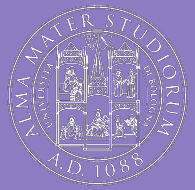


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Intelligenza Artificiale

Data Structures and Computational thinking

Ivan Heibi
Dipartimento di Filologia Classica e Italianistica (FICLIT)
Ivan.heibi2@unibo.it

What is a computer?

A **computer** is a **machine** that can be **instructed** to carry out sequences of **arithmetic or logical operations** automatically for **processing data** represented by alphanumeric characters.

More generic: an agent that is capable of making calculations and producing a response (output) based on some initial information (input)

Writing a program: communicating with a computer using a language (formal) that both the human instructor and the computer itself can understand.

The computer executes **instructions (software)** to manipulate **information (data structures)**

Abstraction and computational thinking

Abstraction is a conceptual process where **general rules and concepts** are derived from the **usage and classification** of specific examples

Abstractions may be formed by filtering out the information content of a concept or an observable phenomenon, selecting only the aspects which are relevant for a particular subjectively valued purpose.

What these two situations have in common?

Abstraction and computational thinking

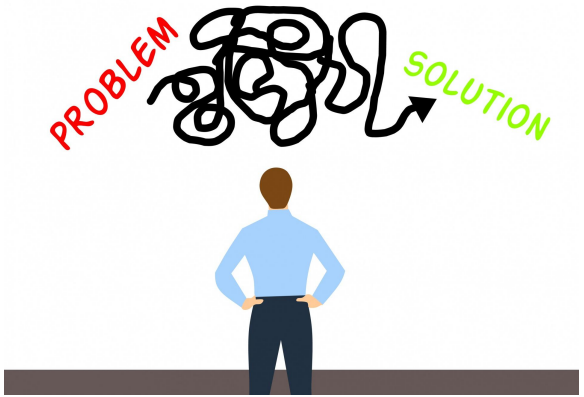
Computational thinking is an approach to **problem-solving, system development, and understanding human behavior** that embraces the fundamental **concepts of computation**

Main abstractions in computer science:

- Data structures
- Models
- Algorithms
- Networks

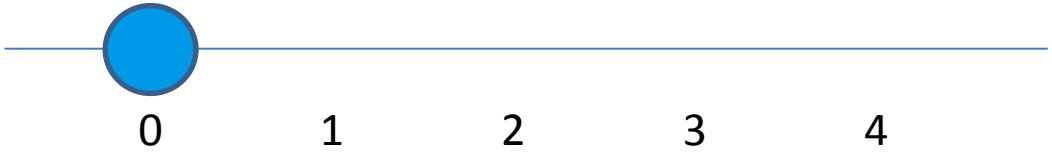
Typical Scenario

1. Represent the problem domain with terms that can be interpreted and manipulated by the machine.
2. Represent the problem with respect to its representation:
   a. Define the initial state as a configuration of the data
   b. Define the final state as a configuration of the data
3. Devise an algorithm able to progress data from an the initial configuration to the final configuration (solution)
4. Implementation of algorithm and data structure



Typical Scenario example

Problem:
I want a train to stop at the third station on its route

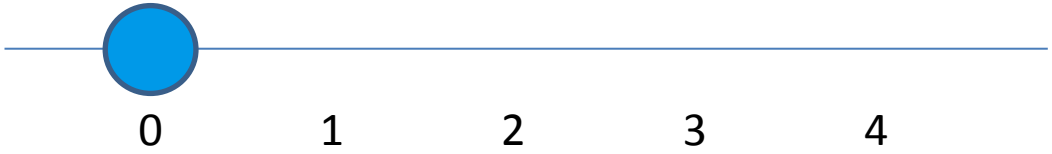


1. Represent the problem domain with terms that can be interpreted and manipulated by the machine.
2. Represent the problem with respect to its representation:
   a. Define the initial state as a configuration of the data
   b. Define the final state as a configuration of the data
3. Devise an algorithm able to progress data from the initial configuration to the final configuration (solution)
4. Implementation of algorithm and data structure

1: **a dot on the x-axis**

Typical Scenario example

Problem:
I want a train to stop at the third station on its route



1. Represent the problem domain with terms that can be interpreted and manipulated by the machine.
2. Represent the problem with respect to its representation:
   a. Define the initial state as a configuration of the data
   b. Define the final state as a configuration of the data
3. Devise an algorithm able to progress data from the initial configuration to the final configuration (solution)
4. Implementation of algorithm and data structure

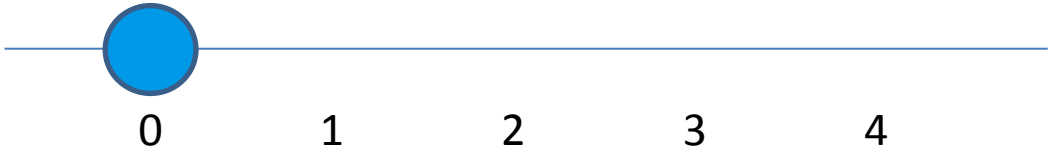
1: **a dot on the x-axis**

2.a: **0**

2.b: **2**

Typical Scenario example

Problem:
I want a train to stop at the third station on its route



1. Represent the problem domain with terms that can be interpreted and manipulated by the machine.
2. Represent the problem with respect to its representation:
   a. Define the initial state as a configuration of the data
   b. Define the final state as a configuration of the data
3. Devise an algorithm able to progress data from the initial configuration to the final configuration (solution)
4. Implementation of algorithm and data structure

1: **a dot on the x-axis**

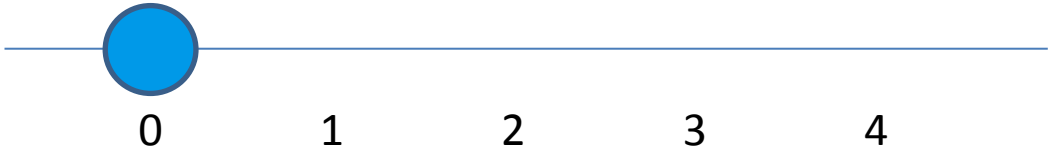
2.a: **0**

2.b: **2**

3. Move right; if position = 2 then stop; otherwise keep moving right

Typical Scenario example

Problem:
I want a train to stop at the third station on its route



1. Represent the problem domain with terms that can be interpreted and manipulated by the machine.
2. Represent the problem with respect to its representation:
   a. Define the initial state as a configuration of the data
   b. Define the final state as a configuration of the data
3. Devise an algorithm able to progress data from the initial configuration to the final configuration (solution)
4. Implementation of algorithm and data structure

1: **a dot on the x-axis**

2.a: **0**

2.b: **2**

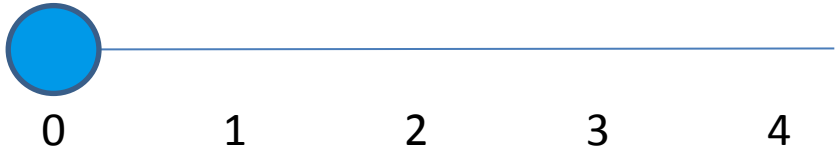
3. Move right; if position = 2 then stop; otherwise keep moving right

4.

```
x = 0
while (x != 2)
    x = x+1
```

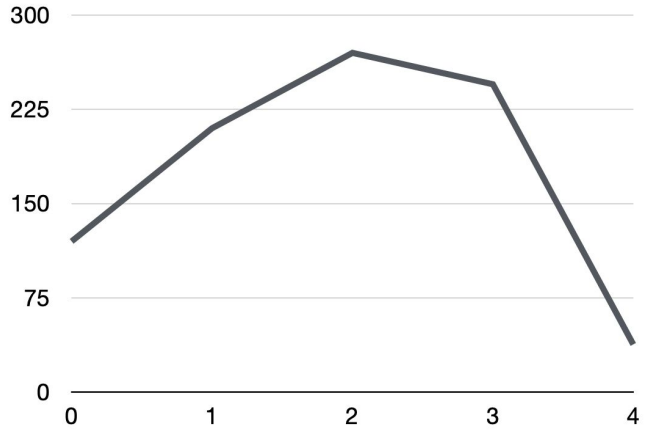
Utility function example (local maximum)

<u>Problem:</u>
The train must proceed as long as the next station has more people than the current one



> the train can call and ask for how many people there is at the next station <u>only</u>

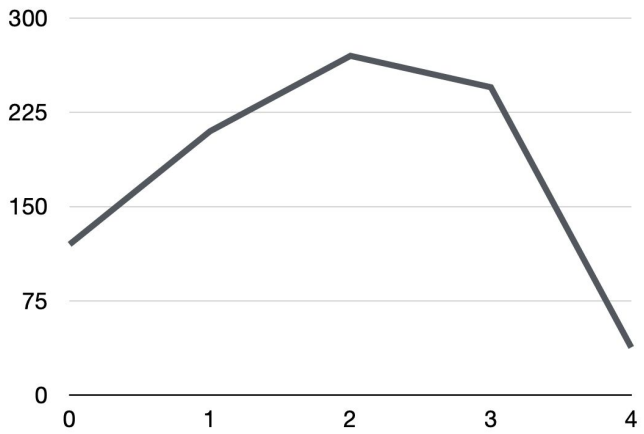
1: **a dot on the x-axis**

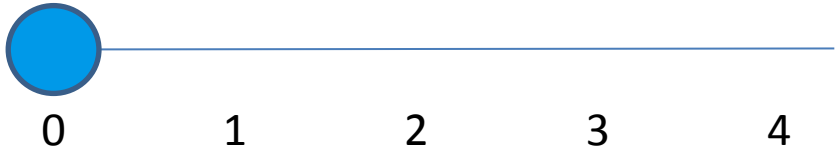


Utility function example (local maximum)

<u>Problem:</u>
The train must proceed as long as the next station has more people than the current one

> the train can call and ask for how many people there is at the next station <u>only</u>





1: **a dot on the x-axis**

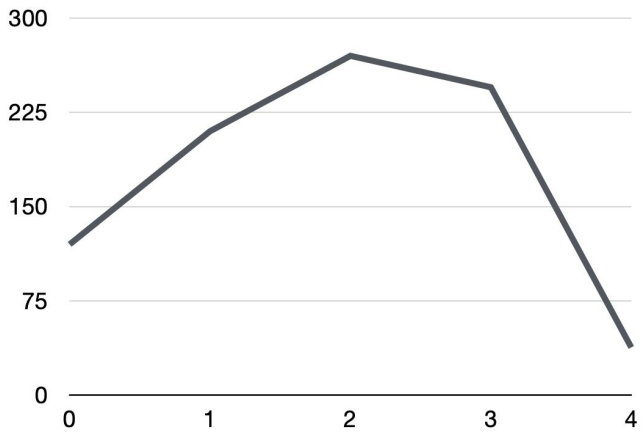
2.a: **0**

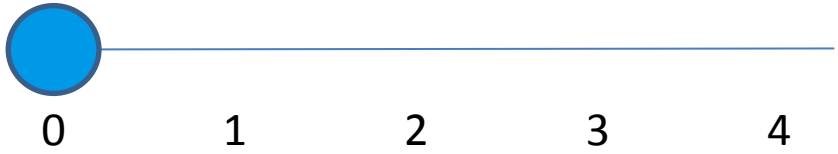
2.b: **the nearest N with higher number of people (utility)**

Utility function example (local maximum)

<u>Problem:</u>
The train must proceed as long as the next station has more people than the current one

> the train can call and ask for how many people there is at the next station <u>only</u>





1: **a dot on the x-axis**

2.a: **0**

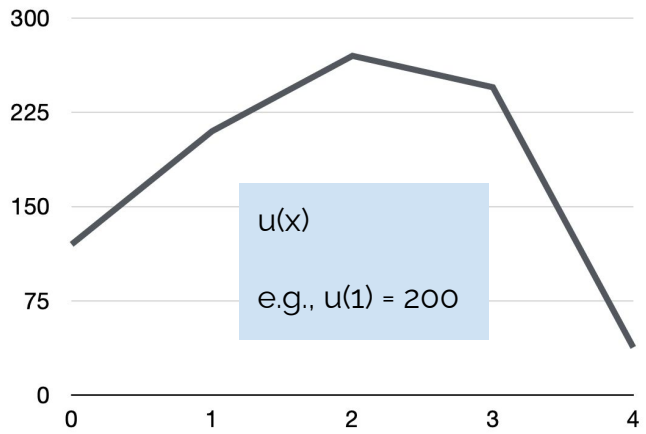
2.b: **the nearest N with higher number of people (utility)**

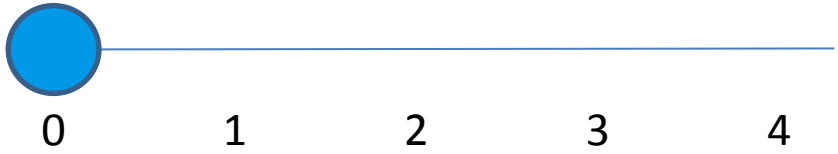
3. if number of people at current position is higher than the one of the next station then stop; otherwise keep moving right

Utility function example (local maximum)

<u>Problem:</u>
The train must proceed as long as the next station has more people than the current one

> the train can call and ask for how many people there is at the next station <u>only</u>





1: **a dot on the x-axis**

2.a: **0**

2.b: **the nearest N with higher number of people (utility)**

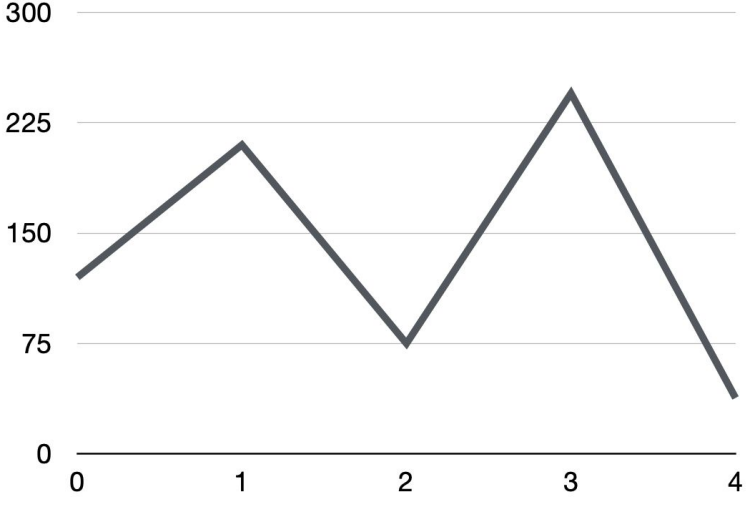
3. if number of people at current position is higher than the one of the next station then stop; otherwise keep moving right

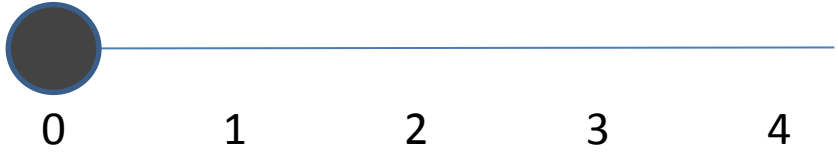
4.

```
while (true)
  if (u(x+1) > u(x))
     x = x+1
  else
     break
```

Utility function example (global maximum)

Problem:
I would like a drone to fly over all five train stations and return to me the station with the highest number of people.





1: **a dot on the x-axis**

2.a: **0**

2.b: **N with max number of people (utility)**

3. Move right; if number of people at current position is higher than the one of the previous station keep track of the station; keep moving right

4.

```
max = u(0)
while (x <= 10)
  x = x + 1
  if (u(x) > max)
     max = u(x)
```

Data

Primitive data types (indicate values)

- Integer: -.... 2, -1, 0, 1, 2 ....
- String (character sequence): “Artificial Intelligence”, “Ivan” (“I”+”v”+”a”+”n”)
- Boolean true/false
- Real e.g. 0.1

An entity is something we may want to say something about

Individuals (identify entities)

- An entity may have attributes
  (E.g. Cillian Murphy as a person with his tax id, birth date, height etc.)
- An entity may have relations with other entities
  (E.g. Christopher Nolan directed Cillian Murphy)
- An entity may belong to a class
  (E.g. Cillian Murphy is an individual of the class Person)

Entities and relations

Use abstraction to define entities.

Entities:

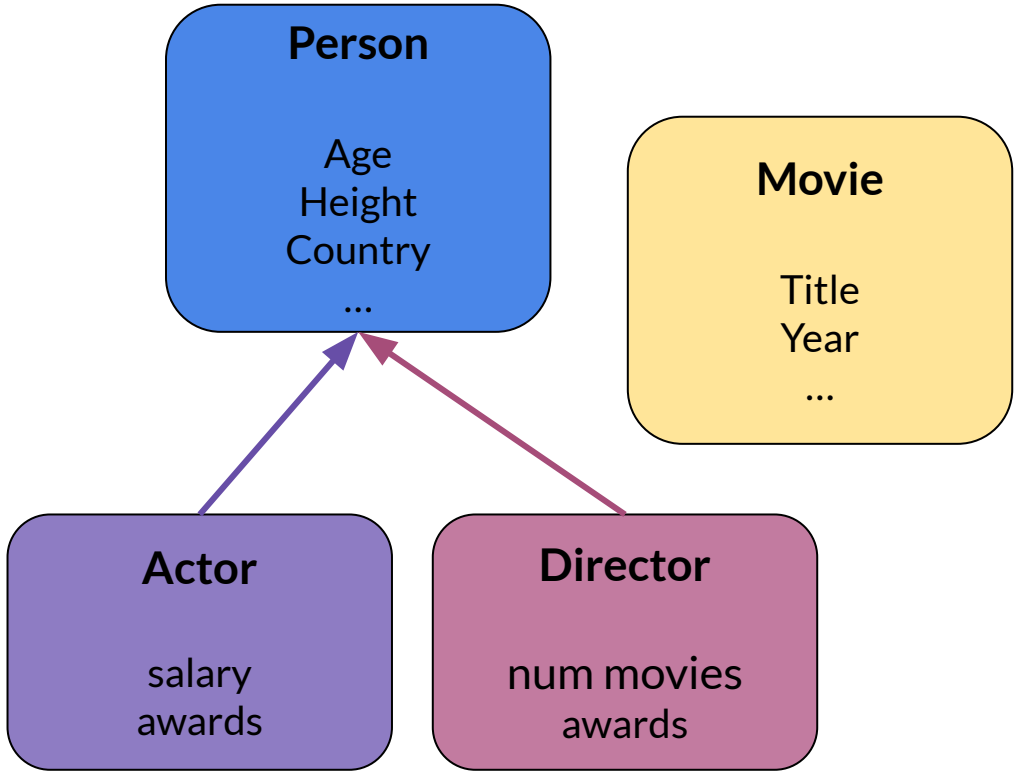
Cillian Murphy is an actor:
47 years old, 175 (cm) tall from Ireland,
he has 3 awards

Christopher Nolan is a director with
53 years old, 185 (cm) tall from UK
he won 8 awards and directed
20 movies

Oppenheimer is a movie released in 2023

Relations:

Christopher Nolan directed Oppenheimer
Cillian Murphy worked with Christopher Nolan

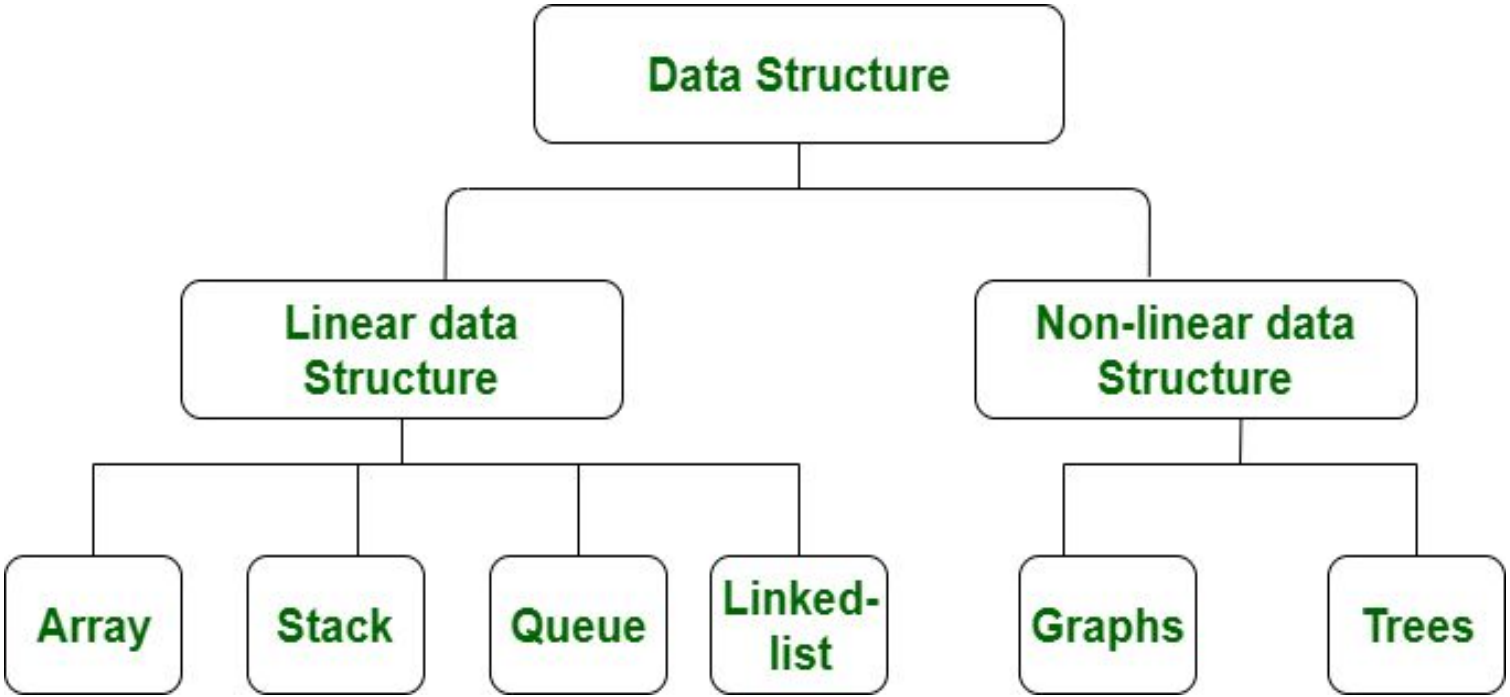


Self Assessment

https://forms.gle/cdghGwTTxfEt65tQ6

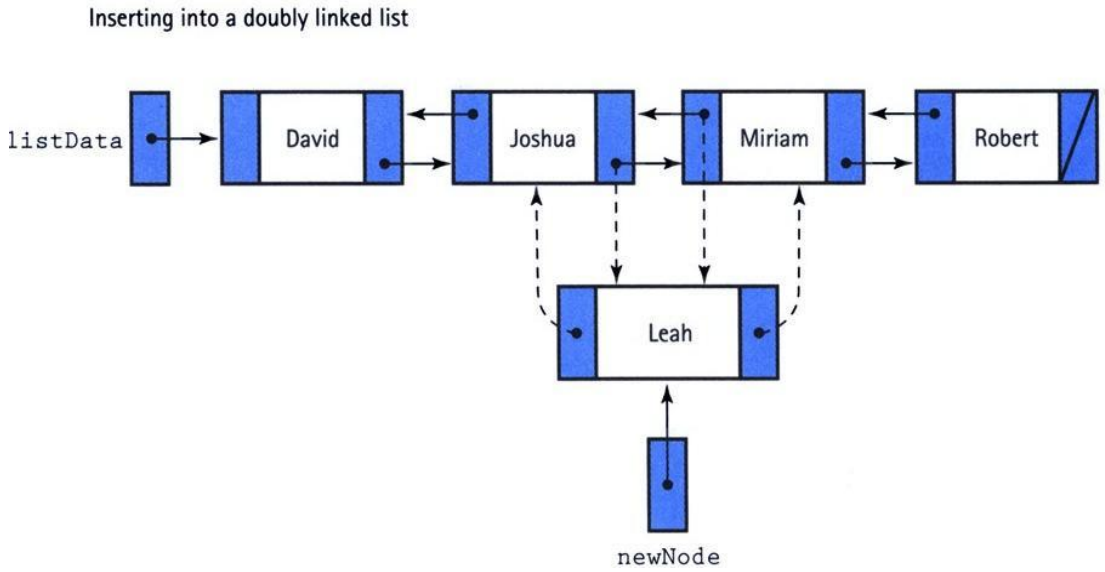
Data structures

A Data Structure is a data organization, management, and storage format



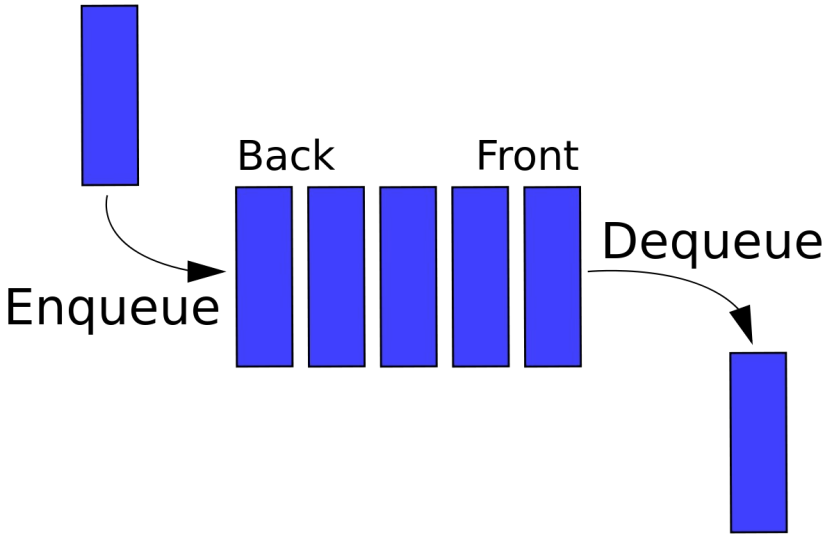
List

A list or sequence is an abstract data type that represents a countable number of ordered values, where the same value may occur more than once.



Queue

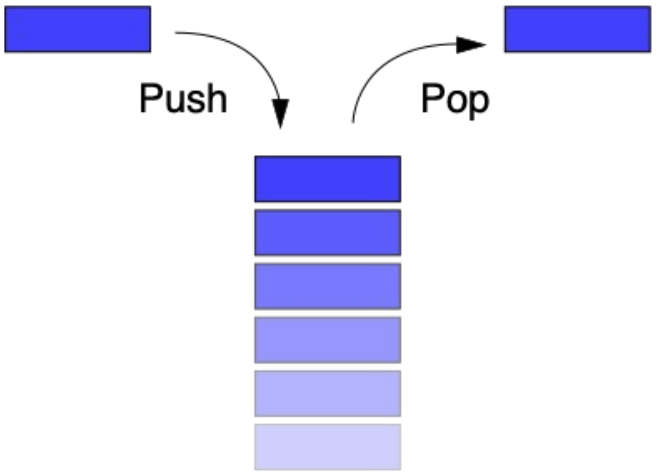
A queue is a collection of entities that are maintained in a sequence and can be modified by the addition of entities at one end of the sequence and the removal of entities from the other end of the sequence.



Stack

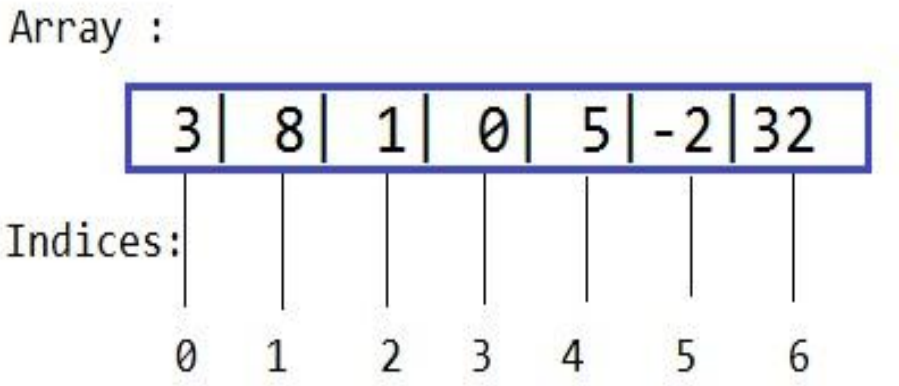
A stack is an abstract data type that serves as a collection of elements, with two main principal operations:

- Push, which adds an element to the collection, and
- Pop, which removes the most recently added element that was not yet removed.



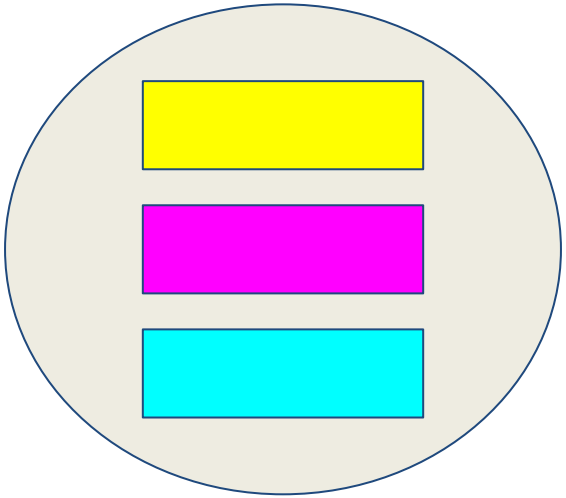
Array

An array is a data structure consisting of a collection of elements (values or variables), each identified by at least one array index or key.



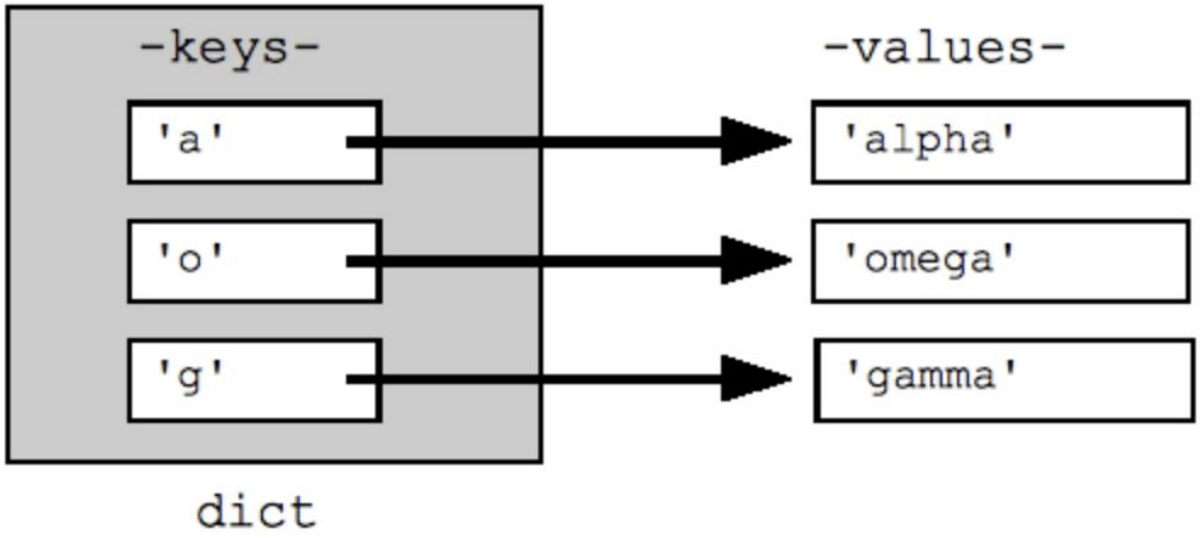
Set

A set is an abstract data type that can store unique values, without any particular order.



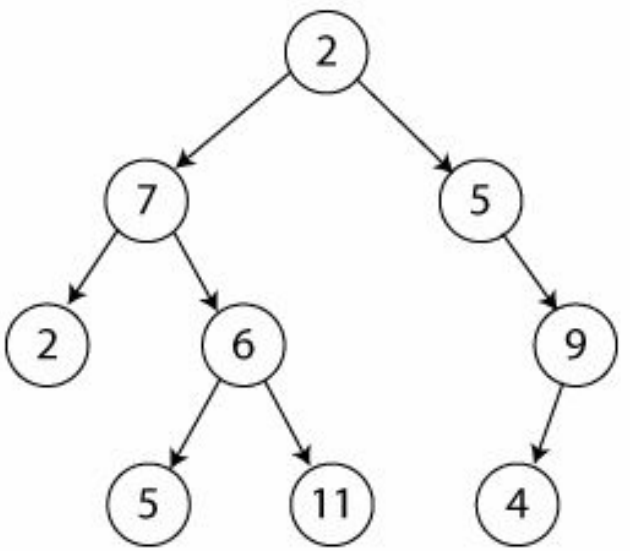
Dictionary

A dictionary is an abstract data type composed of a collection of (key, value) pairs, such that each possible key appears at most once in the collection.



Tree

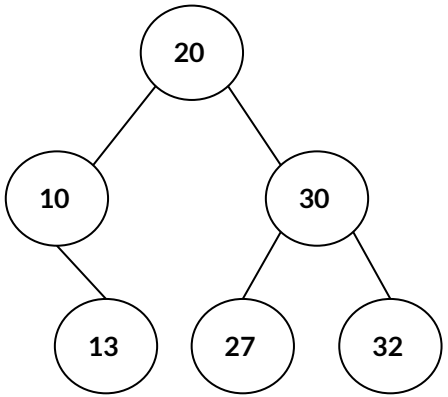
A tree is an abstract data type that simulates a hierarchical tree structure, with a root value and subtrees of children with a parent node, represented as a set of linked nodes.



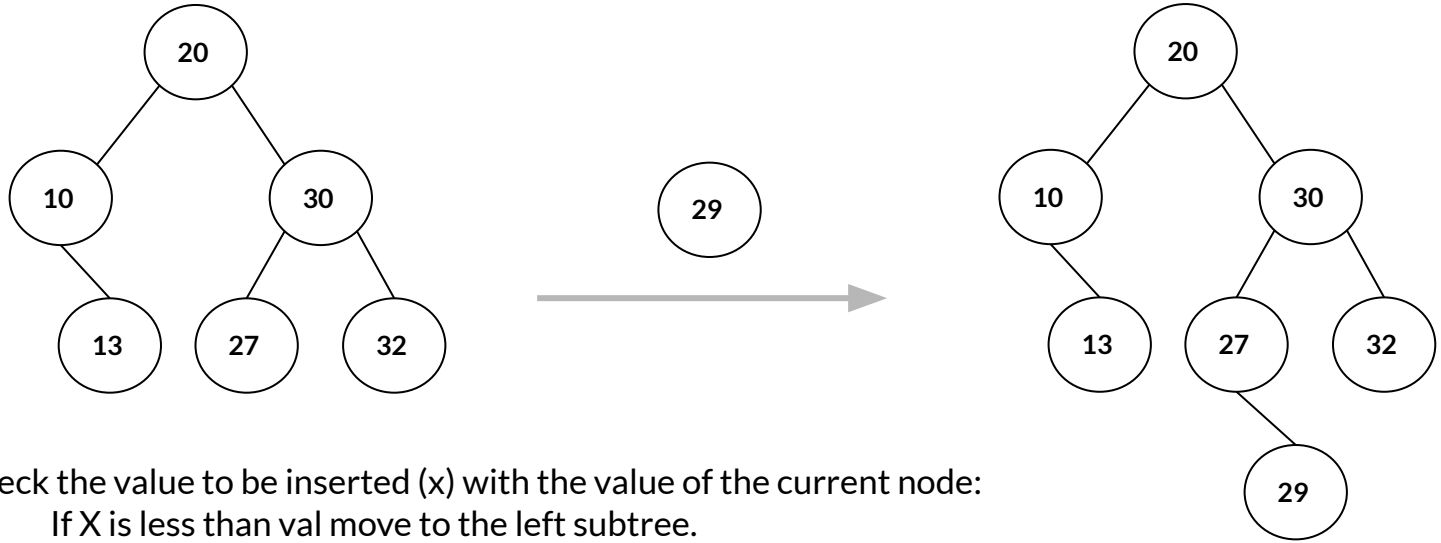
Example: Binary Search Tree

Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.



Example: Binary Search Tree – insert



Check the value to be inserted (x) with the value of the current node:

- If X is less than val move to the left subtree.
- Otherwise, move to the right subtree.
- Once the leaf node is reached, insert X to its right or left based on the relation between X and the leaf node's value.

Graph

A graph is an abstract data type that is meant to implement the undirected graph and directed graph concepts from the field of graph theory within mathematics.

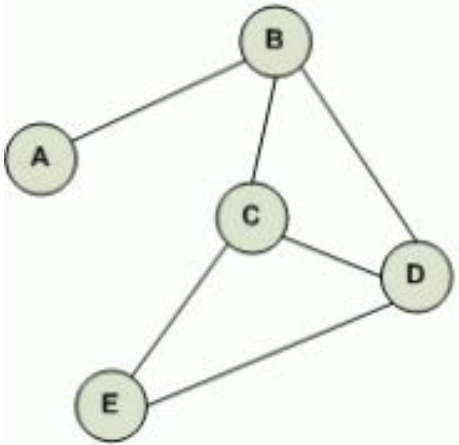


Fig 1. Undirected Graph

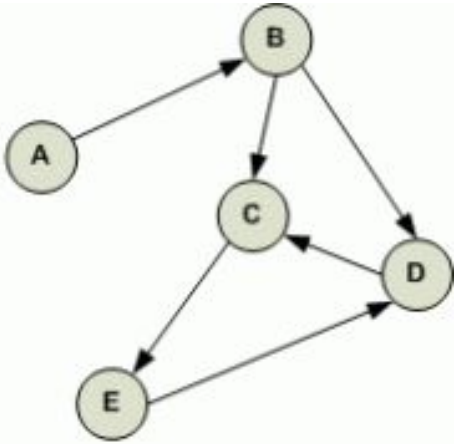
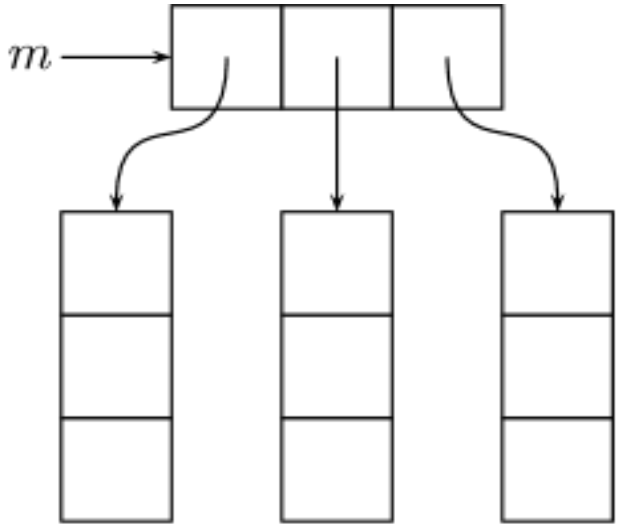


Fig 2. Directed Graph

Matrix

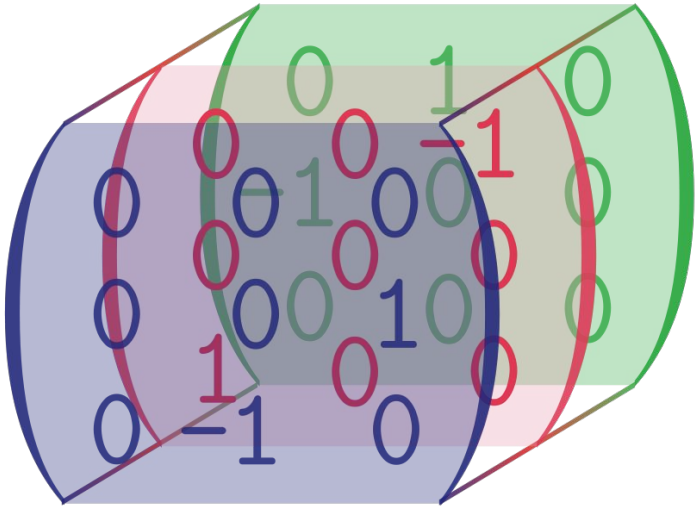
A matrix is a bi-dimensional array (an array of arrays)



Tensor

A tensor is a multi-dimensional array (e.g. an array of arrays of arrays)

$$\epsilon_{ijk} =$$



Example: Library Catalog

Create a data structure to represent a library catalog.

- The catalog needs to store information about books, including their titles, authors, publication years, and availability.
- Design a data structure to efficiently organize and manage this information.

Example: Library Catalog

Create a data structure to represent a library catalog.

- The catalog needs to store information about books, including their titles, authors, publication years, and availability.
- Design a data structure to efficiently organize and manage this information.