Data Structures

Prepared for DSC 314 course at IISER-TVM by Dr. Dhanyamol Antony.

Data Structures and Algorithms

- **Data Structure**: A systematic way of organizing, storing, and accessing data efficiently.
- **Algorithm**: A step-by-step procedure used to solve a problem in a finite amount of time.
- **Relationship between Data Structures and Algorithms**:
  - Algorithms and data structures go hand-in-hand.
  - Certain algorithms require specific data structures to run efficiently.
  - Similarly, the choice of a data structure affects the performance of algorithms.

Algorithm analysis depends critically on the **computational model** used. For example, the statement that an algorithm runs in $O(n^2 \log n)$ time is only valid under a specific model of computation.

Designing Data Structures

- Each data structure has its own costs and benefits.
- Rarely is one data structure better than another in all situations.
- A data structure requires:
  - space for each data item it stores,
  - time to perform each basic operation,
  - programming effort.
- Each problem has constraints on available time and space.
- Only after a careful analysis of problem characteristics and solution requirements can the best data structure for the task be determined.

Selecting Data Structures

- Select a data structure using the following steps:
  1. Analyze the problem to determine the resource constraints that a solution must meet.
  2. Determine the basic operations that must be supported and quantify the resource constraints for each operation.
  3. Select the data structure that best satisfies these requirements.
- Some important questions to consider:
  - Are all data inserted into the structure at the beginning, or are insertions interspersed with other operations?
  - Can data be deleted from the structure?
  - Are the data processed in a well-defined order, or is random access allowed?

Abstract Data Types (ADT)

- An **Abstract Data Type (ADT)** is a theoretical concept or mathematical model that defines a data type by its behavior.
- It specifies:
  - a set of data values, and
  - a set of operations that can be performed on those values,

  without specifying how the data type is implemented.
- Each ADT operation is defined in terms of its inputs and outputs.
- The core idea of an ADT is the separation of:
  - **What**: the logical view and operations of the data type, and
  - **How**: the physical implementation of the data type.
- This hiding of implementation details is known as **encapsulation**.
- **Logical vs. Physical Form of Data**
  - Data items have both a logical and a physical form.
  - **Logical Form**:
    - The definition of the data item within an ADT.
    - Describes what operations are allowed and how the data is viewed logically.
  - **Physical Form**:

    - The actual implementation of the data item using a data structure.
    - Describes how the data is stored in memory.
  - **Example**:
    - The **List ADT** represents an ordered collection of elements (logical form).
    - It can be implemented using an **array** or a **linked list** (physical form).

- **Common Examples of ADTs**:
  - **Stack ADT**: LIFO order; operations include `push()`, `pop()`, `peek()`
  - **Queue ADT**: FIFO order; operations include `enqueue()`, `dequeue()`
  - **List ADT**: sequential data; operations include `insert()`, `delete()`, `find()`
  - **Set ADT**: stores unique elements; operations include `add()`, `remove()`

- **Data Structure** can be defined as:

$$\text{Data Structure} = \text{Organized Data} + \text{Allowed Operations}$$

- A data structure involves two complementary goals:
  1. **Abstract Goal**:
     - To identify and develop useful mathematical entities and operations.
     - To determine the class of problems that can be solved using these entities and operations.
  2. **Implementation Goal**:
     - To determine appropriate representations for the abstract entities.
     - To implement the abstract operations on a concrete data representation.

- Thus, data structures bridge the gap between abstract concepts (such as ADTs) and their efficient implementation in computer memory.

Data Types, Abstract Data Types, and Data Structures

- The **data type** of a variable is the set of values that the variable may assume.
- **Basic Data Types in C**:
  - `int`, `char`, `float`, `double`
- An **Abstract Data Type (ADT)** is a set of elements together with a collection of well-defined operations.
- The operations:

  - may take as operands not only instances of the ADT but also values of other data types or instances of other ADTs,
  - may return results that are not necessarily instances of the ADT,
  - must involve at least one operand or result of the ADT type in question.

- Object-oriented programming languages such as **C++** and **Java** provide explicit support for ADTs through the use of **classes**.

- **Examples of ADTs** include:
  - list, stack, queue, set, tree, graph.

Data Structures

- A **data structure** is an implementation of an ADT using a programming language.
- It is a translation of the ADT into program statements, which includes:
  - declarations that define variables of the ADT type, and
  - procedures or methods that implement the operations defined by the ADT.
- Thus, a data structure provides a concrete representation of an abstract concept, allowing the ADT to be used in actual programs.

ADT Implementation and Data Structures

- An **ADT implementation** selects an appropriate **data structure** to represent the ADT.
- Each data structure is constructed from the **basic data types** of the underlying programming language using available data structuring facilities such as:
  - arrays,
  - records (structures in C),
  - pointers,
  - files,
  - sets, etc.
- **Example: Queue ADT**
  - A **Queue** is an ADT defined as a sequence of elements.
  - Common operations on a Queue include:
    * `full(Q)`
    * `empty(Q)`

    - `enqueue(x, Q)`
    - `dequeue(Q)`
  - The Queue ADT can be implemented using different data structures, such as:
    - array,
    - singly linked list,
    - doubly linked list,
    - circular array.

- Different implementations of the same ADT may vary in terms of time complexity, space requirements, and ease of implementation.