Introducció
Pila
Cua
Conjunt
Grafs

Estructura de dades

ETSEIB/GIE

18 de novembre de 2019

1 Introducció

2 Pila

3 Cua

4 Conjunt

5 Grafs

Estructura de dades

- Per estructures de dades entenem la definició de com s'organitza un conjunt de dades i com s'interrelacionen les dades entre sí per tal de facilitar la seva gestió mitjançant un conjunt d'operacions.
- Estructura de dades freqüents:
  - Pila (últim d'entrar primer en sortir)
  - Cua (primer d'entrar primer en sortir)
  - Llista
  - Arbre
  - Conjunt
  - Diccionari
  - Graf

Classe pila

```
class Pila:
    def __init__(self):
        self.elements = []

    def cim(self):
        return self.elements[len(self.elements)-1]

    def empila(self, element):
        self.elements.append(element)

    def desempila(self):
        return self.elements.pop()

    def esBuida(self):
        return (self.elements == [])
```

Exemple de pila: conversió decimal a binari

1 3	2
1	6

6	2
0	3

3	2
1	1

1	2
1	0

13 en binari $=$ 1101.

Última resta, bit més significatiu.

Exemple de pila: conversió decimal a binari

```
from pila import *

def enBinari(n):
    xifres=Pila()
    while n > 2:
        xifres.empila(n % 2)
        n = n / 2
    xifres.empila(n % 2)
    if n==2:                    # evitem 0 a l'esquer
        xifres.empila(1)        # n / 2
    b=''
    while not xifres.esBuida():
        b = b + str(xifres.desempila())
    return b
```

Exemple de pila: Parèntesis

Dissenyar la funció benParentitzat(e, par) tal que donat un string e, el qual pot representar una expressió amb parèntesis, ens retorni True si l'expressió està ben parentitzada i ens retorni False en cas contrari. Una expressió està ben parentitzada si cada parèntesi que obre té el seu corresponent que tanca. El paràmetre par, conté un diccionari de parèntesis a considerar. Les claus són parèntesis oberts, i els valors respectius, els parèntesis tancats.

Exemple de pila: Parèntesi

```
def benParentitzat(e, par):
    """
    >>> par = {'(': ')', '{': '}', '[': ']', '≪': '≫'}
    >>> benParentitzat('<1+2*(3+4>-5)', par)
    True
    >>> benParentitzat('{1+2*(3+4}-5)', par)
    False
    >>> benParentitzat('{', par)
    False
    >>> benParentitzat('{}', par)
    True
    >>> benParentitzat('', par)
    True
    """
```

Parèntesi


```
def benParentitzat(e, par):

    p = pila.Pila()
    for c in e:
        if esParentesiObert(c, par):
            p.empila(c)
        elif esParentesiTancat(c, par):
            if p.esBuida():
                parentesiOK= False
            else:
                po= p.desempila()
                parentesiOK = fanParella(po, c, par)
            if not parentesiOK:
                break
    else:
        parentesiOK = p.esBuida()
    return parentesiOK
```

Exemple de pila: Parèntesi

```
def esParentesiObert(c, par):
    return c in par

def esParentesiTancat(c, par):
    return c in par.values()

def fanParella(o, t, par):
    return par[o] == t
```

Cua

```
class Cua:
    def __init__(self):
        self.elements = []

    def primer(self):
        return self.elements[0]

    def encua(self, element):
        self.elements.append(element)

    def desencua(self):
        return self.elements.pop(0)

    def esBuida(self):
        return (self.elements == [])
```

Cua

De la biblioteca collections, hi ha la classe *deque* per treballar amb cues de forma més eficient amb els següents mètodes:

c=collections.deque() $\rightarrow$ crea cua buida c

c.append(x) $\rightarrow$ encua x

c.popleft() $\rightarrow$ desencua el primer

len(c)==0 $\rightarrow$ cua c és buida

El mètode *popleft* és més eficient que el *pop(0)* de les llistes.

Conjunts

Creació i inclusió de membres

Conjunts

```
>>> coses={ 'capsa', 'bola',  'cadira', 'caixa', 'arma
ri', 'cadira'}
>>> print(coses)
{'armari', 'capsa', 'caixa', 'cadira', 'bola'}
>>> 'planta' in coses
False
>>> 'caixa' in coses
True
>>> 'pilota' not in coses
True
>>> coses.add('pilota')
>>> 'pilota' not in coses
False
```

Conjunts

Creació i inclusió de membres

Conjunts

```
>>> coses={ 'capsa', 'bola',  'cadira', 'caixa', 'arma
ri', 'cadira'}
>>> print(coses)
{'armari', 'capsa', 'caixa', 'cadira', 'bola'}
>>> 'planta' in coses
False
>>> 'caixa' in coses
True
>>> 'pilota' not in coses
True
>>> coses.add('pilota')
>>> 'pilota' not in coses
False
```

Conjunts

Creació i inclusió de membres

Conjunts

```
>>> coses={ 'capsa', 'bola',  'cadira', 'caixa', 'armari', 'cadira'}
>>> print(coses)
{'armari', 'capsa', 'caixa', 'cadira', 'bola'}
>>> 'planta' in coses
False
>>> 'caixa' in coses
True
>>> 'pilota' not in coses
True
>>> coses.add('pilota')
>>> 'pilota' not in coses
False
```

Conjunts

Creació i inclusió de membres

Conjunts

```
>>> coses={ 'capsa', 'bola',  'cadira', 'caixa', 'arma
ri', 'cadira'}
>>> print(coses)
{'armari', 'capsa', 'caixa', 'cadira', 'bola'}
>>> 'planta' in coses
False
>>> 'caixa' in coses
True
>>> 'pilota' not in coses
True
>>> coses.add('pilota')
>>> 'pilota' not in coses
False
```

Conjunts

Creació i inclusió de membres

Conjunts

```
>>> coses={ 'capsa', 'bola',  'cadira', 'caixa', 'arma
ri', 'cadira'}
>>> print(coses)
{'armari', 'capsa', 'caixa', 'cadira', 'bola'}
>>> 'planta' in coses
False
>>> 'caixa' in coses
True
>>> 'pilota' not in coses
True
>>> coses.add('pilota')
>>> 'pilota' not in coses
False
```

Conjunts

Creació i inclusió de membres

Conjunts

```
>>> coses={ 'capsa', 'bola',  'cadira', 'caixa', 'arma
ri', 'cadira'}
>>> print(coses)
{'armari', 'capsa', 'caixa', 'cadira', 'bola'}
>>> 'planta' in coses
False
>>> 'caixa' in coses
True
>>> 'pilota' not in coses
True
>>> coses.add('pilota')
>>> 'pilota' not in coses
False
```

Conjunts

Creació i inclusió de membres

Conjunts

```
>>> coses={ 'capsa', 'bola',  'cadira', 'caixa', 'arma
ri', 'cadira'}
>>> print(coses)
{'armari', 'capsa', 'caixa', 'cadira', 'bola'}
>>> 'planta' in coses
False
>>> 'caixa' in coses
True
>>> 'pilota' not in coses
True
>>> coses.add('pilota')
>>> 'pilota' not in coses
False
```

Conjunts

Creació i inclusió de membres

Conjunts

```
>>> coses={ 'capsa', 'bola',  'cadira', 'caixa', 'arma
ri', 'cadira'}
>>> print(coses)
{'armari', 'capsa', 'caixa', 'cadira', 'bola'}
>>> 'planta' in coses
False
>>> 'caixa' in coses
True
>>> 'pilota' not in coses
True
>>> coses.add('pilota')
>>> 'pilota' not in coses
False
```

Conjunts

Creació i inclusió de membres

Conjunts

```
>>> coses={ 'capsa', 'bola',  'cadira', 'caixa', 'arma
ri', 'cadira'}
>>> print(coses)
{'armari', 'capsa', 'caixa', 'cadira', 'bola'}
>>> 'planta' in coses
False
>>> 'caixa' in coses
True
>>> 'pilota' not in coses
True
>>> coses.add('pilota')
>>> 'pilota' not in coses
False
```

Conjunts

creació a través de qualsevol iterable

Conjunts

```
>>> nombres= set([10, 20, 20, 30, 40, 50])
>>> print(nombres)
{40, 10, 50, 20, 30}
```

Conjunts

creació a través de qualsevol iterable

Conjunts

```
>>> nombres= set([10, 20, 20, 30, 40, 50])
>>> print(nombres)
{40, 10, 50, 20, 30}
```

Conjunts

creació a través de qualsevol iterable

Conjunts

```
>>> nombres= set([10, 20, 20, 30, 40, 50])
>>> print(nombres)
{40, 10, 50, 20, 30}
```

Conjunts

creació a través de qualsevol iterable

Conjunts

```
>>> nombres= set([10, 20, 20, 30, 40, 50])
>>> print(nombres)
{40, 10, 50, 20, 30}
```

Conjunts

subconjunt o conjunt

Conjunts

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n2.issubset(n1)
True
>>> n2 <= n1
True
>>> n1.issuperset(n2)
True
>>> n1 >= n2
True
```

Conjunts

subconjunt o conjunt

Conjunts

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n2.issubset(n1)
True
>>> n2 <= n1
True
>>> n1.issuperset(n2)
True
>>> n1 >= n2
True
```

Conjunts

subconjunt o conjunt

Conjunts

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n2.issubset(n1)
True
>>> n2 <= n1
True
>>> n1.issuperset(n2)
True
>>> n1 >= n2
True
```

Conjunts

subconjunt o conjunt

Conjunts

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n2.issubset(n1)
True
>>> n2 <= n1
True
>>> n1.issuperset(n2)
True
>>> n1 >= n2
True
```

Conjunts

subconjunt o conjunt

Conjunts

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n2.issubset(n1)
True
>>> n2 <= n1
True
>>> n1.issuperset(n2)
True
>>> n1 >= n2
True
```

Conjunts

subconjunt o conjunt

Conjunts

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n2.issubset(n1)
True
>>> n2 <= n1
True
>>> n1.issuperset(n2)
True
>>> n1 >= n2
True
```

Conjunts

subconjunt o conjunt

Conjunts

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n2.issubset(n1)
True
>>> n2 <= n1
True
>>> n1.issuperset(n2)
True
>>> n1 >= n2
True
```

Conjunts

subconjunt o conjunt

Conjunts

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n2.issubset(n1)
True
>>> n2 <= n1
True
>>> n1.issuperset(n2)
True
>>> n1 >= n2
True
```

Conjunts

Operacions

Conjunts

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n1.intersection(n2)
{3, 5}
>>> n1 & n2
{3, 5}
>>> n1.union(n2)
{1, 3, 5, 7, 9, 11}
>>> n1 | n2
{1, 3, 5, 7, 9, 11}
```

Conjunts

Operacions

Conjunts

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n1.intersection(n2)
{3, 5}
>>> n1 & n2
{3, 5}
>>> n1.union(n2)
{1, 3, 5, 7, 9, 11}
>>> n1 | n2
{1, 3, 5, 7, 9, 11}
```

Conjunts

Operacions

Conjunts

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n1.intersection(n2)
{3, 5}
>>> n1 & n2
{3, 5}
>>> n1.union(n2)
{1, 3, 5, 7, 9, 11}
>>> n1 | n2
{1, 3, 5, 7, 9, 11}
```

Conjunts

Operacions

Conjunts

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n1.intersection(n2)
{3, 5}
>>> n1 & n2
{3, 5}
>>> n1.union(n2)
{1, 3, 5, 7, 9, 11}
>>> n1 | n2
{1, 3, 5, 7, 9, 11}
```

Conjunts

Operacions

Conjunts

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n1.intersection(n2)
{3, 5}
>>> n1 & n2
{3, 5}
>>> n1.union(n2)
{1, 3, 5, 7, 9, 11}
>>> n1 | n2
{1, 3, 5, 7, 9, 11}
```

Conjunts

Operacions

Conjunts

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n1.intersection(n2)
{3, 5}
>>> n1 & n2
{3, 5}
>>> n1.union(n2)
{1, 3, 5, 7, 9, 11}
>>> n1 | n2
{1, 3, 5, 7, 9, 11}
```

Conjunts

Operacions

Conjunts

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n1.intersection(n2)
{3, 5}
>>> n1 & n2
{3, 5}
>>> n1.union(n2)
{1, 3, 5, 7, 9, 11}
>>> n1 | n2
{1, 3, 5, 7, 9, 11}
```

Conjunts

Operacions

Conjunts

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n1.intersection(n2)
{3, 5}
>>> n1 & n2
{3, 5}
>>> n1.union(n2)
{1, 3, 5, 7, 9, 11}
>>> n1 | n2
{1, 3, 5, 7, 9, 11}
```

Conjunt

Estructura de dades

Introducció

Pila

Cua

Conjunt

Grafs

Operacions

Conjunt

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n1.difference(n2)
{1, 9, 11, 7}
>>> n1 - n2
{1, 9, 11, 7}
>>> n2 - n1
set()
```

Conjunt

Estructura de dades

Introducció

Pila

Cua

Conjunt

Grafs

Operacions

Conjunt

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n1.difference(n2)
{1, 9, 11, 7}
>>> n1 - n2
{1, 9, 11, 7}
>>> n2 - n1
set()
```

Conjunt

Operacions

Conjunt

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n1.difference(n2)
{1, 9, 11, 7}
>>> n1 - n2
{1, 9, 11, 7}
>>> n2 - n1
set()
```

Conjunt

Operacions

Conjunt

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n1.difference(n2)
{1, 9, 11, 7}
>>> n1 - n2
{1, 9, 11, 7}
>>> n2 - n1
set()
```

Conjunt

Operacions

Conjunt

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n1.difference(n2)
{1, 9, 11, 7}
>>> n1 - n2
{1, 9, 11, 7}
>>> n2 - n1
set()
```

Conjunt

Operacions

Conjunt

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n1.difference(n2)
{1, 9, 11, 7}
>>> n1 - n2
{1, 9, 11, 7}
>>> n2 - n1
set()
```

Conjunt

Operacions

Conjunt

```
>>> n1= {1, 3, 5, 7, 9, 11}
>>> n2= {3, 5}
>>> n1.difference(n2)
{1, 9, 11, 7}
>>> n1 - n2
{1, 9, 11, 7}
>>> n2 - n1
set()
```

Conjunt

També es poden eliminar elements del conjunt (veure discard/remove) i poden haver conjunts inmutables (frozenset).

Conjunt

```
>>> n2= {3, 5}
>>> n2.remove(3)
>>> n2
{5}
>>> n2.remove(3)
Traceback (most recent call last):
    ...
KeyError: 3
>>> n2.discard(3)
>>>
```

Conjunt

També es poden eliminar elements del conjunt (veure discard/remove) i poden haver conjunts inmutables (frozenset).

Conjunt

```
>>> n2= {3, 5}
>>> n2.remove(3)
>>> n2
{5}
>>> n2.remove(3)
Traceback (most recent call last):
    ...
KeyError: 3
>>> n2.discard(3)
>>>
```

Conjunt

També es poden eliminar elements del conjunt (veure discard/remove) i poden haver conjunts inmutables (frozenset).

Conjunt

```
>>> n2= {3, 5}
>>> n2.remove(3)
>>> n2
{5}
>>> n2.remove(3)
Traceback (most recent call last):
    ...
KeyError: 3
>>> n2.discard(3)
>>>
```

Conjunt

També es poden eliminar elements del conjunt (veure discard/remove) i poden haver conjunts inmutables (frozenset).

Conjunt

```
>>> n2= {3, 5}
>>> n2.remove(3)
>>> n2
{5}
>>> n2.remove(3)
Traceback (most recent call last):
    ...
KeyError: 3
>>> n2.discard(3)
>>>
```

Conjunt

També es poden eliminar elements del conjunt (veure discard/remove) i poden haver conjunts inmutables (frozenset).

Conjunt

```
>>> n2= {3, 5}
>>> n2.remove(3)
>>> n2
{5}
>>> n2.remove(3)
Traceback (most recent call last):
    ...
KeyError: 3
>>> n2.discard(3)
>>>
```

Conjunt

També es poden eliminar elements del conjunt (veure discard/remove) i poden haver conjunts inmutables (frozenset).

Conjunt

```
>>> n2= {3, 5}
>>> n2.remove(3)
>>> n2
{5}
>>> n2.remove(3)
Traceback (most recent call last):
    ...
KeyError: 3
>>> n2.discard(3)
>>>
```

Conjunt

També es poden eliminar elements del conjunt (veure discard/remove) i poden haver conjunts inmutables (frozenset).

Conjunt

```
>>> n2= {3, 5}
>>> n2.remove(3)
>>> n2
{5}
>>> n2.remove(3)
Traceback (most recent call last):
    ...
KeyError: 3
>>> n2.discard(3)
>>>
```

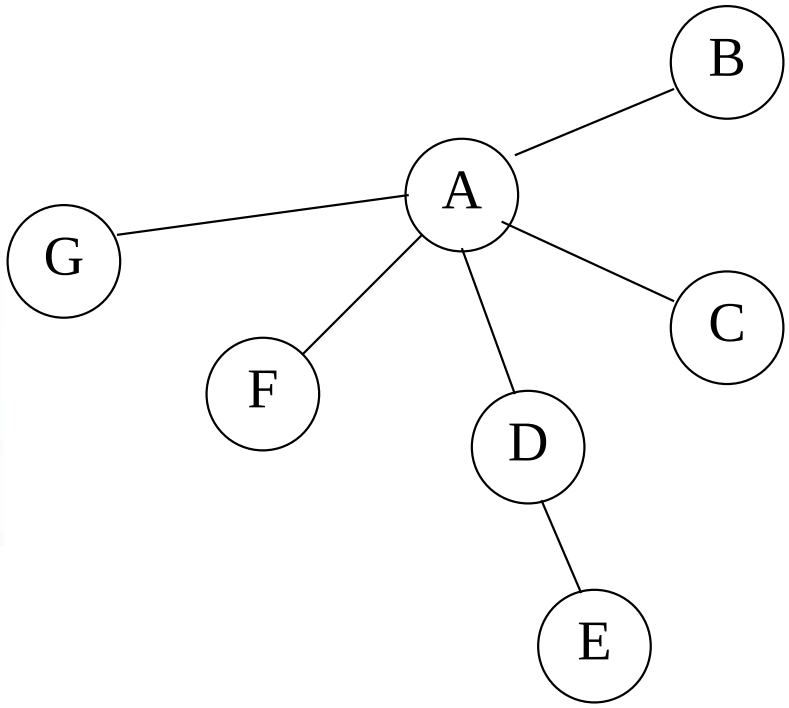
Conjunt

També es poden eliminar elements del conjunt (veure discard/remove) i poden haver conjunts inmutables (frozenset).

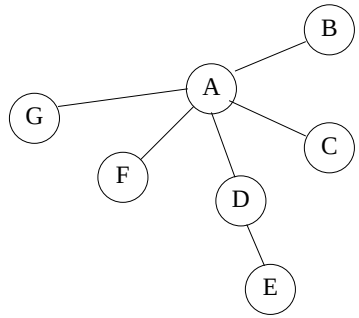
Conjunt

```
>>> n2= {3, 5}
>>> n2.remove(3)
>>> n2
{5}
>>> n2.remove(3)
Traceback (most recent call last):
    ...
KeyError: 3
>>> n2.discard(3)
>>>
```

Exemple de grafs



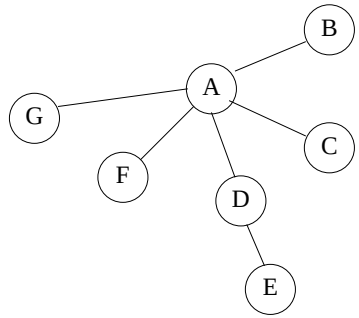
Exemple grafs



Exemple grafs

```
>>> import networkx as nx
>>> G=nx.empty_graph()
>>> G.add_edge('A','B')
>>> G.add_edge('A','C')
>>> G.add_edge('A','D')
>>> G.add_edge('D','E')
>>> G.add_edge('A','F')
>>> G.add_edge('A','G')
>>> cami=nx.shortest_path(G, 'A', 'E')
>>> print(cami)
['A', 'D', 'E']
```

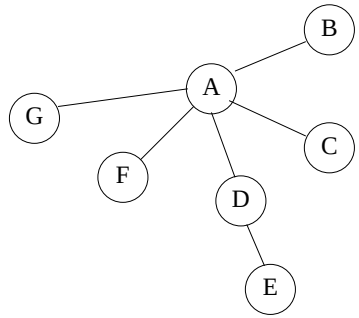
Exemple grafs



Exemple grafs

```
>>> import networkx as nx
>>> G=nx.empty_graph()
>>> G.add_edge('A','B')
>>> G.add_edge('A','C')
>>> G.add_edge('A','D')
>>> G.add_edge('D','E')
>>> G.add_edge('A','F')
>>> G.add_edge('A','G')
>>> cami=nx.shortest_path(G, 'A', 'E')
>>> print(cami)
['A', 'D', 'E']
```

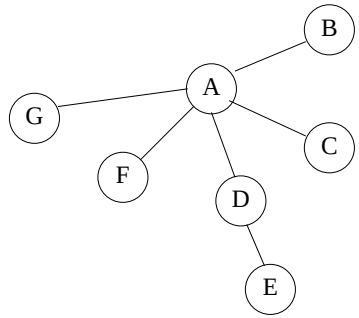
Exemple grafs



Exemple grafs

```
>>> import networkx as nx
>>> G=nx.empty_graph()
>>> G.add_edge('A','B')
>>> G.add_edge('A','C')
>>> G.add_edge('A','D')
>>> G.add_edge('D','E')
>>> G.add_edge('A','F')
>>> G.add_edge('A','G')
>>> cami=nx.shortest_path(G, 'A', 'E')
>>> print(cami)
['A', 'D', 'E']
```

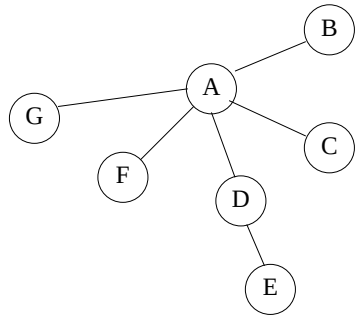
Exemple grafs



Exemple grafs

```
>>> import networkx as nx
>>> G=nx.empty_graph()
>>> G.add_edge('A','B')
>>> G.add_edge('A','C')
>>> G.add_edge('A','D')
>>> G.add_edge('D','E')
>>> G.add_edge('A','F')
>>> G.add_edge('A','G')
>>> cami=nx.shortest_path(G, 'A', 'E')
>>> print(cami)
['A', 'D', 'E']
```

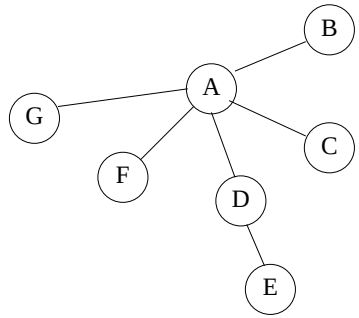
Exemple grafs



Exemple grafs

```
>>> import networkx as nx
>>> G=nx.empty_graph()
>>> G.add_edge('A','B')
>>> G.add_edge('A','C')
>>> G.add_edge('A','D')
>>> G.add_edge('D','E')
>>> G.add_edge('A','F')
>>> G.add_edge('A','G')
>>> cami=nx.shortest_path(G, 'A', 'E')
>>> print(cami)
['A', 'D', 'E']
```

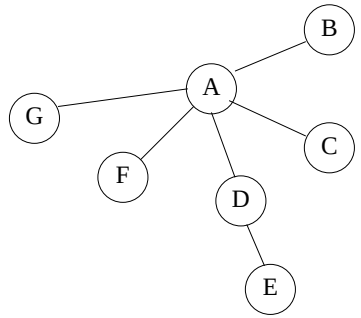
Exemple grafs



Exemple grafs

```
>>> import networkx as nx
>>> G=nx.empty_graph()
>>> G.add_edge('A','B')
>>> G.add_edge('A','C')
>>> G.add_edge('A','D')
>>> G.add_edge('D','E')
>>> G.add_edge('A','F')
>>> G.add_edge('A','G')
>>> cami=nx.shortest_path(G, 'A', 'E')
>>> print(cami)
['A', 'D', 'E']
```

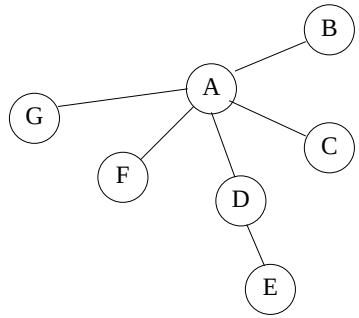
Exemple grafs



Exemple grafs

```
>>> import networkx as nx
>>> G=nx.empty_graph()
>>> G.add_edge('A','B')
>>> G.add_edge('A','C')
>>> G.add_edge('A','D')
>>> G.add_edge('D','E')
>>> G.add_edge('A','F')
>>> G.add_edge('A','G')
>>> cami=nx.shortest_path(G, 'A', 'E')
>>> print(cami)
['A', 'D', 'E']
```

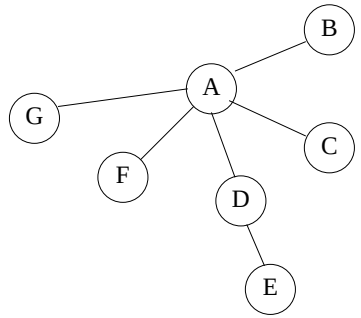
Exemple grafs



Exemple grafs

```
>>> import networkx as nx
>>> G=nx.empty_graph()
>>> G.add_edge('A','B')
>>> G.add_edge('A','C')
>>> G.add_edge('A','D')
>>> G.add_edge('D','E')
>>> G.add_edge('A','F')
>>> G.add_edge('A','G')
>>> cami=nx.shortest_path(G, 'A', 'E')
>>> print(cami)
['A', 'D', 'E']
```

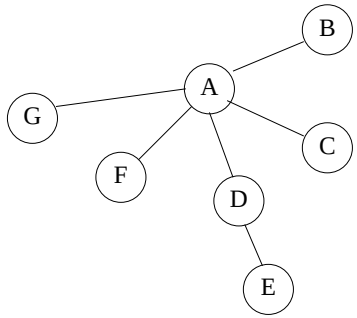
Exemple grafs



Exemple grafs

```
>>> import networkx as nx
>>> G=nx.empty_graph()
>>> G.add_edge('A','B')
>>> G.add_edge('A','C')
>>> G.add_edge('A','D')
>>> G.add_edge('D','E')
>>> G.add_edge('A','F')
>>> G.add_edge('A','G')
>>> cami=nx.shortest_path(G, 'A', 'E')
>>> print(cami)
['A', 'D', 'E']
```

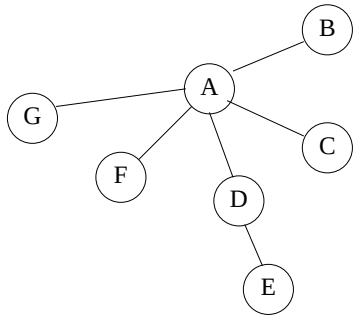
Exemple grafs



Exemple grafs

```
>>> import networkx as nx
>>> G=nx.empty_graph()
>>> G.add_edge('A','B')
>>> G.add_edge('A','C')
>>> G.add_edge('A','D')
>>> G.add_edge('D','E')
>>> G.add_edge('A','F')
>>> G.add_edge('A','G')
>>> cami=nx.shortest_path(G, 'A', 'E')
>>> print(cami)
['A', 'D', 'E']
```

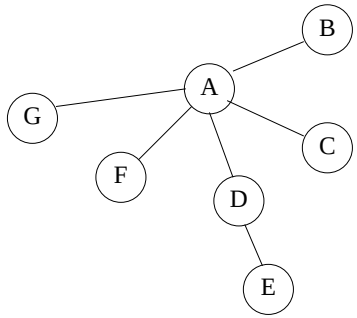
Exemple grafs



Exemple grafs

```
>>> import networkx as nx
>>> G=nx.empty_graph()
>>> G.add_edge('A','B')
>>> G.add_edge('A','C')
>>> G.add_edge('A','D')
>>> G.add_edge('D','E')
>>> G.add_edge('A','F')
>>> G.add_edge('A','G')
>>> cami=nx.shortest_path(G, 'A', 'E')
>>> print(cami)
['A', 'D', 'E']
```

Exemple grafs



Exemple grafs

```
>>> import networkx as nx
>>> G=nx.empty_graph()
>>> G.add_edge('A','B')
>>> G.add_edge('A','C')
>>> G.add_edge('A','D')
>>> G.add_edge('D','E')
>>> G.add_edge('A','F')
>>> G.add_edge('A','G')
>>> cami=nx.shortest_path(G, 'A', 'E')
>>> print(cami)
['A', 'D', 'E']
```