

# 10 – LISTAS DE LISTAS

Jorge Muñoz  
IIC1103 – Introducción a la Programación

# **LISTAS DE LISTAS (nada nuevo)**

```
#Lista de ints (6 elementos)
a = [5, -1, 4, 9, 23, -3]

#Lista de str (4 elementos)
b = ["arbol", "barco", "casa", "dato"]

#Lista de mix de basicos (6 elementos)
c = [5, "hola", 6, 7, True, 6.23]

# Lista de listas (4 elementos)
# Cada elemento es una lista [str,float] con el nombre y nota de un estudiante
d = [["Alice",5.5],["Bob",4.3],["Charlie",6.2],["David",6.0]]
```

```
# Lista de listas (4 elementos)
# Cada elemento es una lista [str,float] con el nombre y nota de un estudiante
d = [["Alice",5.5],["Bob",4.3],["Charlie",6.2],["David",6.0]]
```

```
#Lista de mix de tipos (5 elementos)
#No tienen por que ser todos los elementos del mismo tipo
e = [7,"adios",["precio",1100], True, 5.31]
```

```
#Lista de list (3 elementos)
#No tienen por que ser todas las listas iguales
f = [["arbol","abeja"],["barco","banco","bar"],["caracol"]]
```

```
# Lista (3 elementos)
# Las listas pueden tener listas, y pueden tener listas, ...
g = [["Calculo",[4.2,5.5]],[["Quimica",[4.5]],["Algebra",[5.3,3.2,4.4,6.8]]]]
```

```
# Misma Lista (3 elementos)
# Python te deja escribirlo separado para que lo puedas leer mejor
# pero es exactamente lo mismo
```

```
h = [[["Calculo",[4.2,5.5]],
      ["Quimica",[4.5]],
      ["Algebra",[5.3,3.2,4.4,6.8]]]]
```

# **LOS 1.1**



# INTERESES

“cosas cotidianas de la universidad”



Te escribo en nombre de la coordinación del curso de Introducción a la Programación-IIC1103. Hemos encontrado un alto grado de similitud entre tu Tarea 1 del curso con otras tareas. [REDACTED]

[REDACTED] sanción indicada en el programa del curso (nota final en el curso será de 1.1, y se entregarán los antecedentes correspondientes a la Dirección de Pregrado).



imgflip.com



memesintroalaprogr • Following ...

14h 29 likes Reply



jorge11palma Los semestres pasan y los 1.1 vuelven a aparecer xd

13h 8 likes Reply



gallegos\_brian24 En esos casos uno puede botar el curso y sacarse el 1 de encima?

11h Reply

View replies (4)



Liked by memes.con.cupos.college and 299 others

14 HOURS AGO

Add a comment...

Post



```
# Recibe una lista de estudiantes,  
# donde cada estudiante es una lista [str,float]  
# con el nombre y la nota, y retorna  
# una lista de str con los nombres  
# de los que tienen 1.1
```

```
def unouno(p):
```

```
est = [[ "Alice",5.5],  
       [ "Bob",1.1],  
       [ "Charlie",6.8],  
       [ "David",1.1],  
       [ "Eva",7.0]]
```

```
[ 'Bob', 'David' ]
```





I'LL WAIT  
FOR YOU HERE



```
# Recibe una lista de estudiantes,  
# donde cada estudiante es una lista [str,float]  
# con el nombre y la nota, y retorna  
# una lista de str con los nombres  
# de los que tienen 1.1
```

```
def unouno(p):
```

```
est = [[ "Alice",5.5],  
       [ "Bob",1.1],  
       [ "Charlie",6.8],  
       [ "David",1.1],  
       [ "Eva",7.0]]
```

```
[ 'Bob', 'David' ]
```



```
# Recibe una lista de estudiantes,
# donde cada estudiante es una lista [str,float]
# con el nombre y la nota, y retorna
# una lista de str con los nombres
# de los que tienen 1.1

def unouno(p):
    #Lista donde pondre los nombres que encuentre
    res = []

    #Recorrer estudiante a estudiante
    for e in p:
        # e es un esudiante ([str,float])
        # El nombre esta en la posicion 0, y la nota en la 1
        nombre = e[0]
        nota = e[1]
        # Si tiene 1.1 lo agrego a mi lista de nombre
        if nota == 1.1:
            res.append(nombre)

    #Retorno la lista de nombres
    return res

est = [["Alice",5.5],
       ["Bob",1.1],
       ["Charlie",6.8],
       ["David",1.1],
       ["Eva",7.0]]                                ['Bob', 'David']
```

# **CREANDO LISTAS DE LISTAS (nada nuevo)**

```
#Crear lista de nombres (list de str)
nombres = []

num = int(input("Cuantos nombres? "))
for i in range(0,num):

    nombre = input("Nombre? ")
    nombres.append(nombre)

print(nombres)
```

```
Cuantos nombres? 3
Nombre? Alicia
Nombre? Bernardo
Nombre? Carlos
['Alicia', 'Bernardo', 'Carlos']
```

```
#Crear lista de nombres (list de listas [str,float])
estudiantes = []

num = int(input("Numero estudiantes? "))
for i in range(0,num):

    nombre = input("Nombre? ")
    nota = float(input("Nota? "))

    est = [nombre,nota]
    estudiantes.append(est)

print(estudiantes)
```

```
Numero estudiantes? 3
Nombre? Alicia
Nota? 6.8
Nombre? Bernardo
Nota? 3.5
Nombre? Carlos
Nota? 5.5
[['Alicia', 6.8], ['Bernardo', 3.5], ['Carlos', 5.5]]
```

**PPA**

# INTERESES

“cosas cotidianas de la universidad”



```
def ppa(es):
```

```
    es = [[ "Alice", [6.6,5.4,6.0] ],  
          [ "Barto", [4.0,4.0] ],  
          [ "Carlo", [5.5,6.2,6.9,7.0] ],  
          [ "Peter", [5.5,4.3] ] ]
```

```
[[ 'Alice', 6.0], ['Barto', 4.0], ['Carlo', 6.4], ['Peter', 4.9]]
```





I'LL WAIT  
FOR YOU HERE



```
def ppa(es):
```

```
    es = [[ "Alice", [6.6,5.4,6.0] ],  
          [ "Barto", [4.0,4.0] ],  
          [ "Carlo", [5.5,6.2,6.9,7.0] ],  
          [ "Peter", [5.5,4.3] ] ]
```

```
[[ 'Alice', 6.0], ['Barto', 4.0], ['Carlo', 6.4], ['Peter', 4.9]]
```



```
def ppa(es):  
  
    #Creo una lista donde pondre listas ([nombre,ppa])  
    datos = []  
  
    #Miro estudiante a estudiante  
    for e in es:  
        nombre = e[0]  
        notas = e[1]  
  
        #Calculo el promedio(ppa)  
        suma = 0  
        for n in notas:  
            suma += n  
        ppa = suma / len(notas)  
  
        #Creo una lista  
        dato = [nombre,ppa]  
  
        #La agrego a mi lista de datos  
        datos.append(dato)  
  
    #Retornar  
    return datos
```

```
es = [[ "Alice", [6.6,5.4,6.0]],  
      [ "Barto", [4.0,4.0]],  
      [ "Carlo", [5.5,6.2,6.9,7.0]],  
      [ "Peter", [5.5,4.3]]]
```

```
[[ 'Alice', 6.0], [ 'Barto', 4.0], [ 'Carlo', 6.4], [ 'Peter', 4.9]]
```

# **EUROVISION**



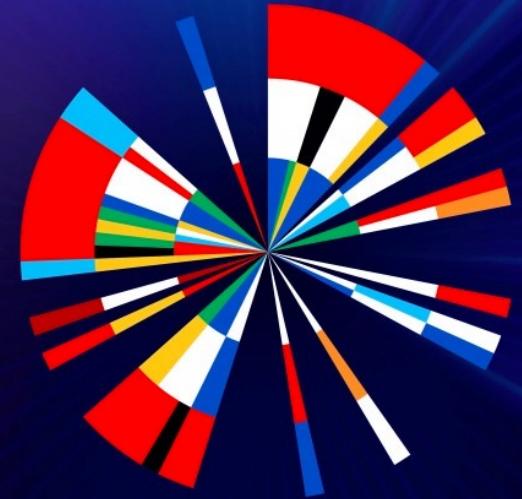
# INTERESES

“Música”

“Festival de Viña”



openup



EUROVISION  
SONG CONTEST  
ROTTERDAM 2020



Situación  
Real

12	NORTH MACEDONIA	91		SERBIA	25
6	SWEDEN	75		3 FRANCE	22
5	ITALY	74		4 GREECE	22
8	AZERBAIJAN	69		SLOVENIA	18
1	RUSSIA	66		ICELAND	11
	THE NETHERLANDS	65		NORWAY	6
2	AUSTRALIA	63		ESTONIA	5
10	SWITZERLAND	60		BELARUS	2
	CZECH REPUBLIC	49		UNITED KINGDOM	2
	MALTA	41		SAN MARINO	1
	DENMARK	36		GERMANY	0
7	CYPRUS	34		ISRAEL	0
	ALBANIA	33		SPAIN	0

12 10 8 7 6 5 4 3 2 1



15 OF 41 COUNTRIES VOTING



```
# Recibe una lista de votos,  
# donde cada voto es una lista [str,str,int]  
# con el nombre del país que ha votado,  
# a quien ha votado y cuantos puntos,  
# y retorna un str con el ganador de Eurovision
```

```
def eurovision(vs):
```

```
votos = [[ "Spain", "Italy", 10],  
          [ "Spain", "Ireland", 12],  
          [ "France", "Ireland", 8],  
          [ "France", "Spain", 1],  
          [ "Italy", "Spain", 10],  
          [ "Italy", "Ireland", 12]]
```

Ireland



## CONSEJO

“En vez de hacerlo directamente, a veces vale la pena descomponer: 1) ir procesando y guardando datos en algún sitio, 2) y luego usar eso para contestar la pregunta fácilmente”





I'LL WAIT  
FOR YOU HERE



```
# Recibe una lista de votos,  
# donde cada voto es una lista [str,str,int]  
# con el nombre del país que ha votado,  
# a quien ha votado y cuantos puntos,  
# y retorna un str con el ganador de Eurovision
```

```
def eurovision(vs):
```

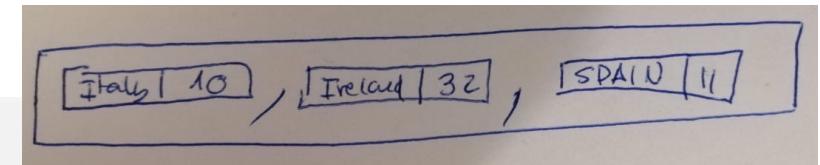
```
votos = [[ "Spain", "Italy", 10],  
          [ "Spain", "Ireland", 12],  
          [ "France", "Ireland", 8],  
          [ "France", "Spain", 1],  
          [ "Italy", "Spain", 10],  
          [ "Italy", "Ireland", 12]]
```

Ireland



```
# Recibe una lista de votos,  
# donde cada voto es una lista [str,str,int]  
# con el nombre del país que ha votado,  
# a quien ha votado y cuantos puntos,  
# y retorna un str con el ganador de Eurovision  
  
def eurovision(vs):  
    # Voy a crear una lista de listas [pais,puntos]  
    # donde voy a ir sumando los puntos que vaya sacando  
    # cada país  
    puntos = []
```

....



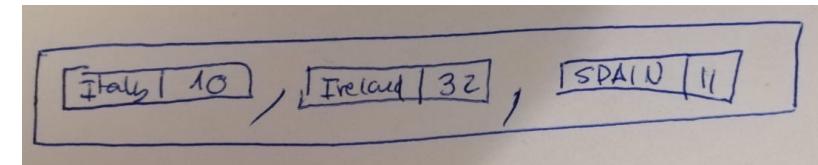
```
votos = [[ "Spain", "Italy", 10],  
          [ "Spain", "Ireland", 12],  
          [ "France", "Ireland", 8],  
          [ "France", "Spain", 1],  
          [ "Italy", "Spain", 10],  
          [ "Italy", "Ireland", 12]]
```

Ireland



```
# Recibe una lista de votos,  
# donde cada voto es una lista [str,str,int]  
# con el nombre del país que ha votado,  
# a quien ha votado y cuantos puntos,  
# y retorna un str con el ganador de Eurovision  
  
def eurovision(vs):  
    # Voy a crear una lista de listas [pais,puntos]  
    # donde voy a ir sumando los puntos que vaya sacando  
    # cada país  
    puntos = []  
  
    #Para cada voto  
    for v in vs:  
  
        #Solo me interesa el país que recibe los puntos y los puntos  
        pais = v[1]  
        pts = v[2]  
  
        #Voy a buscar si ya tengo ese país en mi lista de puntos  
        #Y si lo tengo, le aumento los puntos  
        esta = False  
        for p in puntos:  
            if p[0] == pais:  
                esta = True  
                p[1] += pts
```

....

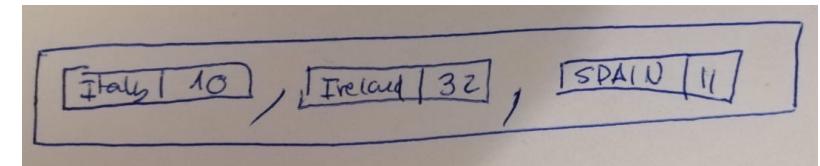


```
votos = [[ "Spain", "Italy", 10],  
          [ "Spain", "Ireland", 12],  
          [ "France", "Ireland", 8],  
          [ "France", "Spain", 1],  
          [ "Italy", "Spain", 10],  
          [ "Italy", "Ireland", 12]]
```

Ireland



```
# Recibe una lista de votos,  
# donde cada voto es una lista [str,str,int]  
# con el nombre del país que ha votado,  
# a quien ha votado y cuantos puntos,  
# y retorna un str con el ganador de Eurovision  
  
def eurovision(vs):  
    # Voy a crear una lista de listas [pais,puntos]  
    # donde voy a ir sumando los puntos que vaya sacando  
    # cada país  
    puntos = []  
  
    #Para cada voto  
    for v in vs:  
  
        #Solo me interesa el país que recibe los puntos y los puntos  
        pais = v[1]  
        pts = v[2]  
  
        #Voy a buscar si ya tengo ese país en mi lista de puntos  
        #Y si lo tengo, le aumento los puntos  
        esta = False  
        for p in puntos:  
            if p[0] == pais:  
                esta = True  
                p[1] += pts  
  
        #Si no esta, me toca crearlo y ponerle los puntos  
        if not esta:  
            x = [pais,pts]  
            puntos.append(x)
```

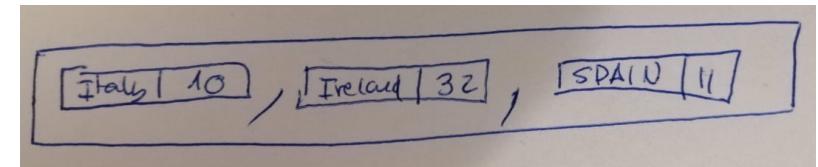


```
votos = [[ "Spain", "Italy", 10],  
          [ "Spain", "Ireland", 12],  
          [ "France", "Ireland", 8],  
          [ "France", "Spain", 1],  
          [ "Italy", "Spain", 10],  
          [ "Italy", "Ireland", 12]]
```

Ireland



```
# Recibe una lista de votos,  
# donde cada voto es una lista [str,str,int]  
# con el nombre del país que ha votado,  
# a quien ha votado y cuantos puntos,  
# y retorna un str con el ganador de Eurovision  
  
def eurovision(vs):  
    # Voy a crear una lista de listas [pais,puntos]  
    # donde voy a ir sumando los puntos que vaya sacando  
    # cada país  
    puntos = []  
  
    #Para cada voto  
    for v in vs:  
  
        #Solo me interesa el país que recibe los puntos y los puntos  
        pais = v[1]  
        pts = v[2]  
  
        #Voy a buscar si ya tengo ese país en mi lista de puntos  
        #Y si lo tengo, le aumento los puntos  
        esta = False  
        for p in puntos:  
            if p[0] == pais:  
                esta = True  
                p[1] += pts  
  
        #Si no esta, me toca crearlo y ponerle los puntos  
        if not esta:  
            x = [pais,pts]  
            puntos.append(x)  
  
    #Aquí tendré en puntos todos los países que han recibido puntos  
    # y los puntos que tienen. Solo tengo que buscar quien tiene más  
    maxip = -1  
    maxin = ""  
    for p in puntos:  
        if p[1] > maxip:  
            maxip = p[1]  
            maxin = p[0]  
  
    return maxin
```



```
votos = [[ "Spain", "Italy", 10],  
          [ "Spain", "Ireland", 12],  
          [ "France", "Ireland", 8],  
          [ "France", "Spain", 1],  
          [ "Italy", "Spain", 10],  
          [ "Italy", "Ireland", 12]]
```

Ireland







- Lordi – Hard Rock Hallelujah (Finlandia)
  - <https://www.youtube.com/watch?v=gAh9NRGNhUU>
- Rodolfo Chikilicuatre – Baila El Chiki Chiki (Spain)
  - <https://www.youtube.com/watch?v=wfeClvOxXBo>

# **ENCADENADOS (nada nuevo)**

```
compra = [[ "Aceite",3500],["Pasta",1300],["Arroz",980],["Pollo",2340]]
```

```
# Producto en la posicion 2 de la lista de la compra
```

```
prod = compra[2]
```

```
# Su Nombre y precio
```

```
nombre = prod[0]
```

```
precio = prod[1]
```

```
# Puedes encadenarlo para ir mas directo
```

```
# compra[2] es una lista, no? pues a una lista le puedes
```

```
# poner el [] pegado para acceder a un elemento.
```

```
# Es exactamente LO MISMO de siempre
```

```
nombre = compra[2][0]
```

```
precio = compra[1][1]
```

```
# Lo mismo con listas de listas, ...
```

```
qs = [[ "USA",["MIT","Harvard","Stanford"]],
```

```
      [ "UK",["Oxford","Cambridge"]],
```

```
      [ "Japan",["Todai","Sokendai"]]]
```

```
#El nombre de la primera universidad de USA
```

```
top = qs[0][1][0]
```

```
#La larga
```

```
usa = qs[0] #Lista [str,list]
```

```
unis = usa[1] # Lista de str con las unis
```

```
top = unis[0] # str con la top uni (pos 0)
```

# GEOTREE



# INTERESES

“Tipo prueba”

“Simpson”

“árboles genealógicos”



## Interrogación 2

### Pregunta 1

Estás realizando tu práctica en **GeoTree**, un emprendimiento UC que usa Python para armar árboles genealógicos. Sin embargo, están teniendo problemas en la implementación, por lo que te piden a ti, flamante alumno de Introducción a la Programación, que los ayudes a terminar su programa.

Hasta ahora, lo único que tienen es una representación con *listas* del árbol de una familia, y la declaración de las funciones que vas a necesitar. El árbol está representado por una lista donde, en cada elemento se guarda un *string* con el nombre del miembro de la familia (no hay *strings* repetidos), seguido de una *lista* de las posiciones de sus hijos dentro de la lista. Puedes ver un ejemplo de la lista en el código más abajo. Las funciones que necesitan que implementes son las siguientes:

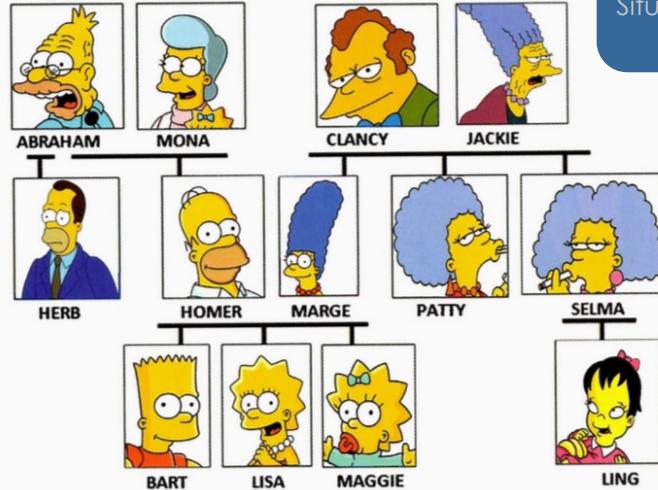
- (a) **(30 puntos)** `hijos(a,n)` : recibe una *lista* `a` con el árbol genealógico y un *string* `n` con el nombre de un integrante de la familia. Retorna una *lista* de *strings* con los nombres de los hijos de ese integrante. Si el integrante no tiene hijos, retorna una *lista* vacía.
- (b) **(30 puntos)** `padres(a,n)` : recibe una *lista* `a` con el árbol genealógico y un *string* `n` con el nombre de un integrante de la familia. Retorna una *lista* de *strings* con los nombres de los padres de ese integrante. Si el integrante no tiene padre, retorna una *lista* vacía.

A continuación, se encuentra el código con el ejemplo de la familia Simpson y el *output* esperado escrito en los comentarios. Tu deber es completar las funciones `hijos(a,n)` y `padres(a,n)`, para que los `print` del final del código impriman lo solicitado. Ten en cuenta que no es necesario que la lista de los nombres salgan exactamente en el orden del *output* esperado.



Situación  
Real

- (a) **(30 puntos)** `hijos(a,n)`: recibe una *lista* *a* con el árbol genealógico y un *string* *n* con el nombre de un integrante de la familia. Retorna una *lista* de *strings* con los nombres de los hijos de ese integrante. Si el integrante no tiene hijos, retorna una *lista* vacía.



```
simp = [
    ["Abraham", [1,2]],      #0
    ["Herb", []],             #1
    ["Homer", [3,4,5]],      #2
    ["Bart", []],             #3
    ["Maggie", []],           #4
    ["Lisa", []],             #5
    ["Marge", [3,4,5]],      #6
    ["Mona", [2]],            #7
    ["Clancy", [6,10,11]],   #8
    ["Jackie", [6,10,11]],   #9
    ["Selma", [12]],          #10
    ["Patty", []],            #11
    ["Ling", []]]             #12
```

```
print(hijos(simp, 'Marge'))      #['Bart', 'Maggie', 'Lisa']
print(hijos(simp, 'Clancy'))     #['Marge', 'Patty', 'Selma']
print(hijos(simp, 'Maggie'))     #[]
```



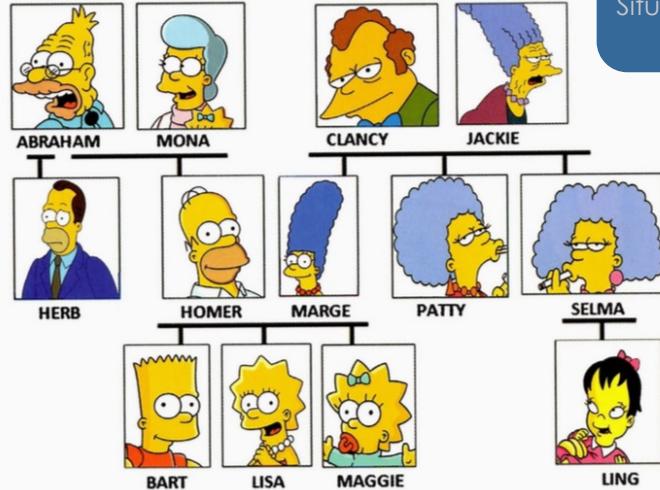


I'LL WAIT  
FOR YOU HERE



Situación  
Real

- (a) **(30 puntos)** `hijos(a,n)`: recibe una *lista* *a* con el árbol genealógico y un *string* *n* con el nombre de un integrante de la familia. Retorna una *lista* de *strings* con los nombres de los hijos de ese integrante. Si el integrante no tiene hijos, retorna una *lista* vacía.



```
simp = [
    ["Abraham", [1,2]],      #0
    ["Herb", []],             #1
    ["Homer", [3,4,5]],      #2
    ["Bart", []],             #3
    ["Maggie", []],           #4
    ["Lisa", []],             #5
    ["Marge", [3,4,5]],      #6
    ["Mona", [2]],            #7
    ["Clancy", [6,10,11]],   #8
    ["Jackie", [6,10,11]],   #9
    ["Selma", [12]],          #10
    ["Patty", []],            #11
    ["Ling", []]]             #12
```

```
print(hijos(simp, 'Marge'))      #['Bart', 'Maggie', 'Lisa']
print(hijos(simp, 'Clancy'))     #['Marge', 'Patty', 'Selma']
print(hijos(simp, 'Maggie'))     #[]
```



- (a) (30 puntos) `hijos(a,n)`: recibe una lista `a` con el árbol genealógico y un string `n` con el nombre de un integrante de la familia. Retorna una lista de strings con los nombres de los hijos de ese integrante. Si el integrante no tiene hijos, retorna una lista vacía.

```
def hijos(a,n):
    res = []

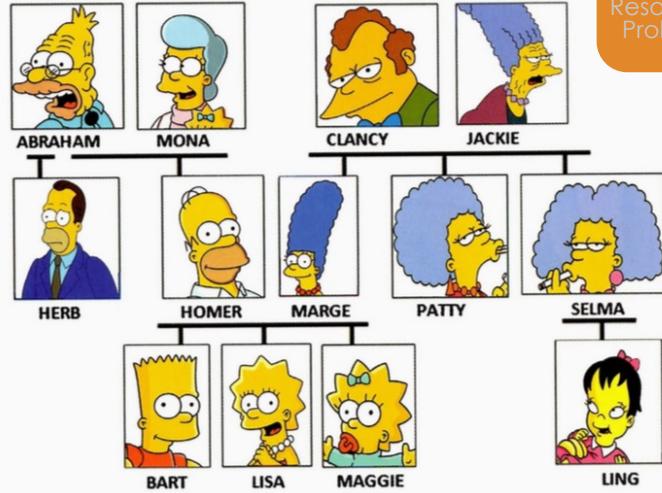
    #Recorrer hasta encontrar la persona (n)
    for p in a:
        nombre = p[0]

        #Si es la persona que busco
        if nombre == n:
            #Lista de posiciones de los hijos
            phs = p[1]

            #Por cada pos de cada hijo
            for ph in phs:
                nombre = a[ph][0]
                res.append(nombre)

    return res

print(hijos(simp, 'Marge'))      #['Bart', 'Maggie', 'Lisa']
print(hijos(simp, 'Clancy'))     #['Marge', 'Patty', 'Selma']
print(hijos(simp, 'Maggie'))     #[]
```

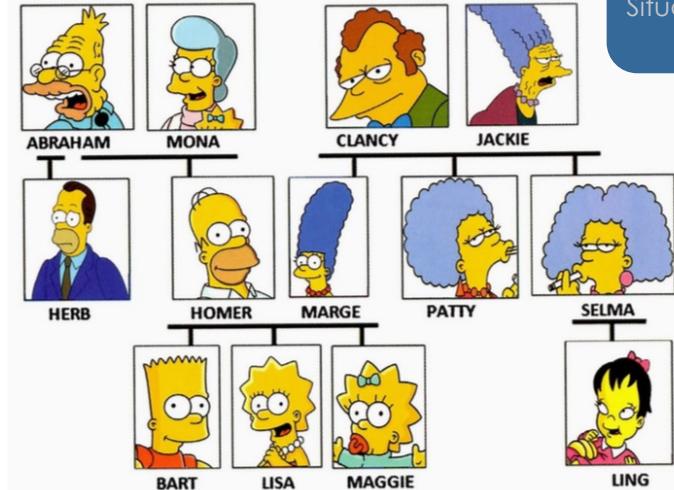


```
simp = [
    ["Abraham", [1,2]],      #0
    ["Herb", []],             #1
    ["Homer", [3,4,5]],      #2
    ["Bart", []],             #3
    ["Maggie", []],           #4
    ["Lisa", []],              #5
    ["Marge", [3,4,5]],       #6
    ["Mona", [2]],             #7
    ["Clancy", [6,10,11]],    #8
    ["Jackie", [6,10,11]],    #9
    ["Selma", [12]],           #10
    ["Patty", []],             #11
    ["Ling", []]]             #12
```

- (b) (30 puntos) `padres(a,n)`: recibe una *lista* *a* con el árbol genealógico y un *string* *n* con el nombre de un integrante de la familia. Retorna una *lista* de *strings* con los nombres de los padres de ese integrante. Si el integrante no tiene padre, retorna una *lista* vacía.



Situación Real



```

print(padres(simp, 'Maggie'))      #['Homer', 'Marge']
print(padres(simp, 'Homer'))       #['Abraham', 'Mona']
print(padres(simp, 'Ling'))        #['Selma']
print(padres(simp, 'Abraham'))     #[]
  
```

```

simp = [
    ["Abraham", [1,2]],      #0
    ["Herb", []],            #1
    ["Homer", [3,4,5]],     #2
    ["Bart", []],            #3
    ["Maggie", []],          #4
    ["Lisa", []],             #5
    ["Marge", [3,4,5]],      #6
    ["Mona", [2]],           #7
    ["Clancy", [6,10,11]],   #8
    ["Jackie", [6,10,11]],   #9
    ["Selma", [12]],         #10
    ["Patty", []],            #11
    ["Ling", []]]            #12
  
```



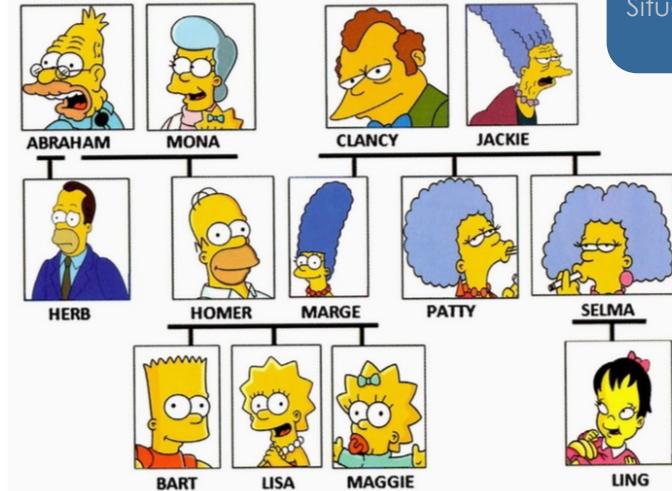


I'LL WAIT  
FOR YOU HERE

- (b) (30 puntos) `padres(a,n)`: recibe una *lista* *a* con el árbol genealógico y un *string* *n* con el nombre de un integrante de la familia. Retorna una *lista* de *strings* con los nombres de los padres de ese integrante. Si el integrante no tiene padre, retorna una *lista* vacía.



Situación Real



```

print(padres(simp, 'Maggie'))      #['Homer', 'Marge']
print(padres(simp, 'Homer'))       #['Abraham', 'Mona']
print(padres(simp, 'Ling'))        #['Selma']
print(padres(simp, 'Abraham'))     #[]
  
```

```

simp = [
    ["Abraham", [1,2]],      #0
    ["Herb", []],            #1
    ["Homer", [3,4,5]],     #2
    ["Bart", []],            #3
    ["Maggie", []],          #4
    ["Lisa", []],             #5
    ["Marge", [3,4,5]],      #6
    ["Mona", [2]],           #7
    ["Clancy", [6,10,11]],   #8
    ["Jackie", [6,10,11]],   #9
    ["Selma", [12]],          #10
    ["Patty", []],            #11
    ["Ling", []]]            #12
  
```

- (b) (30 puntos) `padres(a,n)`: recibe una lista `a` con el árbol genealógico y un string `n` con el nombre de un integrante de la familia. Retorna una lista de strings con los nombres de los padres de ese integrante. Si el integrante no tiene padre, retorna una lista vacía.



Resolución  
Problema

```

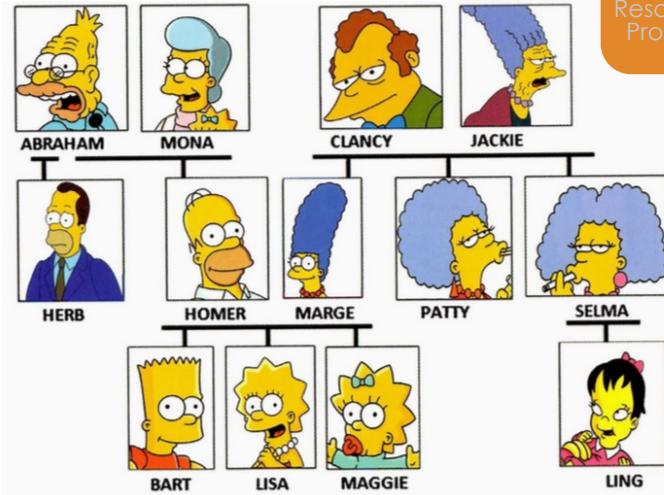
def padres(a,n):
    #Buscar la posicion de la persona
    for i in range(len(a)):
        nombre = a[i][0]
        if nombre == n:
            pos = i

    #Buscar gente que tiene pos en sus hijos
    res = []
    for p in a:
        nom = p[0]
        phjs = p[1]
        if pos in phjs:
            res.append(nom)

    return res

print(padres(simp, 'Maggie'))      #['Homer', 'Marge']
print(padres(simp, 'Homer'))       #[['Abraham', 'Mona']]
print(padres(simp, 'Ling'))        #[['Selma']]
print(padres(simp, 'Abraham'))     #[]

```



```

simp = [
    ["Abraham", [1,2]],      #0
    ["Herb", []],             #1
    ["Homer", [3,4,5]],      #2
    ["Bart", []],             #3
    ["Maggie", []],           #4
    ["Lisa", []],              #5
    ["Marge", [3,4,5]],       #6
    ["Mona", [2]],             #7
    ["Clancy", [6,10,11]],    #8
    ["Jackie", [6,10,11]],    #9
    ["Selma", [12]],           #10
    ["Patty", []],             #11
    ["Ling", []]]             #12
]

```

# **CRUZIGRAMA**



# INTERESES

“Tipo prueba”



## Pregunta 1

El crucigrama es un juego en el cual uno debe llenar todas las casillas de una grilla. Las casillas se llenan según una descripción de la palabra que va en esa fila o columna. Para este problema, considera que tendrás una serie de (cualquier número de) palabras que van horizontalmente y solo una que va verticalmente, como se muestra en la siguiente figura:

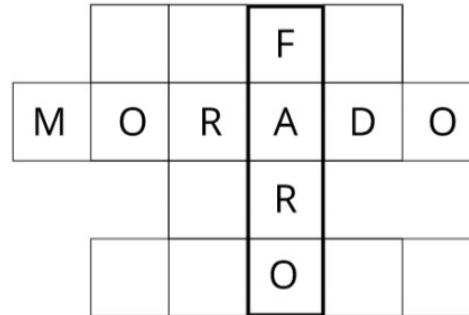


Figura 1: Ejemplo de crucigrama

Tu programa deberá hacer las funciones que permitan llenar cualquier crucigrama. Para esto, considera que un crucigrama se puede modelar de forma que sea una lista de listas, donde cada lista contiene dos cosas: una lista de letras de la palabra, y un índice, que indica la posición donde intersecta la palabra vertical. Siguiendo el ejemplo anterior, el crucigrama C se vería así:

```
C = [  
    [[' ', ' ', 'F', ' ', ], 2],  
    [['M', 'O','R', 'A', 'D', 'O'], 3],  
    [[' ', 'R'], 1],  
    [[' ', ' ', 'O', ' ', ' '], 2]  
]
```



- **(20 puntos) ubicar\_horizontal(C,P,I):** agrega la palabra P horizontalmente al crucigrama C en la lista de índice I, donde P es una lista de letras. Se debe verificar que el largo de P corresponda con el largo de la palabra horizontal en I. Solo se podrá agregar la palabra en el caso que coincida la letra correspondiente a la posición de intersección con la palabra vertical, o en el caso en que ese espacio se encuentre vacío. Si ya había una palabra en la fila I, se reemplaza por la nueva palabra en caso de que cumpla las condiciones anteriores. Retorna el crucigrama actualizado (si la palabra no se pudo agregar, retorna el mismo crucigrama anterior). Por ejemplo, para el crucigrama de la

Figura 1, ubicar\_horizontal(C,['F','L','0','J','0'],3) entrega la lista [[[ "", "", "F", "" ], 2], [[ 'M', '0', 'R', 'A', 'D', 'O' ], 3], [ [" ", "R" ], 1], [[ 'F', 'L', '0', 'J', '0' ], 2]].

```
C = [
    [[ "", "", "F", "" ], 2],
    [[ 'M', '0', 'R', 'A', 'D', 'O' ], 3],
    [ [" ", "R" ], 1],
    [[ "", "", "0", "", "" ], 2]
]
```

			F	
M	O	R	A	D O
			R	
		O		





I'LL WAIT  
FOR YOU HERE



- **(20 puntos) ubicar\_horizontal(C,P,I):** agrega la palabra P horizontalmente al crucigrama C en la lista de índice I, donde P es una lista de letras. Se debe verificar que el largo de P corresponda con el largo de la palabra horizontal en I. Solo se podrá agregar la palabra en el caso que coincida la letra correspondiente a la posición de intersección con la palabra vertical, o en el caso en que ese espacio se encuentre vacío. Si ya había una palabra en la fila I, se reemplaza por la nueva palabra en caso de que cumpla las condiciones anteriores. Retorna el crucigrama actualizado (si la palabra no se pudo agregar, retorna el mismo crucigrama anterior). Por ejemplo, para el crucigrama de la

Figura 1, ubicar\_horizontal(C,['F','L','0','J','0'],3) entrega la lista [[[ "", "", "F", "" ], 2], [[ 'M', '0', 'R', 'A', 'D', 'O' ], 3], [ [" ", "R" ], 1], [[ 'F', 'L', '0', 'J', '0' ], 2]].

```
C = [
    [[ "", "", "F", "" ], 2],
    [[ 'M', '0', 'R', 'A', 'D', 'O' ], 3],
    [ [" ", "R" ], 1],
    [[ "", "", "0", "", "" ], 2]
]
```

			F	
M	O	R	A	D O
			R	
		O		



```
def ubicar_h(C,P,I):  
  
    palh = C[I][0]#Pal H actual  
    posh = C[I][1]#Pos negrita en Pal H  
  
    nv = palh[posh]#Letra negrita Pal V  
    np = P[posh]#Letra negrita Pal dada  
  
    buenlargo = (len(palh) == len(P))  
    buennegrita = (nv == "" or nv == np)  
  
    #Si correcto, actualizar  
    if buenlargo and buennegrita:  
        for i in range(0,len(P)):  
            C[I][0][i] = P[i]  
  
    return C
```

```
C,['F','L','0','J','0'],3
```

			F		
M	O	R	A	D	O
			R		
			O		

```
C = [  
      [[' ', ' ', 'F', ' '], 2],  
      [['M', '0','R', 'A', 'D', '0'], 3],  
      [[' ', 'R'], 1],  
      [[' ', ' ', '0', ' ', ' '], 2]  
]
```



**(20 puntos)** `ubicar_vertical(C,P)`: agrega la palabra `P` de manera vertical al crucigrama `C`, donde `P` es una lista de letras. La palabra **solo se puede agregar si** el largo de `P` corresponde a la altura del crucigrama, y además cada una de las letras de `P` coincide con las letras de las intersecciones con las palabras horizontales, o esos espacios están vacíos. Retorna el crucigrama actualizado (si la palabra no se pudo agregar, retorna el mismo

crucigrama anterior). Por ejemplo, para el crucigrama anterior, `ubicar_vertical(C,['C','A','R','O'])` retorna el mismo crucigrama de la Figura 1, `[[" ", " ", "F", " "], 2], [[M', 'O', 'R', 'A', 'D', 'O'], 3], [[" ", 'R'], 1], [[" ", " ", 'O', " ", " "], 2]]`.

```
C = [
    [['', '', 'F', ''], 2],
    [[M', 'O', 'R', 'A', 'D', 'O'], 3],
    [['', 'R'], 1],
    [['', '', 'O', '', ''], 2]
]
```

			F	
M	O	R	A	D O
			R	
		O		





I'LL WAIT  
FOR YOU HERE



**(20 puntos) ubicar\_vertical(C,P):** agrega la palabra P de manera vertical al crucigrama C, donde P es una lista de letras. La palabra **solo se puede agregar si** el largo de P corresponde a la altura del crucigrama, y además cada una de las letras de P coincide con las letras de las intersecciones con las palabras horizontales, o esos espacios están vacíos. Retorna el crucigrama actualizado (si la palabra no se pudo agregar, retorna el mismo

crucigrama anterior). Por ejemplo, para el crucigrama anterior, `ubicar_vertical(C,['C','A','R','O'])` retorna el mismo crucigrama de la Figura 1, `[[" ", " ", "F", " "], 2], [[M', 'O', 'R', 'A', 'D', 'O'], 3], [[" ", 'R'], 1], [[" ", " ", 'O', " ", " "], 2]]`.

```
C = [
    [['', '', 'F', ''], 2],
    [[M', 'O', 'R', 'A', 'D', 'O'], 3],
    [['', 'R'], 1],
    [['', '', 'O', '', ''], 2]
]
```

			F	
M	O	R	A	D O
			R	
		O		



```
def ubicar_v(C,P):  
  
    nlv = len(C) #Num letras verticales  
  
    buenlargo = (len(P) == nlv)  
  
    #Mirar las letras que hay en la negrita  
    #Y mirar si coinciden con la palabra dada  
    buennegrita = True  
    for i in range(0,nlv):  
        palh = C[i][0]#Pal H actual  
        posh = C[i][1]#Pos negrita en Pal H  
        nv = palh[posh]#Letra negrita Pal V  
        np = P[i]#Letra negrita Pal dada  
  
        #Si no es espacio y no coinciden  
        if nv != " " and nv != np:  
            buennegrita = False  
  
    #Actualizar  
    if buenlargo and buennegrita:  
        for i in range(0,len(P)):  
            posh = C[i][1]  
            C[i][0][posh] = P[i]  
  
    return C
```

[ 'C', 'A', 'R', '0' ]

			F		
M	O	R	A	D	O
			R		
			O		

```
C = [  
    [[' ', ' ', 'F', ' '], 2],  
    [['M', 'O','R', 'A', 'D', 'O'], 3],  
    [[' ', 'R'], 1],  
    [[' ', ' ', 'O', ' ', ' '], 2]  
]
```

# MAPAS (nada nuevo)



# INTERESES

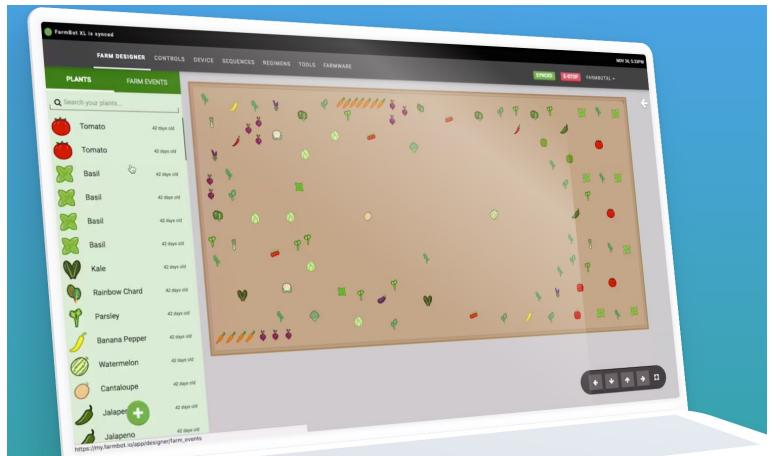
“Agronomía”

“aplicaciones sobre temas agronómicos, por ejemplo algún robot que deba cultivar y cosechar diversas plantas en tales épocas del año, etc...”

“cuidado del medio ambiente y recursos hídricos”



Situación  
Real







```
g = [[ "x", "x" ],  
      [ "x", "x", "x", "x", "o", "o", "x", "x", "x", "x", "x", "x", "x", "x", "x" ],  
      [ "x", "o", "x", "x", "x", "o", "o", "x", "x", "x", "x", "x", "x", "x", "x" ],  
      [ "x", "o", "x", "x", "x", "o", "o", "x", "x", "x", "x", "x", "x", "x", "x" ],  
      [ "x", "o", "o" ]]
```





- FarmBot
  - <https://farm.bot/>
  - [https://www.youtube.com/watch?v=60htrqeI\\_U0](https://www.youtube.com/watch?v=60htrqeI_U0)
  - <https://developer.farm.bot/docs>
  - <https://github.com/openfarmcc>

**ZIPPEDI**



# INTERESES

“robótica”

“robots”

“Inteligencia Artificial (IA), Big Data [...]”

“la industria del retail”



Situación  
Real



## ZIPPEDI: EL PRIMER ROBOT SUPERVISOR DE SUPERMERCADOS

T13

WWW.T13.CL

0:21 / 3:00

Scroll for details





Situación  
Real



## ÁLVARO SOTO

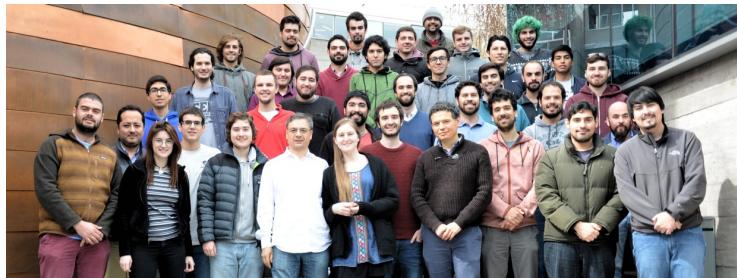
Profesor Asociado

Especialidad: Apredizaje de Máquina,  
Big Data, Robótica Cognitiva,  
Reconocimiento Visual.

Departamento de Ciencia de la  
Computación



asoto@ing.puc.cl  
+56223544440



IA Lab

emol.

Tecnología

Santiago: Jueves 05 de abril del 2018 | Actualizado 09:45



Noticias

Economía

Deportes

Espectáculos Tendencias

Autos

Servicios



Chile

Mundo

Tecnología

Educación

Documentos

Multimedia

Buscar



Zippedi: Cómo es el primer robot chileno que trabaja en supermercados

Gracias al uso de Inteligencia Artificial, el robot revisa precios y el orden de los productos en los estantes de locales comerciales.

10 de Enero de 2018 | 13:29 | Emol

2 2 2 2 2



Zippedi

AHORA SE DEBATE



Hacienda aprieta el cinturón y hará ajuste fiscal por US\$500 millones. ¿Cómo lo ves?

176



¿A qué crees que se debe la crisis que atraviesa la Municipalidad de Viña del Mar y Virginia Reginato?

228

SANTIAGO.- Un grupo liderado por Luis Vera y Ariel Schilkut, fundadores de Zippedi, desarrolló el primer robot chileno que cuenta con Inteligencia Artificial y que tiene el objetivo de verificar que los precios y orden de los productos en supermercados no presenten errores.

Luego de tres años de investigación, el grupo de científicos y alumnos de postgrado de Ciencia de la Computación de la Universidad Católica fue capaz de desarrollar esta innovación a la cual llamaron Zippedi. Junto al apoyo de Corfo, Conicyt y distintas cadenas de retail del país lograron crear el robot íntegramente en Chile.





## Situación Real



Con fecha 21-06-2017 22:42 se publico el siguiente aviso en el curso IIC1103 s.6 Introducción a la Programación:

---

Estimados alumnos,

El examen del día sábado 24 de junio será a las 9:00 hrs. y durará 4 horas.

Se les recomienda que lleven un snack por la duración de éste.

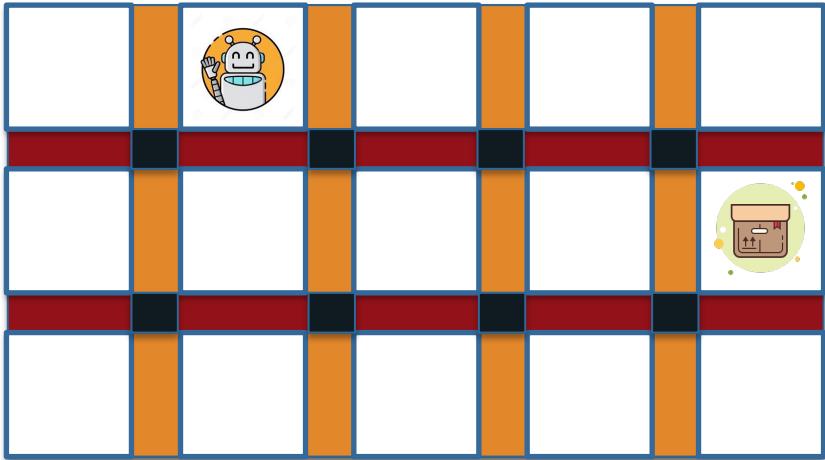
Además mencionarles que el grupo de Inteligencia de Máquina de la PUC busca ayuda para generar datos que ayuden a mejorar las capacidades cognitivas del robot que se encuentran desarrollando. Pueden ayudarlos rotulando en la pagina:

<http://grima-labeling.ing.puc.cl>





Situación  
Real



```
[["-", "R", "-", "-", "-"],  
 ["-", " ", " ", " ", "P"],  
 ["-", " ", " ", " ", "-"]]
```

5

1 seg pasar por naranja y 2 seg por roja

Función que recibe un mapa (list de list) con la situación del robot y el producto, y retorna un int con el tiempo

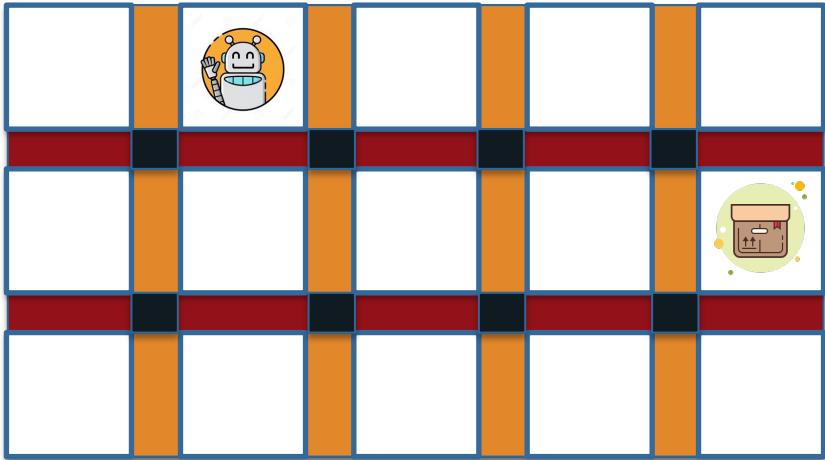




I'LL WAIT  
FOR YOU HERE



Situación  
Real



```
[["-", "R", "-", "-", "-"],  
 ["-", " ", " ", " ", "P"],  
 ["-", " ", " ", " ", "-"]]
```

5

1 seg pasar por naranja y 2 seg por roja

Función que recibe un mapa (list de list) con la situación del robot y el producto, y retorna un int con el tiempo



```
def zippedi(m):  
  
    nfilas = len(m)  
    ncols = len(m[0])  
    #Num de elementos de cualquier fila  
    #por ejemplo, la 0  
  
    #Buscar donde esta el robot y el producto  
    for f in range(0,nfilas):  
        for c in range(0,ncols):  
            if m[f][c] == "R":  
                rf = f  
                rc = c  
            elif m[f][c] == "P":  
                pf = f  
                pc = c  
  
    #Naranjas  
    nar = rc-pc  
    #Puede ser pos o neg dependiendo de si el  
    #robot esta a la izq o derecha del produc  
    #Si es neg lo hacemos pos (mult por -1)  
    if nar < 0:  
        nar = nar * -1  
  
    #Rojas  
    nro = rf - pf  
    if nro < 0:  
        nro = nro * -1  
  
    tiempo = nar*1 + nro*2  
    return tiempo
```

```
[ [ " ", "R", " ", " ", " " ],  
[ " ", " ", " ", " ", "P" ],  
[ " ", " ", " ", " ", " " ] ]
```



## Extensión:

Lista de productos y tengo que hacer un circuito que los recoja todos gastando el mínimo tiempo posible



# Travelling salesman problem

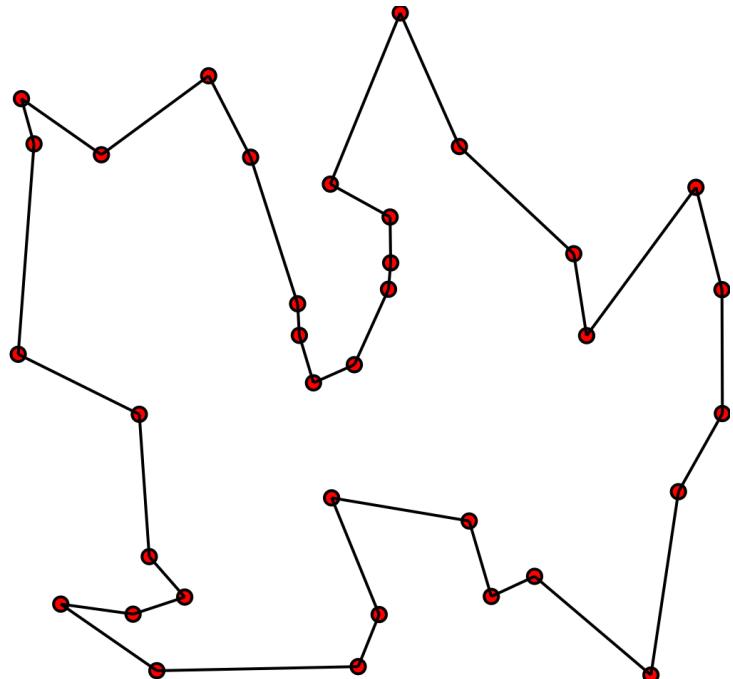
From Wikipedia, the free encyclopedia

The **travelling salesman problem** (also called the **travelling salesperson problem**<sup>[1]</sup> or **TSP**) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" It is an **NP-hard** problem in **combinatorial optimization**, important in **theoretical computer science** and **operations research**.

Demostrado que para encontrar el camino **optimo** hay que probarlos **todos**

20 !=

2.432902e+18





- IA Lab
  - <http://ialab.ing.puc.cl/>
- Zippedi
  - <https://www.youtube.com/watch?v=VCRUCEUoTF8>
  - <https://www.zippedi.com/>
  - <https://www.youtube.com/watch?v=O85Yz8s68u0&feature=youtu.be>
  - <https://www.emol.com/noticias/Tecnologia/2018/01/10/890646/Zippedi-el-primer-robot-chileno-en-trabajar-en-supermercados.html>
- Travelling Salesman Problem (TSP)
  - [https://en.wikipedia.org/wiki/Travelling\\_salesman\\_problem](https://en.wikipedia.org/wiki/Travelling_salesman_problem)

# BUSCAMINAS

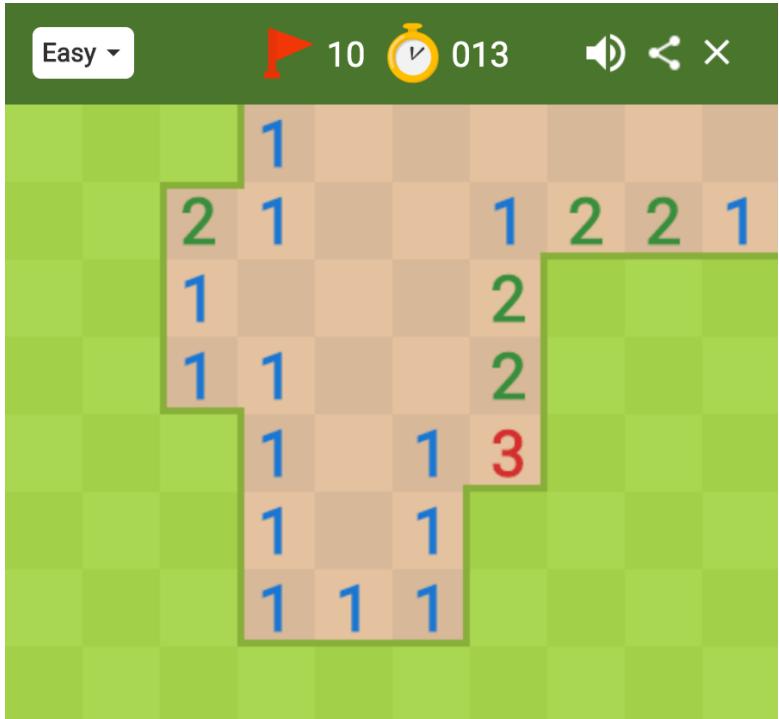


# INTERESES

“videojuegos”



Situación  
Real





```
# Asumir no esta en el borde
def minas(m,f,c):
```



```
    return num
```

```
[ [ "—" , "X" , "X" , "—" ] ,
[ "—" , "—" , "—" , "—" ] ,
[ "X" , "—" , "X" , "X" ] ,
[ "—" , "—" , "X" , "X" ] ,
[ "—" , "—" , "—" , "—" ] ]
```

F: 2 C: 1 Minas: 3  
F: 1 C: 2 Minas: 4





I'LL WAIT  
FOR YOU HERE



```
# Asumir no esta en el borde
def minas(m,f,c):
```



```
    return num
```

```
[ [ "—" , "X" , "X" , "—" ] ,
[ "—" , "—" , "—" , "—" ] ,
[ "X" , "—" , "X" , "X" ] ,
[ "—" , "—" , "X" , "X" ] ,
[ "—" , "—" , "—" , "—" ] ]
```

F: 2 C: 1 Minas: 3  
F: 1 C: 2 Minas: 4



```
# Asumir no esta en el borde
def minas(m,f,c):
    num = 0
    if m[f-1][c-1] == "X":
        num += 1
    if m[f-1][c] == "X":
        num += 1
    if m[f-1][c+1] == "X":
        num += 1
    if m[f][c-1] == "X":
        num += 1
    if m[f][c+1] == "X":
        num += 1
    if m[f+1][c-1] == "X":
        num += 1
    if m[f+1][c] == "X":
        num += 1
    if m[f+1][c+1] == "X":
        num += 1

    return num
```

```
[["-", "X", "X", "-"],  
 ["-", "-", "-", "-"],  
 ["X", "-", "X", "X"],  
 ["-", "-", "X", "X"],  
 ["-", "-", "-", "-"]]
```

F: 2 C: 1 Minas: 3  
F: 1 C: 2 Minas: 4



¿Algo menos manual?

(Imaginar que en vez solo las que tocan, te pidieran mirar las que están distancia 2 o 3)



```
# Asumir no esta en el borde
def minas(m,f,c):
    num = 0

    for i in range(-1,2):
        for j in range(-1,2):

            if m[f+i][c+j] == "X":
                num += 1
    return num
```

```
[["-", "X", "X", "-"],
 ["-", "-", "-", "-"],
 ["X", "-", "X", "X"],
 ["-", "-", "X", "X"],
 ["-", "-", "-", "-"]]
```

F: 2 C: 1 Minas: 3  
F: 1 C: 2 Minas: 4



¿Y si está en un borde?

```
m = [[ "_", "X", "X", "_" ],
      [ " ", " ", " ", " " ],
      [ "X", " ", "X", "X" ],
      [ " ", " ", "X", "X" ],
      [ " ", " ", " ", " " ]]
```

```
r3 = minas(m,4,2)
print("F:",4,"C:",2,"Minas:",r3)
```

```
e/publico/codigos/10-ListasDeListas/buscaminas.py", line 7, in minas
    if m[f+i][c+j] == "X":
IndexError: list index out of range
```

```
r0 = minas(m,0,1)
print("F:",0,"C:",1,"Minas:",r0)
```

F: 0 C: 1 Minas: 2

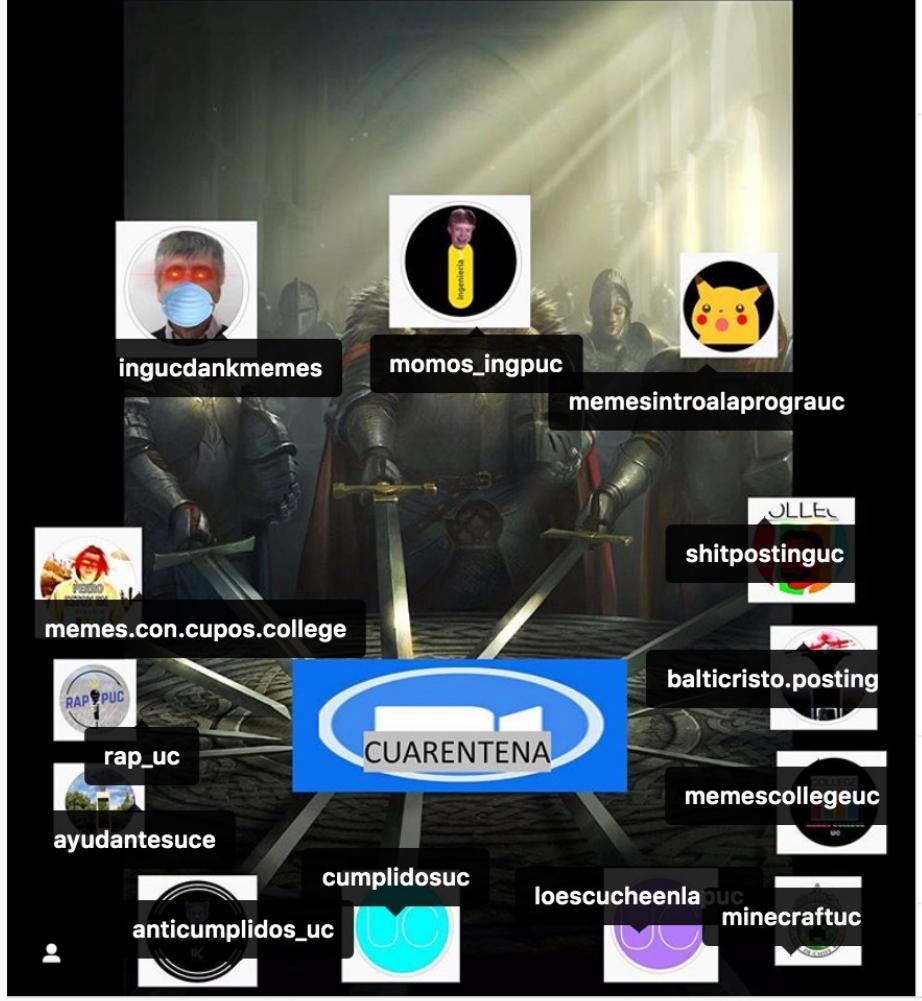


```
def minas(m,f,c):  
  
    nf = len(m)  
    #Num cols son los elemets de cualquier fila  
    #por ejemplo la 0  
    nc = len(m[0])  
  
    num = 0  
  
    for i in range(-1,2):  
        for j in range(-1,2):  
  
            fdentro = (f+i >= 0 and f+i < nf)  
            cdentro = (c+j >= 0 and c+j < nc)  
  
            if fdentro and cdentro:  
                if m[f+i][c+j] == "X":  
                    num += 1  
  
    return num
```

```
[ [ " " , "X" , "X" , " " ] ,  
[ " " , " " , " " , " " ] ,  
[ "X" , " " , "X" , "X" ] ,  
[ " " , " " , "X" , "X" ] ,  
[ " " , " " , " " , " " ] ]
```

```
F: 0 C: 1 Minas: 2  
F: 2 C: 1 Minas: 3  
F: 1 C: 2 Minas: 4  
F: 4 C: 2 Minas: 2
```

# **GRAFOS (nada nuevo)**



memesintroalaprogr • Following ...

memesintroalaprogr 2d 1 like Reply

momos\_ingroupuc Estamos salvando al mundo bros 2d 2 likes Reply

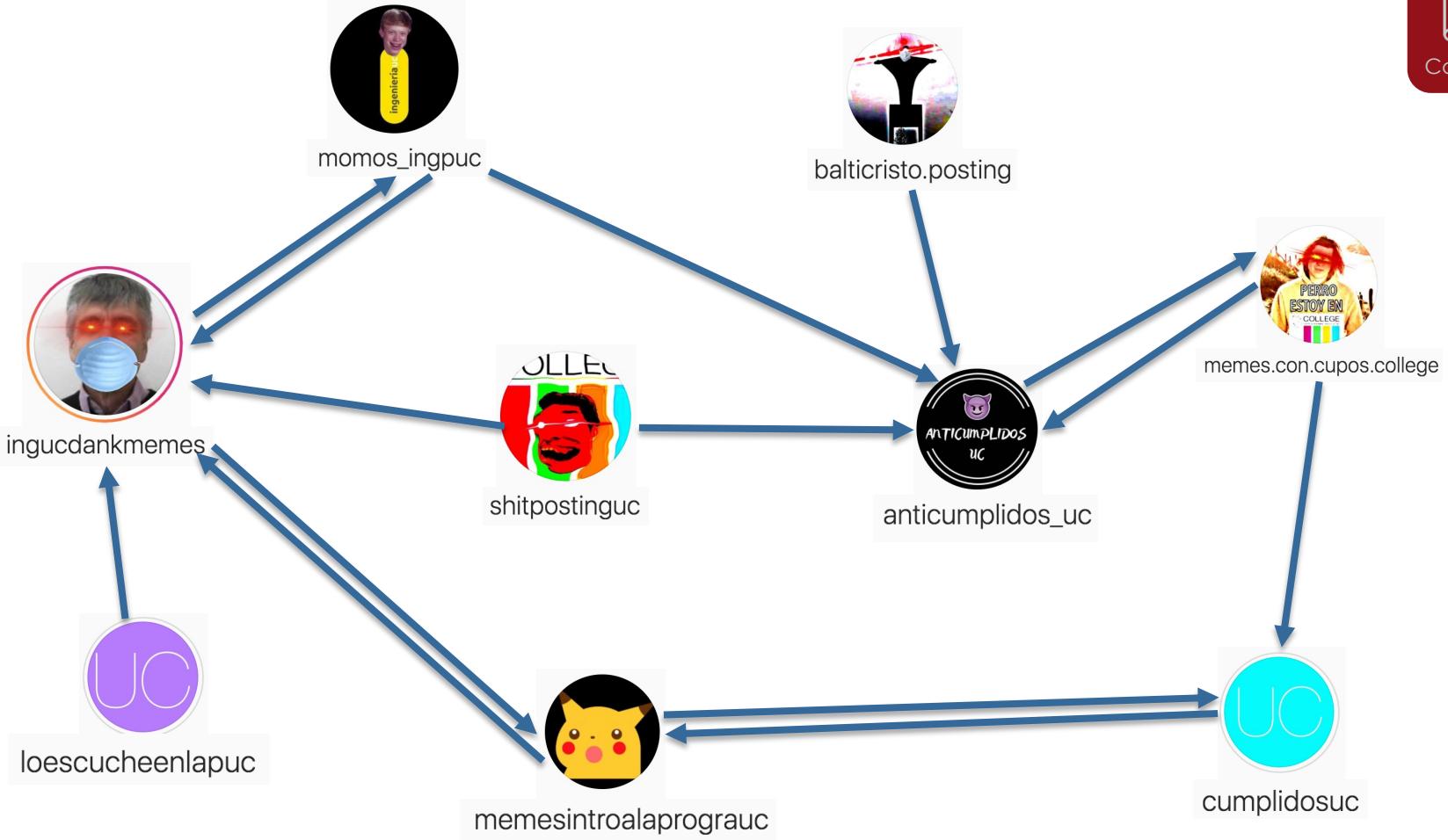
rap\_uc ❤️ 2d 2 likes Reply

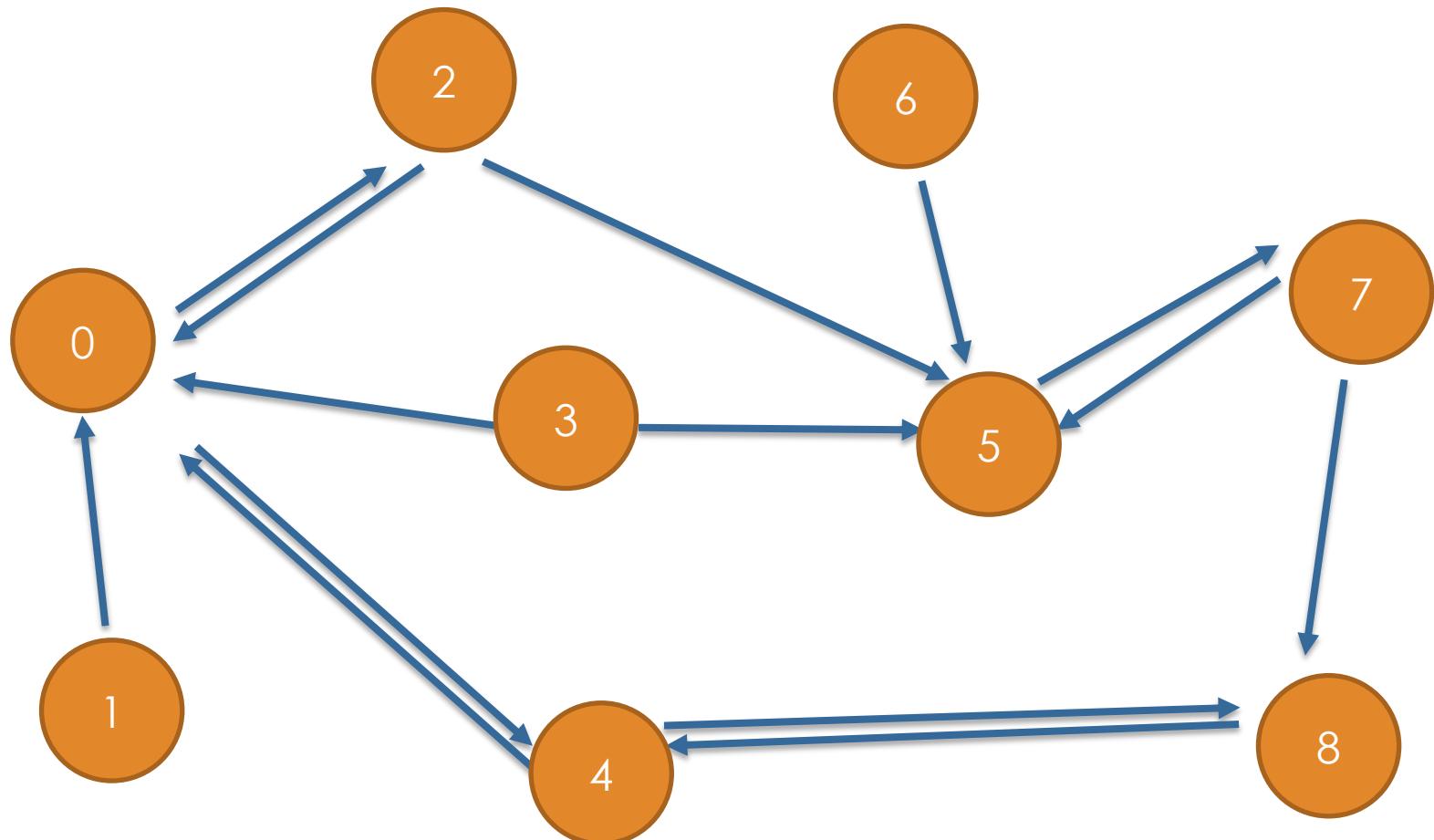
anticumplidos\_uc Somos grandes cabres 2d 4 likes Reply

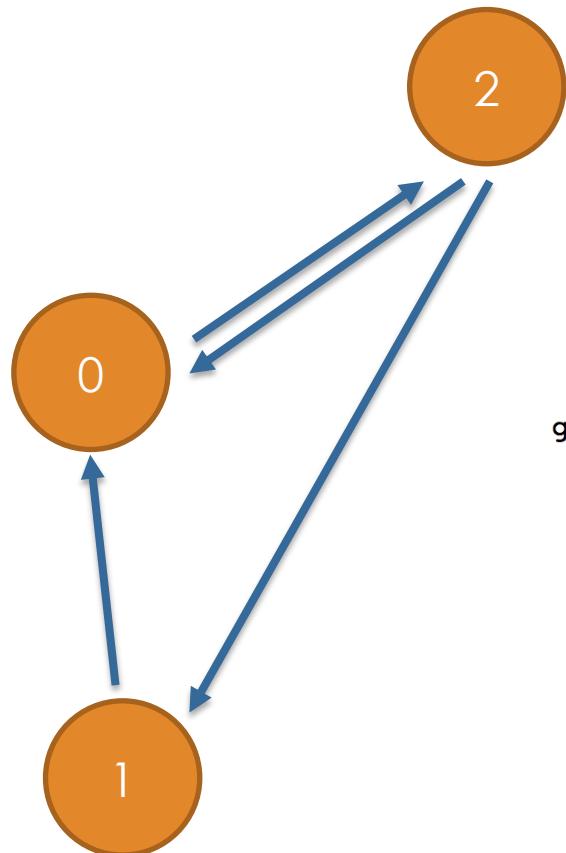
cumplidosuc Holi 2d 2 likes Reply

Liked by shitpostinguc and 162 others 2 DAYS AGO

Add a comment... Post







¿A quien sigue el  
0?

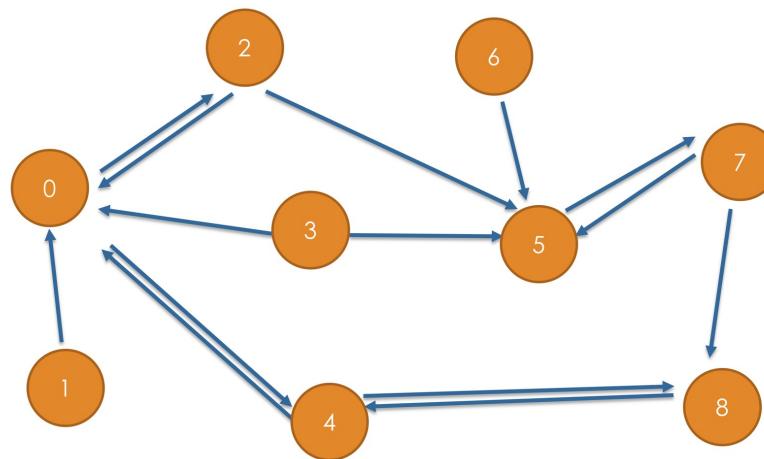
¿A quien sigue el  
1?

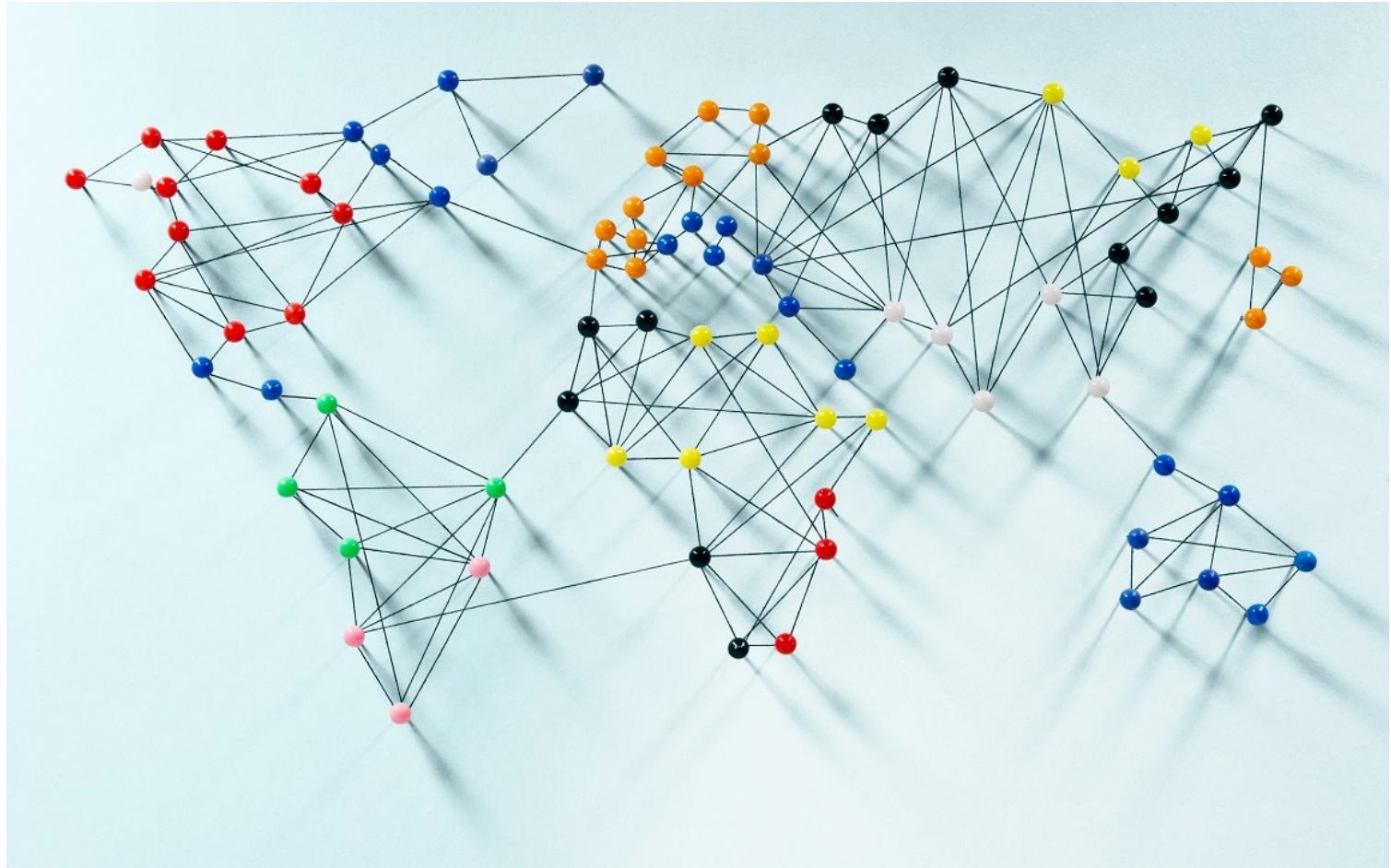
¿A quien sigue el  
2?

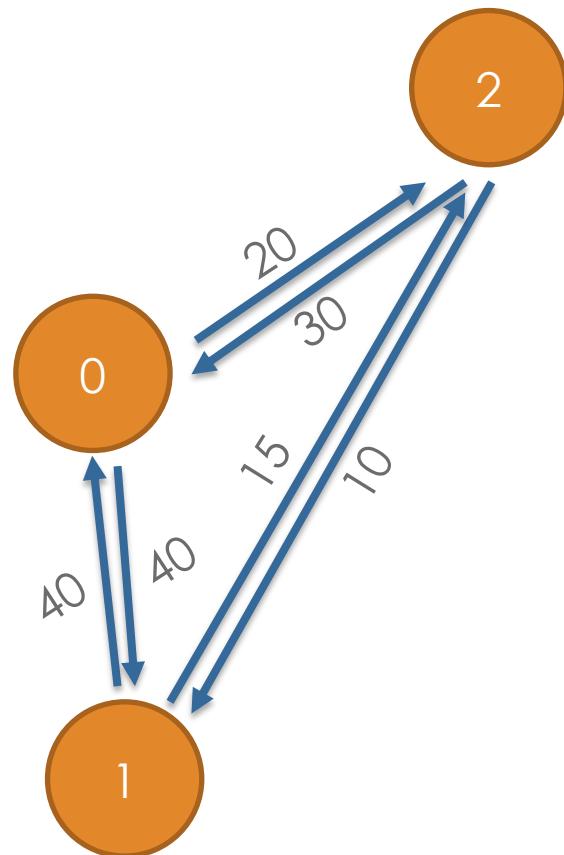
`g = [[False, False, True], [True, False, False], [True, True, False]]`

`g = [[False, False, True],  
[True, False, False],  
[True, True, False]]`

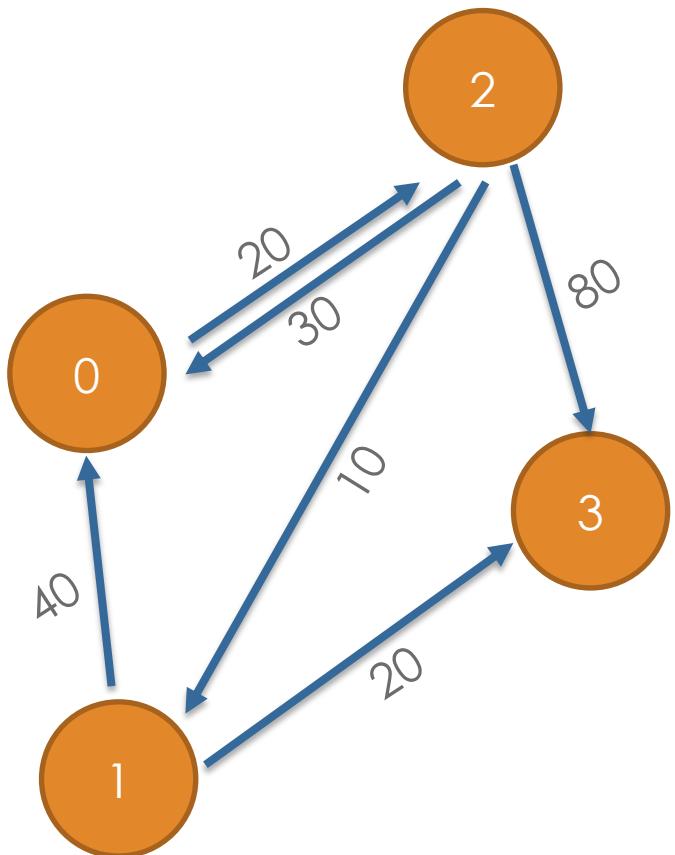
```
g = [[False, False, True, False, True, False, False, False, False],  
      [True, False, False, False, False, False, False, False, False],  
      [True, False, False, False, False, True, False, False, False],  
      [True, False, False, False, False, True, False, False, False],  
      [True, False, False, False, False, False, False, False, True],  
      [False, False, False, False, False, False, False, True, False],  
      [False, False, False, False, False, True, False, False, False],  
      [False, False, False, False, False, True, False, False, True],  
      [False, False, False, False, True, False, False, False, False]]
```







```
g = [[0,40,20],  
     [40,0,15],  
     [30,10,0]]
```



```
g = [[-1,-1,20,-1],  
[40,-1,-1,20],  
[30,10,-1,80],  
[-1,-1,-1,-1]]
```

## Pages in category "Graph algorithms"

The following 125 pages are in this category, out of 125 total. This list may not reflect recent changes ([learn more](#)).

### A

- A\* search algorithm
- Alpha-beta pruning
- Aperiodic graph

### B

- B\*
- Barabási–Albert model
- Belief propagation
- Bellman–Ford algorithm
- Bianconi–Barabási model
- Bidirectional search
- Borůvka's algorithm
- Bottleneck traveling salesman problem
- Breadth-first search
- Bron–Kerbosch algorithm
- Bully algorithm

### C

- Centrality
- Chaitin's algorithm
- Christofides algorithm
- Clique percolation method
- Closure problem
- Color-coding
- Contraction hierarchies
- Courcelle's theorem
- Cuthill–McKee algorithm

### D

- D\*
- Degeneracy (graph theory)
- Depth-first search
- Dijkstra–Scholten algorithm
- Dijkstra's algorithm
- Dinic's algorithm
- Disparity filter algorithm of weighted network
- Double pushout graph rewriting
- DSatur
- Dulmage–Mendelsohn decomposition
- Dynamic connectivity
- Dynamic link matching

### E

- Edmonds–Karp algorithm
- Edmonds' algorithm
- Blossom algorithm
- Euler tour technique
- External memory graph traversal
- Extreme Ensemble Learning

### F

- FKT algorithm
- Flooding algorithm

- Floyd–Warshall algorithm
- Force-directed graph drawing
- Ford–Fulkerson algorithm
- Fringe search

### G

- Gallai–Edmonds decomposition
- Girvan–Newman algorithm
- Goal node (computer science)
- Gomory–Hu tree
- Graph bandwidth
- Graph edit distance
- Graph embedding
- Graph isomorphism
- Graph isomorphism problem
- Graph kernel
- Graph reduction
- Graph traversal

### H

- Havel–Hakimi algorithm
- HCS clustering algorithm
- Hierarchical closeness
- Hierarchical clustering of networks
- Hopcroft–Karp algorithm

### I

- Iterative deepening A\*
- Initial attractiveness
- Iterative compression
- Iterative deepening depth-first search

### J

- Johnson's algorithm
- Journal of Graph Algorithms and Applications
- Jump point search
- Junction tree algorithm

### K

- K shortest path routing
- Karger's algorithm
- KHOPCA clustering algorithm
- Kleinman–Wang algorithms
- Knight's tour
- Knuth's Simpath algorithm
- Kosaraju's algorithm
- Kruskal's algorithm

### L

- Lexicographic breadth-first search
- Longest path problem

### M

- MaxCliqueDyn maximum clique algorithm
- METIS
- Minimax

- Minimum bottleneck spanning tree
- Misra & Gries edge coloring algorithm

### N

- Nearest neighbour algorithm
- Network flow problem
- Network simplex algorithm
- Nonblocking minimal spanning switch

### P

- PageRank
- Parallel all-pairs shortest path algorithm
- Parallel breadth-first search
- Path-based strong component algorithm
- Pre-topological order
- Prim's algorithm
- Proof-number search
- Push–relabel maximum flow algorithm

### R

- Reverse-delete algorithm
- Rocha–Thatté cycle detection algorithm

### S

- Seidel's algorithm
- Semantic Brand Score
- Sethi–Ullman algorithm
- Shortest Path Faster Algorithm
- SMA\*
- Spectral layout
- Spreading activation
- Stoer–Wagner algorithm
- Subgraph isomorphism problem
- Suerballe's algorithm

### T

- Tarjan's off-line lowest common ancestors algorithm
- Tarjan's strongly connected components algorithm
- Theta\*
- Topological sorting
- Transit node routing
- Transitive closure
- Transitive reduction
- Travelling salesman problem
- User:NomenOrmen/sandbox
- Tree traversal

### W

- Widest path problem
- Wiener connector

### Y

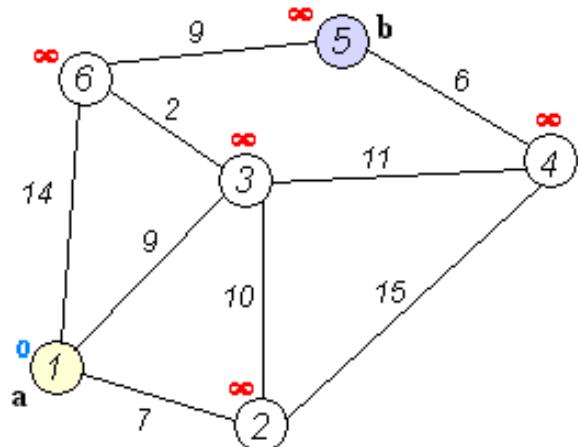
- Yen's algorithm

### Z

- Zero-weight cycle problem

# Algoritmo de Dijkstra

El **algoritmo de Dijkstra**, también llamado **algoritmo de caminos mínimos**, es un **algoritmo** para la determinación del **camino más corto**,





## Conceptos



346 results

Sort by Price per adult

**Stops**

- Direct
  - None
- 1 stop
  - 659 €
- 2+ stops
  - 830 €

**Departure times**

1. Sofia - Barcelona  
00:00 – 23:59

2. Barcelona - Glasgow  
00:00 – 23:59

3. London - New York  
00:00 – 23:59

**Wizz** 18:50 3h 05 → 20:55  
SOF BCN

**BRITISH AIRWAYS** 23:10 10h 35 → 08:45<sup>(+1)</sup>  
BCN GLA

**wow** 11:40 10h 50 → 17:30  
LGW EWR

3 deals from **659 €** Select →

**Wizz** 18:50 3h 05 → 20:55  
SOF BCN

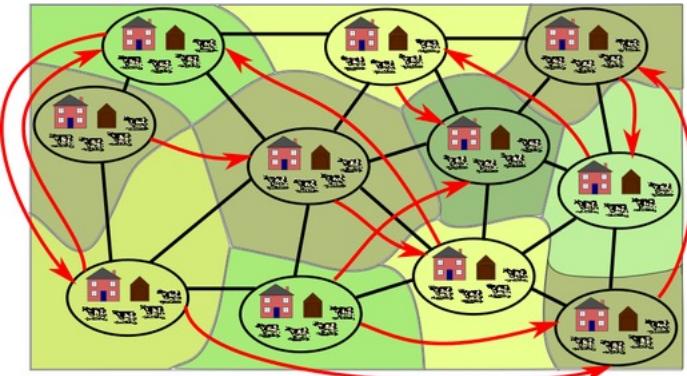
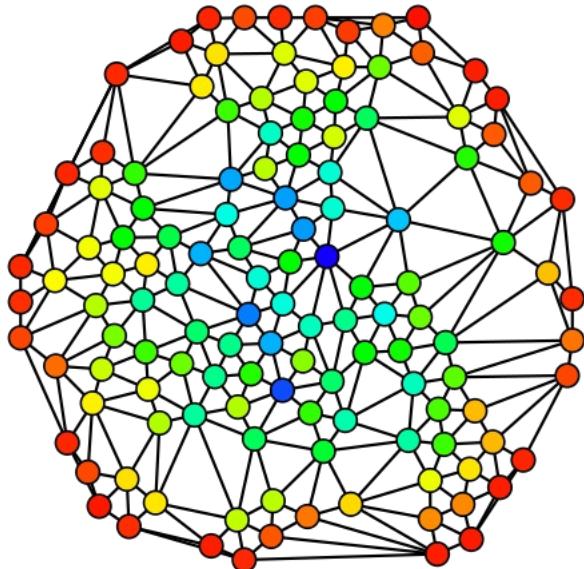
**BRITISH AIRWAYS** 15:30 8h 05 → 22:35  
BCN GLA

**wow** 11:40 10h 50 → 17:30  
LGW EWR

2 deals from **663 €** Select →

# Centralidad

En teoría de grafos y **análisis de redes** la **centralidad** en un **grafo** se refiere a una medida posible de un **vértice** en dicho grafo, que determina su importancia relativa dentro de éste.<sup>1</sup>



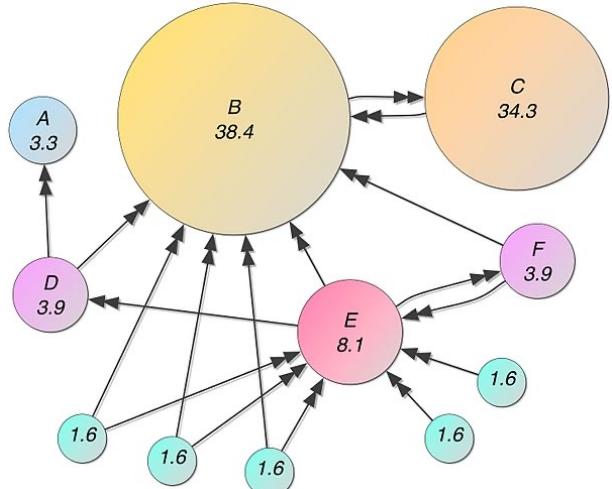
# PageRank

From Wikipedia, the free encyclopedia

*"Google search algorithm" redirects here. For other sea Panda, and Google Hummingbird.*

**PageRank (PR)** is an algorithm used by Google Search to rank web pages in their search engine results.

PageRank was named after Larry Page,<sup>[1]</sup> one of the founders of Google. PageRank is a way of measuring the importance of website pages. According to Google:



# Algoritmo de Kruskal

El **algoritmo de Kruskal** es un [algoritmo](#) de la [teoría de grafos](#) para encontrar un [árbol recubridor mínimo](#)

# Algoritmo de Prim

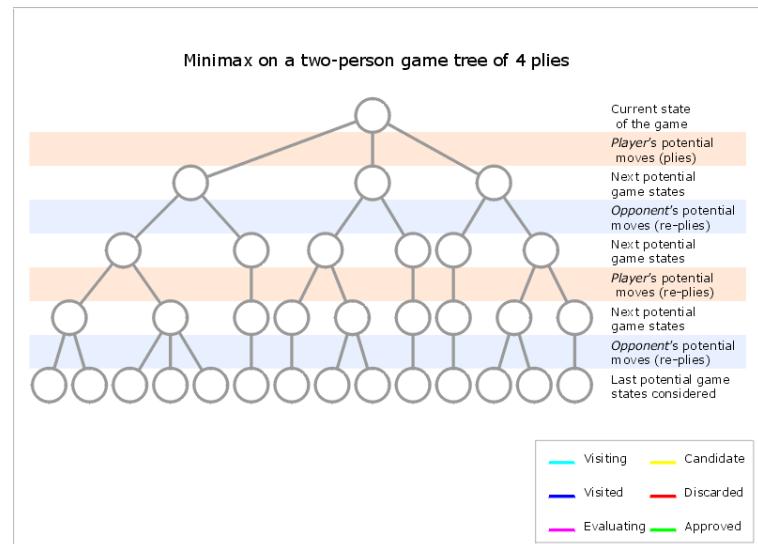
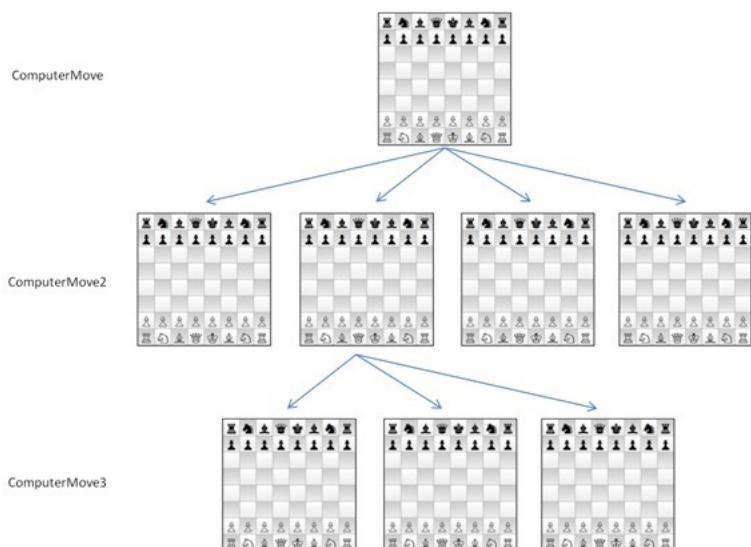
El **algoritmo de Prim** es un [algoritmo](#) perteneciente a la [teoría de los grafos](#) para encontrar un [árbol recubridor mínimo](#)



# Minimax

Para otros usos de este término, véase [Minimax \(infantil\)](#).

En [teoría de juegos](#), **minimax** es un método de decisión para *minimizar* la pérdida *máxima* esperada en juegos con adversario





- Algoritmos
  - [https://en.wikipedia.org/wiki/Category:Graph\\_algorithms](https://en.wikipedia.org/wiki/Category:Graph_algorithms)
  - Internet, Youtube, ...
  - <https://towardsdatascience.com/data-scientists-the-five-graph-algorithms-that-you-should-know-30f454fa5513>
- Hay un libro que me encanta ...
  - ... se me quedó en San Joaquín y no recuerdo el título exacto :S

# **COVID: CENTRALIDAD**

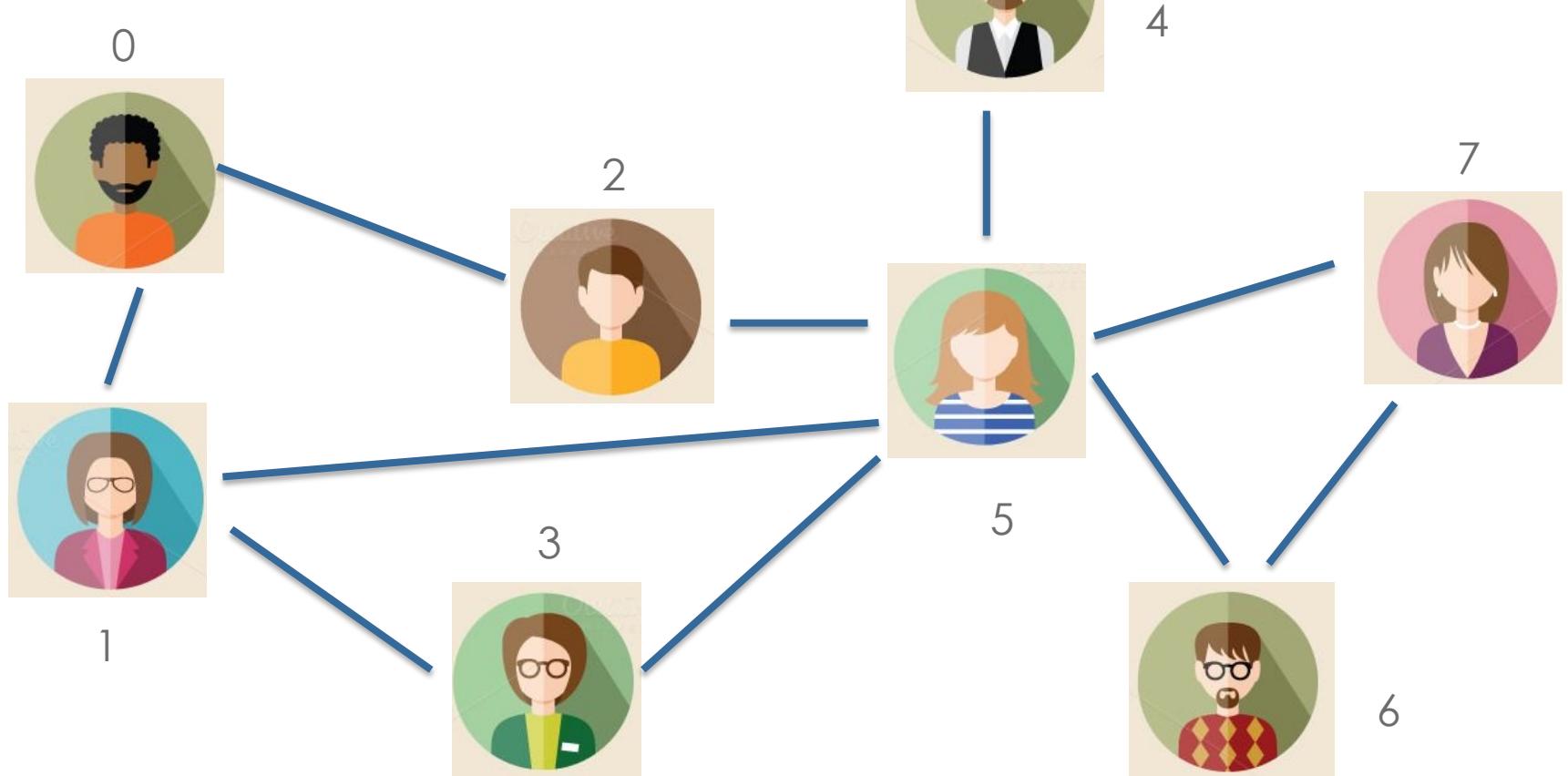


# INTERESES

“COVID”

“Expansión de las enfermedades”

“Actualidad”



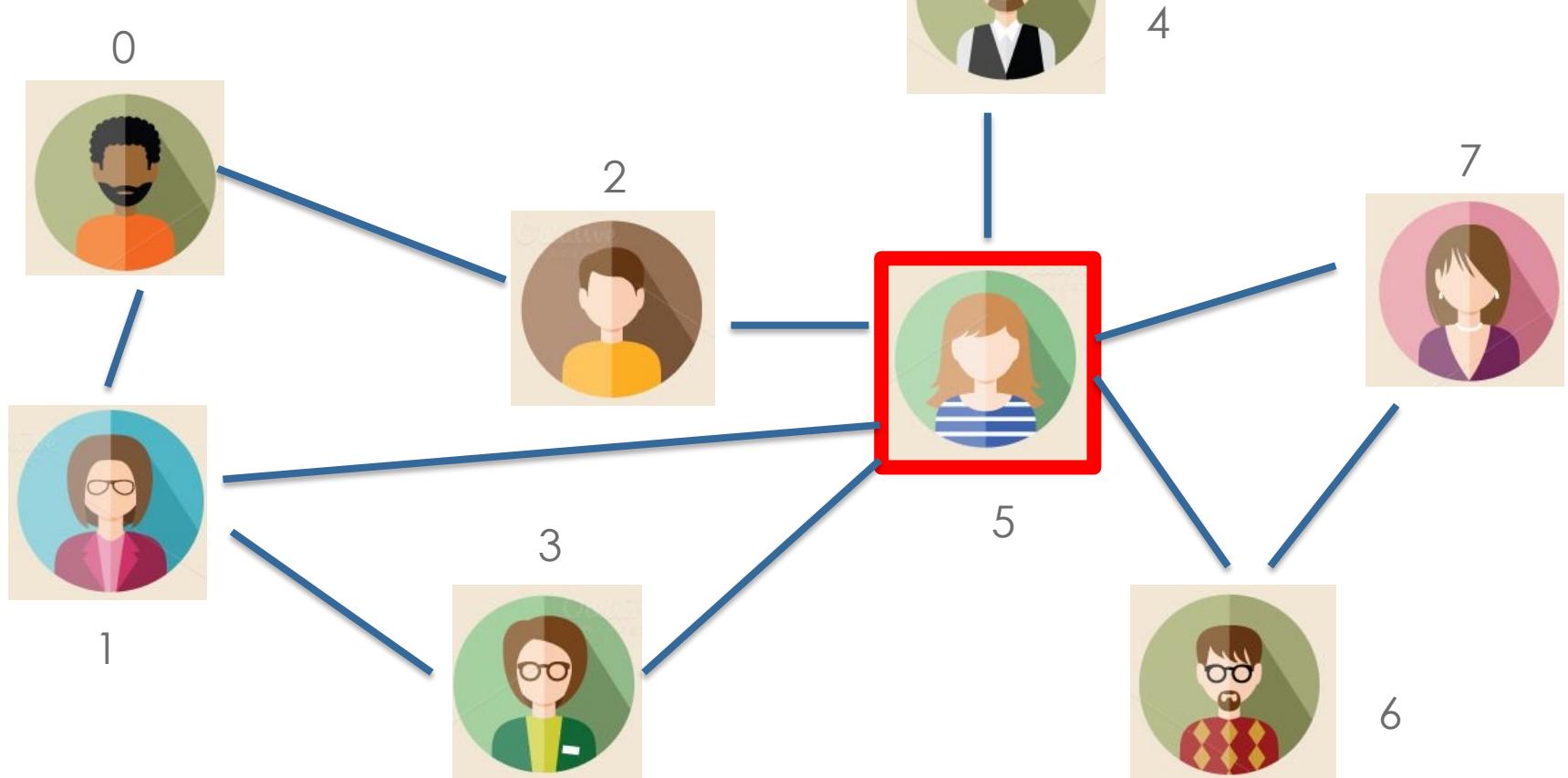
## Degree centrality [ edit ]

---

*Main article: [Degree \(graph theory\)](#)*

Historically first and conceptually simplest is **degree centrality**, which is defined as the number of links incident upon a node (i.e., the number of ties that a node has). The degree can be interpreted in terms of the immediate risk of a node for catching whatever is flowing through the network (such as a virus, or some information). In the

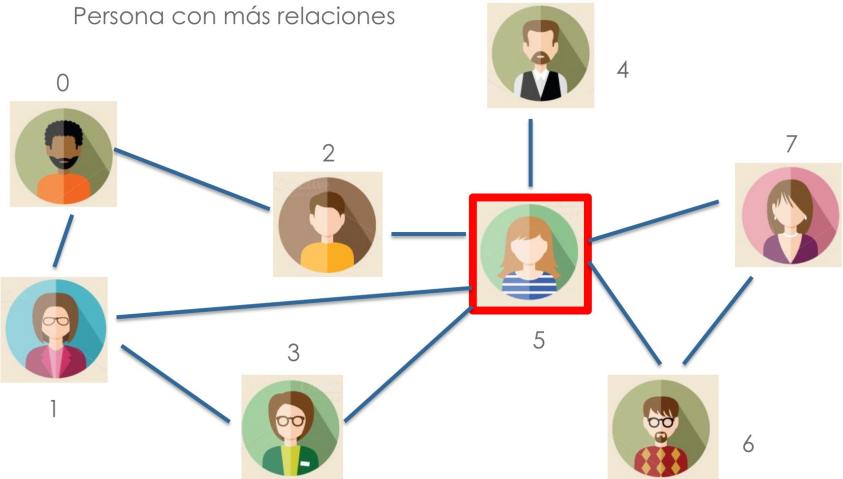
# Persona con más relaciones





5

Persona con más relaciones



```
g = [[False, True, True, False, False, False, False, False],  
[True, False, False, True, False, True, False, False],  
[True, False, False, False, False, True, False, False],  
[False, True, False, False, False, True, False, False],  
[False, False, False, False, False, True, False, False],  
[False, False, False, False, False, True, False, False],  
[False, True, True, True, True, False, True, True],  
[False, False, False, False, False, True, False, True],  
[False, False, False, False, False, True, True, False]]
```

```
def maxrel(g):
```



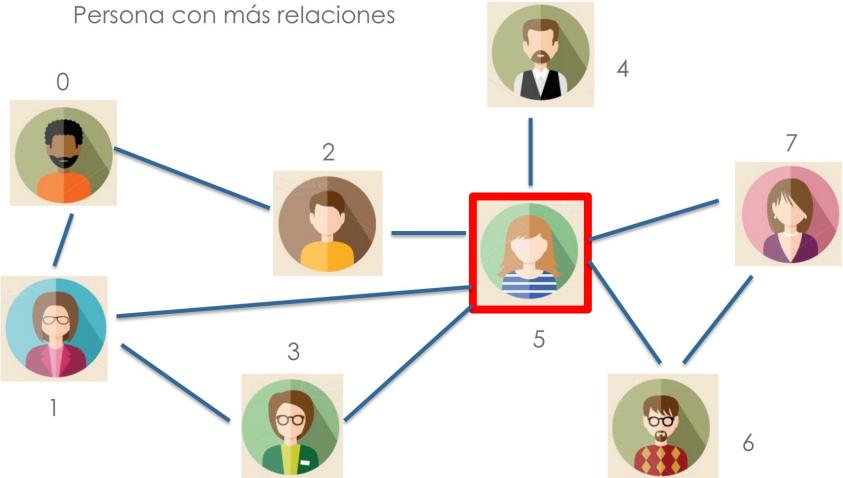


I'LL WAIT  
FOR YOU HERE



5

Persona con más relaciones



```
g = [[False, True, True, False, False, False, False, False],  
[True, False, False, True, False, True, False, False],  
[True, False, False, False, False, True, False, False],  
[False, True, False, False, False, True, False, False],  
[False, False, False, False, False, True, False, False],  
[False, True, True, True, True, False, True, True],  
[False, False, False, False, False, True, False, True],  
[False, False, False, False, False, True, True, False]]
```

```
def maxrel(g):
```



```
def maxrel(g):
    np = len(g) #Número de personas

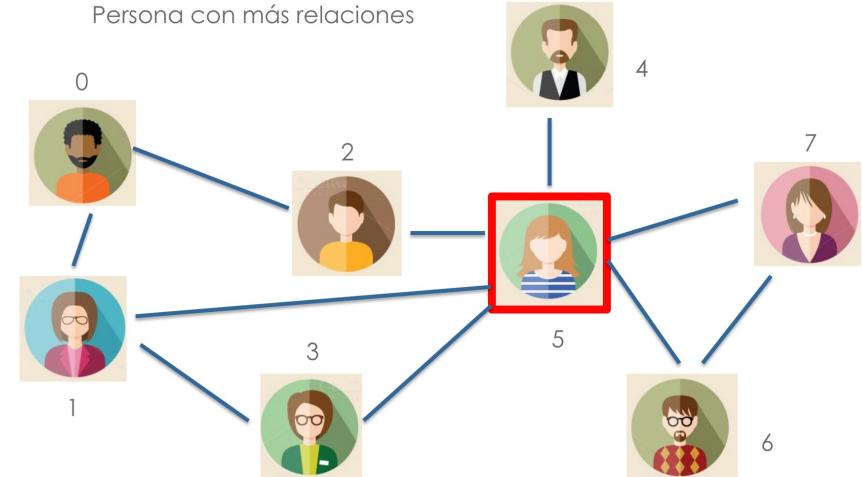
    maxp = -1 #Número de la persona con mas rel
    maxn = -1 #Número de rels de la persona con mas

    #Para cada persona mirar cuantas relaciones
    for i in range(0,np):

        #Contar su numero de Trues
        rel = g[i]
        numT = 0
        for r in rel:
            if r == True:
                numT += 1

        #Mirar si es mayor que lo que llevaba
        if numT > maxn:
            maxn = numT
            maxp = i

    return maxp
```



```
g = [[False, True, True, False, False, False, False, False],
      [True, False, False, True, False, True, False, False],
      [True, False, False, False, False, True, False, False],
      [False, True, False, False, False, True, False, False],
      [False, False, False, False, False, True, False, False],
      [False, True, True, True, True, False, True, True],
      [False, False, False, False, False, True, False, True],
      [False, False, False, False, False, True, True, False]]
```



- Graph Centrality
  - [https://en.wikipedia.org/wiki/Centrality#Degree\\_centrality](https://en.wikipedia.org/wiki/Centrality#Degree_centrality)
  - <https://en.wikipedia.org/wiki/Centrality>

# **COVID: PERSONAS DE CORTE**

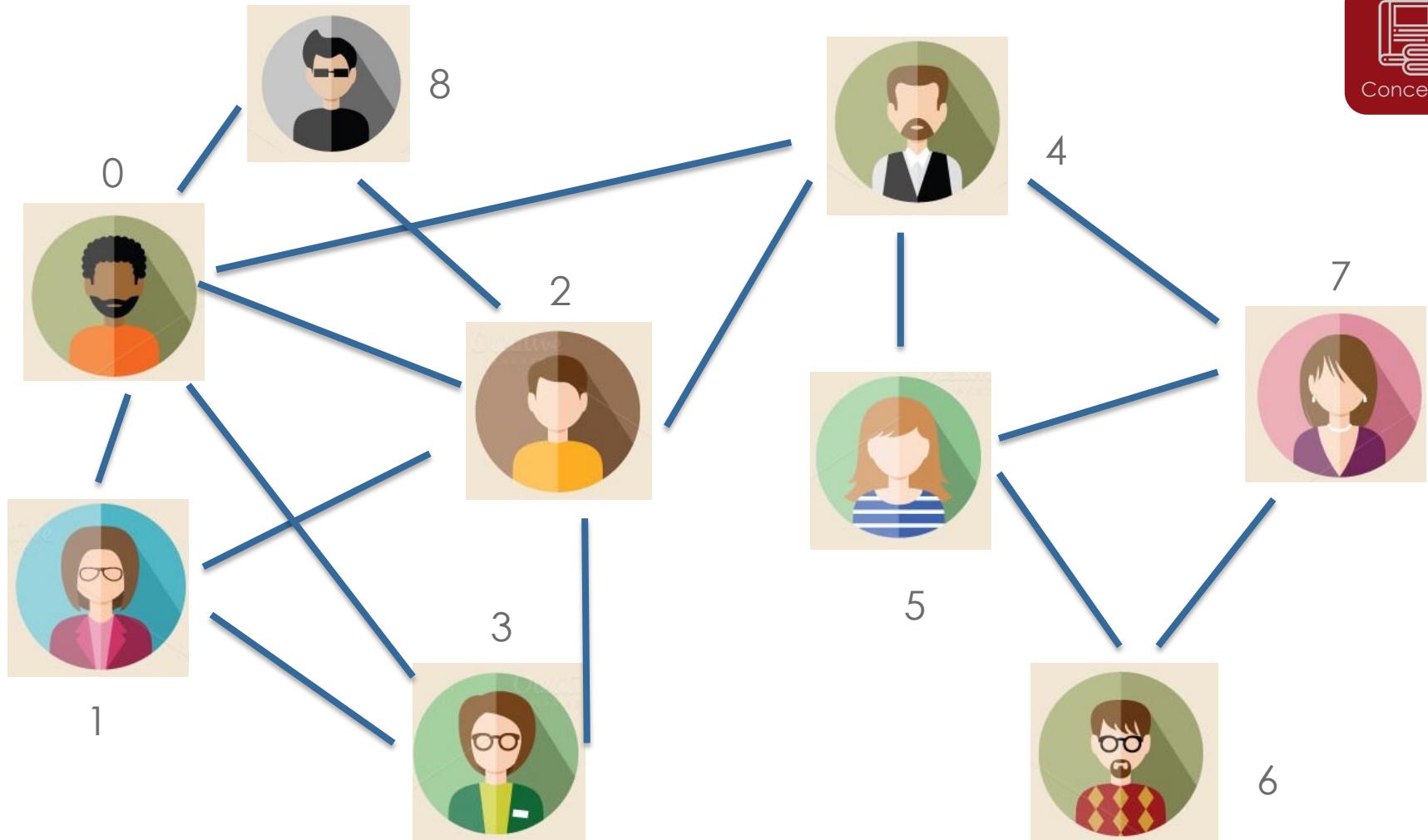


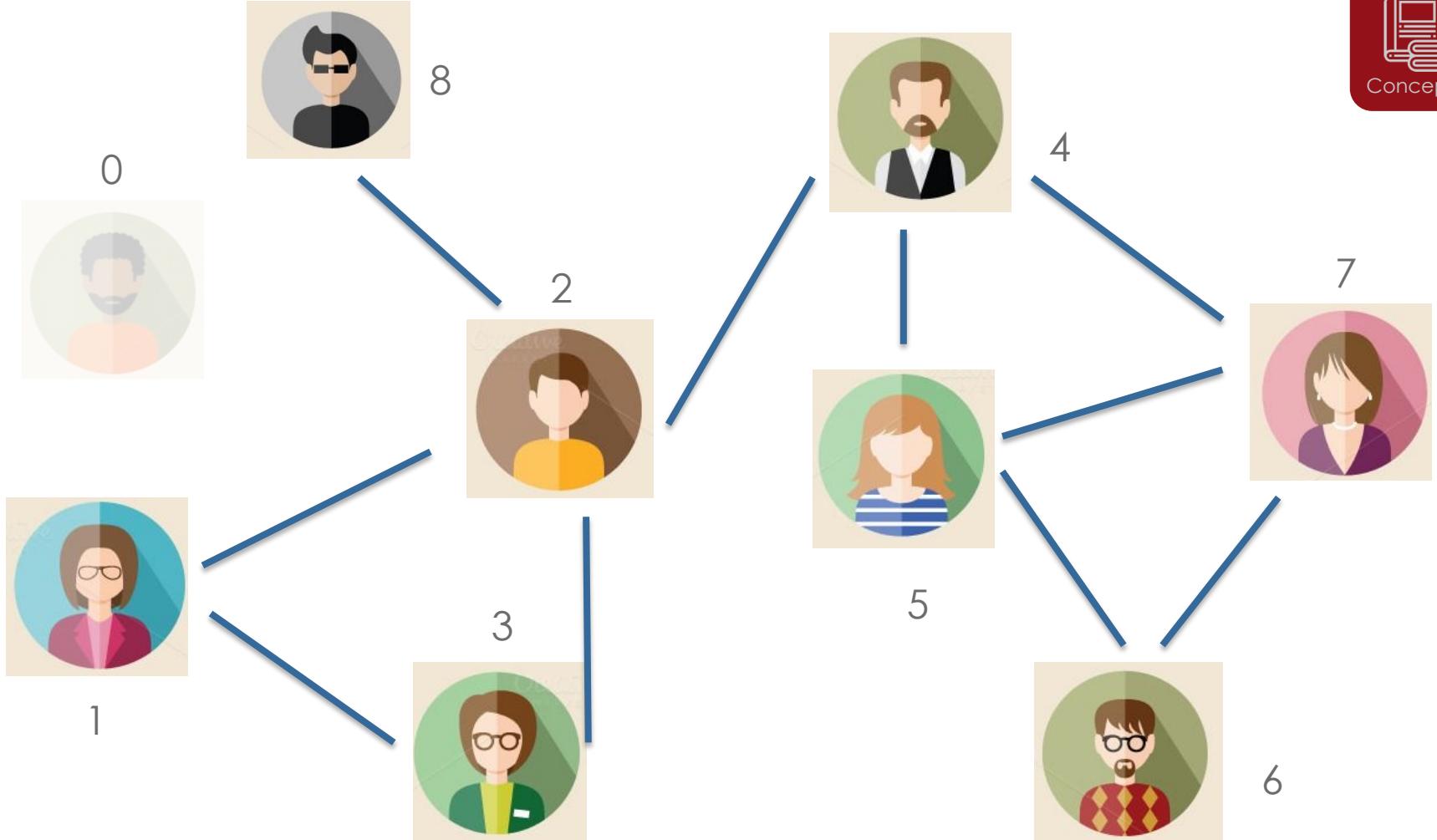
# INTERESES

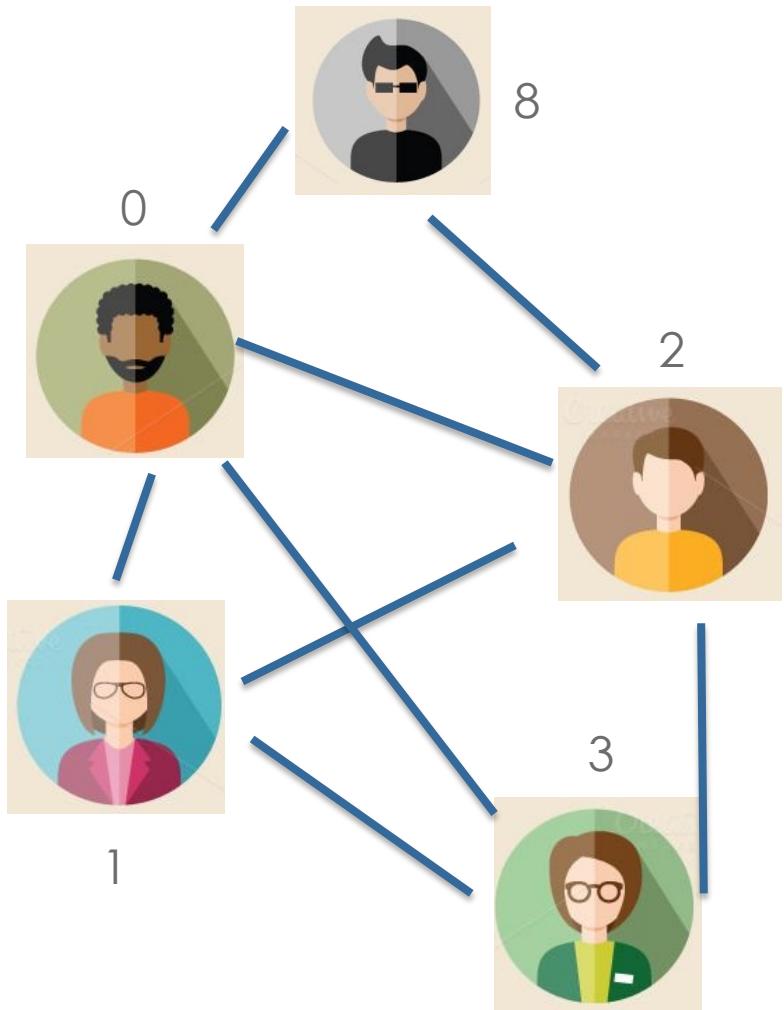
“COVID”

“Expansión de las enfermedades”

“Actualidad”









# ALGORITMO

“Saber que personas son de corte”

Para cada persona ...

1. Quitarla
2. Ver si de una persona cualquiera de las que quedan se puede llegar a todas las demás
  - Si se puede: NO es de corte
  - Si no se puede: es de corte
3. Volver a ponerla



Situación  
Real

Recibe el grafo sin la persona

```
def conectado(g):
```



Retornar True si des de la persona 0 puedes llegar a todas las otras personas

```
[[False,True,False], [[False,True,True],  
[True,False,False], [True,False,False],  
[False,False,False]] [True,False,False]]
```

False

0



1

2



True

0



1

2







I'LL WAIT  
FOR YOU HERE



Parar y pensarlo, y luego os doy una posible pista ...

Sería útil tener 2 listas con los números que ya he visitado y los que debería visitar

Recibe el grafo sin la persona

```
def conectado(g):
```



Retornar True si des de la persona 0 puedes llegar a todas las otras personas

```
[[False,True,False], [[False,True,True],  
[True,False,False], [True,False,False],  
[False,False,False]] [True,False,False]]
```

False

0



1

2



True

0



1

2

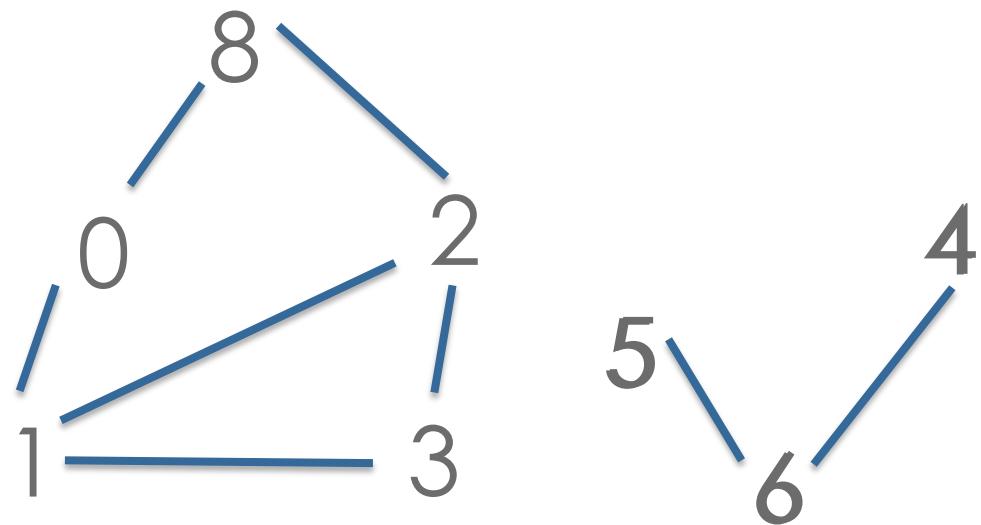




“por mirar”



“ya mirados”

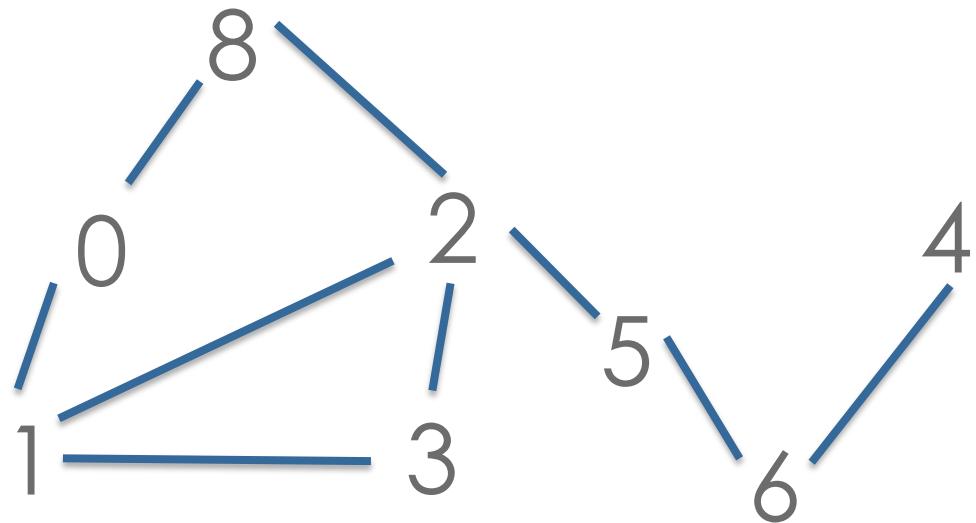




“por mirar”



“ya mirados”





# Recibe el grafo sin la persona

```
def conectado(g):
    pms = [] #personas Por Mirar
    yms = [] #personas Ya Miradas

    #Partire mirando la persona 0
    pms.append(0)

    #Mientras tenga personas por mirar ...
    while len(pms) > 0:

        # Saco a la persona de la lista por mirar
        p = pms.pop()
        # La Pongo en la lista de ya mirados
        yms.append(p)

        #Voy a mirar personas con quien se relaciona
        #Pero solo interesa si no la he mirado ya
        #o no la tengo ya en la lista por mirar
        rel = g[p]
        for i in range(0,len(rel)):
            if (rel[i] == True) and (i not in pms) and (i not in yms):
                pms.append(i)

    #Si el num de personas miradas son todas esta conectado
    return len(yms) == len(g)
```

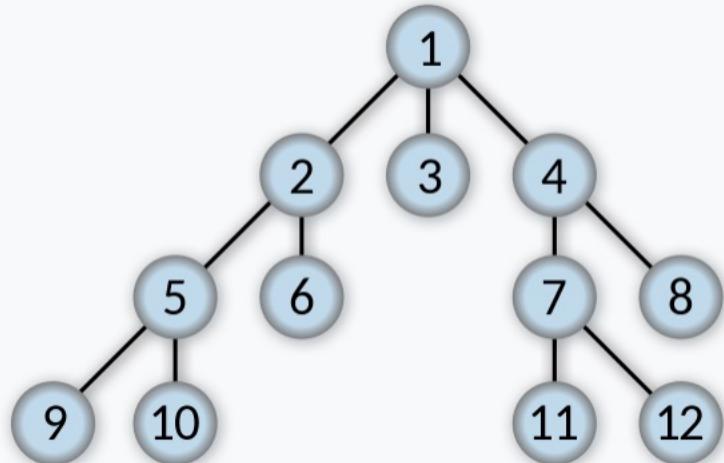
Retornar True si des de la persona 0 puedes llegar a todas las otras personas

```
[[False,True,False], [[False,True,True],
[True,False,False], [True,False,False],
[False,False,False]] [True,False,False]]
```





## Breadth-first search



Order in which the nodes are expanded

<b>Class</b>	Search algorithm
<b>Data structure</b>	Graph
<b>Worst-case performance</b>	$O( V  +  E ) = O(b^d)$
<b>Worst-case space complexity</b>	$O( V ) = O(b^d)$

## Graph and tree search algorithms

[α–β](#) · [A\\*](#) · [B\\*](#) · [Backtracking](#) · [Beam](#) ·  
[Bellman–Ford](#) · [Best-first](#) · [Bidirectional](#) ·  
[Borůvka](#) · [Branch & bound](#) · **BFS** ·  
[British Museum](#) · [D\\*](#) · [DFS](#) · [Dijkstra](#) ·  
[Edmonds](#) · [Floyd–Warshall](#) · [Fringe search](#) ·  
[Hill climbing](#) · [IDA\\*](#) · [Iterative deepening](#) ·  
[Johnson](#) · [Jump point](#) · [Kruskal](#) ·  
[Lexicographic BFS](#) · [LPA\\*](#) · [Prim](#) · [SMA\\*](#) ·  
[SPFA](#)

### Listings

[Graph algorithms](#) · [Search algorithms](#) ·  
[List of graph algorithms](#)

### Related topics

[Dynamic programming](#) · [Graph traversal](#) ·  
[Tree traversal](#) · [Search games](#)



- Algoritmo eficiente para Cut Vertices
  - <https://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph/>
- Algoritmo eficiente para Bridges
  - <https://www.geeksforgeeks.org/bridge-in-a-graph/>
- Breadth-first search (BFS)
  - [https://en.wikipedia.org/wiki/Breadth-first\\_search](https://en.wikipedia.org/wiki/Breadth-first_search)