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

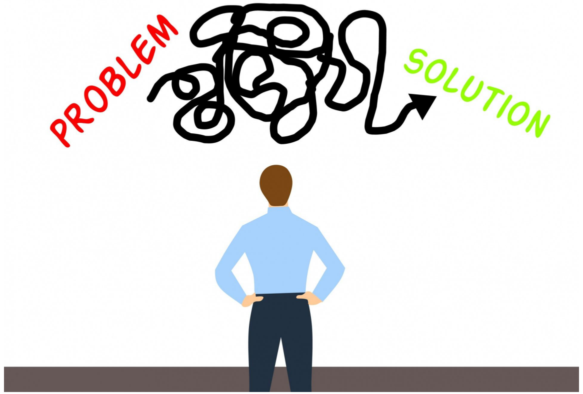
Typical Scenario

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:

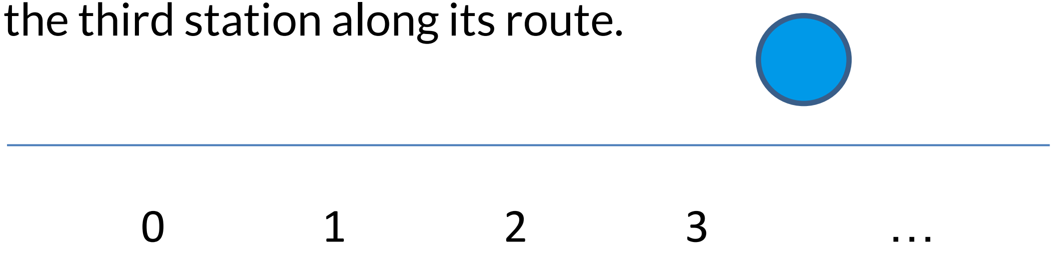
I want to design a program to manage the movement of a train. Specifically, the program should ensure that the train stops at the third station along its route.

Typical Scenario example

Problem:

I want to design a program to manage the movement of a train. Specifically, the program should ensure that the train stops at the third station along its route.

1. Represent the problem domain with terms that can be interpreted and manipulated by the machine.



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

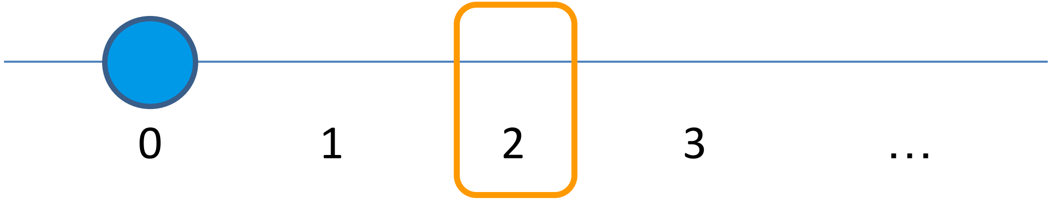
Typical Scenario example

Problem:

I want to design a program to manage the movement of a train. Specifically, the program should ensure that the train stops at the third station along its route.

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



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

2.a: **0**

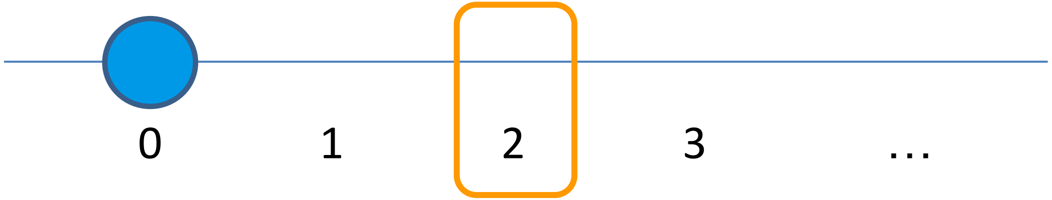
2.b: **2**

Typical Scenario example

Problem:

I want to design a program to manage the movement of a train. Specifically, the program should ensure that the train stops at the third station along its route.

3. Devise an algorithm able to progress data from the initial configuration to the final configuration (solution)



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

2.a: **0**

2.b: **2**

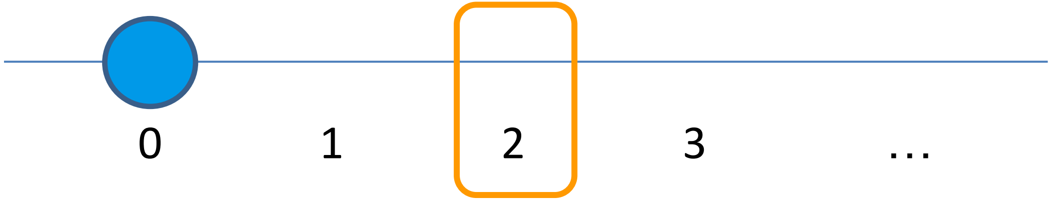
3. Keep moving right; if position = 2 then stop;

Typical Scenario example

Problem:

I want to design a program to manage the movement of a train. Specifically, the program should ensure that the train stops at the third station along its route.

4. Implementation of algorithm and data structure



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

2.a: **0**

2.b: **2**

3. Keep moving right; if position = 2 then stop;

4.

```
x = 0
While x is not equal 2:
    Increment x by 1

return x
```

Utility function example (local minimum)

Problem:

A train moves through 6 stations, and at each station x, it decides whether to continue or stop. The train only knows the current station's passengers and the previous station's passengers. The train should stop at a station where the number of passengers is lower than the previous station (local minimum).

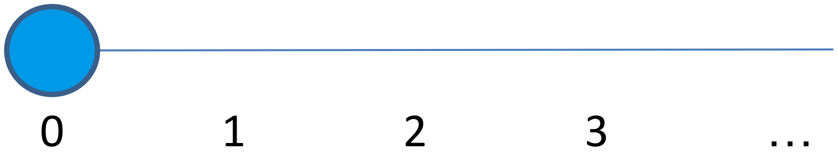
> *The function* ***u(x)*** *takes a station* ***x,*** *and returns the number of* ***passengers***

(utility in this case is related to maximizing passenger count)

Utility function example (local minimum)

<u>**Problem:**</u>

A train moves through 6 stations, and at each station x, it decides whether to continue or stop. The train only knows the current station's passengers and the previous station's passengers. The train should stop at a station where the number of passengers is lower than the previous station (local minimum).



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

> *The function **u(x)** takes a station **x,** and returns the number of **passengers***

(utility in this case is related to maximizing passenger count)

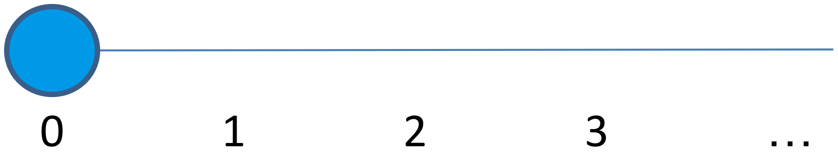
Utility function example (local minimum)

<u>**Problem:**</u>

A train moves through 6 stations, and at each station x, it decides whether to continue or stop. The train only knows the current station's passengers and the previous station's passengers. The train should stop at a station where the number of passengers is lower than the previous station (local minimum).

*> The function **u(x)** takes a station **x**, and returns the number of **passengers***

(utility in this case is related to maximizing passenger count)



1: a dot on the x-axis

2.a: 0

2.b: a station (N)

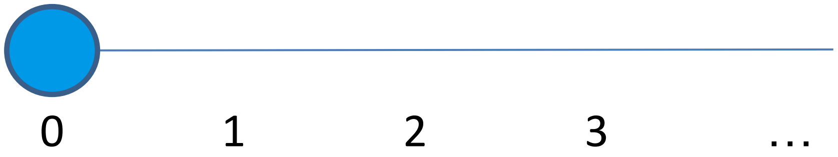
Utility function example (local minimum)

Problem:

A train moves through 6 stations, and at each station x, it decides whether to continue or stop. The train only knows the current station's passengers and the previous station's passengers. The train should stop at a station where the number of passengers is lower than the previous station (local minimum).

> *The function* ***u(x)*** *takes a station* ***x,*** *and returns the number of* ***passengers***

(utility in this case is related to maximizing passenger count)



1: a dot on the x-axis

2.a: 0

2.b: a station (N)

3. keep moving unless the number of passengers at the current station is lower than the number of passengers of the previous station

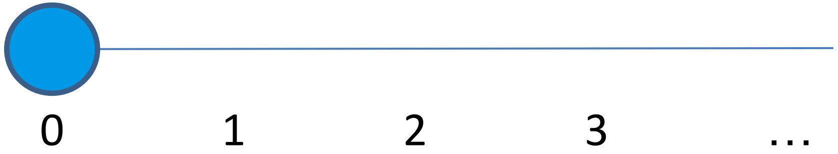
Utility function example (local minimum)

Problem:

A train moves through 6 stations, and at each station x, it decides whether to continue or stop. The train only knows the current station's passengers and the previous station's passengers. The train should stop at a station where the number of passengers is lower than the previous station (local minimum).

> *The function* ***u(x)*** *takes a station* ***x****, and returns the number of* ***passengers***

(utility in this case is related to maximizing passenger count)



1: a dot on the x-axis

2.a: 0

2.b: a station (N)

3. keep moving unless the number of passengers at the current station is lower than the number of passengers of the previous station

4.

```
x = 1
While x is less than 5:
      If u(x) < u(x - 1):
            return x
      else:
            Increment x by 1
```

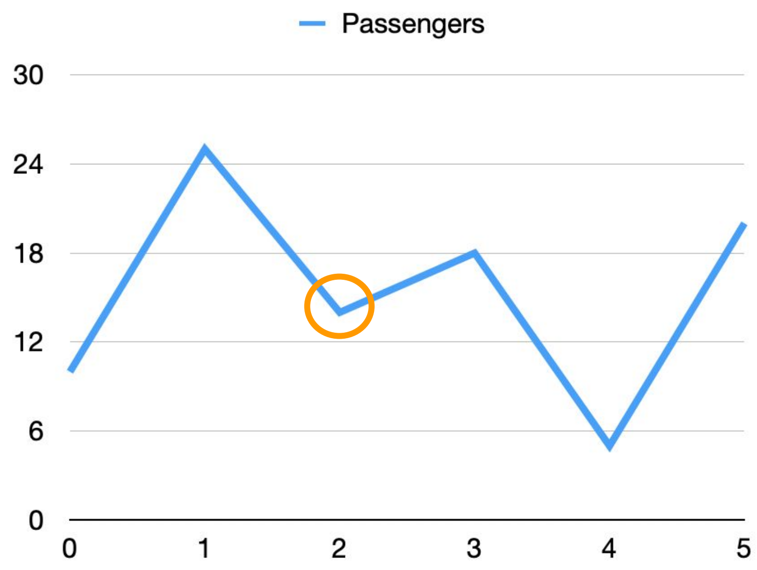
Utility function example (local minimum)

Problem:

A train moves through 6 stations, and at each station x, it decides whether to continue or stop. The train only knows the current station's passengers and the previous station's passengers. The train should stop at a station where the number of passengers is lower than the previous station (local minimum).

> *The function* ***u(x)*** *takes a station* ***x****, and returns the number of* ***passengers***

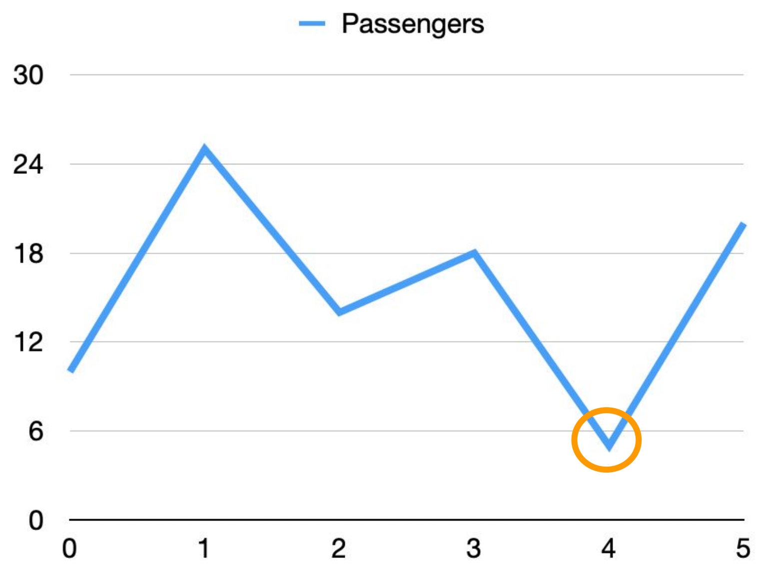
(utility in this case is related to maximizing passenger count)



Utility function example (global minimum)

Problem:

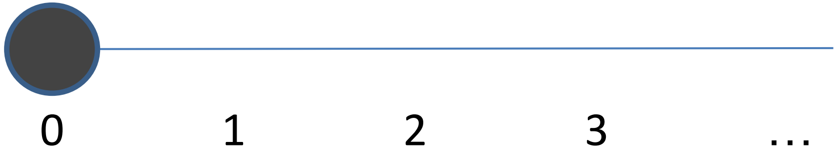
Suppose the train can track the number of passengers boarding at each station. The goal is for the program to identify and return the station with the lower number of passengers boarded by the end of the day.



Utility function example (global minimum)

<u>Problem:</u>

Suppose the train can track the number of passengers boarding at each station. The goal is for the program to identify and return the station with the lower number of passengers boarded by the end of the day.



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

2.a: **0**

2.b: **the last station 5**

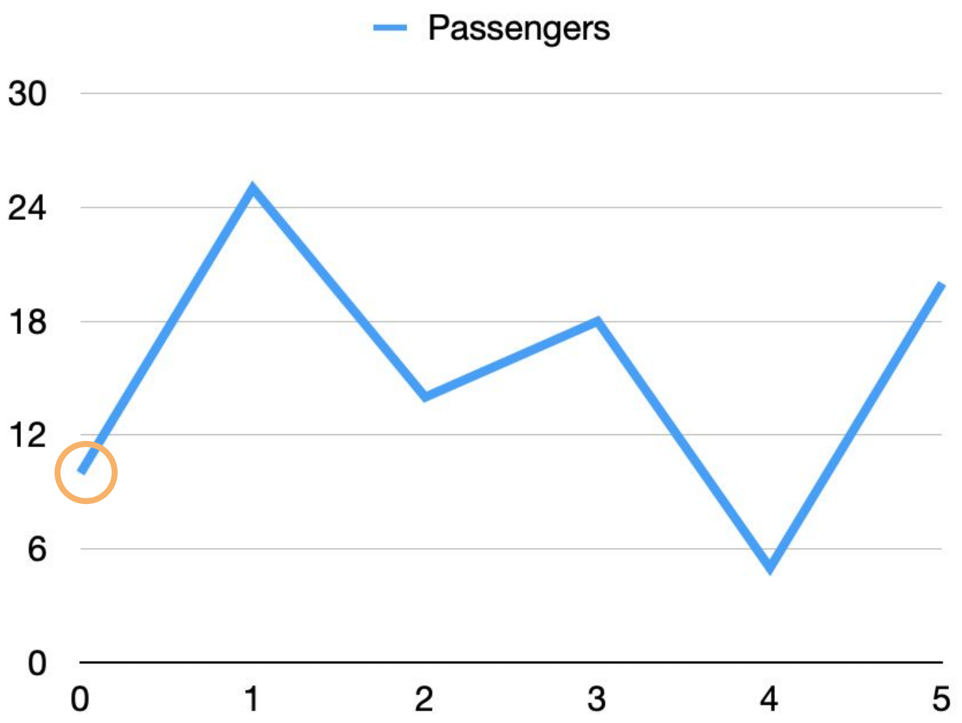
3. Move right; if number of people at current position is higher than the one of the previous station keep track of the station; keep moving right

Utility function example (global minimum)

```
station = 0
min = u(0)
x = 0
While x is less than 5:
   if u(x) < min:
       min = u(x)
       station = x
   Increment x by 1

return station
```

```
station = 0
min = 10
```

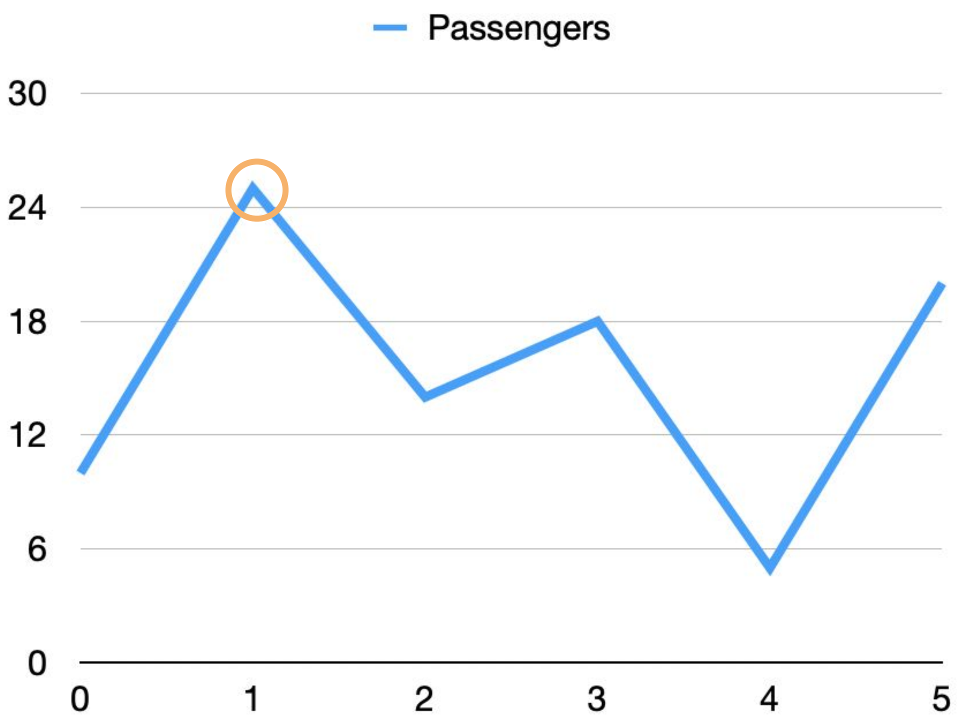


Utility function example (global minimum)

```
station = 0
min = u(0)
x = 0
While x is less than 5:
  if u(x) < min:
     min = u(x)
     station = x
  Increment x by 1

return station
```

```
station = 0
min = 10
```

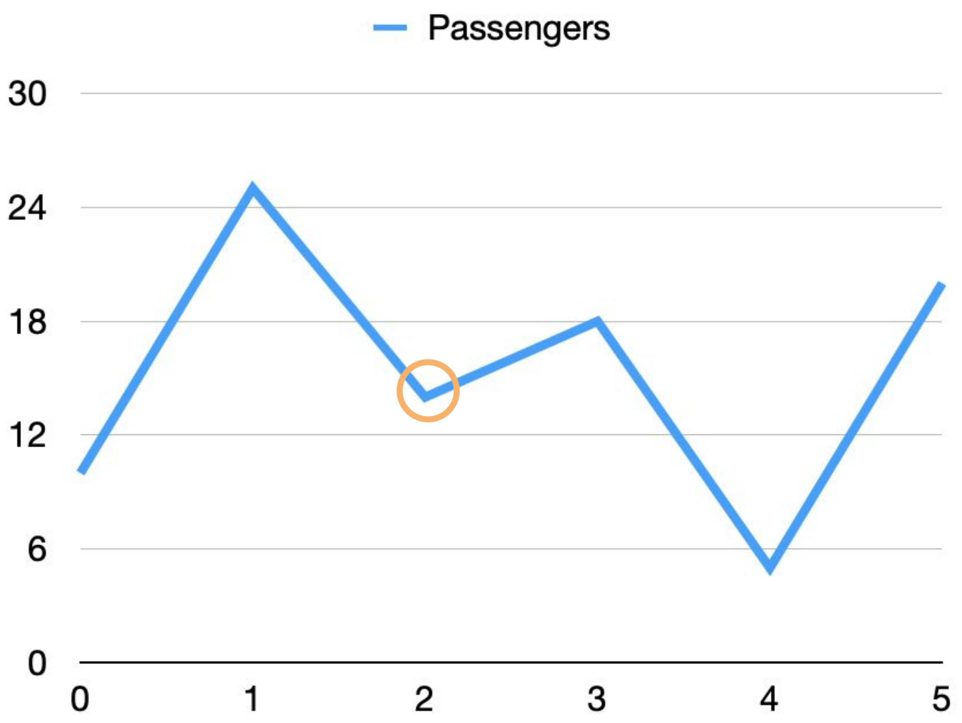


Utility function example (global minimum)

```
station = 0
min = u(0)
x = 0
While x is less than 5:
   if u(x) < min:
       min = u(x)
       station = x
   Increment x by 1

return station
```

```
station = 0
min = 10
```

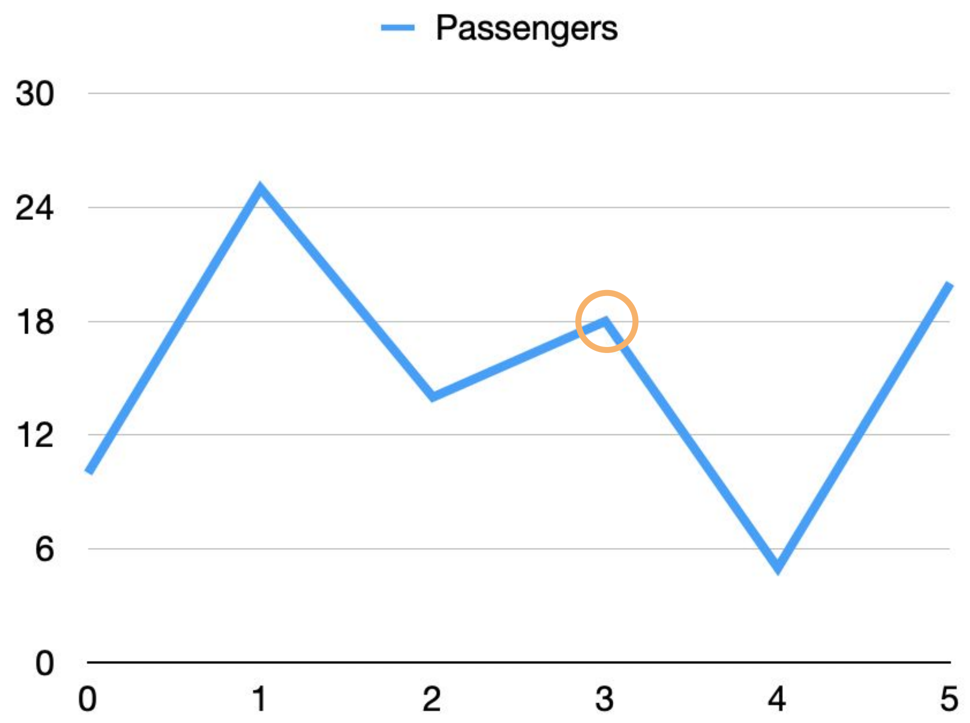


Utility function example (global minimum)

```
station = 0
min = u(0)
x = 0
While x is less than 5:
  if u(x) < min:
     min = u(x)
     station = x
  Increment x by 1

return station
```

```
station = 0
min = 10
```

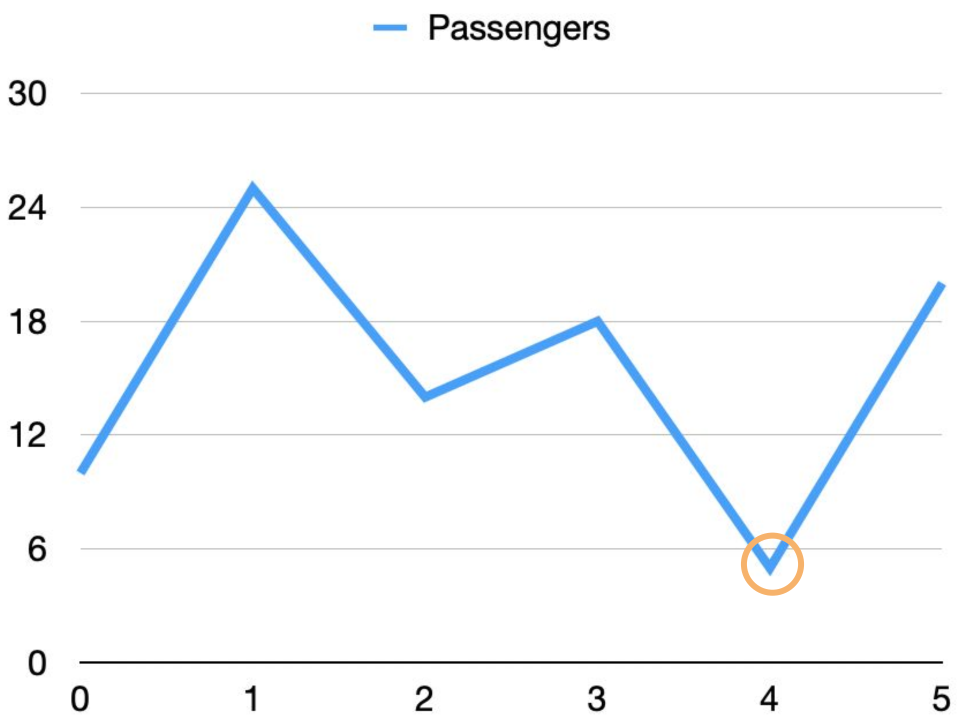


Utility function example (global minimum)

```
station = 0
min = u(0)
x = 0
While x is less than 5:
  if u(x) < min:
     min = u(x)
     station = x
  Increment x by 1

return station
```

```
station = 4
min = 5
```

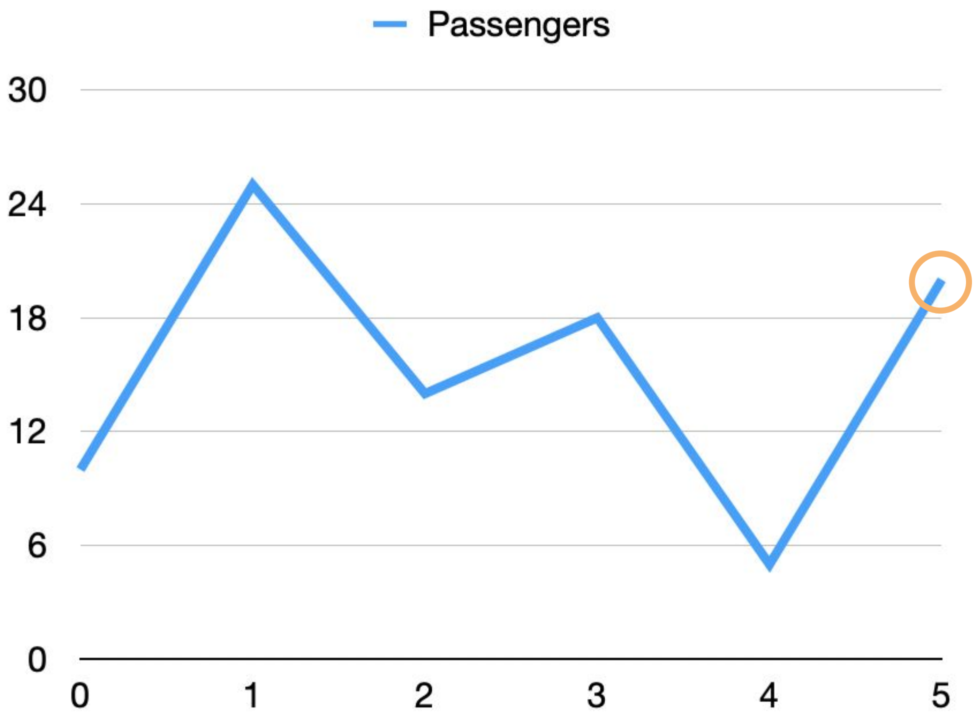


Utility function example (global minimum)

```
station = 0
min = u(0)
x = 0
While x is less than 5:
  if u(x) < min:
      min = u(x)
      station = x
  Increment x by 1

return station
```

```
station = 4
min = 5
```



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:** 0.1, 0.05

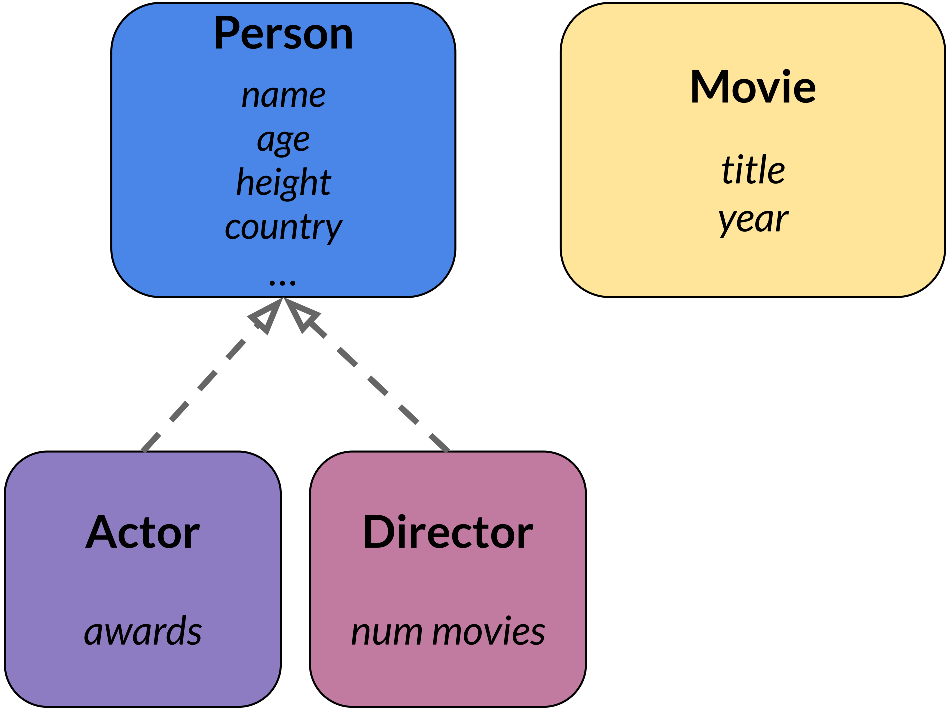
*An **entity** is something we may want to say something about*

Individuals (identify entities)

- An entity may have attributes
  (E.g. Alberto is a person with his tax id, birth date, height etc.)
- An entity may have relations with other entities
  (E.g. Alberto works at the University of Bologna)
- An entity may belong to a class
  (E.g. Alberto is a professor)

Entities and relations: example

Use abstraction to define entities.



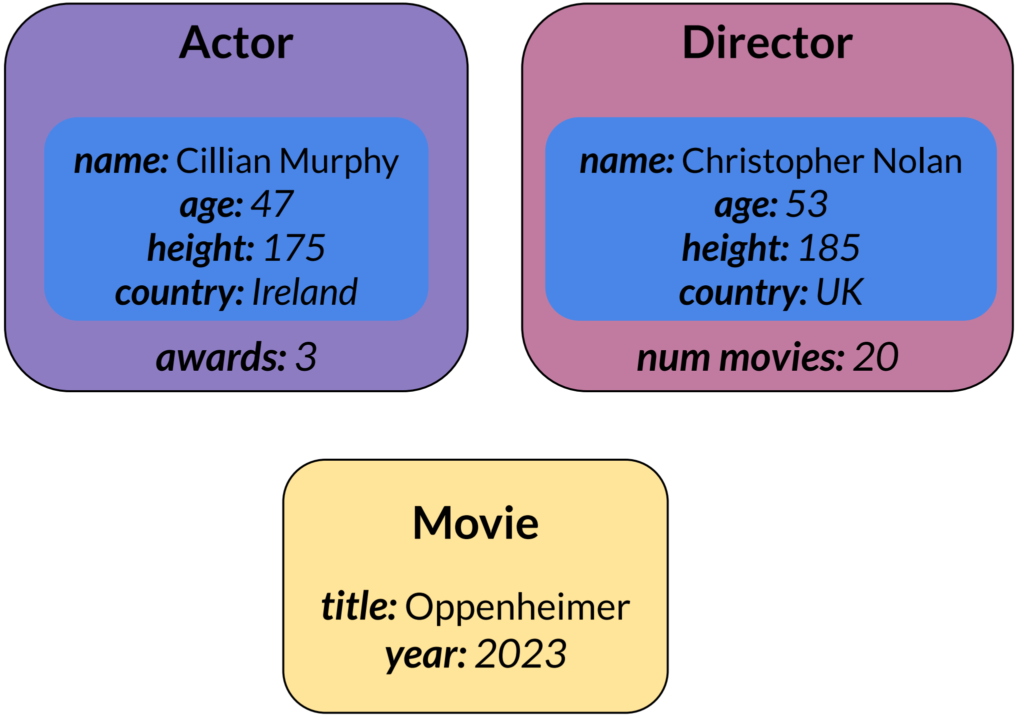
Entities and relations: example

Entities:

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

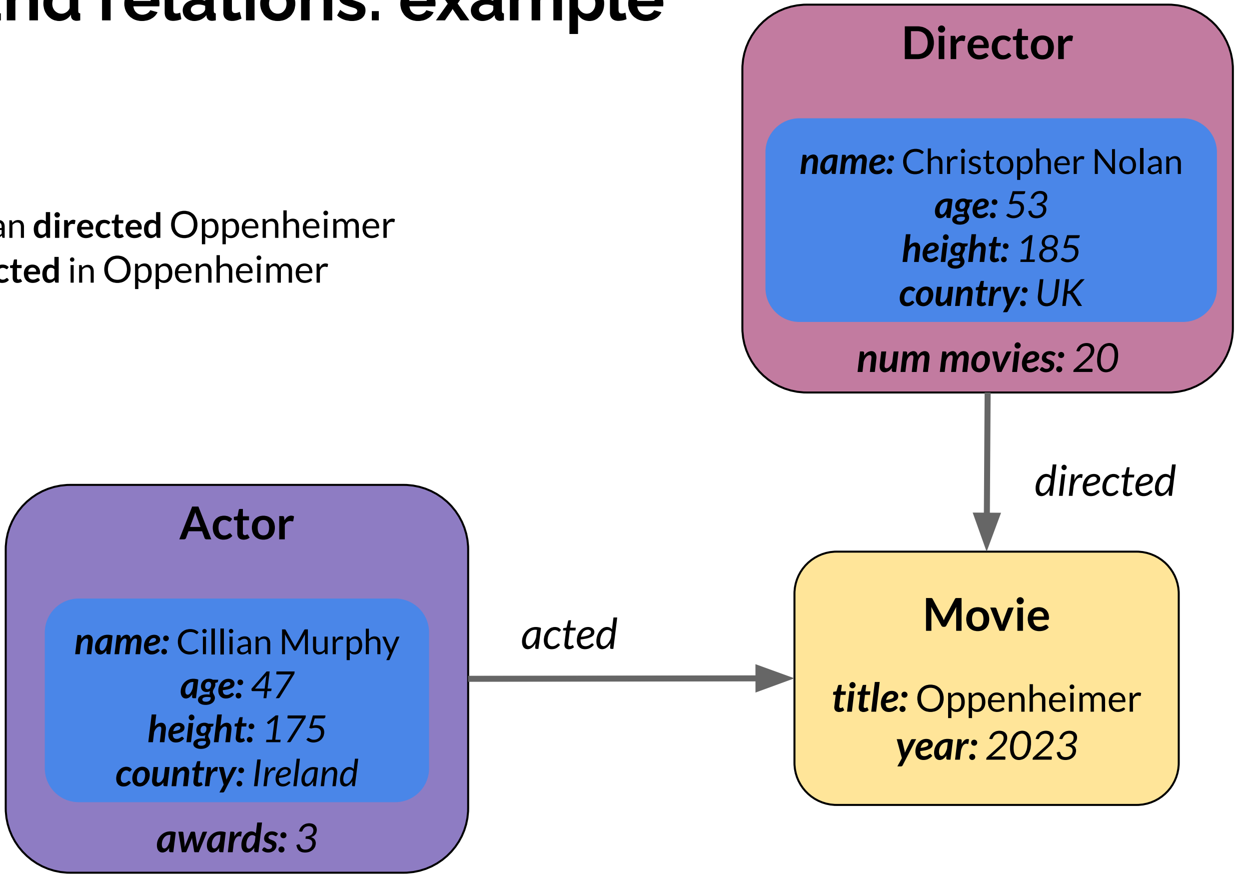
Christopher Nolan is a director with
53 years old, 185 (cm) tall from UK
he directed 20 movies

Oppenheimer is a movie released in 2023



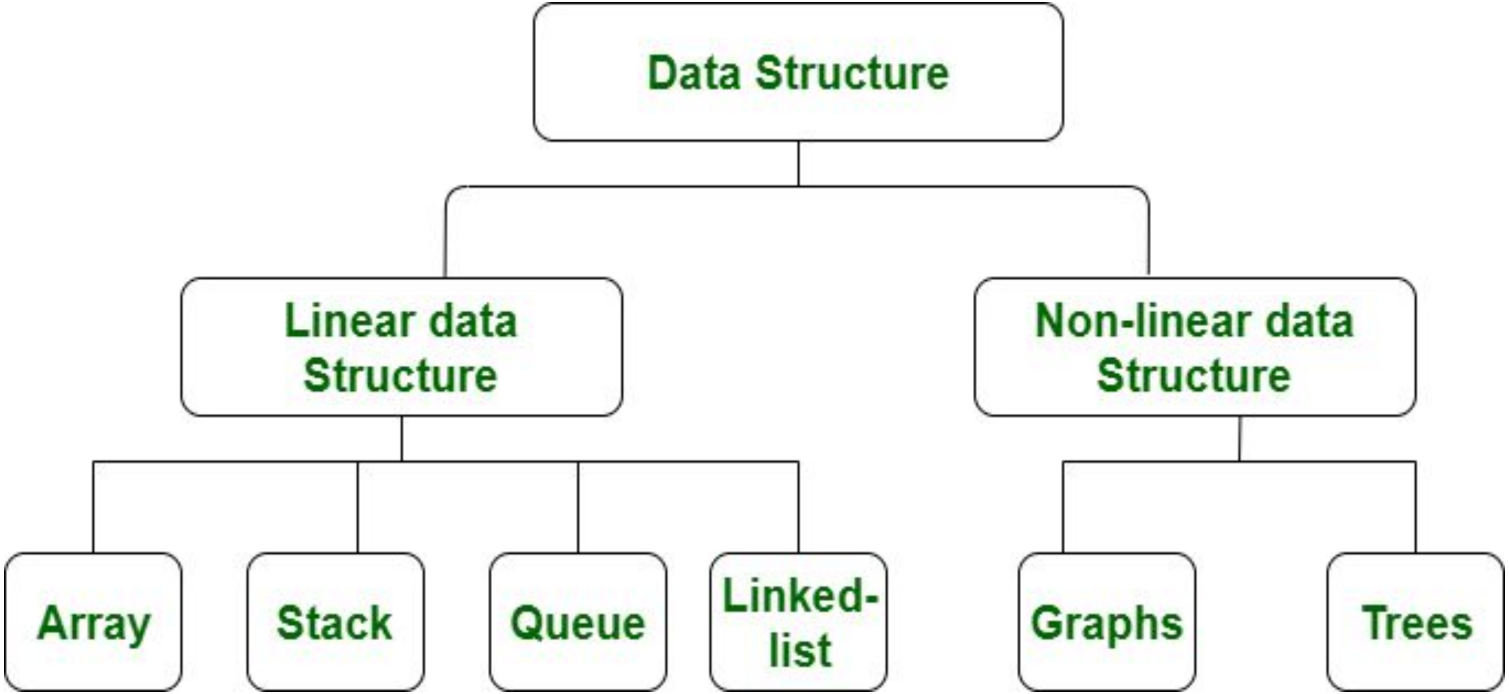
Entities and relations: example

Relations:
Christopher Nolan **directed** Oppenheimer
Cillian Murphy **acted** in Oppenheimer



Data structures

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

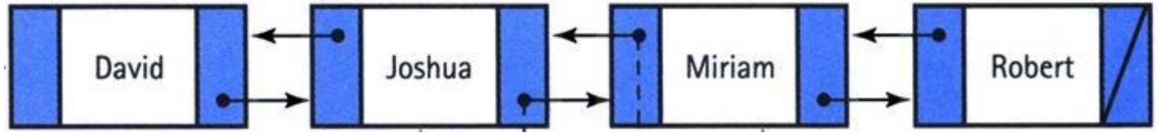


Linked list

In computer science, a **linked list** is a linear data structure in which elements, called nodes, are connected sequentially. Each node contains two parts:

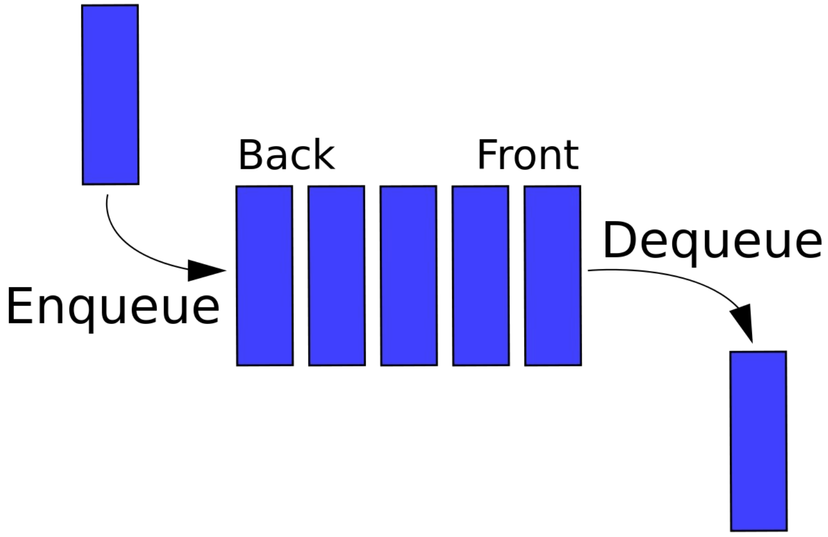
- **Data**: The value or information stored in the node.
- **Pointer (or reference)**: A reference to the next node in the sequence.

A linked list could be **singly, doubly, circular**



Queue

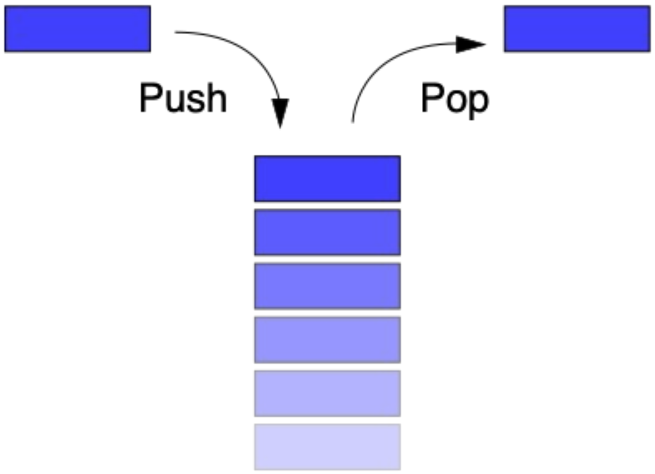
A queue is a collection of entities that are maintained in a sequence and can be modified by the addition of entities at one end of the sequence and the removal of entities from the other end of the sequence.



Stack

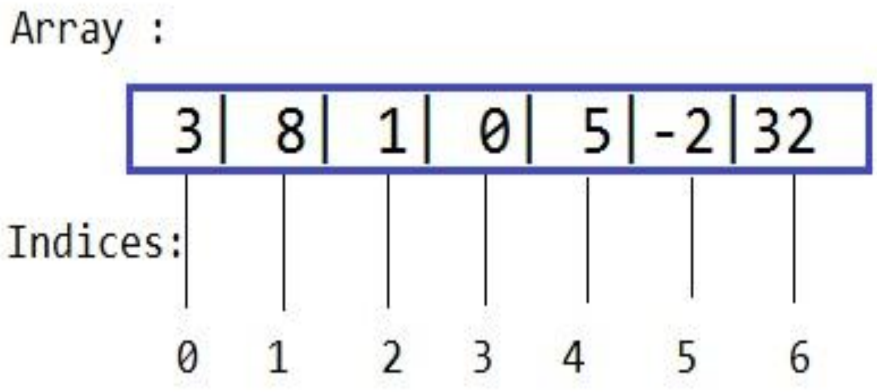
A stack is an abstract data type that serves as a collection of elements, with two main principal operations:

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



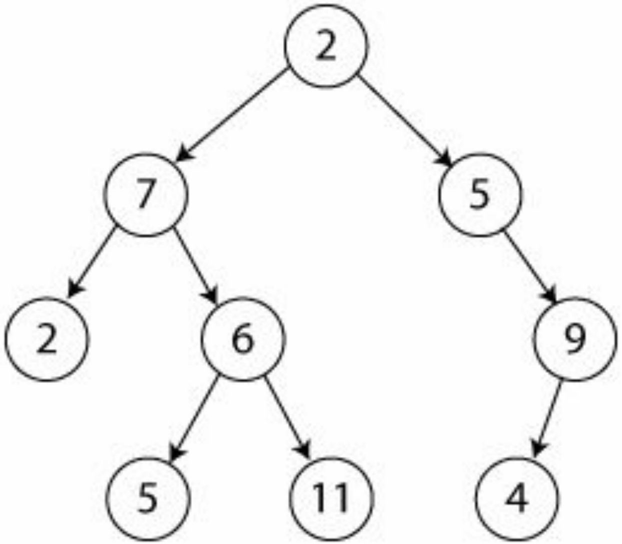
Array

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.



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.



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.

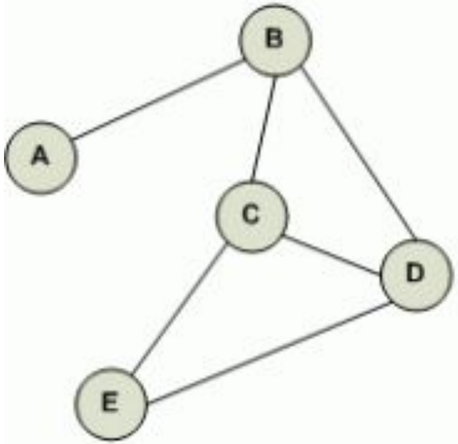


Fig 1. Undirected Graph

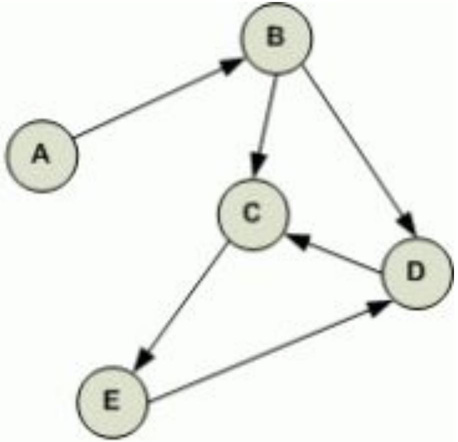
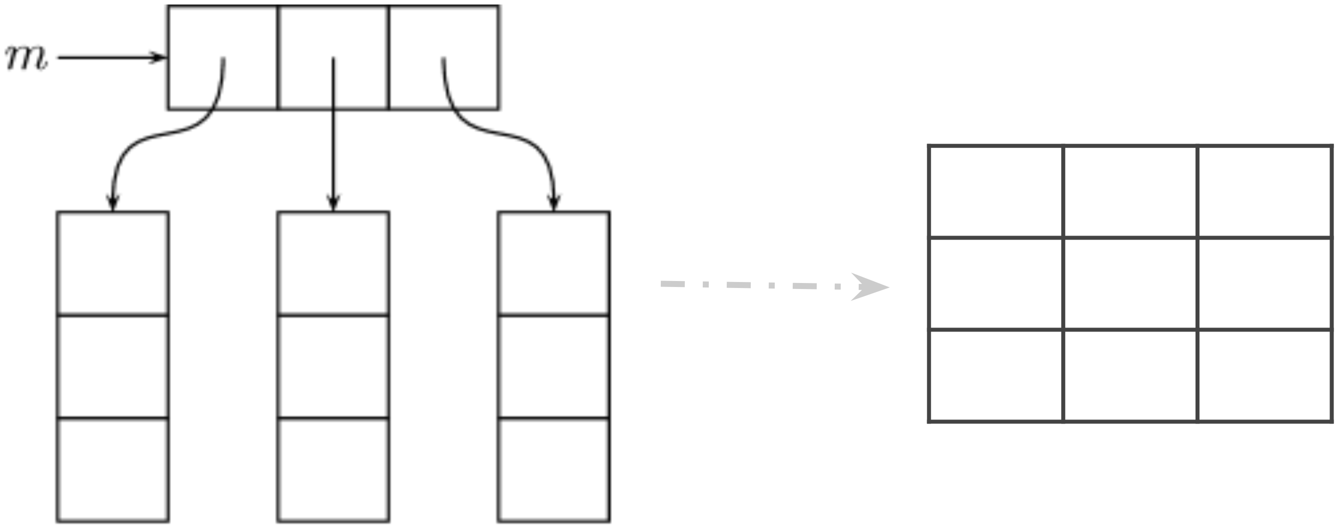


Fig 2. Directed Graph

Matrix

A matrix is a <u>bi-dimensional</u> array (an array of arrays)



Tensor

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

A tensor with one dimension can be thought of as a **vector**, a tensor with two dimensions as a **matrix** and a tensor with three dimensions can be thought of as a **cuboid**.

$$\epsilon_{ijk} =$$