: When checking properties, draw small examples or use matrix multiplication (`M_R * M_R`) for transitivity. Remember that for a relation to be symmetric, if an edge exists from A to B, an edge MUST exist from B to A. For antisymmetric, if an edge exists from A to B (and A≠B), an edge CANNOT exist from B to A.

ALGORITHMS & COMPLEXITY

Algorithm Analysis: Measures efficiency (time/space) based on input size `n`.

Asymptotic Notations

Describe function growth as `n → ∞`.

Big-O (O): Upper Bound
`f(n) = O(g(n))` if `|f(n)| ≤ C|g(n)|` for `n > k`.
`f` grows no faster than `g`.

Big-Omega (Ω): Lower Bound
`f(n) = Ω(g(n))` if `|f(n)| ≥ C|g(n)|` for `n > k`.
`f` grows no slower than `g`.

Big-Theta (Θ): Tight Bound
`f(n) = Θ(g(n))` if `f(n) = O(g(n))` AND `f(n) = Ω(g(n))`.
`f` grows at the same rate as `g`.

Common Growth Rates (Slowest to Fastest)

- `O(1)`: Constant (e.g., array element access)
- `O(log n)`: Logarithmic (e.g., binary search)
- `O(n)`: Linear (e.g., sequential search)
- `O(n log n)`: Log-linear (e.g., merge sort)

Common Growth Rates (Cont.)

- `O(n^2)`: Quadratic (e.g., nested loops, bubble sort)
- `O(n^k)`: Polynomial
- `O(2^n)`: Exponential (e.g., brute-force subset sum)
- `O(n!)`: Factorial (e.g., brute-force TSP)

Recurrence Relations

Define a sequence where each term depends on previous terms. Used for recursive algorithms.

Common Methods:

- **Iteration:** Expand the recurrence, find a pattern, sum it up.
- **Master Theorem:** For `T(n) = aT(n/b) + f(n)`.
  - Compare `f(n)` with `n^(log_b a)`.

Recurrence Examples

- **Binary Search:** `T(n) = T(n/2) + O(1)` -> `O(log n)`
- **Merge Sort:** `T(n) = 2T(n/2) + O(n)` -> `O(n log n)`
- **Fibonacci (naive):** `T(n) = T(n-1) + T(n-2) + O(1)` -> `O(φ^n)` where `φ` is the golden ratio.

Common Mistake: Treating Big-O as a precise measurement rather than an upper bound. `f(n) = O(g(n))` does *not* mean `f(n)` is *exactly* `g(n)`. It means `f(n)` is *at most* proportional to `g(n)` for large `n`. Only Big-Theta implies a tight bound.

Graphs, Counting & Proofs

GRAPH THEORY

Definitions

- **Graph** `G = (V, E)`: `V`=vertices, `E`=edges.
- **Simple Graph:** No loops (edge from vertex to itself), no multiple edges between same pair.
- **Multigraph:** Allows multiple edges.
- **Directed Graph (Digraph):** Edges have direction.
- **Weighted Graph:** Edges have values (weights).
- **Degree of a Vertex (** `deg(v)` **):** Number of edges incident to `v` (double for loops).

Paths & Cycles

- **Path:** Sequence of distinct vertices where consecutive vertices are connected by an edge.
- **Cycle:** A path that starts and ends at the same vertex, visiting other vertices at most once.

Connectivity

- **Connected (Undirected):** Path between any two vertices.
- **Strongly Connected (Directed):** Path from `u` to `v` AND `v` to `u` for any `u,v`.

Trees: Connected, acyclic (no cycles) undirected graph.

- `n` vertices, `n-1` edges.
- Unique simple path between any two vertices.
- Acyclic implies no cycles.
- A **Forest** is a collection of disjoint trees.

Eulerian Paths/Circuits

Visits every *edge* exactly once.

- **Circuit:** Starts/ends at same vertex.
- **Path:** Starts/ends at different vertices.

Conditions (Undirected Graphs):

- **Circuit exists:** All vertices have `even` degree.
- **Path exists:** Exactly two vertices have `odd` degree.

Hamiltonian Paths/Cycles

Visits every *vertex* exactly once.

- **Cycle:** Starts/ends at same vertex.
- **Path:** Starts/ends at different vertices.

Note: No simple condition like Euler's. Generally an NP-complete problem to find.

Key Insight: For Eulerian paths/circuits, focus solely on the *degrees* of vertices. For Hamiltonian paths/cycles, it's about covering all vertices. Trees are fundamental: think of them as graphs without redundant connections that would form cycles.

COMBINATORICS & COUNTING

Fundamental Principles

- **Sum Rule:** If a task can be done in `n_1` ways OR `n_2` ways (disjoint), total ways `n_1 + n_2`.
  - **Example:** Choose a fruit: 3 apples or 4 oranges = 7 options.
- **Product Rule:** If a task has `n_1` ways for the first step and `n_2` ways for the second, total ways `n_1 × n_2`.
  - **Example:** Choose shirt (3) and pants (2) = 6 outfits.

Permutations

Arrangement of `k` items from `n` *distinct* items where **order matters**.

`P(n, k) = n! / (n-k)!`

Example: Number of ways to arrange 3 books from 5 unique books:
`P(5, 3) = 5! / (5-3)! = 5! / 2! = 120 / 2 = 60`

Combinations

Selection of `k` items from `n` *distinct* items where **order DOES NOT matter**.

`C(n, k) = n! / (k! * (n-k)!) = (n choose k)`

Example: Number of ways to choose 3 books from 5 unique books:
`C(5, 3) = 5! / (3! * 2!) = 120 / (6 * 2) = 10`

Inclusion-Exclusion Principle

Used to count elements in unions of sets.

- **Two Sets:** `|A ∪ B| = |A| + |B| - |A ∩ B|`
- **Three Sets:** `|A ∪ B ∪ C| = |A| + |B| + |C| - (|A ∩ B| + |A ∩ C| + |B ∩ C|) + |A ∩ B ∩ C|`

Pigeonhole Principle

If `N` items are placed into `K` containers:

- If `N > K`, at least one container must contain more than one item.
- Generally: At least one container must contain `⌈N/K⌉` items.

Example: In any group of 13 people, at least two were born in the same month (`N=13` people, `K=12` months).

Common Mistake: Confusing permutations and combinations. If the specific roles or positions matter (e.g., president, VP, secretary), it's a permutation. If it's just a group or a committee where roles aren't assigned, it's a combination. *Permutation = Position, Combination = Choice.*

PROOFS & MATHEMATICAL REASONING

Direct Proof

To prove `P → Q`:

1. Assume `P` is true.
2. Use definitions, axioms, logical equivalences, and previously proven theorems to show `Q` must be true.

Example: Prove that if `n` is an odd integer, then `n^2` is an odd integer.

- Assume `n` is odd, so `n = 2k + 1` for some integer `k`.
- Then `n^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1`.
- Since `(2k^2 + 2k)` is an integer, `n^2` is odd.

Indirect Proof (Contraposition)

To prove `P → Q`, prove its contrapositive `¬Q → ¬P`.

1. Assume `¬Q` is true.
2. Show that `¬P` must be true.

Example: Prove that if `n^2` is an even integer, then `n` is an even integer.

- Contrapositive: If `n` is odd, then `n^2` is odd.
- (This is the same example as Direct Proof; the proof is identical once the contrapositive is stated).

Proof by Contradiction

To prove `P`:

1. Assume `¬P` is true.
2. Derive a contradiction (e.g., `R ∧ ¬R`), which means the initial assumption `¬P` must be false.
3. Therefore, `P` must be true.

Example: Prove `√2` is irrational.

- Assume `√2` is rational. So `√2 = a/b` for integers `a, b` with no common factors, `b ≠ 0`.
- `2 = a^2/b^2` => `2b^2 = a^2`. Thus `a^2` is even, implying `a` is even. Let `a = 2k`.
- `2b^2 = (2k)^2 = 4k^2` => `b^2 = 2k^2`. Thus `b^2` is even, implying `b` is even.
- Contradiction! `a` and `b` have `2` as a common factor, but we assumed no common factors.

Proof by Induction

To prove `P(n)` for all integers `n ≥ base_case`.

1. **Base Case:** Show `P(base_case)` is true (usually `P(0)` or `P(1)`).
2. **Inductive Hypothesis (IH):** Assume `P(k)` is true for an arbitrary integer `k ≥ base_case`.
3. **Inductive Step (IS):** Show that `P(k+1)` is true, using the inductive hypothesis.

Strong Induction: IH assumes `P(j)` is true for ALL integers `j` such that `base_case ≤ j ≤ k`. Used when `P(k+1)` depends on more than just `P(k)`.

Tips for Proofs:

- **Understand Definitions:** Know exactly what terms mean.
- **Start & End:** Clearly state what you are assuming and what you need to show.
- **Structure:** Follow the logical flow of the proof type.
- **Scratchpad:** Do scratch work to find the path before writing the formal proof.
- **Induction:** Be meticulous with the inductive step. Clearly use the IH to transform `P(k+1)`.

Common Mistake: In induction, assuming `P(k+1)` is true at the start of the inductive step instead of proving it. The goal is to *derive* `P(k+1)` from `P(k)` (or `P(j)` for strong induction).