**Task 1: Definitions and Characteristics**

1. **Uninformed Search:**
   * It doesn't use any domain-specific knowledge.
   * Searches blindly through the problem space.
   * **Examples:**  
     a) Breadth-First Search (BFS)  
     b) Depth-First Search (DFS)
2. **Informed Search:**
   * Uses heuristics (extra knowledge) to find solutions faster.
   * More efficient than uninformed strategies.
   * **Examples:**  
     a) A\* Search  
     b) Greedy Best-First Search (GBFS)
3. **Comparison: BFS vs DFS**

| **Feature** | **BFS** | **DFS** |
| --- | --- | --- |
| Completeness | Yes (if solution exists) | No (may go in infinite loop) |
| Optimality | Yes (if cost is uniform) | No |
| Time Complexity | O(b^d) | O(b^d) |
| Space Complexity | O(b^d) (stores all nodes at level) | O(b\*d) (only path in memory) |

(*b = branching factor, d = depth of solution*)

1. *Why is A better than GBFS?*\*
   * A\* considers both path cost (g(n)) and heuristic (h(n)), giving optimal results.
   * GBFS only considers h(n) (can lead to suboptimal path).
   * A\* is **complete and optimal** if the heuristic is admissible.

**Task 2: Problem-Solving Scenarios**

1. **Finding shortest path in a grid:**  
   **Use:** A\*  
   **Why:** A\* uses heuristics (e.g., Manhattan distance) to efficiently find the shortest path.
2. **Solving 8-puzzle with heuristic:**  
   **Use:** A\*  
   **Why:** Heuristics like misplaced tiles or Manhattan distance help A\* solve efficiently.
3. **Exploring all moves in a game tree without heuristic:**  
   **Use:** DFS  
   **Why:** Without heuristic, DFS is better for deep exploration and memory efficient.

**Part 2: Implementation-Based Tasks**

**Task 3: BFS and DFS in Python**

**1. BFS Implementation**

from collections import deque

def bfs(graph, start, goal):

visited = set()

queue = deque([[start]])

while queue:

path = queue.popleft()

node = path[-1]

if node == goal:

return path

if node not in visited:

visited.add(node)

for neighbor in graph[node]:

new\_path = list(path)

new\_path.append(neighbor)

queue.append(new\_path)

return None

**2. DFS Implementation (Same Structure)**

def dfs(graph, start, goal):

visited = set()

stack = [[start]]

while stack:

path = stack.pop()

node = path[-1]

if node == goal:

return path

if node not in visited:

visited.add(node)

for neighbor in graph[node]:

new\_path = list(path)

new\_path.append(neighbor)

stack.append(new\_path)

return None

**3. Analysis:**

* **BFS** finds the shortest path in an unweighted graph.
* **DFS** may find longer paths or get stuck in deep trees.
* **BFS is better** for shortest path, but **DFS uses less memory**.

***Task 4: GBFS and A Implementation*\***

**1. Greedy Best-First Search**

python

import heapq

def gbfs(grid, start, goal, heuristic):

open\_list = [(heuristic(start, goal), start)]

came\_from = {}

visited = set()

while open\_list:

\_, current = heapq.heappop(open\_list)

if current == goal:

path = []

while current in came\_from:

path.append(current)

current = came\_from[current]

path.append(start)

return path[::-1]

visited.add(current)

for neighbor in get\_neighbors(grid, current):

if neighbor not in visited:

visited.add(neighbor)

came\_from[neighbor] = current

heapq.heappush(open\_list, (heuristic(neighbor, goal), neighbor))

return None

**2. *A Search*\***

def astar(grid, start, goal, heuristic):

open\_list = [(0 + heuristic(start, goal), 0, start)]

came\_from = {}

cost\_so\_far = {start: 0}

while open\_list:

\_, g, current = heapq.heappop(open\_list)

if current == goal:

path = []

while current in came\_from:

path.append(current)

current = came\_from[current]

path.append(start)

return path[::-1]

for neighbor in get\_neighbors(grid, current):

new\_cost = g + 1

if neighbor not in cost\_so\_far or new\_cost < cost\_so\_far[neighbor]:

cost\_so\_far[neighbor] = new\_cost

priority = new\_cost + heuristic(neighbor, goal)

heapq.heappush(open\_list, (priority, new\_cost, neighbor))

came\_from[neighbor] = current

return None

*(Assume get\_neighbors and heuristic functions are defined.)*

**3. Comparison:**

* **GBFS** explores fewer nodes but may miss optimal path.
* **A**\* explores more nodes but **guarantees shortest path**.

**Part 3: Research and Analysis**

**Task 5: Real-World Applications (200–300 words)**

**BFS/DFS in Web Crawling:**  
BFS is used in web crawling to explore links level by level. It ensures that all pages reachable within fewer clicks are visited before going deeper. It helps search engines index popular pages first. DFS may be used for deep crawling when specific nested data is needed. However, DFS can get stuck if cycles are not handled.

*A in GPS Navigation:*\*  
A\* is widely used in GPS systems to find the shortest and fastest route between two points. It combines actual travel cost and heuristic (like straight-line distance). This makes it efficient and accurate for real-time navigation, especially in road networks.

**Why Informed Search is Preferred:**  
In real-world problems like GPS or robotics, time and resources are limited. Informed search like A\* reduces unnecessary exploration and provides faster, better results using domain knowledge.

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

1. **Limitations of GBFS:**
   * Not optimal.
   * Can get stuck in local minima.
   * Ignores actual path cost.
2. *When A fails to be optimal:*\*
   * If the heuristic is not **admissible** (overestimates cost).
   * If the heuristic is **inconsistent**.
3. **Why DFS is impractical in large spaces:**
   * May go deep without finding the goal.
   * No guarantee of finding the shortest path.
   * Can get stuck in cycles if not handled.