AI

**✅ Comprehensive Viva & Theory Questions with Answers – N-Queens (CSP) & Related AI Topics (SPPU)**

**🔹 1. Core N-Queens / CSP Based Viva**

1. **What is a Constraint Satisfaction Problem?**  
   A CSP is a mathematical problem defined as a set of objects whose state must satisfy a number of constraints or limitations. Examples: Map coloring, N-Queens, Sudoku.
2. **How does backtracking work in the N-Queens problem?**  
   Backtracking places queens one by one in different columns and rows, backtracks when no safe placement is possible, and tries the next option recursively.
3. **Explain how branch and bound optimizes the search space in CSPs.**  
   Branch and Bound eliminates paths that cannot yield better solutions than already found ones by using bounds to prune the search tree.
4. **What is the time complexity of solving N-Queens using backtracking?**  
   Worst case is O(N!) as each queen can go into N positions. With optimizations, it can be reduced.
5. **How does the bounding function prune branches in the search tree?**  
   It evaluates if a partial solution can lead to a valid full solution. If not, that branch is cut off early.
6. **Can we solve the N-Queens problem using BFS or DFS? Why or why not?**  
   Technically yes using DFS with pruning. BFS is not memory-efficient here. Backtracking is preferred due to structured constraint checks.

**🔹 2. Search Strategy Concepts**

1. **Compare uninformed vs informed search strategies.**  
   Uninformed (e.g., BFS, DFS) use no domain knowledge; Informed (e.g., A\*) use heuristics to improve efficiency.
2. **Where do CSPs fit in AI problem taxonomy – informed or uninformed search?**  
   CSPs are usually solved using uninformed methods like backtracking, but heuristics can make them informed.
3. *What is the difference between DFS with backtracking and A algorithm?*\*  
   DFS explores depth; backtracking undoes decisions. A\* uses g(n) + h(n) to evaluate best path toward goal.
4. **Explain Greedy Search and how it differs from Branch and Bound.**  
   Greedy selects best option based on heuristic only (h(n)); Branch and Bound uses both cost so far and bound.
5. *Why is A complete and optimal? What role does the heuristic play?*\*  
   A\* is complete if h(n) is admissible (never overestimates). Heuristic guides the search efficiently to the goal.

**🔹 3. Heuristics & Optimization**

1. **What is a heuristic function?**  
   A heuristic estimates the cost from current node to goal. Used in informed search like A\*.
2. **Can N-Queens be solved using heuristic-based approach like Hill Climbing?**  
   Yes. Hill Climbing starts with one configuration and improves until a solution or local maxima. May get stuck.
3. **What is local search? How is it different from complete search?**  
   Local search evaluates one solution and its neighbors, not complete tree. It uses less memory but may miss global optimum.
4. **What problems occur in local search?**  
   Local maxima, plateaus, and ridges may trap the search with no improvement.

**🔹 4. Problem Characteristics**

1. **Explain characteristics of AI problems: single-agent vs multi-agent, deterministic vs stochastic.**  
   Single-agent involves one decision maker (e.g., puzzle solving); multi-agent includes opponents. Deterministic has predictable results; stochastic has uncertainty.
2. **Is N-Queens deterministic and single-agent? Justify.**  
   Yes. One agent places queens, and actions have predictable results.
3. **What is state space representation in CSP?**  
   It’s the set of all possible configurations (e.g., queen placements) with nodes as states and edges as actions.
4. **What kind of goal test is used in the N-Queens problem?**  
   All queens placed without conflicts in rows, columns, or diagonals.
5. **How do constraints affect the search tree in CSPs?**  
   Constraints reduce the number of valid branches at each level, leading to pruning and efficiency.

**🔹 5. Advanced CSP & AI Links**

1. **What are arc consistency, node consistency, and path consistency in CSPs?**

* Node consistency: Single variable satisfies unary constraints.
* Arc consistency: Every value in one variable has valid value in the other.
* Path consistency: Three or more variables satisfy constraints jointly.

1. **How would forward checking and constraint propagation work for N-Queens?**  
   Forward checking eliminates unsafe positions ahead of time. Constraint propagation updates remaining domains dynamically as values are assigned.
2. **Discuss how CSPs are solved in Prolog or declarative languages.**  
   Prolog uses rules and backtracking to assign values. Declarative nature fits well with CSP definition.
3. **Can neural networks solve CSP problems? If yes, how?**  
   Yes. For example, Hopfield networks or constraint embedding in loss function can approximate solutions.
4. **How does AI handle over-constrained problems?**  
   Through relaxation (soft constraints), weighted constraints, or optimization for best satisfaction rather than perfect.

**🔹 6. Real-world & Interdisciplinary Applications**

1. **Where are CSPs used in the real world?**  
   Scheduling, exam timetables, map coloring, sudoku solving, circuit design, and resource allocation.
2. **How does N-Queens relate to task scheduling?**  
   Tasks (queens) must be assigned to time slots (columns) without conflict (diagonals or rows).
3. **Can we use CSP techniques for Sudoku solving? How?**  
   Yes. Each cell is a variable with domain 1-9 and constraints on rows, columns, and boxes.
4. **How is N-Queens useful in compiler design or OS scheduling?**  
   It resembles register allocation and resource sharing problems where conflicts must be avoided.
5. **What role do search strategies play in game AI and planning?**  
   AI in games use Minimax, A\*, and heuristics for decision making, move planning, and state exploration.

**🔍 Additional Practical Concepts – Selection Sort and MST using Greedy**

**🧠 Selection Sort (Greedy Approach)**

* Select the smallest element and place it in the sorted portion iteratively.
* Greedy choice: selecting the minimum in each iteration.
* Time Complexity: O(n²); Space: O(1)
* Best fit for scenarios where memory is limited, and input size is small.

**🧠 Minimum Spanning Tree (MST – Greedy Strategy)**

* Use Kruskal's or Prim’s algorithm.
* Greedy choice: always select the lowest weight edge that doesn't form a cycle.
* Applications: Network design, circuit layout, clustering.
* Time Complexity: O(E log E) (Kruskal with Union-Find); O(V²) (Prim’s with adjacency matrix).

**🔹 5. DFS and BFS (Graph Search)**

**🔸 DFS (Depth First Search) – Recursive**

**Steps:**

1. Start from the initial node.
2. Mark it visited.
3. Recursively visit all unvisited neighbors.
4. Backtrack when no unvisited neighbors remain.

python

CopyEdit

def dfs(graph, node, visited):

if node not in visited:

print(node, end=' ')

visited.add(node)

for neighbor in graph[node]:

dfs(graph, neighbor, visited)

**🔸 BFS (Breadth First Search) – Queue**

**Steps:**

1. Start from the initial node.
2. Use a queue to store next nodes to visit.
3. Visit all neighbors level by level.

python

CopyEdit

from collections import deque

def bfs(graph, start):

visited = set()

queue = deque([start])

while queue:

node = queue.popleft()

if node not in visited:

print(node, end=' ')

visited.add(node)

queue.extend(graph[node])

**🔹 *6. A Algorithm for Maze Solver*\***

**Steps:**

1. Use Manhattan Distance as heuristic.
2. Use a priority queue for open nodes.
3. Expand the node with the lowest cost (f = g + h).
4. Keep track of the path using came\_from.

python

CopyEdit

from queue import PriorityQueue

def heuristic(a, b):

return abs(a[0] - b[0]) + abs(a[1] - b[1])

def a\_star(maze, start, goal):

open\_set = PriorityQueue()

open\_set.put((0, start))

came\_from = {}

g\_score = {start: 0}

while not open\_set.empty():

\_, current = open\_set.get()

if current == goal:

path = []

while current in came\_from:

path.append(current)

current = came\_from[current]

path.append(start)

return path[::-1]

for dx, dy in [(0,1),(1,0),(0,-1),(-1,0)]:

neighbor = (current[0]+dx, current[1]+dy)

if 0 <= neighbor[0] < len(maze) and 0 <= neighbor[1] < len(maze[0]) and maze[neighbor[0]][neighbor[1]] == 0:

temp\_g = g\_score[current] + 1

if neighbor not in g\_score or temp\_g < g\_score[neighbor]:

came\_from[neighbor] = current

g\_score[neighbor] = temp\_g

f = temp\_g + heuristic(neighbor, goal)

open\_set.put((f, neighbor))

**🔹 7. Selection Sort using Greedy**

**Steps:**

1. Traverse the array.
2. Find the minimum element in unsorted part.
3. Swap it with the first unsorted element.
4. Repeat until sorted.

python

CopyEdit

def selection\_sort(arr):

for i in range(len(arr)):

min\_index = i

for j in range(i+1, len(arr)):

if arr[j] < arr[min\_index]:

min\_index = j

arr[i], arr[min\_index] = arr[min\_index], arr[i]

return arr

**🔹 8. Minimum Spanning Tree (Kruskal’s Algorithm)**

**Steps:**

1. Sort all edges by weight.
2. Pick smallest edge that doesn’t form a cycle.
3. Use Union-Find to detect cycles.
4. Stop when MST has (V-1) edges.

python

CopyEdit

def find(parent, i):

if parent[i] == i:

return i

return find(parent, parent[i])

def union(parent, rank, x, y):

rootX = find(parent, x)

rootY = find(parent, y)

if rank[rootX] < rank[rootY]:

parent[rootX] = rootY

elif rank[rootX] > rank[rootY]:

parent[rootY] = rootX

else:

parent[rootY] = rootX

rank[rootX] += 1

def kruskal(n, edges):

edges.sort(key=lambda x: x[2])

parent = [i for i in range(n)]

rank = [0]\*n

mst = []

for u, v, w in edges:

if find(parent, u) != find(parent, v):

mst.append((u, v, w))

union(parent, rank, u, v)

return mst

**🔹 9. 4-Queens (Backtracking & Branch and Bound)**

**🔸 Backtracking Steps:**

1. Try placing a queen in each row of current column.
2. If it's safe, place and move to next column.
3. If not safe, backtrack.
4. Stop when all columns are filled.

python

CopyEdit

def is\_safe(board, row, col):

for i in range(col):

if board[i] == row or abs(board[i]-row) == abs(i-col):

return False

return True

def solve\_n\_queens\_bt(n):

def backtrack(col, board):

if col == n:

print(board)

return

for row in range(n):

if is\_safe(board, row, col):

board[col] = row

backtrack(col+1, board)

backtrack(0, [-1]\*n)

**🔸 Branch and Bound – Additional Step:**

* Use bounding function to check if current path can ever succeed.
* If it can't, don't explore further – this improves efficiency.

Would you like me to make a 1-page summary version of this for your viva sheet or notes?

**You said:**

What is manhattan distance

**ChatGPT said:**

**Manhattan Distance** (also called **L1 distance** or **Taxicab distance**) is a way to measure the distance between two points in a grid-based system like a maze, where movement is restricted to horizontal and vertical paths (not diagonal).

**📌 Formula:**

If you have two points:  
**Point A = (x₁, y₁)**  
**Point B = (x₂, y₂)**  
Then:

Manhattan Distance=∣x1−x2∣+∣y1−y2∣\text{Manhattan Distance} = |x₁ - x₂| + |y₁ - y₂|Manhattan Distance=∣x1​−x2​∣+∣y1​−y2​∣

**🔹 Example:**

From point **(1, 2)** to **(4, 5)**:

Distance=∣1−4∣+∣2−5∣=3+3=6\text{Distance} = |1 - 4| + |2 - 5| = 3 + 3 = 6Distance=∣1−4∣+∣2−5∣=3+3=6

**🔍 Why it's called "Manhattan"?**

Because it mimics how you move in a city like Manhattan, where roads are laid out in a grid—so you can’t cut diagonally across blocks, only go around using right-angle turns.

**✅ Where is it used?**

* **A\*** algorithm as a heuristic for grid-based pathfinding (like maze solving).
* **Puzzle solving** (e.g., 8-puzzle, 15-puzzle).
* **AI games** where movement is in four directions only.

**1. What is Artificial Intelligence?**

**Answer:**  
Artificial Intelligence is the simulation of human intelligence in machines that are programmed to think, learn, and make decisions like humans.

**2. What are the main types of AI?**

**Answer:**

1. **Narrow AI:** Performs a specific task (e.g., Siri, Google Maps).
2. **General AI:** Can perform any intellectual task like a human.
3. **Super AI:** Hypothetical AI that surpasses human intelligence.

**3. What is the difference between supervised and unsupervised learning?**

**Answer:**

* **Supervised:** Data is labeled (e.g., spam vs. not spam).
* **Unsupervised:** Data is not labeled; model finds patterns (e.g., clustering).

**4. What is the difference between BFS and DFS?**

**Answer:**

* **BFS (Breadth-First Search):** Explores level by level (queue-based).
* **DFS (Depth-First Search):** Explores deep first, then backtracks (stack/recursion).

**5. What is a heuristic?**

**Answer:**  
A heuristic is an estimate or rule of thumb used to guide search algorithms like A\* toward the goal efficiently.

***6. What is the difference between Greedy and A algorithm?*\***

**Answer:**

* **Greedy:** Uses only heuristic (h(n)) — faster, less accurate.
* **A\*:** Uses cost so far (g(n)) + heuristic (h(n)) — more accurate.

**7. What is a Constraint Satisfaction Problem (CSP)?**

**Answer:**  
A problem where the goal is to find a state that satisfies all constraints (e.g., N-Queens, Sudoku).

**8. What are examples of real-world applications of AI?**

**Answer:**

* Voice Assistants (e.g., Alexa)
* Self-driving cars
* Medical Diagnosis
* Chatbots
* Face Recognition

**9. What is a state space?**

**Answer:**  
The set of all possible states or configurations a system can be in.

**10. What is the role of Manhattan Distance in AI?**

**Answer:**  
Used as a heuristic in A\* algorithm for grid-based pathfinding; it calculates cost without diagonal moves.

Would you like another set of *advanced but still safe-to-answer questions* for preparation?

**You said:**

Any difficult one's left?

**ChatGPT said:**

Yes, here are some **difficult or higher-level viva questions** that an external might ask to test **conceptual depth**—especially if they want to see whether you truly understand AI beyond just coding:

**🔍 Advanced AI Viva Questions & Answers**

***1. Why is A both complete and optimal?*\***

**Answer:**  
A\* is complete because it explores all possible paths if needed. It is optimal if the heuristic used is **admissible** (never overestimates) and **consistent**, ensuring the best path is found.

**2. Explain the difference between local search and global search algorithms.**

**Answer:**

* **Local search** (e.g., Hill Climbing) explores only the current state and its neighbors; low memory use.
* **Global search** (e.g., DFS, BFS) explores the full search space; higher memory and better completeness.

**3. How do you handle over-constrained CSPs?**

**Answer:**  
By using **soft constraints** (not all must be satisfied), **prioritizing constraints**, or applying **optimization techniques** like Min-Conflicts heuristic.

**4. What’s the difference between Backtracking and Branch & Bound?**

**Answer:**

* **Backtracking** explores every possibility and undoes decisions when constraints are violated.
* **Branch & Bound** uses bounds (cost limits) to avoid exploring paths that can’t give better solutions.

**5. In which cases does BFS fail or become inefficient?**

**Answer:**  
When the **search space is large or infinite**, BFS becomes memory-intensive due to storing all levels of nodes.

***6. What is the significance of admissible and consistent heuristics in A?*\***

**Answer:**

* **Admissible:** Heuristic never overestimates → ensures optimality.
* **Consistent:** Heuristic value decreases along a path → ensures no reprocessing of nodes.

**7. Can you convert DFS or BFS into informed search algorithms?**

**Answer:**  
Not directly. But they can be modified using heuristics (e.g., **Best-First Search** is like Greedy BFS).

**8. What are real-world cases where greedy algorithms fail?**

**Answer:**  
In problems like **Traveling Salesman**, greedy may not find the shortest route due to shortsighted decisions.

**9. How does forward checking help in CSPs?**

**Answer:**  
It eliminates invalid future values immediately after assigning a variable, preventing dead-end paths early.

**10. How does AI differ from traditional programming?**

**Answer:**  
Traditional programming uses fixed rules; AI learns from data, adapts, and deals with uncertainty or incomplete knowledge.

CC

**✅ 1. Amazon EC2**

**🔸 What is EC2?**

EC2 (Elastic Compute Cloud) is Amazon Web Services' IaaS offering that provides **resizable virtual machines** (instances) in the cloud. It allows you to run applications just like on a physical server but hosted on AWS.

**🔸 What are AMIs?**

AMI (Amazon Machine Image) is a pre-configured **template containing an OS and application software** used to launch EC2 instances. It serves as a blueprint for instance creation.

**🔸 Difference between Security Group and Network ACL?**

| **Feature** | **Security Group** | **Network ACL** |
| --- | --- | --- |
| Type | Virtual firewall at **instance** level | Firewall at **subnet** level |
| Rules Direction | **Stateful** (return traffic allowed automatically) | **Stateless** (return traffic must be explicitly allowed) |
| Applies To | EC2 Instances | Subnets |
| Rules Type | Only **allow** rules | Allow and deny rules |

**🔸 Steps to Make EC2 Host a Public Website:**

1. Launch EC2 instance (Ubuntu).
2. Configure **security group**: allow HTTP (80), HTTPS (443), SSH (22).
3. Connect via SSH using .pem file.
4. Install Apache:

bash

CopyEdit

sudo apt update

sudo apt install apache2

1. Upload HTML file to /var/www/html/index.html.
2. Open browser → enter EC2’s **public IPv4**.

**✅ 2. Google App Engine (GAE)**

**🔸 What file defines app config?**

The app.yaml file. It declares:

* Runtime (Python, Java, etc.)
* Instance class
* Entry point
* Environment variables
* Scaling settings

**🔸 What is instance class in GAE?**

It defines **CPU, memory, and pricing tier**. Examples:

* F1, F2 (Standard)
* B1, B2 (Flexible)  
  Choosing a class affects performance and cost.

**🔸 Difference: GAE Standard vs GAE Flexible**

| **Feature** | **Standard** | **Flexible** |
| --- | --- | --- |
| Scaling | Automatic | Manual or automatic |
| Startup Time | Very fast | Slower |
| Customization | Limited | Full customization via Docker |
| Supported Runtimes | Only selected (e.g. Python, Java) | Any (via Docker) |

**🔸 Auto-scaling vs Manual-scaling**

* **Auto-scaling**: Adds/removes instances based on traffic.
* **Manual-scaling**: Fixed number of instances, user-defined.
* Auto-scaling is cost-efficient and dynamic.

**✅ 3. Salesforce – Apex**

**🔸 What is Apex? How is it different from Java?**

Apex is Salesforce’s **object-oriented programming language**, similar to Java. It is tightly integrated with the Salesforce platform.  
Differences:

* Apex is optimized for database operations (SOQL).
* Runs on Salesforce cloud, not standalone.
* Enforces **Governor Limits**.
* Designed for **multitenancy**.

**🔸 What are Governor Limits?**

Governor Limits are runtime limits Salesforce enforces to ensure no single user can monopolize shared cloud resources.  
Examples:

* Max SOQL queries per transaction: 100
* Max DML statements: 150
* Max CPU time: 10,000 ms

**🔸 What is SOQL vs SOSL?**

| **Feature** | **SOQL (Salesforce Object Query Language)** | **SOSL (Salesforce Object Search Language)** |
| --- | --- | --- |
| Use Case | Query one object/table at a time | Search multiple objects at once |
| Similar To | SQL SELECT | Full-text search |
| Syntax | SELECT Name FROM Account | FIND 'John' IN ALL FIELDS |

**🔸 Can Apex run asynchronously?**

Yes. Apex supports asynchronous processing via:

* **Future methods**
* **Queueable Apex**
* **Batch Apex**
* **Scheduled Apex**  
  Used for long-running jobs like bulk processing.

**✅ 4. Salesforce – App Development**

**🔸 Lookup vs Master-Detail?**

| **Feature** | **Lookup Relationship** | **Master-Detail Relationship** |
| --- | --- | --- |
| Ownership | Child can exist without parent | Child is deleted with parent |
| Sharing | Separate | Inherited from master |
| Roll-up Summary | Not supported | Supported |
| Required Field | Optional | Mandatory |

**🔸 What is Lightning Record Page?**

It’s a **custom UI layout** for a specific Salesforce object, built using drag-and-drop **Lightning App Builder**. You can:

* Add components
* Set visibility rules
* Customize user experience without coding

**🔸 What’s the use of Process Builder?**

It is a **point-and-click automation tool** in Salesforce that can:

* Create records
* Send emails
* Launch flows
* Update fields  
  All based on rule conditions.

**🔸 How do dashboards update in real time?**

Dashboards are refreshed **manually or scheduled**. Real-time updates are **not automatic** unless built using Lightning components with live data refresh (e.g., using LWC or push topics via Streaming API).

**☁️ Cloud Computing Core Concepts**

**🔹 What is Cloud Computing?**

Cloud computing is the on-demand delivery of IT resources (compute, storage, databases, etc.) over the Internet with pay-as-you-go pricing.

**🔹 Cloud Models:**

| **Model** | **Description** | **Example** |
| --- | --- | --- |
| IaaS | Provides virtualized hardware over the internet | Amazon EC2 |
| PaaS | Platform for developers to build & deploy apps | Google App Engine |
| SaaS | Fully functional software on cloud | Salesforce CRM, Gmail |

**🔸 Comparison: IaaS vs PaaS vs SaaS**

| **Feature** | **IaaS** | **PaaS** | **SaaS** |
| --- | --- | --- | --- |
| Control | Full control over OS & software | Focus on app logic only | Use software via browser |
| Example Providers | AWS EC2, Azure VM | Google App Engine, Heroku | Salesforce, Gmail |
| Infrastructure | User manages | Managed by provider | Fully managed |
| Flexibility | High | Medium | Low |
| Dev Skill Needed | Advanced (DevOps) | Moderate (Dev/Build) | Minimal (End-user config) |

**📦 EC2 (Elastic Compute Cloud) – IaaS**

* Virtual server in AWS cloud.
* You manage: OS, runtime, apps.
* Can scale vertically (bigger instance) or horizontally (more instances).
* **Elastic IPs** allow static public IPs for your server.
* Often paired with **EBS (Elastic Block Storage)** for persistent storage.

**☁️ Google App Engine – PaaS**

* Automatically manages:
  + Scaling
  + Patching
  + Load balancing
* Just push code → app is live on cloud.
* Built-in security, monitoring, and version control.
* Ideal for rapid web or REST API development.

**🌀 Salesforce Platform – SaaS + PaaS**

* Offers CRM, custom objects, reporting.
* **Apex** = server-side logic.
* **SOQL/SOSL** = querying Salesforce data.
* **Trigger** = automatic logic on insert/update/delete.
* **Lightning Builder** = drag-and-drop UI designer.

**💡 Important Viva Questions by Practical**

**1. Amazon EC2**

* What is EC2?
* What are AMIs?
* Difference between Security Group and Network ACL?
* Steps to make EC2 host a public website?

**2. Google App Engine**

* What file defines app config? (→ app.yaml)
* What is instance class in GAE?
* Difference: GAE Standard vs GAE Flexible?
* Auto-scaling vs Manual-scaling?

**3. Salesforce – Apex**

* What is Apex? How is it different from Java?
* What are Governor Limits?
* What is SOQL vs SOSL?
* Can Apex run asynchronously?

**4. Salesforce – App Dev**

* Lookup vs Master-Detail?
* What is Lightning Record Page?
* What’s the use of Process Builder?
* How do dashboards update in real time?

**🚀 Deployment Architecture Diagrams (Simplified)**

**EC2 Web Hosting:**

scss

CopyEdit

User

↓

[Browser]

↓ (HTTP)

[EC2 Instance - Apache]

↑ SSH

Developer (via .pem key)

**GAE Python App:**

csharp

CopyEdit

User

↓

[GAE Managed Instance]

↓

[App (Flask/Django)]

↓

Google Cloud Storage / Firestore (optional)

**Salesforce App:**

scss

CopyEdit

User

↓ (Web UI)

[Lightning Page]

↓

[Apex Logic]

↓

[Salesforce DB (Object Data)]

**🧠 Must-Prepare Real-World Use Cases**

| **Platform** | **Use Case Example** |
| --- | --- |
| EC2 | Host a Node.js app with MongoDB |
| GAE | Deploy ML inference API in Flask |
| Salesforce | Event CRM for college festival with automated reminders |