Estructures de dades

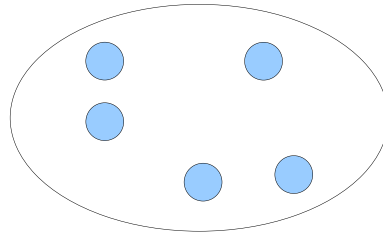
Informàtica
curs 23/24
Part 1/2

Prof. D. Tost

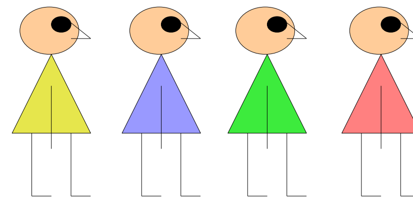
El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Estructures de dades

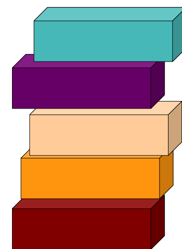
Sets



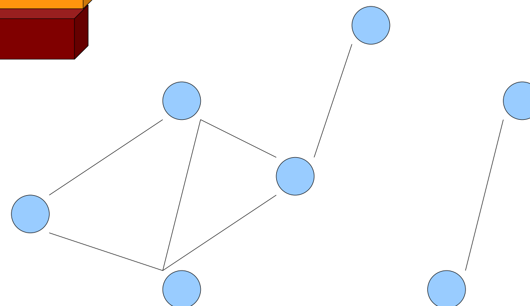
Cues



Piles



Grafs



El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Sets

Conjunts d'elements: no ordenats, no indexats, sense repeticions

Python proporciona una classe *builtin* per representar-los

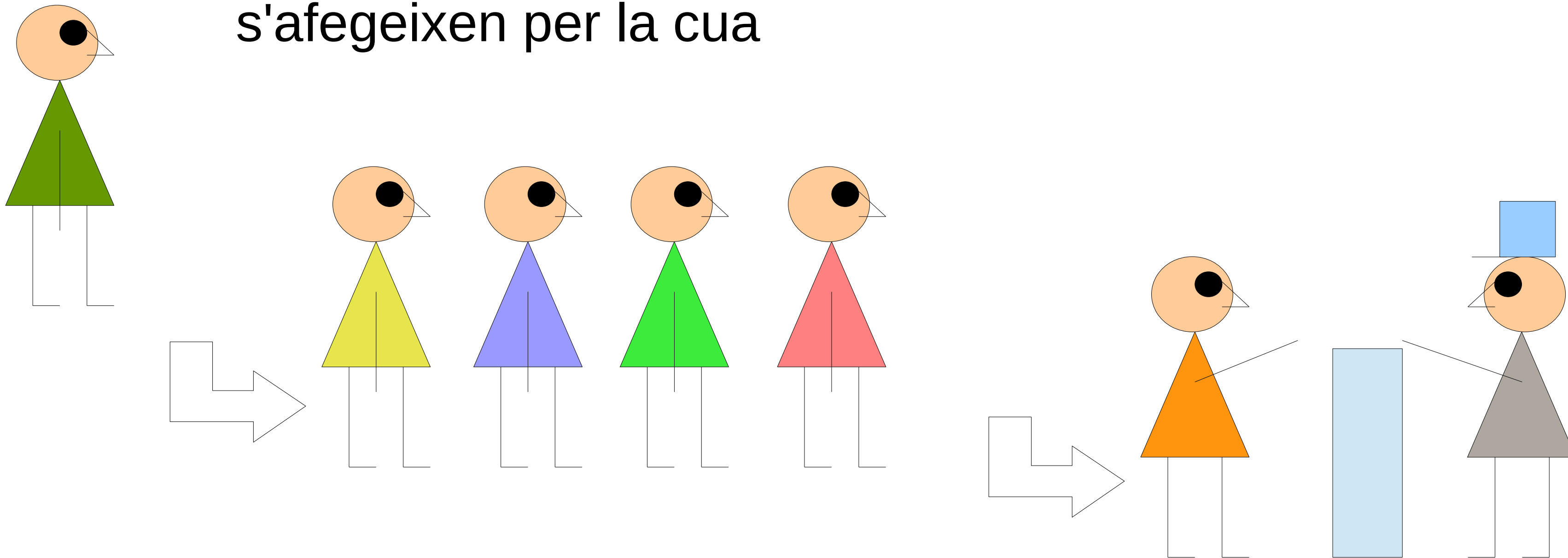
```
>>> s = set()
>>> s
set()
>>> s.add(5)
>>> s.add(5)
>>> s.add(5)
>>> s
{5}
>>> s2 = set([7, 8, 5, 9])
>>> s2.union(s)
{5, 7, 8, 9}
>>> s2.difference(s)
{8, 9, 7}
>>> s2.intersection(s)
{5}
>>> s2.pop()
8
>>> s2.remove(7)
>>> s2
{9, 5}
```


El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Cues

Estructura de dades de tipus FIFO (First In First Out). Té nombrosos usos, per exemple, en la gestió d'esdeveniments en una aplicació interactiva.

Els elements s'afegeixen per la cua



Els elements es treuen per davant

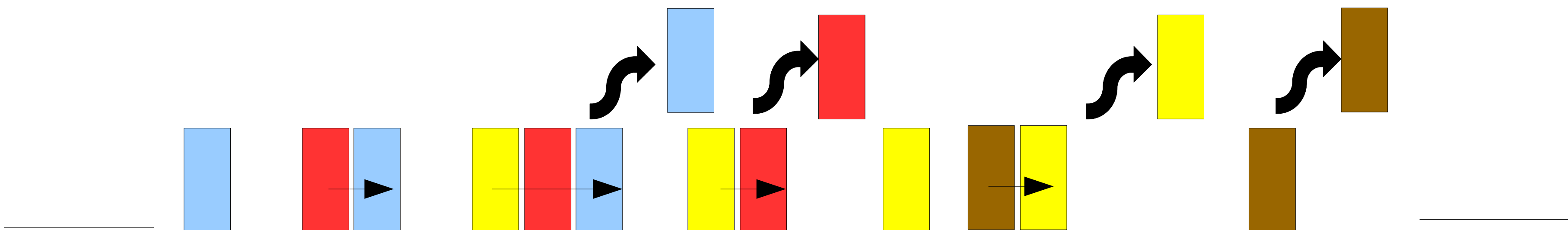
El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Cues

Una possible implementació

```
class Cua(object):

    def __init__(self):
        self.__elems = []

    def encuar(self, e):
        self.__elems.append(e)

    def desencuar(self):
        return self.__elems.pop(0)

    def __len__(self):
        return len(self.__elems)
```

```
>>> c = cua.Cua()
>>> c.encuar('blau')
>>> c.encuar('vermell')
>>> c.encuar('groc')
>>> len(c)
3
>>> c.desencuar()
'blau'
>>> c.desencuar()
‘vermell'
>>> c.encuar(‘marró')
>>> c.desencuar()
'groc'
>>> c.desencuar()
'marró'
>>> len(c)
0
```



El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Cues

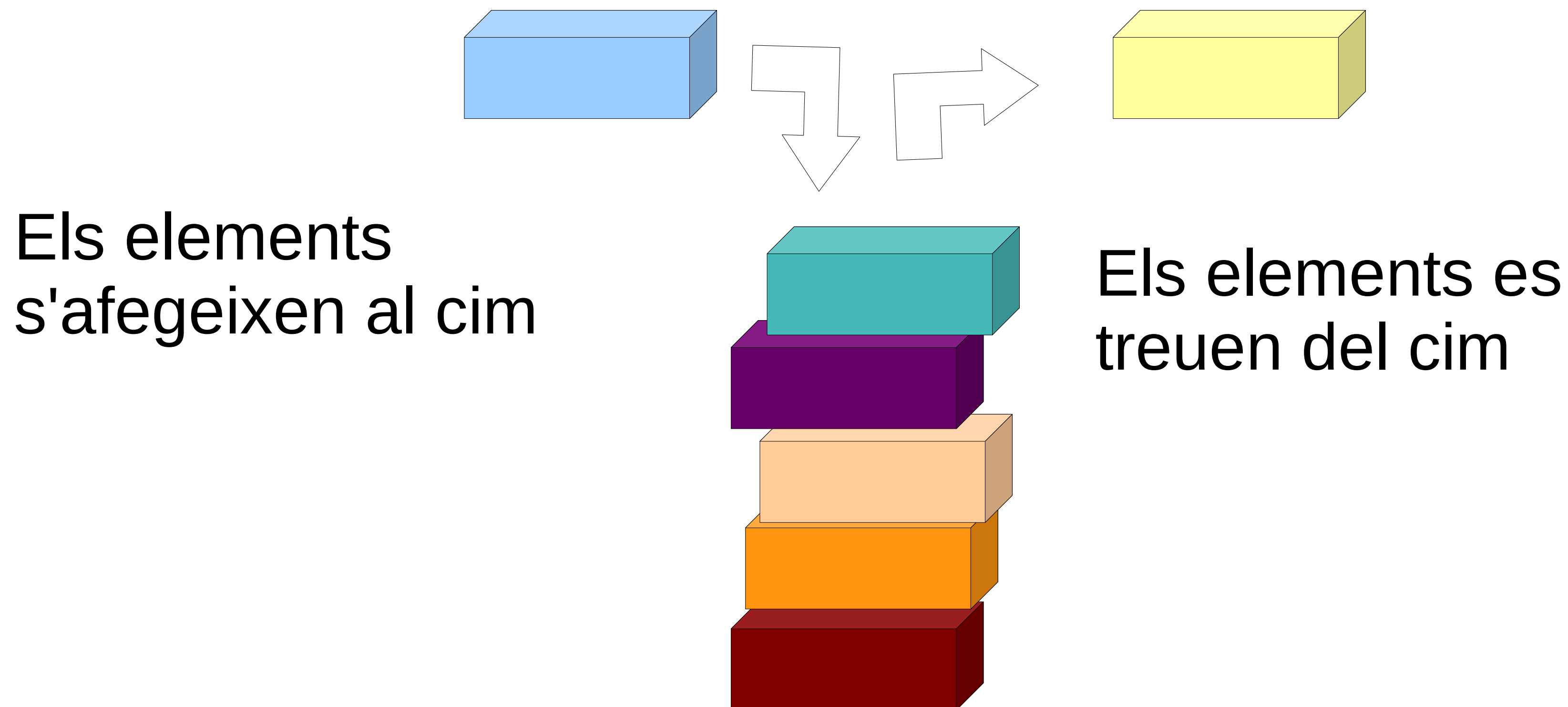
```
>>> import queue
>>> q = queue.Queue()
>>> q.put(1)
>>> q.put(14)
>>> q.put(15)
>>> q.get()
1
>>> q.put(67)
>>> q.get()
14
>>> q.get()
15
>>> q.get()
67
>>> q.empty()
True
```

Diversos mòduls de python implementen cues, oferint opcions per gestionar temps d'espera, límits d'elements, etc.

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Piles

Estructura de dades de tipus LIFO Last In First Out. Té nombrosos usos, per exemple en la gestió d'esdeveniments en una aplicació interactiva.



El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Piles

```
>>> import collections
>>> dq = collections.deque(['tom', 'anna', 'lluis'])
>>> dq.append(1)
>>> dq.append(12)
>>> dq.append(14)
>>> dq.pop()
14
>>> dq.pop()
12
>>> dq.pop()
1
>>> len(dq)
0
>>> dir(dq)
[.... 'append', 'appendleft', 'clear', 'copy',
'count', 'extend', 'extendleft', 'index', 'insert',
'maxlen', 'pop', 'popleft', 'remove', 'reverse',
'rotate']
```

Utilitzant popleft també es pot fer servir per cues.

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Grafs: definicions

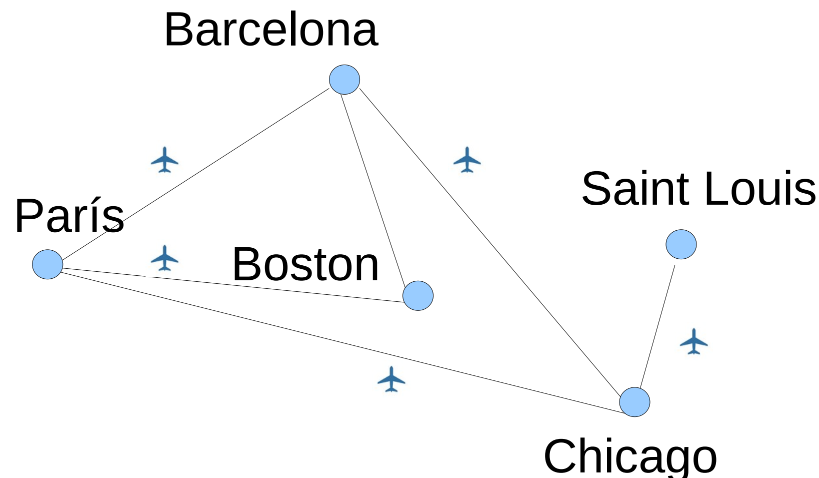
G = (V, E)

V = {**nodes**} (vèrtexs)
E = {**arestes** entre nodes} (arcs)

Un graf està associat a una relació binària `R` entre nodes: hi ha una aresta entre dos nodes `a` i `b` si `a R b`

Els nodes i les arestes d'un graf poden tenir una o més etiquetes (informació associada)

Exemple: nodes: Barcelona, Boston, Chicago, Saint Louis, Paris (ciutats)
relació: existència d'un vol directe entre dues ciutats
arestes: hi ha una aresta entre a i b si hi ha un vol directe entre a i b.



El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Grafs: definicions

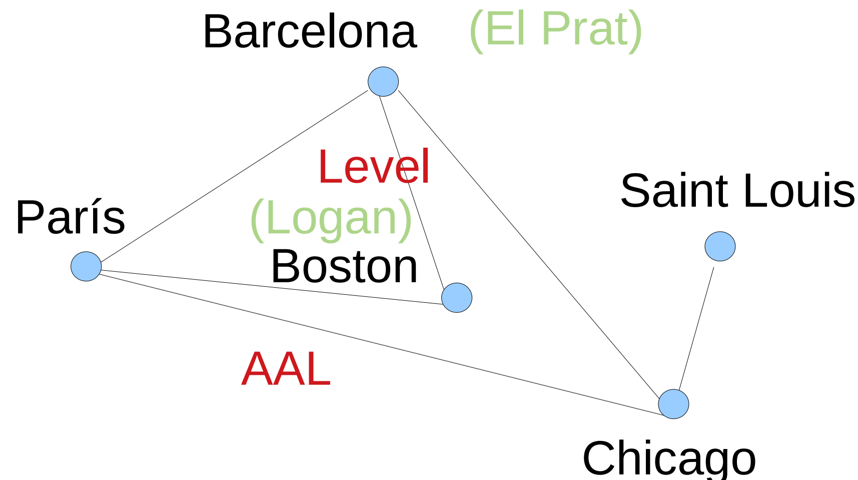
G = (V, E)

V = {**nodes**} (vèrtexs)
E = {**arestes** entre nodes} (arcs)

Un graf està associat a una relació binària `R` entre nodes: hi ha una aresta entre dos nodes `a` i `b` si `a R b`

Els nodes i les arestes d'un graf poden tenir una o més etiquetes (informació associada)

Exemple: nodes: Barcelona, Boston, Chigago, Saint Louis, Paris (ciutats)
relació: existència d'un vol directe entre dues ciutats
arestes: hi ha una aresta entre a i b si hi ha un vol directe entre a i b.



En vermell atribut d'aresta (companyia aèria)
En verd atribut de node (nom de l'aeroport).

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Grafs: tipus de grafs

Graf no dirigit: les arestes no tenen orientació (u, v) = (v, u)
Exemple: R representa «ser amic de»

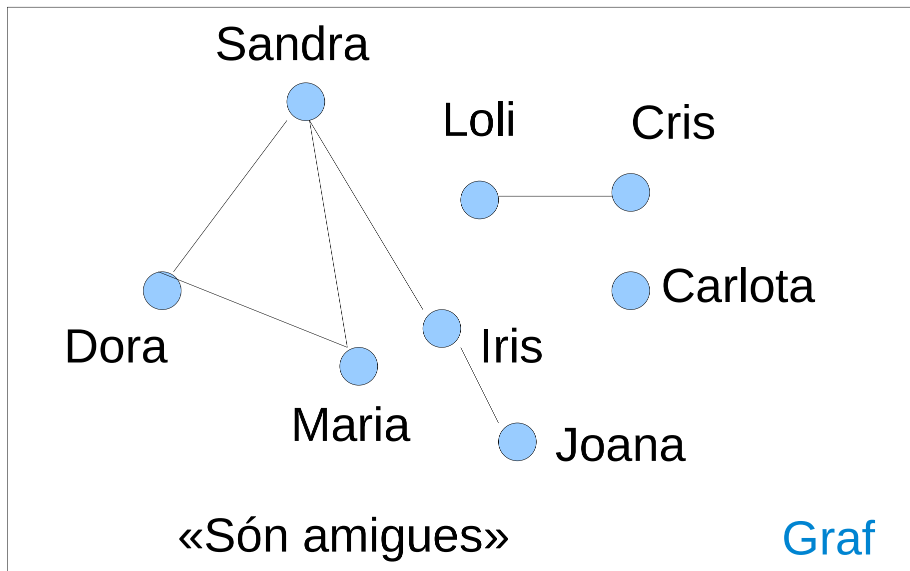
Graf dirigit (digraf): les arestes tenen orientació (u, v) ≠ (v, u)
Exemple: R representa «ser parent (pare o mare) de»

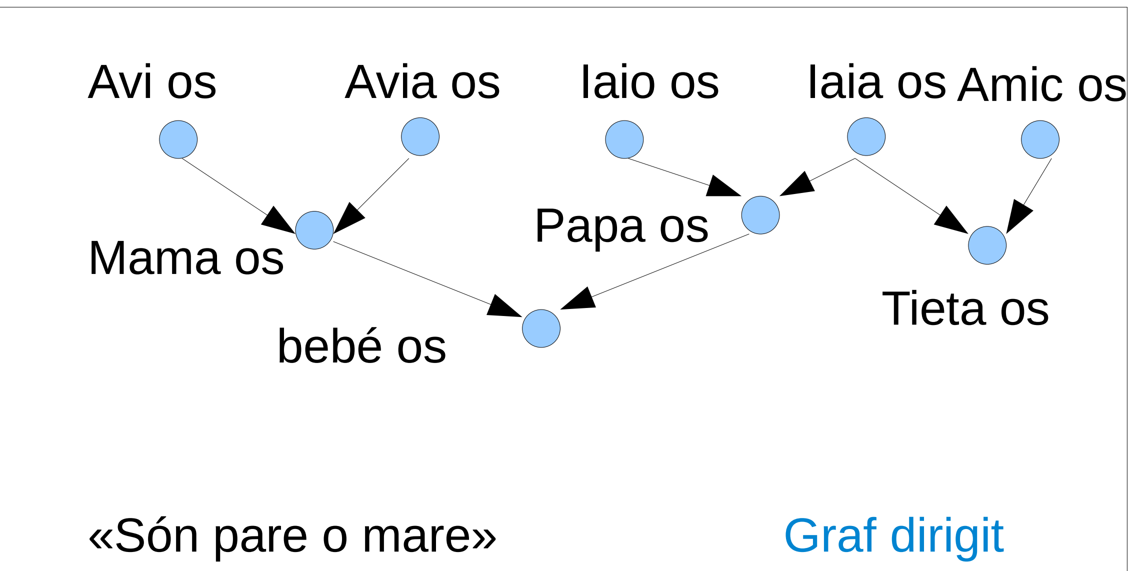
Multigraf (no dirigit): hi pot haver més d'una aresta entre dos nodes
Exemple: R representa «haver quedat amb»

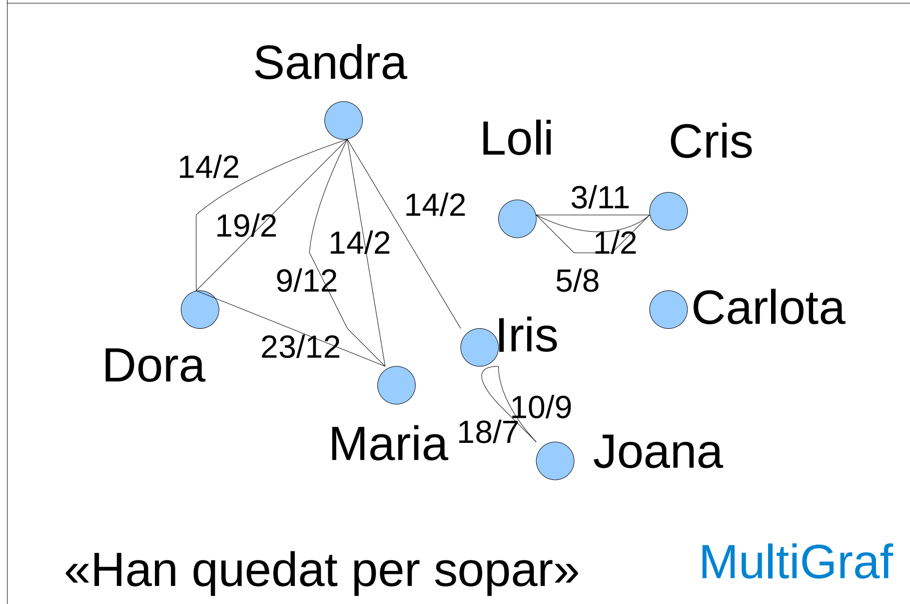
Multigraf dirigit: les arestes tenen orientació i hi pot haver més d'una entre dos nodes
Exemple: R representa «haver invitat a»

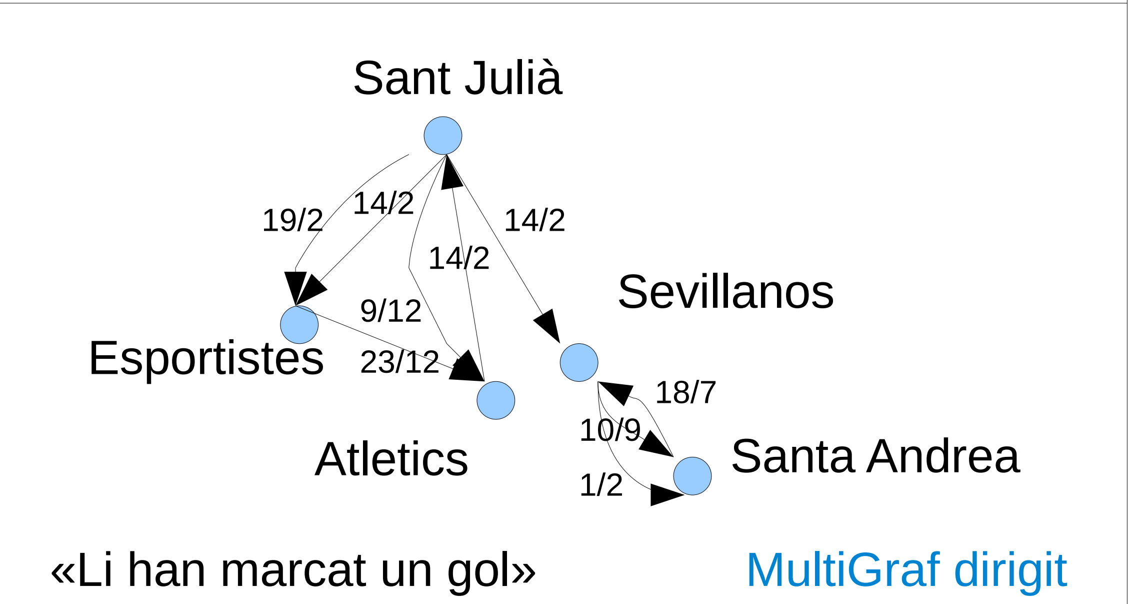
El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Grafs: exemples









El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Grafs: definicions

Label, etiqueta o atribut d’una aresta: informació complementària sobre la aresta

Label, etiqueta o atribut d’un vèrtex/node: informació complementària sobre el node

Nodes/vèrtexs adjacents: N i M són adjacents si existeix una aresta entre ells

Nodes/vèrtexs veïns d'un node N: tots els nodes M_i tals que existeix l'aresta (N, M_i)

Ordre d'un graf: nombre de nodes

Mida d'un graf: nombre d'arestes

Grau d'un node/vèrtex: nombre d'arestes del node (vèrtex)

Camí (path) entre dos nodes/vèrtexs: successió d'arestes que porten de l'un a l'altre

Camí simple (simple path) entre dos nodes: successió d'arestes que porten de l'un a l'altre sense tornar a passar per cap node

Camí més curt entre dos nodes: de tots els camins entre dos nodes, els que passen per menys nodes (o d’altres criteris relacionats amb les etiquetes de les arestes)

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Networkx: documentació

- Una llibreria molt extensa
- Utilitzeu la documentació: https://networkx.github.io/documentation/stable/. Conté tutorials i sobretot el manual de referència

- En llegir la documentació, identifiqueu els enllaços (sovint en vermell)
- Distingiu entre **mètodes del graf** i **funcions que reben un graf com a paràmetre**
- Alerta amb les versions, poden haver-hi canvis

- Per instal.lar:

```
pip3 install networkx
```

- Instal.leu també matplotlib per visualitzar els grafs:

```
pip3 install matplotlib
```

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Networkx: documentació

- Utilitzeu la documentació: https://networkx.github.io/documentation/stable/
- Identifiqueu els enllaços
- Distingiu entre **mètodes del graf** i **funcions que reben un graf com a paràmetre**
- Alerta amb les versions, poden haver-hi canvis

Reference — Netwo...
networkx.readthedocs.io/en/stable/reference/index.html
Más visitados Getting Started Informàtica Google Calendar Campus Virtual UPC: ... 2 - PROBL

NetworkX
stable
Search docs

Overview
Download
Installing
Tutorial
Reference
Introduction
Graph types
Algorithms
Functions
Graph generators
Linear algebra
Converting to and from other data formats
Relabeling nodes
Reading and writing graphs
Drawing
Exceptions
Utilities
License
Citing
Credits
Glossary
Reference
Testing
Developer Guide
History
Bibliography
NetworkX Examples

Read the Docs v: stable

Docs » Reference

Reference

Release: 1.11

Date: January 31, 2016

- Introduction
  - NetworkX Basics
  - Nodes and Edges
- Graph types
  - Which graph class should I u
  - Basic graph types
- Algorithms
  - Approximation
  - Assortativity
  - Bipartite
  - Blockmodeling
  - Boundary
  - Centrality
  - Chordal
  - Clique
  - Clustering
  - Coloring
  - Communities
  - Components
  - Connectivity
  - Cores
  - Cycles
  - Directed Acyclic Graphs
  - Distance Measures
  - Distance-Regular Graphs
  - Dominance
  - Dominating Sets
  - Eulerian
  - Flows
  - Graphical degree sequence
  - Hierarchy
  - Hybrid
  - Isolates
  - Isomorphism

Graph – Undirected ...
networkx.readthedocs.io/en/stable/reference/classes.graph.html
Más visitados Getting Started Informàtica Google Calendar Campus Virtual UPC: ... 2 - PROBLEMA U OPO... gitlab SGCREB - Seriou

NetworkX
stable
Search docs

Overview
Download
Installing
Tutorial
Reference
Introduction
Graph types
Which graph class should I use?
Basic graph types
Graph – Undirected graphs with self loops
Methods
DiGraph - Directed graphs with self loops
Methods
MultiGraph - Undirected graphs with self loops and parallel edges
Methods
MultiDiGraph - Directed graphs with self loops and parallel edges
Methods
Algorithms
Functions
Graph generators
Linear algebra
Converting to and from other data formats
Relabeling nodes
Reading and writing graphs
Drawing
Exceptions

Docs » Reference » Graph types » Graph – Undirected graphs with self loops   Edit on GitHub

Graph – Undirected graphs with self loops

Overview

Graph(*data=None, **attr*) [source]

Base class for undirected graphs.

A Graph stores nodes and edges with optional data, or attributes.

Graphs hold undirected edges. Self loops are allowed but multiple (parallel) edges are not.

Nodes can be arbitrary (hashable) Python objects with optional key/value attributes.

Edges are represented as links between nodes with optional key/value attributes.

Parameters:
- **data** (*input graph*) – Data to initialize graph. If data=None (default) an empty graph is created. The data can be an edge list, or any NetworkX graph object. If the corresponding optional Python packages are installed the data can also be a NumPy matrix or 2d ndarray, a SciPy sparse matrix, or a PyGraphviz graph.
- **attr** (*keyword arguments, optional (default= no attributes)*) – Attributes to add to graph as key=value pairs.

See also

`DiGraph()`, `MultiGraph()`, `MultiDiGraph()`

Examples

Create an empty graph structure (a "null graph") with no nodes and no edges.

```
>>> G = nx.Graph()
```

G can be grown in several ways.

Nodes:

Add one node at a time:

```
>>> G.add_node(1)
```

El contingut està disponible sota la lli

Networkx: documentació

Network Analysis in Python

2.5

Search docs

- Install
- Tutorial
- Gallery
- Reference
  - Introduction
  - Graph types
    - Which graph class should I use?
    - Basic graph types
    - Graph Views
    - Filters
  - Algorithms
  - Functions
  - Graph generators
  - Linear algebra
  - Converting to and from other data formats
  - Relabeling nodes
  - Reading and writing graphs
  - Drawing
  - Randomness
  - Exceptions
  - Utilities
  - Glossary
- Developer Guide
- Release Log
- License
- About Us

» Reference » Graph types

Graph types

NetworkX provides data structures and methods for storing graphs.

All NetworkX graph classes allow (hashable) Python objects as nodes and any Python object can be assigned as an edge attribute.

The choice of graph class depends on the structure of the graph you want to represent.

Which graph class should I use?

Networkx Class	Type	Self-loops allowed	Parallel edges allowed
Graph	undirected	Yes	No
DiGraph	directed	Yes	No
MultiGraph	undirected	Yes	Yes
MultiDiGraph	directed	Yes	Yes

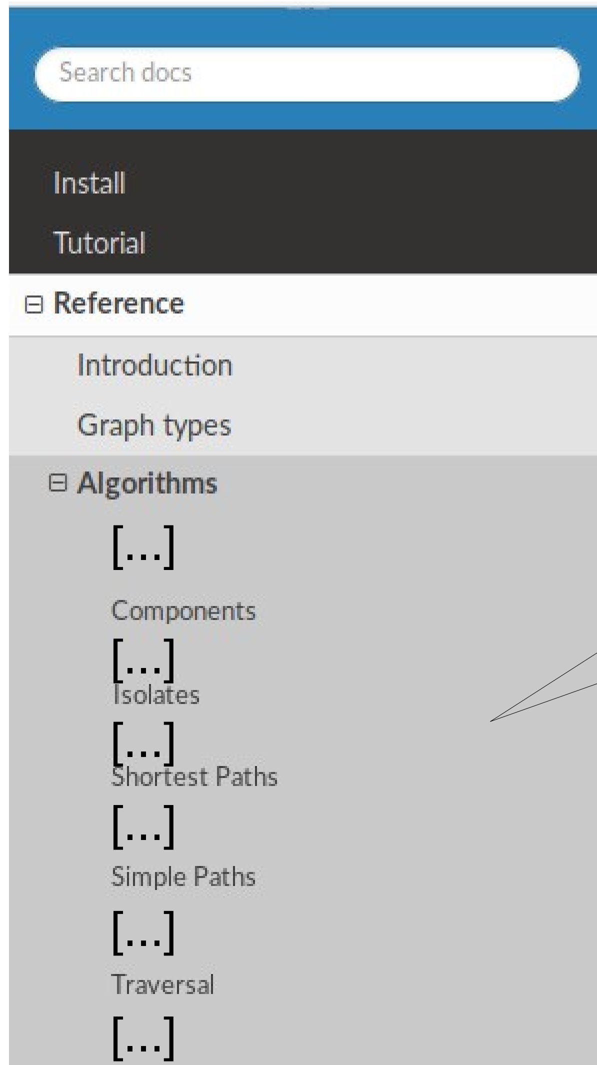
Basic graph types

- Graph—Undirected graphs with self loops
  - Overview
  - Methods
- DiGraph—Directed graphs with self loops
  - Overview
  - Methods
- MultiGraph—Undirected graphs with self loops and parallel edges
  - Overview
  - Methods
- MultiDiGraph—Directed graphs with self loops and parallel edges

Aquí trobareu els mètodes dels diferents tipus de grafs.

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Networkx: documentació



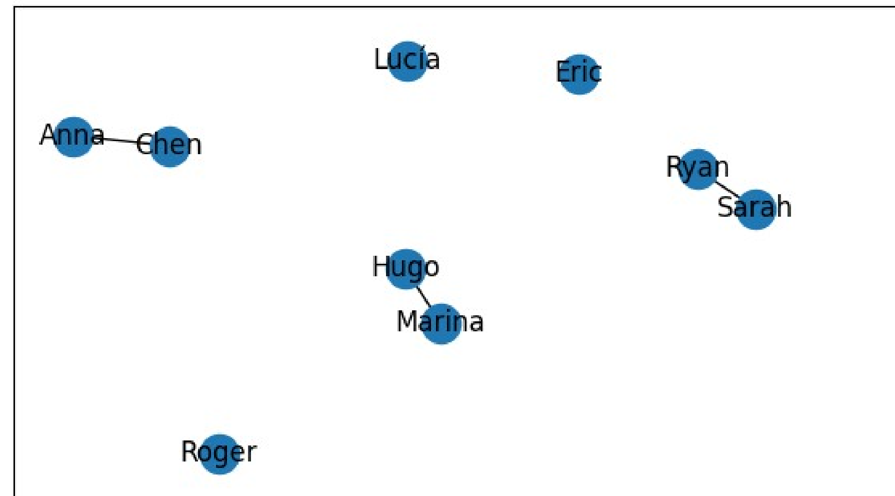
Aquí trobareu funcions que tenen com a paràmetre un graf. Aquestes pestanyes de la documentació són les imprescindibles... però moltes altres són també interessants

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Networkx: creació d'un graf simple

```
>>> import networkx as nx
>>> g = nx.Graph()
>>> g.add_node('Roger')
>>> g.add_nodes_from([‘Sarah’, ‘Eric’,‘Lucía’, 'Hugo'])
>>> g.add_edge('Chen', 'Anna')
>>> g.add_edges_from([('Sarah', 'Ryan'), ('Hugo', 'Marina')])
>>> import matplotlib.pyplot as plt
>>> nx.draw_networkx(g)
>>> plt.show()
```

- `g.add_node('Roger')` ← 1 sol node
- `g.add_nodes_from(...)` ← Tots els nodes d'una llista
- `g.add_edge('Chen', 'Anna')` ← 1 sola aresta: si el node no era al graf, l'afegeix
- `g.add_edges_from(...)` ← Totes les arestes d'una llista



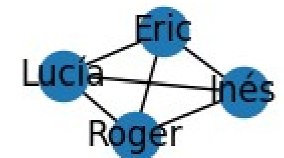
El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Networkx: creació d'un graf simple

```
... continua ...
```

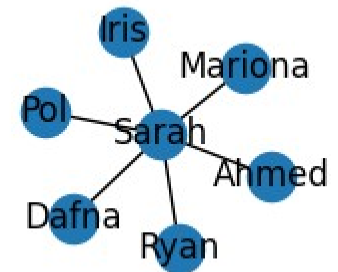
```
>>> llista = ['Lucía', 'Eric', 'Roger', 'Inés']
>>> for i, node in enumerate(llista):
...      for vei in llista[i+1:] :
...           g.add_edge(node, vei)
...
```

← Tots amb tots



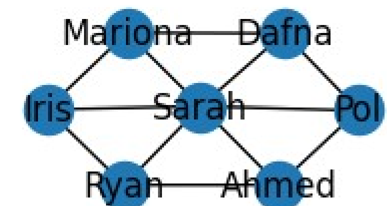

```
>>> llista = ['Ahmed', 'Pol', 'Dafna', 'Mariona', 'Iris']
>>> for node in llista:
...    g.add_edge('Sarah', node)
...
```

← Un node amb tots els altres (estrella)



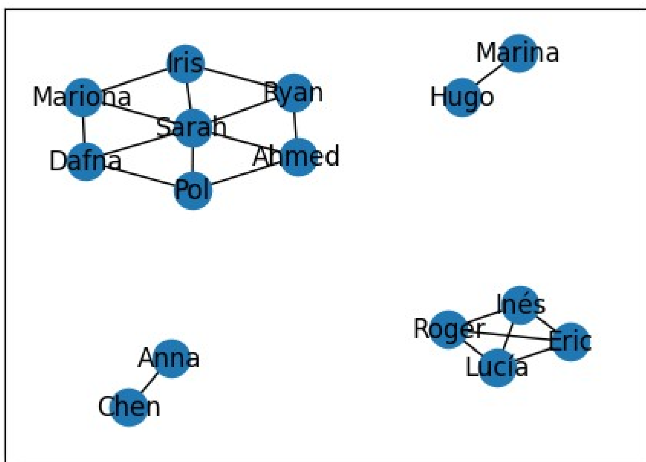

```
>>> llista = ['Ahmed', 'Pol', 'Dafna', 'Mariona', 'Iris', 'Ryan']
>>> for i in range(1, len(llista)):
...      g.add_edge(llista[i-1], llista[i])
...
>>> g.add_edge(llista[-1], llista[0])
```

Un camí circular



El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Networkx i diccionaris



> **Note**
>
> NetworkX uses `dicts` to store the nodes and neighbors in a graph.

La implementació de networkx es basa en diccionaris.

```
>>> g['Chen']
AtlasView({'Anna': {}})
>>> g['Sarah']
AtlasView({'Ryan': {}, 'Ahmed': {}, 'Pol': {}, 'Dafna':
{}, 'Mariona': {}, 'Iris': {}})
```

Els veis d'un node s'emmagatzemen en un diccionari. La clau del diccionari és el node i els valors són també diccionaris. Aquests diccionaris de veïns tenen com a claus els veïns i com a valors diccionaris d'atributs d'arestes. En els diccionaris d'atributs, les claus són els noms dels atributs i els valors són els dels atributs en cada aresta.

```
>>> g['Sarah']['Ryan']['any'] = 2013
>>> g['Sarah']['Ryan']['tipus'] = 'familia'
>>> g['Sarah']['Ahmed']['tipus'] = 'amistat'
>>> g['Sarah']['Ahmed']['any'] = '2001'
>>> g['Sarah']
AtlasView({'Ryan': {'any': 2013, 'tipus': 'familia'}, 'Ahmed': {'tipus': 'amistat',
'any': '2001'}, 'Pol': {}, 'Dafna': {}, 'Mariona': {}, 'Iris': {}})
>>> g['Sarah']['Ryan']['tipus']
'familia'
>>> g['Sarah']['Ahmed']['any']
'2001'
```

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Mètodes dels grafs
Atributs de nodes

```
>>> 'Tom' in g
False
```

Consultar si un node és del graf

```
>>> g.add_node('Tom', edat=45, aficions=['cartes'])
```

En crear un node s'especifiquen els seus atributs

```
>>> g.nodes['Tom’]
{'edat': 45, 'aficions': ['cartes']}
```

Consultar els atributs d'un node

```
>>> g.nodes['Tom']['edat']
45
```

Consultar el valor d'un atribut d'un node

```
>>> g.nodes[‘Sarah’][‘edat’] = 36
```

Crear un atribut per un node ja existent

```
>>> llista = []
>>> for node in g.nodes:
...     if 'edat' in g.nodes[node] and g.nodes[node]['edat'] > 40:
...         llista.append(node)
...
>>> llista
[‘Tom']
```

Llistar els nodes que tenen l'atribut edat i amb un valor superior a 40

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Mètodes dels grafs
Atributs de nodes

Consultar el valor de l’atribut ‘edat’ per tots els nodes

```
>>> g.nodes(data='edat')
NodeDataView({'Roger': None, 'Sarah': 36, 'Eric': None, 'Lucía':
None, 'Hugo': None, 'Chen': None, 'Anna': None, 'Ryan': None,
'Marina': None, 'Inés': None, 'Ahmed': None, 'Pol': None, 'Dafna':
None, 'Mariona': None, 'Iris': None, 'Tom': 45}, data='edat')
```

Llistar els nodes que tenen l’atribut edat i amb un valor superior a 40 (versió 2)

```
>>> it = filter(lambda t: t[1]!=None and t[1]>40, g.nodes(data='edat'))
>>> list(map(lambda t: t[0], it)
[‘Tom’]
```

L’iterador de g.nodes(data=’edat’) itera sobre tuples (nom, edat). Per això t és un tuple (nom, edat).

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Mètodes dels grafs
Atributs d'arestes

```
>>> g.add_edge('Tom', 'Joana', any=1993, tipus='professional')
>>> g['Tom']
AtlasView({'Joana': {'any': 1993, 'tipus': 'professional'}})
>>> g['Tom']['Joana']['any']
1993
```

En crear una aresta s'especifiquen els seus atributs

```
>>> g['Ahmed']['Pol']['any'] = 2015
```

Es poden afegir atribus a arestes ja creades

```
>>> g.add_edges_from([('Ali', 'Paula'), ('Paula', 'Johnny'), ('Johnny', 'Pol')], any=2020, tipus='estudis')
```

Es poden crear diverses arestes amb el mateix atribut a la vegada

```
>>> g.edges(data='any', default = -1)
EdgeDataView([('Roger', 'Lucía', -1), ('Roger', 'Eric', -1), ('Roger', 'Inés', -1), ('Sarah', 'Ryan', -1), ('Sarah', 'Ahmed', -1), ('Sarah', 'Pol', -1), ('Sarah', 'Dafna', -1), ('Sarah', 'Mariona', -1), ('Sarah', 'Iris', -1), ('Eric', 'Lucía', -1), ('Eric', 'Inés', -1), ('Lucía', 'Inés', -1), ('Hugo', 'Marina', -1), ('Chen', 'Anna', -1), ('Ryan', 'Iris', -1), ('Ryan', 'Ahmed', -1), ('Ahmed', 'Pol', -1), ('Pol', 'Dafna', -1), ('Pol', 'Johnny', 2020), ('Dafna', 'Mariona', -1), ('Mariona', 'Iris', -1), ('Tom', 'Joana', 1993), ('Ali', 'Paula', 2020), ('Paula', 'Johnny', 2020)])
```

Es pot llistar el valor d'un atribut d'aresta per totes les arestes

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Networkx grafs i diccionaris

Es pot recorre un graf seguint la seva estructura de diccionari:

```
>>> for node in g:
...     print("Node:", node)
...     for vei in g[node]:
...          print("\t Vei: ", vei)
...          for atribut in g[node][vei]:
...                print("\t\t Atribut: ", atribut, " Valor: ", g[node][vei][atribut])
...

Node: Roger
     Vei:  Lucía
     Vei:  Eric
     Vei:  Inés
Node: Sarah
     Vei:  Ryan
         Atribut:  any  Valor:  2013
         Atribut:  tipus  Valor:  familia
     Vei:  Ahmed
         Atribut:  tipus  Valor:  amistat
         Atribut:  any  Valor:  2001
     Vei:  Pol
     Vei:  Dafna
     Vei:  Mariona
     Vei:  Iris
Node: Eric
     Vei:  Lucía
     Vei:  Roger
     Vei:  Inés

..... continua.....
```

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

Mètodes dels grafs

Estan documentats en el manual de referència > Basic graph types> Graph> Methods

Estan classificats en 4 apartats:

· Afegir i eliminar nodes i arestes

· Informar sobre nodes, arestes i veïns

· Comptar nodes, arestes i veïns

· Fer còpies i subgrafs

Basic graph types

- Graph—Undirected graphs with self loops
  - Overview
  - Methods

Exemples:

```
>>> len(g)
15
>>> g.size()
20
>>> g.remove_edge('Chen', 'Anna')
>>> g.size()
19
```

El nombre de nodes

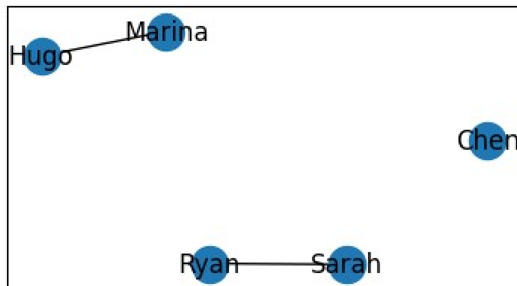
El nombre d'arestes

Com eliminar una aresta

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

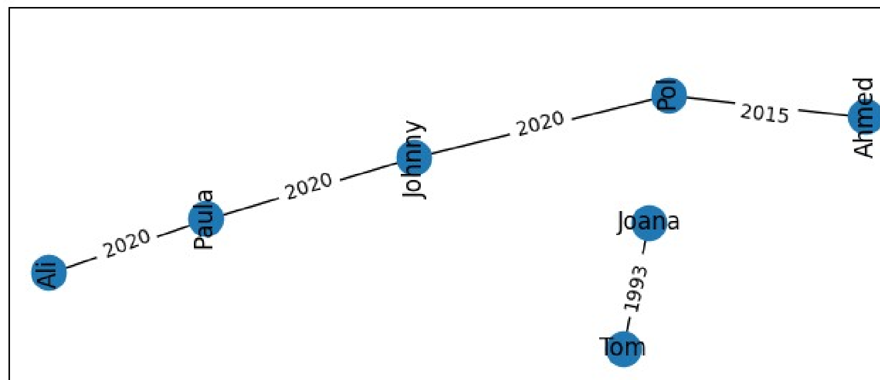
Mètodes dels grafs
Subgrafs

```
>>> gp = g.subgraph(['Sarah', 'Ryan', 'Marina', 'Hugo', 'Chen'])
```



Subgraf que inclou només els nodes seleccionats (i les arestes entre nodes seleccionats

```
>>> arestes_amb_any = filter(lambda t: t[2] != -1, g.edges(data='any', default=-1))
>>> gp1 =g.edge_subgraph(map(lambda x: (x[0], x[1]), arestes_amb_any))
```



Subgraf que inclou només les arestes seleccionades

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.

La setmana vinent ...

Algorismes sobre grafs

El contingut està disponible sota la llicència Creative Commons Attribution Share Alike si no s'indica el contrari.