**Part 1: Understanding Search Strategies (Conceptual Questions)**

**Task 1: Definitions and Characteristics**

1. Define **Uninformed Search** and provide two examples.
2. Define **Informed Search** and provide two.
3. Compare **Breadth-First Search (BFS)** and **Depth-First Search (DFS)** in terms of:
   * Completeness
   * Optimality
   * Time Complexity
   * Space Complexity
4. Why is *A Search*\* considered better than **Greedy Best-First Search (GBFS)**?



**1. Define Uninformed Search and provide two examples.**

**Uninformed Search** (also called *blind search*) refers to search strategies that have **no additional information** about states beyond what is provided in the problem definition. These algorithms **do not use any domain-specific knowledge** (heuristics).

**Examples:**

* **Breadth-First Search (BFS)**: Explores all nodes at the present depth before moving to the next level.
* **Depth-First Search (DFS)**: Explores as far as possible along each branch before backtracking.

**2. Define Informed Search and provide two examples.**

**Informed Search** (also called *heuristic search*) uses **problem-specific knowledge** (heuristics) to make more efficient decisions during the search. These algorithms aim to reduce the search time by estimating how close a state is to the goal.

**Examples:**

* **A\* Search**: Uses both actual cost from the start and estimated cost to the goal (f(n) = g(n) + h(n)).
* **Greedy Best-First Search (GBFS)**: Chooses the node that appears to be closest to the goal (f(n) = h(n))

**3. Compare Breadth-First Search (BFS) and Depth-First Search (DFS):**

| **Property** | **Breadth-First Search (BFS)** | **Depth-First Search (DFS)** |
| --- | --- | --- |
| **Completeness** | Yes (if branching factor is finite) | No (may get stuck in infinite branch) |
| **Optimality** | Yes (if all step costs are equal) | No (does not guarantee shortest path) |
| **Time Complexity** | O(b^d) | O(b^d) |
| **Space Complexity** | O(b^d) | O(b \* d) |

Where:  
b = branching factor  
d = depth of the shallowest goal

**4. Why is A\* Search considered better than Greedy Best-First Search (GBFS)?**

A\* Search is considered better than GBFS because:

* **A\* is both complete and optimal** (if h(n) is admissible and consistent), while GBFS is **not guaranteed to find the optimal path**.
* A\* considers both:
  + **Cost so far (g(n))** and
  + **Estimated cost to goal (h(n))**  
    Whereas GBFS considers only the heuristic (h(n)), which can lead to poor choices (e.g., getting stuck in local minima).
* **A\*** balances **exploration** and **exploitation**, making it more robust across different problem types.

**Task 2: Problem-Solving Scenarios**

For each scenario below, suggest the most appropriate search strategy (from the allowed list) and justify your choice:

1. Finding the shortest path in a grid.
2. Solving an 8-puzzle problem with a heuristic (Research based).
3. Exploring all possible moves in a game tree without heuristic knowledge (Research based).



**1. Finding the shortest path in a grid**

* **Recommended Search Strategy:** **A\* Search**
* **Justification:**
  + The goal is to find the *shortest path*, so an *optimal* strategy is required.
  + A\* combines both the actual cost (g(n)) and an estimated cost to the goal (h(n)), making it **both complete and optimal** (as long as the heuristic is admissible).
  + Common heuristics like **Manhattan distance** work well on grid-based problems.
  + It is more efficient than BFS as it uses heuristics to guide the search.

**2. Solving an 8-puzzle problem with a heuristic (Research based)**

* **Recommended Search Strategy:** **A\* Search**
* **Justification:**
  + The 8-puzzle is a classic problem where the goal is to reach a specific configuration using valid tile moves.
  + A\* is ideal because it guarantees optimality if the heuristic is admissible.
  + Widely used **heuristics** include:
    - **Misplaced tiles**: Counts the number of tiles not in their correct position.
    - **Manhattan distance**: Sum of the distances each tile is from its goal position.
  + These heuristics help reduce the number of states explored compared to uninformed searches.

**3. Exploring all possible moves in a game tree without heuristic knowledge (Research based)**

* **Recommended Search Strategy:** **Depth-First Search (DFS)** or **Minimax (with DFS traversal)**
* **Justification:**
  + Since there is **no heuristic knowledge**, informed strategies like A\* or GBFS cannot be used.
  + **DFS** is effective in **exploring all possible paths** in large game trees because it uses less memory than BFS.
  + If this is a two-player adversarial game (e.g., chess), a **Minimax algorithm** can be used with **DFS-style traversal** to evaluate all possible move sequences up to a certain depth.
  + DFS is preferred over BFS in game trees due to space constraints and the depth-first nature of most game strategies.

**Part 2: Implementation-Based Tasks**

**Task 3: Implementing BFS and DFS**

1. Write a Python program to implement **BFS** for finding the shortest path in a graph.
2. Modify the same program to implement **DFS** and compare the paths obtained.
3. Analyze which algorithm is more efficient for this problem and why (Research based).

**Task 4: Implementing Greedy Best-First Search (GBFS) and A**\*

1. Implement **Greedy Best-First Search** using a simple heuristic (e.g., Manhattan distance for a grid).
2. Extend the program to implement *A Search*\* with the same heuristic.
3. Compare the number of nodes explored by GBFS and A\* in a given maze.



**Task 3 :**

**3. Analysis: Which is more efficient?**

* **BFS is more efficient** for finding the *shortest path*, especially in unweighted graphs.
* **DFS** may find a path faster in some cases but **not necessarily the shortest**, and it risks going deep into irrelevant paths.
* BFS guarantees the **shortest path** (in terms of number of steps), making it ideal for such problems.

**Task 4 :**

**3. Compare the number of nodes explored by GBFS and A\* in a given maze.**

| **Metric** | **Greedy Best-First Search** | **A\* Search** |
| --- | --- | --- |
| **Nodes Explored** | Fewer (in many cases) | More |
| **Path Optimality** | Not guaranteed | Guaranteed |
| **Use Case** | Fast approximate solutions | Reliable optimal solutions |

**Part 3: Research and Analysis**

**Task 5: Case Study on Real-World Applications (Research based)**

Research and write a short report (200-300 words) on:

* **One real-world application of BFS/DFS** (e.g., web crawling, social networks).
* **One real-world application of A**\* (e.g., robotics, GPS navigation).
* Discuss why an informed search is preferred over an uninformed search in these cases.

**Task 6: Limitations and Trade-offs**

1. What are the main limitations of **Greedy Best-First Search**?
2. Under what conditions does **A**\* fail to find an optimal solution?
3. Why might **DFS** be impractical for large search spaces despite its low memory usage?

**Task 5: Case Study on Real-World Applications (Research Based)**

**BFS/DFS Application: Web Crawling**

Web crawlers used by search engines like Google employ **Breadth-First Search (BFS)** to systematically explore and index web pages. Starting from a seed URL, BFS visits all immediate links (children) before moving to the next level. This ensures that higher-priority, more frequently linked pages are indexed first. **DFS**, on the other hand, may be used when a crawler is exploring a specific domain deeply before switching to others, but it risks going too deep into irrelevant pages.

**A\* Application: GPS Navigation Systems**

**A\*** is widely used in GPS-based navigation (e.g., Google Maps, Waze). It finds the shortest path between two locations by combining the actual cost of the path already traveled (g(n)) and an estimate of the remaining distance (h(n)) using heuristics like **Euclidean** or **road-network-based** distance. This ensures the suggested route is both **efficient and realistic**, considering known constraints like traffic, road closures, and one-ways.

**Why Informed Search is Preferred**

Informed searches like A\* are preferred in these real-world applications because they significantly **reduce the search space** and **improve efficiency**. While BFS/DFS can work for smaller or structured data, they become **infeasible** when the number of possible paths or nodes grows exponentially. A\*, guided by heuristics, focuses only on promising paths, making it more practical and scalable for real-time systems.

**Task 6: Limitations and Trade-offs**

**1. Limitations of Greedy Best-First Search (GBFS):**

* **Not Optimal**: GBFS can choose paths that look good in the short term but lead to dead ends.
* **Incomplete**: May fail in infinite search spaces or when heuristics are misleading.
* **Heuristic-Dependent**: Poor heuristic quality severely affects performance.
* **No Path Cost Consideration**: It ignores actual path cost (g(n)), relying only on the estimated cost (h(n)).

**2. When A\* Fails to Find Optimal Solution:**

* If the **heuristic is not admissible** (i.e., it overestimates the true cost), A\* can return **non-optimal** paths.
* If the **heuristic is not consistent (monotonic)**, A\* may revisit nodes and lose efficiency or correctness.
* In **resource-constrained systems** (e.g., real-time navigation), A\* may be approximated or interrupted before completion, leading to suboptimal results.

**3. Why DFS Is Impractical for Large Search Spaces:**

* **Can go infinitely deep** in infinite trees/graphs without reaching a goal (especially in cyclic or deep structures).
* **No guarantee of shortest path**, which can be problematic in route planning or optimization.
* Though **memory-efficient** (O(b\*d)), DFS’s blind depth-first nature often leads to **wasted computation** and poor decision-making in vast or complex spaces.