

ABBOTTABAD UNIVERSITY OF SCIENCE AND TECHNOLGY

SUBMITTED BY: SANAULLAH BAIG

SUBMITTED TO: MR JAMAL ABDUL AHAD

SECTION: C

ROLL NO: 12384

ASSIGNMENT:5

SUBJECT: DATA STRUCTURES AND ALGORTHMS

Q1: What are the predominant representational structures commonly used to capture connections in a graph, and what distinctions can be observed in their visual representations? Additionally, how do these chosen structures impact the conceptual understanding of the graph's relationships?

**Answer**:

Common Representations: The two predominant structures are adjacency matrices and adjacency lists.

Adjacency Matrices: Visualized as a square matrix where rows and columns represent vertices. A cell contains a value (e.g., 1) if an edge exists; otherwise, it's 0.

Adjacency Lists: Visualized as a collection of lists, one for each vertex, containing its adjacent vertices.

Impact on Conceptual Understanding:

Adjacency Matrices: Suitable for dense graphs. Easy to check connectivity and find neighbors. Inefficient for sparse graphs due to wasted space.

Adjacency Lists: Efficient for sparse graphs. Memoryfriendly but may be slower for edge lookups.

Q2: Explore the structural characteristics that define a graph as a tree and elucidate the criteria for differentiating between a graph and a tree structure.

**Answer:**

Structural Characteristics of a Tree:

Acyclic: No cycles or loops.

Connected: Every pair of vertices is connected.

Single Root: A designated starting vertex (root).

Hierarchical: Vertices are organized in levels.

Differentiating Graphs and Trees:

Cyclic Structure: Trees are acyclic; graphs can have cycles.

Connectivity: Graphs may not be fully connected; trees are always fully connected.

Rooted Hierarchy: Trees have a single root; graphs may have multiple components.

Q3: How does the efficiency of bubble sort change under different scenarios for an array of size n? Explore its performance in:

***The bestcase scenario is when the array is presorted.***

An averagecase scenario with a randomly arranged array.

The worstcase scenario is when the array is arranged in reverse order. Additionally, delve into the underlying concepts that influence the algorithm's behavior in these diverse situations.

**Answer**:

BestCase Scenario (Presorted): O(n), as the algorithm only requires one pass to confirm that the array is sorted.

AverageCase Scenario (Randomly Arranged): O(n^2), where n is the number of elements.

WorstCase Scenario (Reverse Order): O(n^2), as each element needs to traverse the array multiple times.

**Underlying Concepts:**

Bubble Sort Principle: Compares adjacent elements and swaps them if they are in the wrong order.

Inefficiency: Bubble sort's simplicity leads to inefficiency, especially for large datasets.

Q4: Explore the fundamental concepts underlying the selection sort algorithm and delve into its stepbystep process, elucidating the key principles guiding the sorting procedure.

**Answer:**

Fundamental Concepts:

Selection: Finds the minimum element and places it at the beginning, then repeats for the remaining unsorted portion.

InPlace Sorting: Requires constant additional space.

StepbyStep Process:

1. Selection: Identify the minimum element.

2. Swap: Swap it with the first element.

3. Repeat: Repeat for the remaining unsorted portion.

Q5: Envision an unordered dataset. Could you lead me through the conceptual steps of implementing selection sort, emphasizing the fundamental principles that drive the transformation from disorder to a meticulously organized arrangement? Additionally, how does selection sort compare to other sorting algorithms in terms of efficiency and applicability?

Answer:

Conceptual Steps:

1. Find Minimum: Identify the minimum element in the unsorted portion.

2. Swap: Swap it with the first element of the unsorted portion.

3. Expand Sorted Portion: Move the boundary between sorted and unsorted portions.

4. Repeat: Continue until the entire array is sorted.

Comparison with Other Sorting Algorithms:

Efficiency: Inefficient for large datasets (O(n^2)).

Applicability: Simple and easy to implement but less efficient than more advanced algorithms like quicksort or mergesort.

Stability: Selection sort is not stable.

These algorithms offer different tradeoffs in terms of simplicity, stability, and efficiency, making them suitable for various use cases and constraints.