Lecture 1.1

Data Structures with Python - Lecture 1

Agenda

- 1.1 Data Structure Basic Concepts
- 1.2 Types of data structures
- 1.3 Analysis Terms (definitions only)

What is a Data Structure?

- A data structure is a specialized format for organizing, processing, retrieving and storing data.
- It provides a means to manage large amounts of data efficiently for uses such as large databases and internet indexing services.
- It's the logical or mathematical model of a particular organization of data.

Why do we need Data Structures?

- Data Search: To search data quickly when dealing with massive datasets.
- Processor Speed: Efficient data structures reduce the burden on processor speed.
- Multiple Requests: Organizing data helps in responding to concurrent requests efficiently.

Real-world Example of Data Structure

- Dictionary: Words are sorted alphabetically. This organization makes searching for a word fast.
- Queue at a ticket counter: First person in, first person served (FIFO).
- Browser History: Stored like a stack. Pressing 'back' goes to the most recent page (LIFO).

Types of Data Structures

- Broadly categorized into two main types based on how elements are stored and arranged.
- 1. Linear Data Structures
- 2. Non-Linear Data Structures

Linear Data Structures

- Elements are arranged in a sequential or linear order.
- Each element is attached to its previous and next adjacent elements.
- Memory can be a single contiguous block or scattered but linked sequentially.

Examples of Linear Data Structures

- Arrays: Fixed size, sequential memory.
- Linked Lists: Dynamic size, nodes link to each other.
- Stacks: Last-In, First-Out (LIFO).
- Queues: First-In, First-Out (FIFO).

Non-Linear Data Structures

- Elements are not arranged sequentially or linearly.
- One element can be connected to multiple other elements.
- Data is organized in a hierarchical or interconnected manner.

Examples of Non-Linear Data Structures

- Trees: Hierarchical structure, with a root and branches (e.g., family tree, file system).
- Graphs: Interconnected nodes representing networks (e.g., social networks, maps).

Static vs. Dynamic Data Structures

- Static Data Structures: Size is fixed at compile time (e.g., Array). Memory is allocated beforehand.
- Dynamic Data Structures: Size can grow or shrink at runtime (e.g., Linked List). Memory is allocated dynamically.

Analysis Terms: Why Analyze?

- To solve a problem, multiple algorithms may exist.
- We need to choose the most efficient one.
- Efficiency is usually measured in terms of time taken and memory space used.

Time Complexity & Space Complexity

- Time Complexity: The amount of time an algorithm takes to run as a function of the length of the input.
- Space Complexity: The amount of memory space an algorithm needs to run to completion.

Asymptotic Notations

- Mathematical tools to represent the time and space complexity of algorithms.
- Focuses on the behavior of the algorithm as the input size approaches infinity.
- Abstracts away machine-specific details.

Big 'O' Notation

- The most commonly used notation.
- Represents the upper bound of the runtime of an algorithm.
- Describes the worst-case scenario. The algorithm will not take more time than this bound.

Best, Average, and Worst Case

- Best Case Complexity: Minimum time required. (e.g., Target found at the first position).
- Average Case Complexity: Expected time required across all possible inputs.
- Worst Case Complexity: Maximum time required. (e.g., Target is at the end or not present).

Summary

- Data structures organize data for efficient use.
- They are primarily Linear or Non-Linear.
- Algorithm analysis uses Time and Space complexity.
- Big 'O' notation is used to describe the worst-case performance.

Next Lecture Preview

- We will dive into Python Specific Data Structures.
- Topics include: List, Tuple, Set, and Dictionary.