GUJARAT TECHNOLOGICAL UNIVERSITY

Program Name: Bachelor of Engineering
Level: UG
Subject Code: BE04000261
Subject Name: Discrete Mathematics and Graph Theory

w. e. f. Academic Year:	2024-25
Semester:	4
Category of the Course:	BSC

Prerequisite:	Algebra; logic
Rationale:	This course introduces the basic concepts of discrete mathematics in the field of computer science. It covers sets, logic, functions, relations, graph theory and algebraic structures. These basic concepts of sets, logic, functions and graph theory are applied to Boolean Algebra and logic networks, while the advanced concepts of functions and algebraic structures are applied to finite state machines and coding theory.

Course Outcomes:

Sr. No.	CO statement	Marks % Weightage
CO-1	Write an argument using logical notation and determine if the argument is or is not valid. To simplify and evaluate basic logic statements including compound statements, implications, inverses, converses, and contra positives using truth tables and the properties of logic. To express a logic sentence in terms of predicates, quantifiers, and logical connectives.	18%
CO-2	Understand the basic principles of sets and operations in sets and apply counting principles to determine probabilities. Be familiar with recurrence relations	7%
CO-3	Use the properties of algebraic structures.	25%
CO-4	Understand the basic concept of functions like domain and range of a function, identify one-to- one function, perform the composition of functions and apply the properties of functions to application problems. Apply relations and to determine their properties.	25%
CO-5	Interpret different traversal methods for trees and graphs. Model problems in Computer Science using graphs and trees.	25%

Teaching and Examination Scheme:

Teaching / Learning Scheme (in Hours per semester)					Total Credits	Assessment Pattern and Marks					Total Marks
						Theory		Tutorial / Practical			
L	T	P	SL	Total no of hours per semester		ESE (E)	PA / CA (M)	PA/ CA (I)	TW/ SL (I)	ESE (V)	
45	00	30	15	90	3	70	30	20	30	50	200

- **Problem Based Learning (PBL) aims to accommodate learning beyond syllabus as per clause 9.4 of NBA manual.**

Course Content:

Sr. No.	Content	Total Hrs	% of Weightage
1	**Propositional Logic**: Definition, Statements & Notation, Truth Values, Connectives, Statement Formulas & Truth Tables, Well-formed Formulas, Tautologies, Equivalence of Formulas, Duality Law, Tautological Implications, Examples **Predicate Logic:** Definition of Predicates; Statement functions, Variables, Quantifiers, Predicate Formulas, Free & Bound Variables; The Universe of Discourse, Examples, Valid Formulas & Equivalences, Examples	**08**	**18**
2	**Set Theory:** Basic Concepts of Set theory like inclusion, Cartesian product, The Power Set **Counting and Combinatorics:** Basic counting principles, Permutations and combinations, Binomial coefficients and the Binomial theorem, Pigeonhole principle, Inclusion-exclusion principle, Recurrence relations (Definitions and simple examples only)	**03**	**7**
3	**Functions:** Basic concepts of Functions like domain, range, surjective, injective, bijective; Composition of functions, Inverse of function **Algebraic Structures:** Algebraic structures with one binary operation- Semigroup, Monoid, Group, Subgroup, normal subgroup, Coset, homomorphic subgroups, Lagrange's theorem, Congruence relation and quotient structures. Algebraic structures with two binary operation- Ring, Integral domain and field. (Definitions and simple examples only)	**11**	**25**
4	**Relations**: Definition, Binary Relation, Representation, Domain, Range, Universal Relation, Void Relation, Union, Intersection, and Complement Operations on Relations, Properties of Binary Relations in a Set: Reflexive, Symmetric, Transitive, Anti-symmetric Relations, Partition and Covering of a Set, Equivalence Relation, Equivalence Classes, Compatibility Relation, Maximum Compatibility Block, Composite Relation, Converse of a Relation, Transitive Closure of a Relation R in Set X **Partial Ordering**: Definition, Examples, Simple or Linear Ordering, Totally Ordered Set (Chain), Frequently Used Partially Ordered Relations, Representation of Partially Ordered Sets, Hesse Diagrams, Least & Greatest Members, Minimal & Maximal Members, Least Upper Bound (Supremum), Greatest Lower Bound (infimum), Well-ordered Partially Ordered Sets (Posets). Lattice as Posets, Complete, Distributive, Modular and Complemented lattices, Boolean and pseudo Boolean lattices. (Definitions and simple examples only)	**11**	**25**

5	**Graphs:** Introduction, definition, examples; Nodes, edges, adjacent nodes, directed and undirected edge, Directed graph, undirected graph, examples; Initiating and terminating nodes, Loop (sling), Distinct edges, Parallel edges, Multi-graph, simple graph, weighted graphs, examples, Isolated nodes, Null graph; Isomorphic graphs, examples; Degree, Indegree, out-degree, total degree of a node, examples; Subgraphs: definition, examples; Converse (reversal or directional dual) of a digraph, examples; Path: Definition, Paths of a given graph, length of path, examples; Simple path (edge simple), elementary path (node simple), examples; Cycle (circuit), elementary cycle, examples; **Reachability:** Definition, geodesic, distance, examples; Properties of reachability, the triangle inequality; Reachable set of a given node, examples, Node base, examples **Connectedness:** Definition, weakly connected, strongly connected, unilaterally connected, examples; Strong, weak, and unilateral components of a graph, examples, Applications to represent Resource allocation status of an operating system, and detection and correction of deadlocks; Matrix representation of graph: Definition, Adjacency matrix, 2oolean (or bit) matrix, examples; Determine number of paths of length n through Adjacency matrix, examples; Path (Reachability) matrix of a graph, examples; Warshall's algorithm to produce Path matrix, Flowchart. **Trees:** Definition, branch nodes, leaf (terminal) nodes, root, examples; Different representations of a tree, examples; Binary tree, m-ary tree, Full (or complete) binary tree, examples; Converting any m-ary tree to a binary tree, examples; Representation of a binary tree: Linked-list; Tree traversal: Pre-order, in-order, post-order traversal, examples, algorithms; Applications of List structures and graphs	**12**	**25**

Suggested Specification table with Marks (Theory): (For B.E. only)

Distribution of Theory Marks					
R Level	U Level	A Level	N Level	E Level	C Level
10	**20**	**20**	**10**	**10**	**00**

R: Remembrance; U: Understanding; A: Application, N: Analyze and E: Evaluate C: Create and above Levels (Revised Bloom's Taxonomy)

Note: This specification table shall be treated as a general guideline for students and teachers. The actual distribution of marks in the question paper may vary slightly from above table.

Reference Books:

1. J. P. Tremblay and R. Manohar, Discrete Mathematical Structures with Applications to Computer Science, Tata McGraw-Hill,1997.
2. S. Lipschutz and M. L. Lipson, Schaum's Outline of Theory and Problems of Discrete

Mathematics, 2nd Ed., Tata McGraw-Hill,1999.
3. K. H. Rosen, Discrete Mathematics and its applications, Tata McGraw-Hill, 6th Ed., 2007.
4. David Liben-Nowell, Discrete Mathematics for Computer Science, Wiley publication, July 2017. 5. Eric Gossett, Discrete Mathematics with Proof, 2nd Edition,Wiley publication, July 2009.
5. Josheph A. Gallian, Contemporary Abstract Algebra, Chapman & Hall, 11th edition, July 2025.

Suggested MOOC Courses: https://onlinecourses.nptel.ac.in/noc25_cs27/preview

List of Experiments:

Sr. No	Practical	CO
1	**Propositional logic: parser, truth table & tautology checker** **Objective:** Parse Boolean formulas, generate truth table, decide tautology/contradiction/contingent. **Prerequisite:** Recursion or stack parsing. **Language:** Python (parser + eval) or C++ (shunting-yard + AST). **Procedure:** 1. Implement parser supporting ~(NOT), &(AND), |(OR), ->(IMPLIES), <->(IFF), parentheses. 2. Enumerate truth assignments; evaluate formula and print table. Declare tautology if all true, contradiction if all false. **Sample I/O:** Formula `(p -> q) <-> (~q -> ~p)` → Tautology? TRUE. **Deliverables:** Source, test formulas, truth tables.	1
2	**Predicate logic evaluation over finite domain** **Objective:** Evaluate quantified formulas over a finite universe and illustrate free/bound variables. **Prerequisite:** Lab 1, basic function/predicate definitions. **Language:** Python. **Procedure:** 1. Allow user to define universe U (e.g., {1..5}) and primitive predicates `P(x)`, `Q(x,y)` as code callbacks or lambda expressions. 2. Evaluate formulas with ∀ and ∃ by brute force checking. Provide examples showing different binding outcomes. **Sample I/O:**∀`x` ∃`y` `(x+y==5)` over U={1..4} → print TRUE/FALSE with witness `y`. **Deliverables:** Source, 3 example formula evaluations with explanations.	1

3	**Set operations & verifying identities** **Objective:** Implement set data structure; perform union/intersection/difference/symmetric product/power set; programmatically verify De Morgan and distributive identities. **Prerequisite:** Arrays / lists, basic I/O. **Language:** Python (recommended) or C/C++. **Procedure (steps):** 1. Read two sets A, B (elements integers/strings). 2. Implement functions: union(A,B), intersection(A,B), diff(A,B), sym_diff(A,B), cartesian_product(A,B), power_set(A). Generate random small sets and programmatically verify identities (e.g., (A ∪ B)' = A' ∩ B' for universal set provided). **Sample I/O:** Input: A={1,2,3}, B={2,4} → prints union `{1,2,3,4}`, power set of A, identity check TRUE/FALSE. **Deliverables:** Source, README, sample runs, short report demonstrating identity verifications.	2
4	**Counting: nPr, nCr, Pascal triangle & pigeonhole** **Objective:** Implement robust combinatorics: nPr, nCr (DP/Pascal), generate Pascal triangle up to n=30, demonstrate pigeonhole principle with simulation. **Prerequisite:** Loops, big-integer handling (Python ints ok). **Language:** C++ or Python. **Procedure:** 1. Implement DP Pascal table for nCr to avoid overflow. 2. Compute nPr via nCr*factorial or DP. Simulate pigeonhole example (e.g., show for 13 people and 12 months there must be two with same birth month). **Sample I/O:** n=5 → Pascal row `1 5 10 10 5 1`. **Deliverables:** Source, sample outputs, one solved counting problem.	2
5	**Functions: mapping, composition & inverse** **Objective:** Represent finite functions (domain→codomain) and detect injective/surjective/bijective; compute composition and inverse when exists. **Prerequisite:** Dictionaries/maps, arrays. **Language:** C++ (std::map/vector) or Python. **Procedure:** 1. Input domain X, codomain Y, mapping f as pairs. 2. Check injective (distinct images), surjective (image==Y), bijective. Compose `f∘g` from two mappings; compute inverse if bijective. **Sample I/O:** X={1,2}, Y={a,b}, f={1→a,2→b} → Bijective; inverse `{a→1,b→2}`. **Deliverables:** Source, testcases including non-bijective.	3

6	**Relations: adjacency matrix, properties & examples** **Objective:** Input binary relations; output adjacency matrix and test reflexive/symmetric/transitive/antisymmetric with counterexamples. **Prerequisite:** Matrix basics. **Language:** C / C++ / Python. **Procedure:** 1. Read set X (size $n \leq 20$) and relation R as pairs. 2. Build n×n adjacency matrix and print. Check and report which properties hold; if property fails, show counterexample pair(s). **Sample I/O:** X={1,2,3}, R={(1,1),(1,2),(2,1)} → symmetric? TRUE, reflexive? FALSE (3,3 missing). **Deliverables:** Source, matrix printouts, report.	3
7	**Transitive closure: Warshall algorithm & reachability matrix** **Objective:** Implement Warshall algorithm for transitive closure and answer reachability queries. **Prerequisite:** Lab 6, matrix manipulation. **Language:** C++ or Python. **Procedure:** 1. Use adjacency matrix from Lab 6. 2. Implement Warshall (boolean matrix update) to compute reachability matrix. 3. Validate results by performing BFS from nodes and compare. **Sample I/O:** Print closure matrix and reachability sets for each node. **Deliverables:** Source, closure matrix, BFS verification logs.	3
8	**Partial orders & Hasse diagram generation (Graphviz)** **Objective:** From a given partial order generate the Hasse diagram (cover edges), minimal/maximal elements, and represent `.dot` file for visualization. **Prerequisite:** Warshall/transitive closure. **Language:** Python or C++. **Procedure:** 1. Input poset as relation R. Remove reflexive pairs and transitive edges to get cover relation. 2. Compute minimal/maximal elements and least/greatest members where present. Output Graphviz `.dot` for Hasse diagram. **Sample I/O:** Edge list for Hasse; minimal elements `{...}`. **Deliverables:** Source, `.dot` file (students produce PNG via `dot -Tpng`).	3

9	**Lattice checks: distributive & complemented tests (small finite lattices)** **Objective:** Given a small finite lattice defined as poset, check if distributive, modular, complemented (provide witness/counterexample). **Prerequisite:** Lab 8; join/meet computation. **Language:** Python. **Procedure:** 1. Define lattice elements and partial order (≤). 2. Implement meet (inf) and join (sup) via poset searches. Verify distributive identity `x ∧ (y ∨ z) == (x ∧ y) ∨ (x ∧ z)` for all triples; search for complements for each element. **Sample I/O:** Boolean lattice of 2 elements → distributive TRUE, complemented TRUE. M3 example → distributive FALSE (show counterexample). **Deliverables:** Source, lattice examples with results.	3
10	**Recurrence relations: iteration & closed-form (characteristic equation)** **Objective:** Compute terms for linear homogeneous recurrences (constant coefficients) iteratively and derive closed form via characteristic polynomial (simple roots). **Prerequisite:** Polynomial roots (numeric), linear algebra basics helpful. **Language:** Python (use `numpy` optionally) or C++. **Procedure:** 1. Input recurrence coefficients and initial terms (e.g., a_n = 3a_{n-1} -2a_{n-2}). 2. Produce first 20 terms by iteration. Solve characteristic polynomial, compute closed form for distinct roots, compare values for initial terms and up to n=20. Provide error table. **Sample I/O:** Fibonacci recurrence → sequence `0,1,1,2,3,...` and closed-form check. **Deliverables:** Source, iteration vs closed-form table.	3
11	**Permutations & group basics: cycle notation, inverse, subgroup generation** **Objective:** Represent permutations, convert to cycle notation, compute inverses and generate subgroup by repeated composition (demonstrate Lagrange theorem example). **Prerequisite:** Arrays/mappings, loops. **Language:** C++ or Python. **Procedure:** 1. Read permutation on n elements. Output cycle decomposition. 2. Compute inverse permutation and composition p∘q. Generate subgroup produced by a permutation (powers) and compute its size; compare with n. **Sample I/O:** Permutation `[2,3,1]` → cycle `(1 2 3)`, order 3. **Deliverables:** Source, sample subgroup computations.	4

12	**Modular arithmetic & field test (Z_n)** **Objective:** Implement arithmetic in Z_n and test for field property by checking multiplicative inverses for all nonzero elements. **Prerequisite:** Loops, gcd function. **Language:** C / C++ / Python. **Procedure:** 1. For given n, compute modular inverses using extended Euclid for each a with $1 \leq a < n$. 2. If every nonzero a has inverse → declare Z_n a field; otherwise show counterexamples. Test for n = 2..20. **Sample I/O:** Z5 → field TRUE; Z6 → field FALSE (2 has no inverse). **Deliverables:** Source, inverse tables for n values.	4
13	**Graph basics: adjacency list/matrix, degree, connected components** **Objective:** Implement graph input, produce adjacency matrix/list, compute degree sequences, and find connected components (undirected) or strongly connected components (Kosaraju/Tarjan optional). **Prerequisite:** Data structures (lists, stacks). **Language:** C++ (STL) or Python. **Procedure:** 1. Read n, m edges (directed/undirected option). Build adjacency list & matrix. 2. Compute degree/indegree/outdegree and print. Use DFS/BFS to find connected components (or Kosaraju for SCC). **Sample I/O:** Graph with edges → degree list, components printed. **Deliverables:** Source, degree tables, component lists.	5
14	**Graph algorithms: BFS/DFS, shortest path (unweighted), cycle detection** **Objective:** Implement BFS & DFS traversals, reconstruct shortest path in unweighted graphs, detect cycles (directed/undirected). **Prerequisite:** Lab 13, queues/stacks. **Language:** C++ or Python. **Procedure:** 1. Implement BFS (queue) to compute distances and parents; reconstruct shortest path from source to target. 2. Implement DFS and use it to detect cycles (use visited/recursion-stack for directed graphs). **Sample I/O:** BFS from node 1 → distances array; cycle detection TRUE with example cycle nodes. **Deliverables:** Source, traversal logs, example path reconstructions.	5

15	**Weighted graphs: Dijkstra (nonnegative weights) & MST (Kruskal) + tree traversals** **Objective:** Implement Dijkstra for shortest paths (non-negative weights) and Kruskal (with DSU) for MST; also include binary tree representations and pre/in/post order traversals. **Prerequisite:** Priority queue, DSU (union-find), tree node structures. **Language:** C++ (recommended for performance) or Python. **Procedure:** 1. Read weighted graph; run Dijkstra from chosen source; print distance table and parent tree. 2. Run Kruskal to compute MST edges & total cost. For tree traversal: read binary tree (level-order), implement pre/in/post recursive and iterative traversals and compare outputs. **Sample I/O:** Weighted graph → Dijkstra distances, MST edge list and cost; tree traversals strings. **Deliverables:** Source, example graph/tree files, outputs.	5

Final Evaluation scheme (suggested)

- **Each practical:** 25 marks (except Lab 15: 30 marks).
  - Program correctness & results — 12 (15 for Lab 15)
  - Code quality & modularity (comments, naming) — 4
  - Test cases, sample I/O, edge-case handling — 4
  - Report, screenshots, viva — 5
- **Semester total:** Choose best 12 of 15 practicals (or evaluate all & scale to 100 marks based on GTU practical weight as appropriate). Align with GTU marking & credits per guidelines.

Submission requirements (per practical)

1. **Source code** file(s) with header comments (name, roll no, lab title).
2. **README** with language & run instructions (e.g., `python lab4.py` or `g++ lab14.cpp -o lab14 && ./lab14`).
3. **Input files** used for testing (if any) and **sample output** screenshots / terminal logs.
4. **Short report (1–2 pages)**: objective, algorithm, complexity notes, sample runs, conclusions.
5. **Graphviz .dot** files for Hasse/graph visuals where requested.

Implementation & Instructor Notes

- **Language choice:** Encourage Python for logic, parsing, matrices and faster prototyping; C++ for graph algorithms, DSU, and pointer/tree exercises to teach memory management and performance.
- **Tools:** Graphviz (`dot`) for Hasse/graph visuals; Jupyter notebooks recommended for combining code and explanation.
- **Academic integrity:** Require students to submit unique test cases and answer viva to ensure understanding.
- **References:** GTU syllabus & course outcomes (for alignment).

List of suggested activities for Problem Based Learning:

Sr. No	Name of Activity	No. of Hours	Evaluation Criteria
1	Assignments and Tutorials	10 assignments × 1h each = Total = 10h	Based on completeness, correctness and submission of assignments
2	Technical Video-Based Learning – Watch online lectures/tutorials on topics covered	Duration = 5h, Report & Presentation = 5h; Total = 10h	Report / Presentation on key learning outcomes
3	Seminar / Presentation – Study and present a technical topic beyond syllabus	Study/Prep = 10h, Report = 3h, Presentation = 2h; Total = 15h	Based on technical depth, quality of report and presentation skills
4	Real-World Case Study problems based on syllabus	study = 5h, Report prep = 5h; Total = 10h	Based on correctness, completeness and analysis of the report

Note:

- Above mentioned activities are suggestive, faculty can carry other activity related to the subject which can enhance theoretical and practical understanding of the students.
- The number of hours is suggestive. Faculty can sub-divide the number of hours based on the activity. However, total number of hours is fixed.
- Rubrics for the evaluation can be prepared by the faculty.
- All records pertaining to the evaluation and assessment of self-learning activities must be properly maintained and preserved at the institute level. These records should be made available to the university upon request.
- Institutes are encouraged to utilize digital platforms, such as Microsoft Teams, for effective record-keeping and to ensure transparency in the evaluation and assessment of self-learning activities.

****