CIS 5210 Midterm Cheatsheet

Scope: Python Skills, Rational Agents, Uninformed Search, Informed Search, Adversarial Search, CSPs, Logical Agents.
Generated (UTC): 2026-02-22T00:17:20+00:00
Source Labels: A=Assignments, Q=Quizzes, S=Slides/Module Resources, P=Practice Exam, T=Textbook, R=Reading Requirements, E=Exam Requirements, Y=Synthesized.

Python Skills

Priority concepts: Data structures, Comprehensions, Recursion, Generators, PEP8 style
Key Definitions: Mutable object: object whose value can change in place (list, dict, set). ; Immutable object: object whose value cannot change after creation (tuple, str). ; Hashable object: object with stable hash/equality, usable as dict key or set element.
Concept Definitions: Iterator protocol uses __iter__ and __next__ to stream values lazily. ; Comprehensions compactly transform/filter iterables while preserving declarative intent. ; Recursion correctness needs a base case, progress step, and invariant argument.
Original Algorithms (material-grounded): State-Space Search Template (Assignments)
Source: CIS5210-Assignments + course search coding patterns

1. Encode each state in an immutable representation suitable for hash-based duplicate detection.
2. Initialize frontier with start state and initialize explored/reached with the same start key.
3. Pop one node from frontier according to queue discipline and goal-test immediately after pop.
4. Generate legal successor states, attach parent pointer and action metadata, and enqueue unseen states.
5. Reconstruct final path by following parent pointers from goal node back to the start.

Recursive Backtracking Template (Assignments)
Source: CIS5210-Assignments + CSP preparation patterns

1. Check base condition for solved state and return solution immediately when satisfied.
2. Select next decision variable/state choice using deterministic ordering policy.
3. Iterate candidate values/actions and skip candidates that violate current constraints.
4. Apply candidate, recurse, and propagate success return upward if recursive call succeeds.
5. Undo candidate side effects before trying next candidate; return failure if all fail.

Data Structures/Representations: deque for FIFO frontier; list for stack-like LIFO usage. ; dict for memoization/parent pointers; set for explored-state membership checks. ; tuple/frozenset for immutable state encoding in search problems.
Complexity/Guarantees: set/dict membership is average-case O(1), while list membership is O(n). ; Generator pipelines reduce peak memory from O(n) materialization toward O(1) incremental iteration. ; Recursion depth is O(depth) call-stack space and must stay within interpreter limits.
Source mix in selected bullets: Y=6 **High-yield notes:**

- Use list for ordered mutable sequences, tuple for fixed records, set for unique elements, and dict for key-value lookup. [Y]
- Average-case membership is O(1) for set and dict, and O(n) for list and tuple. [Y]
- Prefer list and dict comprehensions for compact transformations while keeping side effects out of expressions. [Y]
- Use generators for streaming data to avoid storing full intermediate lists in memory. [Y]
- In recursion, define a correct base case and guarantee each recursive call makes progress toward it. [Y]
- Represent search states with hashable immutable values such as tuples so they can be stored in visited sets. [Y]

Rational Agents

Priority concepts: PEAS, Task environments, Agent function, Rationality, Task Environment, Rational Agent
Key Definitions: Rational agent: chooses action that maximizes expected performance from percept history. ; PEAS: Performance measure, Environment, Actuators, Sensors. ; Agent function maps percept sequence to action; agent program implements this mapping.
Concept Definitions: Task environments are classified by observability, determinism, episodic/sequential, static/-dynamic, and discrete/continuous. ; Autonomy increases as behavior depends more on learned experience than fixed prior rules. ; Utility-based reasoning compares expected outcome quality, not only goal reachability.
Original Algorithms (material-grounded): Table-Driven Agent Procedure
Source: AIMA Chapter 2 (agent program forms) + rational agents slides

1. Append current percept to the complete percept sequence maintained by the agent.
2. Lookup action in a precomputed table keyed by the full percept sequence.
3. Return the table action as the chosen behavior for the current step.
4. Repeat for each new percept without explicit model update logic.

Model-Based Reflex Agent Procedure
Source: AIMA Chapter 2 + rational agents slides

1. Update internal state using previous state, previous action, current percept, and transition model.
2. Match updated state against condition-action rules for the task environment.
3. Select action associated with the first applicable rule.
4. Store chosen action for next state update cycle and return action.

Data Structures/Representations: Percept sequence/history for policy input. ; Internal state or

belief-state representation for partially observable settings. ; Utility table/model parameters for outcome scoring.
Complexity/Guarantees: Policy lookup time is O(1) if action rules are table-driven and percept keys are indexed. ; Belief-state updates can grow combinatorially with hidden-state cardinality in partially observable tasks. ; Rationality guarantee is expectation-based with respect to the defined performance measure.
Source mix in selected bullets: Y=6 **High-yield notes:**

- A rational agent selects the action that maximizes expected performance given percept history and built-in knowledge. [Y]
- PEAS defines a task environment: Performance measure, Environment, Actuators, and Sensors. [Y]
- Agent function maps percept sequences to actions; agent program is the concrete implementation running on hardware. [Y]
- Task environments vary along observability, determinism, episodic versus sequential, static versus dynamic, and discrete versus continuous. [Y]
- Randomized behavior can still be rational when it improves expected utility under uncertainty. [Y]
- Autonomy increases as an agent relies more on experience and learning rather than fixed prior rules. [Y]

Uninformed Search

Priority concepts: BFS, DFS, Uniform-cost, Iterative deepening, Graph vs tree search, Uninformed Search, Breadth-First Search, Uniform-Cost Search
Key Definitions: Node stores state, parent, action, path cost g(n), and depth. ; Complete algorithm finds a solution if one exists under stated assumptions. ; Optimal algorithm returns minimum-cost solution under stated assumptions.
Concept Definitions: Tree search may revisit states; graph search prevents repeats via explored checks. ; BFS expands shallowest nodes first; DFS expands deepest frontier node first. ; UCS expands node with minimum g(n) when step costs are nonnegative.
Original Algorithms (material-grounded): Breadth-First Search (Graph Version)
Source: AIMA Chapter 3 (Uninformed Search) + module search slides

1. Create initial node, push it into FIFO frontier queue, and mark initial state as reached.
2. Loop while frontier is non-empty: pop oldest node from frontier.
3. If popped node is goal, terminate and return this node as solution.
4. Expand node and for each successor state not in reached, mark reached and append successor to frontier.
5. If frontier empties with no goal found, return failure.

Uniform-Cost Search (Graph Version)
Source: AIMA Chapter 3 (Dijkstra/UCS) + module search slides

1. Initialize priority frontier ordered by path cost g and set reached[start]=start node.
2. Pop minimum-g node from frontier; if it is a goal, return it as optimal under positive step costs.
3. Expand node and compute each child path cost g_child.
4. If child state is unseen or g_child is lower than reached entry, update reached and push child.
5. Terminate with failure when frontier becomes empty.

Data Structures/Representations: Queue for BFS, stack (explicit/implicit) for DFS. ; Priority queue for UCS keyed by g(n). ; Explored set and parent map for duplicate detection and path reconstruction.
Complexity/Guarantees: BFS time and space are O(b^d) for branching factor b and shallowest solution depth d. ; DFS time is O(b^m) with O(bm) space for maximum depth m. ; UCS is complete and optimal when step costs are bounded below by a positive constant.
Source mix in selected bullets: Y=6 **High-yield notes:**

- Breadth-first search expands shallowest nodes first and is complete for finite branching factors. [Y]
- BFS is optimal when all step costs are equal. [Y]
- Depth-first search uses little memory but is not optimal and can fail to terminate in infinite-depth spaces. [Y]
- Uniform-cost search expands the node with minimum path cost g(n). [Y]
- UCS is complete and optimal when every step cost is strictly positive. [Y]
- Iterative deepening combines DFS memory usage with BFS shallow-solution guarantees. [Y]

Informed Search

Priority concepts: Greedy best-first, A*, Admissibility, Consistency, Heuristic design, Informed Search, Heuristic Function, Uninformed Search
Key Definitions: Heuristic h(n): estimate of cheapest remaining cost from n to goal. ; Admissible heuristic never overestimates true remaining cost. ; Consistent heuristic satisfies h(n) <= c(n,a,n') + h(n').
Concept Definitions: A* evaluates f(n)=g(n)+h(n); greedy best-first uses f(n)=h(n). ; Heuristic dominance (h2>=h1 pointwise) usually reduces expansions if both admissible. ; Consistency implies admissibility and monotone f-values along paths.
Original Algorithms (material-grounded): Greedy Best-First Search
Source: AIMA Chapter 3 (Informed Search) + heuristic search slides

1. Set evaluation function to f(n)=h(n) and initialize priority frontier by lowest h.
2. Pop node with smallest heuristic estimate from frontier.
3. Goal-test popped node and return if goal condition is satisfied.
4. Expand popped node and insert qualifying successors into frontier (with reached checks in graph mode).
5. Return failure only if frontier becomes empty.

A* Search (Graph Version)
Source: AIMA Chapter 3 (A and admissibility/consistency) + heuristic search slides*
1. Use f(n)=g(n)+h(n); initialize frontier with start node and reached with start best g.
2. Pop node with smallest f-value and goal-test on pop.
3. If goal popped, return it as optimal when heuristic assumptions hold.
4. Expand node and compute child g/f values; keep child only if it improves best-known g for that state.
5. Continue until frontier empty; then return failure.

Data Structures/Representations: Open frontier as min-heap keyed by f(n). ; Closed/explored map storing best known g(n) per state. ; Heuristic lookup tables or feature-based evaluation functions.
Complexity/Guarantees: A* with admissible h is optimal in tree search; with consistent h it is optimal in graph search. ; A* worst-case time/space remain exponential in depth despite good heuristics. ; Heuristic dominance improves expansion efficiency while preserving admissibility guarantees.
Source mix in selected bullets: Y=6 **High-yield notes:**
- Greedy best-first search uses f(n) = h(n) and chooses nodes that appear closest to a goal. [Y]
- A* search uses f(n) = g(n) + h(n) to balance path cost so far and estimated remaining cost. [Y]
- An admissible heuristic never overestimates the true remaining cost to a goal. [Y]
- A consistent heuristic satisfies h(n) <= c(n,a,n') + h(n') for every transition. [Y]
- Consistency implies admissibility and ensures nondecreasing f-values along paths. [Y]
- With graph search and a consistent heuristic, A* is complete and optimal under standard positive-cost assumptions. [Y]

Adversarial Search

Priority concepts: Minimax, Alpha-beta pruning, Expectimax, Evaluation function, Utility Theory, Alpha-Beta Pruning, Adversarial Search, Expected Utility
Key Definitions: Minimax value: backed-up utility assuming optimal play by both players. ; Alpha: lower bound on MAX value; beta: upper bound on MIN value. ; Expectimax chance node backs up expected utility under probability model.
Concept Definitions: Zero-sum deterministic games use alternating MAX/MIN backups. ; Alpha-beta pruning preserves minimax value while skipping irrelevant branches. ; Depth cutoffs require evaluation function approximating long-horizon utility.
Original Algorithms (material-grounded): Minimax Search
Source: AIMA adversarial search chapter + minimax lecture slides
1. At terminal state or depth cutoff, return utility/evaluation value.
2. At MAX node, recursively evaluate each child and return maximum child value.
3. At MIN node, recursively evaluate each child and return minimum child value.
4. At root, choose action leading to child with returned minimax value.

Alpha-Beta Pruning Search
Source: AIMA alpha-beta section + alpha-beta lecture slides
1. Initialize alpha=-inf and beta=+inf at root; recurse like minimax.
2. At MAX node, update alpha with best value seen so far.
3. At MIN node, update beta with best value seen so far for MIN.
4. Prune remaining children whenever alpha>=beta because ancestor decision cannot change.
5. Return same final move as minimax but with fewer expanded nodes.

Data Structures/Representations: Game tree node representation: state, player-to-move, legal actions. ; Transposition table caches evaluated states to avoid recomputation. ; Move ordering structures prioritize strong actions early for better pruning.
Complexity/Guarantees: Minimax explores O(b^d) nodes for branching factor b and depth d. ; Alpha-beta best-case effective complexity approaches O(b^(d/2)) under strong move ordering. ; Alpha-beta preserves exact minimax value while pruning provably irrelevant branches.
Source mix in selected bullets: Y=6 **High-yield notes:**
- Minimax computes the value of a state assuming optimal play by MAX and MIN. [Y]
- At MAX nodes choose the maximum child value; at MIN nodes choose the minimum child value. [Y]
- Alpha-beta pruning preserves minimax optimality while reducing explored nodes. [Y]
- Alpha is a lower bound on MAX's achievable value; beta is an upper bound on MIN's achievable value. [Y]
- Prune whenever alpha >= beta because further exploration cannot affect the ancestor decision. [Y]
- Move ordering is critical: better ordering leads to more pruning and lower effective branching factor. [Y]

CSPs

Priority concepts: Backtracking, Forward checking, Arc consistency, MRV and LCV, Constraint Satisfaction Problem, Backtracking Search, AC-3, Arc Consistency
Key Definitions: CSP consists of variables, domains, and constraints over variable assignments. ; Consistent assignment satisfies all constraints over currently assigned variables. ; Arc consistency for Xi->Xj means each Xi value has supporting value in Xj.
Concept Definitions: MRV picks variable with smallest remaining legal domain. ; Degree heuristic breaks MRV ties using number of unassigned constrained neighbors. ; LCV chooses value eliminating the fewest options for re-

maining variables.

Original Algorithms (material-grounded): Backtracking Search for CSP

Source: AIMA CSP chapter (backtracking procedure) + CSP slides

1. If assignment is complete, return assignment as solution.
2. Select an unassigned variable (typically MRV with degree tie-break).
3. Order candidate values (typically LCV), then iterate values.
4. If value is consistent, assign it and run inference (forward checking or AC propagation).
5. Recurse on updated assignment; if recursion fails, undo assignment/inference and continue.
6. Return failure if all values for selected variable fail.

AC-3 Arc Consistency

Source: AIMA CSP chapter (AC-3) + CSP slides

1. Initialize queue with all arcs (Xi, Xj) in constraint graph.
2. Pop one arc and call REVISE(Xi, Xj) to remove unsupported values from domain Xi.
3. If domain Xi becomes empty, return failure immediately.
4. If Xi was revised, enqueue all neighbor arcs (Xk, Xi) for Xk in Neighbors(Xi) except Xj.
5. When queue is empty, return success (arc-consistent under current domains).

Data Structures/Representations: Constraint graph with variables as nodes and constraints as edges. ; Domain map variable->current legal values. ; Arc queue/deque for AC-3 propagation workflow.

Complexity/Guarantees: Backtracking is exponential in worst case, but propagation and heuristics reduce practical search. ; AC-3 runs in O(e*d^3) in the standard bound with e arcs and domain size d. ; Tree-structured CSPs are solvable in linear time in variable count after ordering.

Source mix in selected bullets: Y=6 **High-yield notes:**

- A CSP is defined by variables, domains, and constraints; a solution assigns every variable consistently. [Y]
- Backtracking search incrementally assigns variables and backtracks on constraint violations. [Y]
- MRV chooses the variable with the fewest legal values left. [Y]
- Degree heuristic breaks MRV ties by choosing the variable that constrains the most unassigned neighbors. [Y]
- LCV prefers values that eliminate the fewest options for neighboring variables. [Y]
- Forward checking removes values from neighbor domains immediately after each assignment. [Y]

Logical Agents

Priority concepts: Knowledge base, Entailment, Inference, Resolution, Wumpus world, Propositional Logic,

Logical Agents

Key Definitions: Entailment KB |= alpha means alpha true in all models where KB is true. ; Sound inference derives only entailed sentences; complete inference derives all entailed sentences. ; Resolution combines clauses with complementary literals to infer new clause.

Concept Definitions: Syntax concerns symbol forms; semantics concerns truth in models. ; CNF transforms sentences into conjunction of disjunctions for resolution. ; Horn clauses support efficient forward and backward chaining.

Original Algorithms (material-grounded): Truth-Table Entailment Check (TT-Entails)

Source: AIMA logical agents chapter + logic lecture materials

1. Enumerate all propositional models over symbols in KB and query.
2. For each model where KB is true, require query to also evaluate true.
3. If any model satisfies KB but falsifies query, return not entailed.
4. If no countermodel exists after enumeration, return entailed.

Propositional Resolution (PL-Resolution)

Source: AIMA logical agents chapter (resolution algorithm) + logic slides

1. Convert KB and negated query into conjunctive normal form clauses.
2. Repeatedly resolve clause pairs that contain complementary literals.
3. If empty clause is derived, return entailed by contradiction.
4. If no new clauses can be added, return not entailed.

Data Structures/Representations: Knowledge base as set of clauses/sentences. ; Clause representation as sets/lists of literals. ; Agenda/goal stack structures for chaining-based inference.

Complexity/Guarantees: Propositional entailment/-model checking is exponential in number of symbols in the worst case. ; Resolution is refutation-complete for propositional logic over CNF clauses. ; Forward/backward chaining over Horn clauses is polynomial-time in KB size.

Source mix in selected bullets: Y=6 **High-yield notes:**

- A logical agent stores declarative knowledge in a knowledge base and derives new facts by inference. [Y]
- KB entails alpha when alpha is true in every model where KB is true. [Y]
- Sound inference derives only entailed sentences; complete inference can derive every entailed sentence. [Y]
- Propositional model checking is complete but can be computationally expensive in worst case. [Y]
- Resolution uses proof by contradiction: show KB and not alpha is unsatisfiable. [Y]
- To use resolution, convert sentences to conjunctive normal form and resolve complementary literals. [Y]