GUJARAT TECHNOLOGICAL UNIVERSITY

BE-5 SEMESTER – OLD PAPER – S22 TO W25 – QUESTION BANK ANSWER

Subject Name & Code:

ANALYSIS AND DESIGN OF ALGORITHMS (3150703)

(Disclaimer: The purpose of these AI-generated responses is just education and reference. Utilise them to grasp topics and structure, but always rewrite in your own words and double-check)

Unit 7: Backtracking

Q1 (and Q3–Q181) – Explain the backtracking method and solve the 4-Queens problem. (7 marks)

Ans:

Backtracking Method – A systematic algorithmic technique for solving constraint satisfaction problems (CSPs) and optimization problems. It incrementally builds candidates for a solution and abandons a candidate ("backtracks") as soon as it determines that the candidate cannot possibly lead to a valid complete solution.

Key characteristics:

- **State space tree** – Explores the search space recursively.
- **Pruning** – Eliminates branches that violate constraints.
- **Depth-first search** – Typically implemented using recursion.
- **Recovery** – Undoes choices when backtracking.

General backtracking algorithm:

text

```
Backtrack(candidate):
   if candidate is a complete solution:
      output candidate or return true
   else:
      for each possible next move:
         if move is valid (satisfies constraints):
            make move
            Backtrack(next candidate)
            undo move (backtrack)
```

4-Queens Problem – Place 4 queens on a 4×4 chessboard such that no two queens attack each other (no same row, column, or diagonal).

Step-by-step solution:

Place queens row by row, each queen in a column where it is safe.

Board representation: rows 1–4, columns 1–4.

Constraints for a new queen at (row, col):

- No other queen in same column.
- No other queen on same diagonal:
  |row - row_other| != |col - col_other|

Backtracking trace:

First solution found (after further backtracking):

- Q1 at (1,2)
- Q2 at (2,4)
- Q3 at (3,1)
- Q4 at (4,3)

Visual representation of one solution:

Row	Col1	Col2	Col3	Col4
1		Q		
2				Q
3	Q			
4			Q	

Total solutions for 4-Queens: 2 distinct solutions (others are rotations/reflections).
Time complexity: For n-Queens, worst-case O(n!) but pruning makes it much faster in practice.
Real-world application: Puzzle solving (Sudoku, crossword), graph coloring, constraint satisfaction (scheduling, timetabling), AI game playing.

Q2 – Differentiate between backtracking and branch-and-bound algorithms. (4 marks)
Ans:

Parameter	Backtracking	Branch-and-Bound
Primary use	Constraint satisfaction problems (find any or all solutions)	Optimization problems (find optimal solution)
Pruning criterion	Feasibility constraints only	Bounding function (upper/lower bound) to prune suboptimal branches
Search order	Typically DFS (depth-first)	Can use BFS, best-first, or DFS with bounding
State space	Generates all feasible solutions	Generates solutions but discards those that cannot beat best known
Bound calculation	No bound needed	Requires estimating bound (e.g., cost so far + heuristic)
Optimality guarantee	Finds all solutions but not necessarily optimal (for optimization, would need exhaustive)	Guarantees optimal solution if bounding is correct
Memory usage	O(depth) recursion stack	Can be higher (e.g., priority queue in best-first)
Examples	N-Queens, Sudoku, Hamiltonian cycle,	Travelling Salesman Problem (TSP),

Parameter	Backtracking	Branch-and-Bound
	graph coloring	knapsack (0/1), job scheduling

Example illustrating difference:

- **Backtracking** for 0/1 knapsack: explores all subsets, prunes only when weight exceeds capacity. Finds all feasible solutions, then picks best.
- **Branch-and-bound** for 0/1 knapsack: maintains best value so far. At a node, computes upper bound (e.g., fractional knapsack value). If bound ≤ best, prune that branch – much faster.

Key insight: Branch-and-bound = backtracking + intelligent bounding (optimization). Backtracking = satisfaction without cost optimization.

Other Important Question 1 – What is state space tree? Explain with Eight Queens problem. (7 marks)

Appeared in: S23 (Q5c, 07 marks), W22 (Q4b, 04 marks) – highest 7

Ans:

State space tree – A tree that represents all possible partial and complete solutions to a problem. Each node corresponds to a state (partial solution), and each edge represents a decision/choice. The root is the initial empty state, leaves are complete solutions or dead ends. Backtracking performs a depth-first traversal of this tree, pruning branches that violate constraints.

Eight Queens problem – Place 8 queens on an 8×8 chessboard so that no two attack each other (same row, column, or diagonal).

State space tree for 8-Queens:

- **Root:** No queens placed.
- **Level 1 (row 1):** 8 children, each representing placing a queen in one of the 8 columns of row 1.
- **Level 2 (row 2):** For each node at level 1, up to 8 children (queen in row 2, column c), but many are pruned due to column/diagonal conflicts.
- **Levels 3 to 8:** Continue placing queens row by row.
- **Leaf:** Either a complete valid solution (8 queens placed) or a dead end (no safe column for next row).

Example partial tree (for 4-Queens, but principle same):

[DG PROMPT]
Title: State Space Tree for 4-Queens (Partial)
Description: Draw a tree with root labeled "{}". Level1: four branches labeled "Q1 at col1", "Q1 at col2", "Q1 at col3", "Q1 at col4". From "Q1 at col1", level2: show branches for row2 col1 (conflict – cross out), col2 (conflict – cross out), col3 (valid – keep), col4 (valid – keep). Continue similarly. Valid paths lead to leaves representing full board configurations.

Importance of state space tree:

- Visualizes all possibilities.
- Backtracking prunes invalid branches, reducing exponential search.
- For 8-Queens, total nodes without pruning = huge; with pruning, only ~15,000 nodes visited.

Real-world application: AI game tree search, constraint satisfaction (scheduling, Sudoku).

Other Important Question 2 – Draw state space tree for 4-Queens problem. (4 marks)
Appeared in: S24 (Q5b, 04 marks)
Ans:
State space tree for 4-Queens (complete – showing only valid children, pruning at conflicts):
Tree description (text for drawing):

- **Root:** [ ] (no queen placed)
- **Level 1 (Row 1):** Four nodes for columns 1,2,3,4.
  All are initially valid: [1], [2], [3], [4]
- **From** [1] **(Queen at (1,1)):**
  Row2 possible columns:
  col1: conflict (same column) → prune
  col2: diagonal conflict (|2-1|=1, |2-1|=1) → prune
  col3: safe → node [1,3]
  col4: safe → node [1,4]
- **From** [2] **(Queen at (1,2)):**
  Row2:
  col1: safe → [2,1]
  col2: conflict column → prune
  col3: diagonal? (|2-1|=1,|3-2|=1) → prune
  col4: safe? check diagonal: |2-1|=1,|4-2|=2 → safe → [2,4]
- **From** [3] **(Queen at (1,3)):**
  Row2:
  col1: safe → [3,1]
  col2: diagonal (|2-1|=1,|2-3|=1) → prune
  col3: column conflict → prune
  col4: diagonal (|2-1|=1,|4-3|=1) → prune → only [3,1]
- **From** [4] **(Queen at (1,4)):**
  Row2:
  col1: safe (diag? |2-1|=1,|1-4|=3) → [4,1]
  col2: diagonal (|2-1|=1,|2-4|=2) → safe → [4,2]
  col3: diagonal (|2-1|=1,|3-4|=1) → prune
  col4: column conflict → prune → nodes [4,1], [4,2]
- **Continue to row3 and row4** pruning similarly. Final valid leaves (solutions) are: [2,4,1,3] and [3,1,4,2]

Drawing instructions:
Draw root. Level1: 4 nodes labeled "Q1=c1"…"Q1=c4". From each, draw branches to level2 as above (only safe nodes). From each level2 node, draw level3 safe placements, then level4. Mark pruned branches with ✗.

Other Important Question 3 – Explain minimax principle with example (tic-tac-toe). (4 marks)
Appeared in: S22 (Q5b, 03 marks), W22 (Q4b, 04 marks) – highest 4
Ans:
Minimax principle – A decision rule used in two-player zero-sum games (e.g., tic-tac-toe, chess). The maximizing player (MAX) tries to maximize the score; the minimizing player (MIN) tries to minimize it. The algorithm assumes both play optimally. At MAX's turn, it chooses the move that maximizes the minimum possible result (i.e., the worst-case best outcome). MIN chooses the move that minimizes the maximum result.

How it works:

1. Build game tree to a certain depth (or to terminal states).
2. Evaluate leaf nodes with a utility function (e.g., +1 for MAX win, -1 for MIN win, 0 for draw).
3. Propagate values upward:
    - MAX node = max(child values)
    - MIN node = min(child values)
4. MAX chooses move leading to the highest value.

Example – Tic-tac-toe (simplified):
Assume board state where MAX (X) has one move left to win.
text
Board:
X O _
X _ O
_ _ _
MAX places X at (0,2) to win. Leaf evaluation: +1 for MAX win. Minimax propagates that value.

Partial tree for a different scenario:
MAX turn at root. Three possible moves. MIN responds. Leaves evaluated.

MAX move	MIN best response	Leaf value
Move A	leads to draw	0
Move B	leads to loss	-1
Move C	leads to win	+1

MAX picks **Move C** (max of {0, -1, +1} = +1).
Real-world application: Chess engines, game AI (Alpha-Beta pruning improves minimax).

Other Important Question 4 – Explain Travelling Salesman Problem and Hamiltonian problem in backtracking context. (7 marks)
Appeared in: S25 (Q4a, 03 marks), W22 (Q5c, 07 marks) – highest 7
Ans:
Travelling Salesman Problem (TSP): Given n cities and distances between them, find the shortest possible route that visits each city exactly once and returns to the starting city (minimum-cost Hamiltonian cycle).
Hamiltonian problem (Hamiltonian cycle): Given a graph, find a cycle that visits each vertex exactly once (without regard to cost). Decision problem: existence of such a cycle.
Backtracking approach for both:
State space tree:

- Root: starting city (e.g., 1).
- Each level: next city to visit.
- Path from root to a node represents partial tour.
- Leaf: tour covering all vertices and returning to start.

Constraints for Hamiltonian cycle:

- No repeated vertices (except start at the end).
- Edge must exist between consecutive vertices.
- Final vertex must have edge back to start.

For TSP (optimization):
Add bounding function (branch-and-bound) to prune paths that cannot beat best tour found so far. Bounding estimate: current path cost + minimum possible remaining cost (e.g., sum of smallest outgoing edges from unvisited cities).
Backtracking algorithm for Hamiltonian cycle (decision):
text
```
Hamiltonian(curr, visited, path, start):
  if visited all vertices:
    if edge exists from curr to start:
      print path, return true
    return false
  for each neighbor v of curr not visited:
    path.append(v), mark visited
    if Hamiltonian(v, visited, path, start):
      return true
    backtrack (unmark, remove from path)
  return false
```
Complexity: O(n!) worst-case, but pruning helps.
TSP using branch-and-bound:

- Maintain best (shortest tour length found).
- At node with partial path of length currCost, compute lower bound = currCost + lowerBound(remaining).
- If bound ≥ best, prune.

Real-world application: Route planning, logistics, microchip manufacturing, DNA sequencing.

Other Important Question 5 – Explain knapsack problem using backtracking. (3 marks)
Appeared in: S24 (Q4a, 03 marks)
Ans:
0/1 Knapsack using backtracking – Explores all subsets of items, pruning when weight exceeds capacity or when optimistic bound shows cannot beat best value.
State space tree:

- Root: no item considered.
- Level i: decision for item i (take or skip).
- Left branch: include item i (if weight ≤ remaining capacity).
- Right branch: skip item i.

Backtracking algorithm:
text
```
KnapsackBT(i, profit, weight, capacity, best):
  if weight > capacity: return
  if i == n: best = max(best, profit); return
  // Include item i
  KnapsackBT(i+1, profit+v[i], weight+w[i], capacity, best)
  // Skip item i
  KnapsackBT(i+1, profit, weight, capacity, best)
```
Pruning (optional but recommended): Compute upper bound (e.g., fractional knapsack for remaining items). If bound ≤ best, prune.
Example: n=3, capacity=5, items (w,v): (2,3), (3,4), (4,5)

Backtracking explores:

- Include 1 (profit3,w2) → include2 (profit7,w5) → leaf (best=7)
- Then skip2, include3 (weight6>5) → skip, etc.
- Skip1 branch also explored.
  Optimal = 7 (items 1 and 2).

Complexity: $O(2^n)$ worst-case, but pruning improves.

Other Important Question 6 – Explain states, constraints, types of nodes, and bounding function in backtracking/branch-and-bound. (4 marks)
Appeared in: W22 (Q4b, 04 marks)
Ans:

Term	Definition
States	Partial solutions at a given point in the search. Represented as nodes in state space tree. Example: in N-Queens, state = columns occupied in first k rows.
Constraints	Rules that a valid solution must satisfy. Two types: **explicit constraints** (e.g., each queen in different row/col) and **implicit constraints** (e.g., no two queens on same diagonal).
Types of nodes	- **Live node:** Generated but not yet expanded. - **E-node (expanding node):** Currently being explored. - **Dead node:** Cannot lead to feasible solution (pruned). - **Solution node:** Represents a complete valid solution.
Bounding function	Used in branch-and-bound to prune nodes that cannot improve the best solution found so far. For maximization: compute an **upper bound** on the best possible from that node; if bound ≤ current best, prune. For minimization: compute lower bound.

Example – TSP bounding function:
At a node with partial tour cost c and unvisited set U, lower bound = c + (sum of minimum outgoing edges from each vertex in U)/2 (or simpler: sum of smallest edges from each unvisited vertex).
Real-world application: All optimization problems (TSP, knapsack, job scheduling) use bounding functions to prune search space.

****