LP2 Viva Qs Practical 1 - 6

🔹 Depth First Search (DFS) – Recursive on Undirected Graph

Q1. What is DFS and how does it work in a recursive manner?
A1. DFS explores each branch fully before backtracking, using recursion to visit unvisited neighbors.

Q2. How do you handle cycles in an undirected graph during DFS?
A2. By marking visited nodes in a boolean array or set.

Q3. What data structure is used to keep track of visited nodes in DFS?
A3. A `visited[]` array or a hash set.

Q4. How does the recursive call stack behave in DFS?
A4. Each recursive call adds to the stack until a leaf node is reached.

Q5. Can DFS be used to detect connected components? How?
A5. Yes, run DFS from each unvisited node and count the runs.

Q6. How would you modify DFS to find the path between two nodes?
A6. Use a parent map or list to track the path during recursion.

🔹 Breadth First Search (BFS) – Recursive on Undirected Graph

Q1. What is the primary difference between BFS and DFS in terms of traversal?
A1. BFS explores level-by-level; DFS goes deep first.

Q2. Why is BFS usually implemented iteratively, and how do you handle it recursively?
A2. BFS uses a queue; recursive BFS can use a helper function that processes the queue.

Q3. How do you implement BFS recursively without an explicit queue?
A3. Difficult and inefficient; simulate a queue with recursive calls.

Q4. What problems may arise with recursive BFS in terms of performance?
A4. Stack overflow or poor efficiency due to call overhead.

Q5. Can BFS be used to find the shortest path in an unweighted graph? How?
A5. Yes, because it finds the shortest path by levels.

Q6. Compare BFS and DFS in terms of space complexity and use cases.
A6. BFS uses more space (queue); BFS is better for shortest path, DFS for deep search.

🔹 *A Algorithm for 8-Queens Problem**

*Q1. What is the state representation for the 8-Queens problem in A?**
A1. Each state is a partial board configuration.

*Q2. What is the heuristic function used in A for 8-Queens?**
A2. Number of non-attacking pairs or conflicts.

*Q3. Why is A not commonly used for 8-Queens, and what makes it suitable/unsuitable?**
A3. The problem is constraint-based, not path-finding, so backtracking is better.

Q4. How do you generate successor states in the 8-Queens problem?
A4. Place a queen in the next row, try all safe columns.

*Q5. How does A ensure the optimal solution for constraint satisfaction problems like 8-Queens?**
A5. It explores lowest-cost states first using `f(n) = g(n) + h(n)`.

🔹 *A Algorithm for 8-Puzzle Problem**

Q1. What is the state space representation for the 8-puzzle?
A1. A 3x3 grid representing tile positions.

*Q2. What are admissible heuristics used in A for 8-puzzle?**
A2. Manhattan distance or misplaced tiles count.

*Q3. How does the A algorithm ensure completeness and optimality?**
A3. By using admissible heuristics and exploring lowest total cost paths first.

*Q4. How do you detect repeated states in A search?**
A4. Use a hash set or visited map to store explored states.

*Q5. How is priority queue used in A and what does it store?**
A5. It stores states with their `f(n)` cost; pops the lowest one.

*Q6. What is the time and space complexity of A in worst case?**
A6. Exponential: O(b^d), where `b` is branching factor and `d` is depth.

🔹 Greedy – Selection Sort & Minimum Spanning Tree

Q1. How does the greedy strategy apply in Selection Sort?
A1. Selects the minimum element and puts it at the beginning in each pass.

Q2. What makes a problem suitable for a greedy approach?
A2. If it has the **greedy choice property** and **optimal substructure**.

Q3. What is the difference between greedy and dynamic programming?
A3. Greedy makes locally optimal choices; DP considers all possibilities.

Q4. Explain why greedy works for MST but not for shortest path in general graphs.
A4. MST doesn't depend on path continuity; shortest path may have side effects from local choices.

Q5. Compare Selection Sort and Insertion Sort in terms of greedy strategy.
A5. Selection Sort is greedy; Insertion Sort is more adaptive to already sorted data.

🔹 Greedy – Dijkstra's & Job Scheduling

Q1. Why is Dijkstra's algorithm considered greedy?
A1. It picks the node with the smallest tentative distance each time.

Q2. How do you differentiate between Dijkstra's and Bellman-Ford in terms of applicability?
A2. Dijkstra doesn't handle negative weights; Bellman-Ford does.

Q3. What is the time complexity of Dijkstra's algorithm using a min-heap?
A3. O((V + E) log V)

Q4. Explain the greedy approach for Job Scheduling with deadlines and profits.
A4. Sort jobs by profit, schedule each in the latest possible slot before deadline.

Q5. What happens if jobs are not sorted in Job Scheduling? Will greedy still work?
A5. No, it may miss higher-profit jobs; sorting by profit is essential.

🔹 Greedy – Prim's & Kruskal's MST

Q1. Explain the basic steps of Prim's algorithm.
A1. Start from a node, add the minimum edge connecting visited to unvisited nodes.

Q2. How does Kruskal's algorithm ensure no cycles are formed?
A2. By checking disjoint sets before adding an edge.

Q3. What data structure is used in Kruskal's algorithm to detect cycles?
A3. Union-Find (Disjoint Set Union - DSU).

Q4. When is Kruskal's preferred over Prim's and vice versa?
A4. Kruskal's is better for sparse graphs; Prim's for dense graphs.

Q5. What is the role of union-find in Kruskal's algorithm?
A5. To check whether two nodes are in the same set (to prevent cycles).

🔹 Greedy – Kruskal's vs Dijkstra's

Q1. Both algorithms are greedy – how do their goals differ?
A1. Kruskal's finds MST; Dijkstra's finds shortest paths.

Q2. Can Dijkstra's be used to build a MST? Why or why not?
A2. No, because Dijkstra focuses on shortest path, not spanning all nodes.

Q3. Why does Dijkstra's algorithm fail with negative weights?
A3. It assumes once a node is visited with shortest distance, it won't change — which is false with negatives.

Q4. Can you implement Dijkstra's using BFS when all edge weights are the same?
A4. Yes, because all weights are equal — BFS gives shortest paths in that case.

Q5. Explain the priority queue role in both Kruskal's and Dijkstra's.
A5. In Kruskal's, it's used for sorting edges by weight; in Dijkstra's, for picking the node with least distance.

Here's the **complete short answer** on your requested points:

🔹 Greedy Algorithm – Time Complexity

- **Depends on the problem**, but common cases:
    - **Selection Sort**: $O(n^2)$
    - **Kruskal's MST**: O(E log E) (because edges are sorted)
    - **Prim's MST (with Min-Heap)**: O(E log V)

- **Dijkstra's (with Min-Heap)**: O((V + E) log V)
- **Job Scheduling**: O(n log n) (due to sorting)

🔹 Greedy vs A* Algorithm – Differences

Feature	Greedy Search	A* Search
Strategy	Chooses node with lowest **heuristic (h(n))**	Chooses node with lowest **f(n) = g(n) + h(n)**
Cost Consideration	Ignores past cost (g(n))	Considers both cost so far and estimated cost
Optimality	Not guaranteed	Guaranteed if heuristic is admissible
Completeness	Not always complete	Complete if heuristic is admissible
Example Use	Job Scheduling, MST	Shortest path, puzzle-solving

---

🔹 Greedy Search – Heuristic Function

- **Greedy uses only `h(n)`** – the estimated cost to reach the goal from node `n`.
- Example in 8-puzzle:
  `h(n) = number of misplaced tiles` or `h(n) = Manhattan distance`
- **It does not consider** the cost already spent (`g(n)`), unlike A*.
