Algorithms and Data Structures

**Algorithm**
- Sequence of steps used to solve a problem
- Operates on collection of data
- Each element of collection $\Rightarrow$ data structure

**Data structure**
- Combination of simple / composite data types
- Design $\Rightarrow$ information stored for each element
- Choice affects characteristic & behavior of algorithm
- May severely impact efficiency of algorithm
Data Structures

Taxonomy
- Classification scheme
- Based on relationships between element

<table>
<thead>
<tr>
<th>Category</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>one ⇒ one</td>
</tr>
<tr>
<td>Hierarchical</td>
<td>one ⇒ many</td>
</tr>
<tr>
<td>Graph</td>
<td>many ⇒ many</td>
</tr>
<tr>
<td>Set</td>
<td>none ⇒ none</td>
</tr>
</tbody>
</table>

Data Structures

Core operations
- Add element
- Remove element
- Iterate through all elements
- Compare elements
Linear Data Structures

- One-to-one relationship between elements
  - Each element has unique predecessor
  - Each element has unique successor

---

Linear Data Structures

- Core operations
  - Find first element (head)
  - Find next element (successor)
  - Find last element (tail)

- Terminology
  - Head $\Rightarrow$ no predecessor
  - Tail $\Rightarrow$ no successor
Example Linear Data Structures

- **List**
  - Collection of elements in order

- **Queue**
  - Elements removed in order of insertion
  - First-in, First-out (FIFO)

- **Stack**
  - Elements removed in opposite order of insertion
  - First-in, Last-out (FILO)

Hierarchical Data Structures

- One-to-many relationship between elements
  - Each element has unique predecessor
  - Each element has multiple successors
Hierarchical Data Structures

Core operations
- Find first element (root)
- Find successor elements (children)
- Find predecessor element (parent)

Terminology
- Root ⇒ no predecessor
- Leaf ⇒ no successor
- Interior ⇒ non-leaf
- Children ⇒ successors
- Parent ⇒ predecessor

Example Hierarchical Data Structures

- Tree
  - Single root

- Forest
  - Multiple roots

- Binary tree
  - Tree with 0–2 children per node
Graph Data Structures

- Many-to-many relationship between elements
  - Each element has multiple predecessors
  - Each element has multiple successors

Core operations
- Find successor nodes
- Find predecessor nodes
- Find adjacent nodes (neighbors)

Terminology
- Directed ⇒ traverse edges in one direction
- Undirected ⇒ traverse edges in both directions
- Neighbor ⇒ adjacent node
- Path ⇒ sequence of edges
- Cycle ⇒ path returning to same node
- Acyclic ⇒ no cycles
Example Graph Data Structures

- **Undirected graph**
  - Undirected edges

- **Directed graph**
  - Directed edges

- **Directed acyclic graph (DAG)**
  - Directed edges, no cycles

---

Set Data Structures

- **No relationship between elements**
  - Elements have no predecessor / successor
  - Only one copy of element allowed in set
Set Data Structures

- Core operations
  - Set versions of core operations
  - Add set, remove set, compare set

- Terminology
  - Subset ⇒ elements contained by set
  - Union ⇒ select elements in either set
  - Intersection ⇒ select elements in both sets
  - Set difference ⇒ select elements in one set only

Example Set Data Structures

- Set
  - Basic set
- Map
  - Map value to element in set
- Hash Table
  - Maps value to element in set using hash function

![Set Diagram](#)

![Map Diagram](#)

![Hash Table Diagram](#)
Java Collections Framework

- **Collection**
  - Object that groups multiple elements into one unit
  - Also called container

- **Collection framework consists of**
  - Interfaces
    - Abstract data type
  - Implementations
    - Reusable data structures
  - Algorithms
    - Reusable functionality

Java Collections Framework

- **Goals**
  - Reduce programming effort
  - Make APIs easier to learn
  - Make APIs easier to design and implement
  - Reuse software
  - Increase performance
Core Collection Interfaces

- **Collection**
  - Group of elements

- **Set**
  - No duplicate elements

- **List**
  - Ordered collection

- **Map**
  - Maps keys to elements

- **SortedSet, SortedMap**
  - Sorted ordering of elements

Core Collection Hierarchy
Collections Interface Implementations

- **General implementations**
  - Primary public implementation
  - Example
    - List – ArrayList, LinkedList
    - Set – TreeSet, HashSet
    - Map – TreeMap, HashMap

- **Wrapper implementations**
  - Combined with other interfaces
  - Example
    - synchronizedArrayList, unmodifiableHashMap

Collections Interface Methods

- **boolean add(Object o)**
  - Add specified element

- **boolean contains(Object o)**
  - True if collection contains specified element

- **boolean remove(Object o)**
  - Removes specified element from collection

- **boolean equals(Object o)**
  - Compares object with collection for equality

- **Iterator iterator()**
  - Returns an iterator over the elements in collection
Collections Interface Methods

- boolean `addAll(Collection c)`
  - Adds all elements in specified collection

- boolean `containsAll(Collection c)`
  - True if collection contains all elements in collection

- boolean `removeAll(Collection c)`
  - Removes all elements in specified collection

- boolean `retainAll(Collection c)`
  - Retains only elements contained in specified collection

Collections Interface Methods

- void `clear()`
  - Removes all elements from collection

- boolean `isEmpty()`
  - True if collection contains no elements

- int `size()`
  - Returns number of elements in collection

- `Object[] toArray()`
  - Returns array containing all elements in collection
Iterator Interface

- **Iterator**
  - Common interface for all Collection classes
  - Used to examine all elements in collection

- **Properties**
  - Order of elements is unspecified (may change)
  - Can remove current element during iteration
  - Works for any collection

---

Iterator Interface

- **Interface**
  ```java
  public interface Iterator {
    boolean hasNext();
    Object next();
    void remove();   // optional, called once per next()
  }
  ```

- **Example usage**
  ```java
  Iterator i = myCollection.iterator();
  while (i.hasNext()) {
    myCollectionElem x = (myCollectionElem) i.next();
  }
  ```