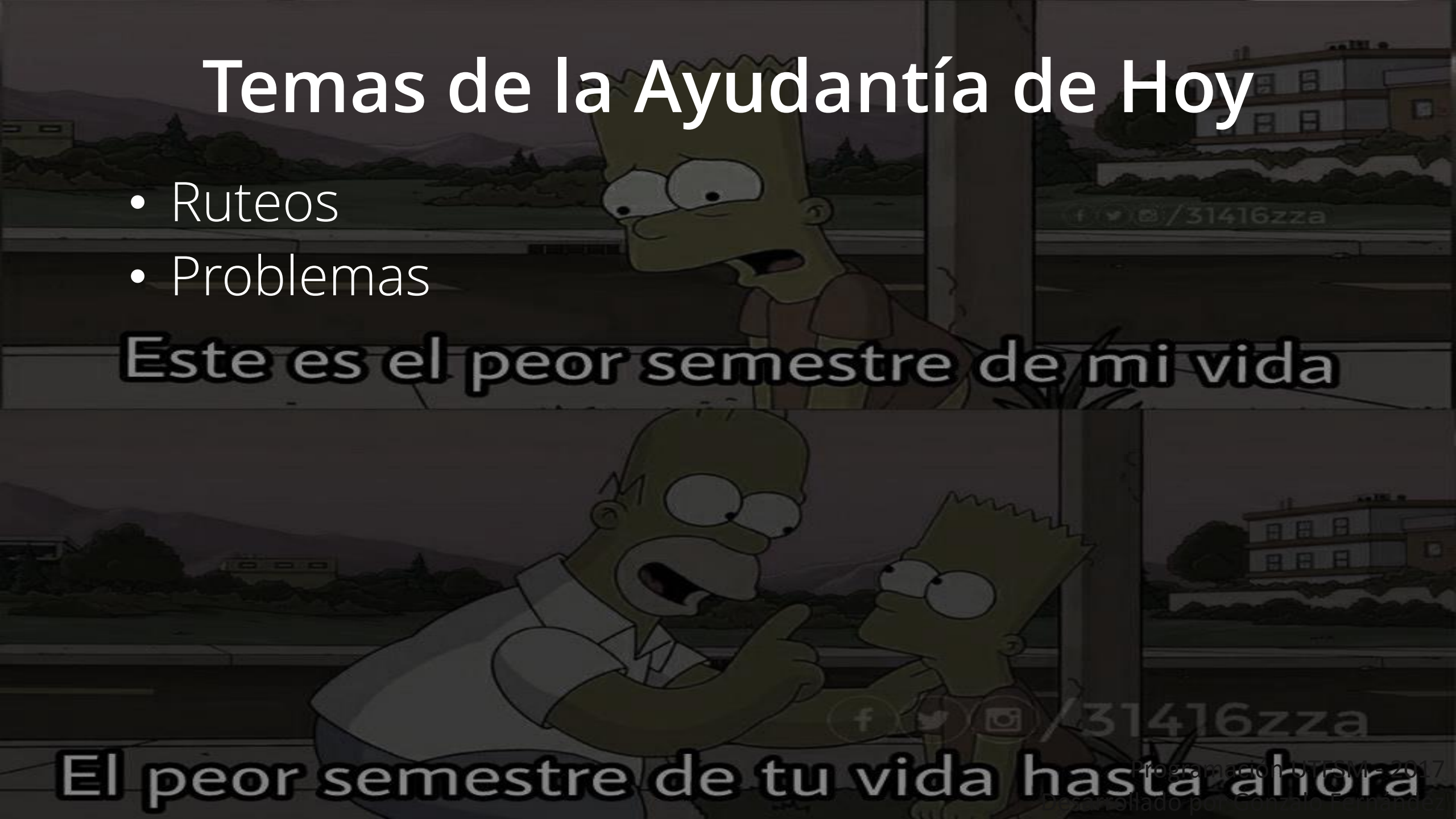
Ayudantía Pre-Certamen 2

Gonzalo Fernández

IWI-131 07/11/2017

Temas de la Ayudantía de Hoy

- Ruteos
- Problemas



```
def f1(li):
    for i in range(1, len(li)):
        v = li[i]
        p = i
        while p > 0 and li[p-1] > v:
            li[p] = li[p-1]
            p = p - 1
            li[p] = v
    return li

print f1([3, 2, 1, 7, 4])
```

```
def f1(li):
    for i in range(1, len(li)):
        v = li[i]
        p = i
        while p > 0 and li[p-1] > v:
            li[p] = li[p-1]
            p = p - 1
            li[p] = v
    return li

print f1([3, 2, 1, 7, 4])
```

f1			
li	i	v	p
[3,2,1,7,4]			
	1		
		2	
			1
[3,3,1,7,4]			
			0
[2,3,1,7,4]			
	2		
		1	
			2
[2,3,3,7,4]			
			1
[2,1,3,7,4]			
[2,2,3,7,4]			
			0
[1,2,3,7,4]			
	3		
		7	
			3
	4		
		4	
			4
[1,2,3,7,7]			
			3
[1,2,3,4,7]			

[1, 2, 3, 4, 7]

```
def f1(lista):
    d = {}
    i = 3
    while i > 0:
        for x in lista:
            if x[1] == i:
                if x[1] not in d:
                    d[x[1]] = []
                d[x[1]].append(x)
        i -= 1
    return d

lista=[(1,3),(1,2),(2,1),(2,3)]
print f1(lista)
```

```
def f1(lista):
  d = {}
  i = 3
  while i > 0:
    for x in lista:
      if x[1] == i:
        if x[1] not in d:
          d[x[1]] = []
        d[x[1]].append(x)
    i -= 1
  return d

lista=[(1,3),(1,2),(2,1),(2,3)]
print f1(lista)
```

{3:[(1,3),(2,3)], 2:[(1,2)], 1:[(2,1)] }

Global	f1			
lista	lista	i	x	d
[(1,3),(1,2),(2,1),(2,3)]				
	[(1,3),(1,2),(2,1),(2,3)]			
				{}
		3		
			(1,3)	
				{3:[]}
				{3:[(1,3)] }
			(1,2)	
			(2,1)	
			(2,3)	
				{3:[(1,3),(2,3)]}
		2		
			(1,3)	
			(1,2)	
				{3:[(1,3),(2,3)],2:[]}
				{3:[(1,3),(2,3)], 2:[(1,2)] }
			(2,1)	
			(2,3)	
		1		
			(1,3)	
			(1,2)	
			(2,1)	
				{3:[(1,3),(2,3)], 2:[(1,2)], 1:[] }
				{3:[(1,3),(2,3)], 2:[(1,2)], 1:[(2,1)] }
			(2,3)	
		0		

```
def ss(L):
    for i in range(0, len(L)):
        j = i
        k = i
        while j < len(L):
            if L[k] > L[j]:
                k = j
            j += 1
        L[i], L[k] = L[k], L[i]
    return L

L = [3,1,2]
print ss(L)
```

```
def ss(L):
    for i in range(0, len(L)):
        j = i
        k = i
        while j < len(L):
            if L[k] > L[j]:
                k = j
            j += 1
        L[i], L[k] = L[k], L[i]
    return L

L = [3,1,2]
print ss(L)
```

```
[1, 2, 3]
```

Global	ss			
L	L	i	j	k
[3, 1, 2]				
	[3, 1, 2]			
		0		
			0	
				0
			1	
				1
			2	
			3	
	[1, 3, 2]			
		1		
			1	
				1
			2	
				2
			3	
	[1, 2, 3]			
		2		
			2	
				2
			3	
	[1, 2, 3]			

Problema 100% Real No Fake

En Pythonia, descubrieron que el **Trap** estimula el cerebro de aquellos que lo escuchan, permitiéndoles procesar el mundo que los rodea en no más de un **KripySegundo**, gracias a esto, la gente dejó de programar, pues sus capacidades cerebrales eran suficientes para resolver problemas que antes requerían de cientos de líneas de código.

Un día, un grupo de **Alienígenas** invaden Pythonia en busca de las últimas fuentes de **cumbia** antes de ser destruidas por los **KushKushianos**, pero por respeto a la raza creadora del mejor género musical, deciden primero establecer una comunicación para resolver este problema de forma pacífica, el **problema**, es que los **Aliens** sólo son capaces de comunicarse mediante estructuras de **Python**, y como los **KushKushianos** ya no programan, no son capaces de descifrar qué dicen.

Es por esto que la brigada **BadBunny** contrata a los alumnos de **IWI-131** para poder decodificar los mensajes y así evitar una guerra contra los alienígenas.

```
M = {  "¿" : {
          1 : ["Asd", "Queremos", 1],
          0 : ["Humanos", "s"],
          5 : ["Asd", 1, 4, [], "as", "Decirles"],
          "C" : ( set("59s"), set("aks4") )
       },
       "s" : {
          6 : ["", 1, 2, 3, 1, 3, "Al Oído"],
          "C" : ( set("asd42"), set("dm3ia") )
       },
       "X" : {
          3 : [ 0, 12, "LoL", "Un Cuitoh en Maincra","/rule"],
          "C" : ( set("Nanda Kore"), set("Wa") ),
       },
       "a" : {
          0 : ["Tantas", 0, [2], {4 : 5}, {1, 2, 3},"34/"],
          9 : ["Cos", "C", 12, 2, 1, 2, 5, 3, 9, "Cosas Preciosas"]
       }
    }
```

1. Dados un mensaje y una **llave inicial**, se leé el mensaje partiendo por ella.
2. Cada valor del mensaje es un diccionario, el cual contiene llaves numéricas que indican **la posición** de la lista donde está el mensaje a, **las llaves se leen de menor a mayor**.
3. Si el diccionario contiene la llave '**C**', la intersección de los conjuntos en ella es **la siguiente llave a leer**
4. El mensaje es descifrado al llegar a un diccionario que no tiene la llave **C**.

```
L = "¿"
M = {  "¿" : {
          1 : ["Asd", "Queremos", 1],
          0 : ["Humanos", "s"],
          5 : ["Asd", 1, 4, [], "as", "Decirles"],
       "C" : ( set("59s"), set("aks4") )
    },
    "s" : {
          6 : ["", 1, 2, 3, 1, 3, "Al Oído"],
       "C" : ( set("asd42"), set("dm3ia") )
    },
    "X" : {
          3 : [ 0, 12, "LoL", "Un Cuitoh en
                Maincra"],
       "C" : ( set("Nanda Kore"), set("Wa") )
    },
    "a" : {
          0 : ["Tantas", 0, [2], {4 : 5}, {1, 2, 3}],
          9 : ["Cos", "C", 12, 2, 1, 2, 5, 3, 9,
               "Cosas Preciosas"]
    }
}
```

Desarrolle la función **descifrar**(**M**, **L**), que, dado un diccionario de diccionarios **M** (el mensaje) y una llave **L** (llave inicial), imprima en pantalla el mensaje que están enviando los aliens.

Ejemplo (usando **M** y **L** definidos anteriormente):

```
>>> descrifrar(M, L)
"Queremos Decirles Al Oído, Tantas Cosas Preciosas"
>>> ...
>>> # Chalo** por favor deja de inventar ***
```

[40 %] Inversiones RCaray está siendo investigada por una presunta estafa millonaria. Su presidente generó una asociación ilícita utilizando los dineros de cientos de personas durante los tres últimos años por medio de varias empresas fantasma. Las investigaciones pudieron reunir una información parcial acerca de las personas y las sumas de dineros involucradas. La policía ha recaudado esta información en una lista llamada *estafados*, la que almacena tuplas de la forma (*rut*, *deuda*, *empresa*, *fecha*), donde rut está en formato string, deuda en formato entero, empresa en formato string y fecha es una tupla (dd, mm, aaaa). A continuación se muestra un ejemplo de la estructura *estafados*:

```
estafados = [
('12.234.567-8', 2000000, 'pelados_furiosos', (25, 5, 2015)),
('9.111.567-k', 5500000, 'multibank', (1, 10, 2014)),
('14.987.007-1', 100000000, 'inversiones_seguras', (30, 11, 2016)),
('12.234.567-8', 10000000, 'multibank', (2, 7, 2015)),
('12.234.567-8', 2500000, 'multibank', (18, 1, 2016)), ...]
```

Una misma persona pudo ser estafada por varias empresas en distintas fechas. De acuerdo a la información anterior, se le solicita implementar las siguientes funciones:

a.- *estafado_por*(*lista*, *rut*), función que recibe 2 parámetros, la lista de tuplas **estafados** y un string correspondiente al rut de la persona. Esta debe retornar una lista con las distintas empresas que estafaron a la persona con dicho rut. En caso de que el rut ingresado no se encuentre, entregar el mensaje: "Rut no estafado".

```
>>> estafado_por(estafados, '12.234.567-8')
>>> ['pelados_furiosos', 'multibank']
>>> estafado_por(estafados, '19.678.222-2')
>>> Rut no estafado
```

b.- *ranking*(*estafados*), función que recibe como parámetro la lista de estafados y retorna un diccionario donde cada llave corresponde al nombre de una empresa, y el valor asociado a cada llave es la cantidad de dinero que cada empresa logró acumular.

```
>>> ranking(estafados)
>>> {'inversiones_seguras': 100000000, 'multibank': 18000000,
     'pelados_furiosos': 2000000, ...}
```

c.- *estafados_por_fecha*(*estafados*, *inicial*, *final*), función que recibe como parámetros la lista de estafados, la tupla fecha inicial y la tupla fecha final en formato tuplas (dd, mm, aaaa) y retorna una lista de tuplas (rut, deuda) ordenadas en forma descendente según la deuda total generada entre las fechas entregadas.

```
>>> estafados_por_fecha(estafados,(03,01,2016),(01,01,2017))
>>> [('14.987.007-1', 100000000), ('12.234.567-8', 2500000), ...]
```

NOTA: Los puntos suspensivos en las estruturas de datos indican que existen más datos y que solo se muestran unos pocos a modo de ejemplo.

[40 %] Las *Bibliotecas USM* poseen un sistema de préstamo y devolución de libros para alumnos, funcionarios y profesores. Su catálogo de libros está dado por la lista de tuplas `catálogo`, donde cada tupla tiene el número de clasificación, nombre, campus y tipo de préstamo para un libro en particular.

```
catalogo = [('010.245C1', 'Programacion en Python', 'CCV', 'AD'),
            ('145.261D7', 'Algebra Lineal', 'CSJ', 'CB'),
            ('121.748F2', 'Fisica Universitaria', 'CSV', 'CG'), ... ]
```

El tipo de préstamo puede ser `'AD'` (*Alta Demanda*), `'CG'` (*Colección General*) o `'CB'` (*Colección Básica*) con devolución en 2, 7 y 14 días después de la fecha de préstamo, respectivamente. Los préstamos vigentes de libros se manejan en la estructura `prestamos`, donde se guarda el rut del usuario, el número de clasificación y la fecha de préstamo en el formato `(dia, mes, año)`.

```
prestamos = {('12422532-2', '145.261D7', (24, 5, 2017)),
             ('14025935-K', '145.261D7', (13, 5, 2017)),
             ('12422532-2', '010.245C1', (10, 5, 2017)), ... }
```

A continuación, se desea conocer para **una fecha de consulta posterior** a la fecha de todos los préstamos del sistema, la multa que le corresponde a cada usuario por no devolver uno o varios libros a tiempo. Para ello debe crear la función `obtener_multa(catalogo, prestamos, fecha_consulta)` que reciba como parámetros el catálogo de libros, los préstamos registrados en el sistema y la fecha de consulta como una tupla en el formato `(año, mes, dia)`. Esta función debe entregar un diccionario, cuyas llaves corresponden a los ruts de los usuarios de la biblioteca y el valor corresponde a un entero con el monto adeudado. Además, considere lo siguiente:

- La multa por día de atraso corresponde a $1000, $500 y $100 para los libros de Alta Demanda, Colección General y Colección Básica, respectivamente.
- Todos los meses del año tienen 30 días.

Usted puede crear las funciones que estime necesarias para la resolucion de este problema.

```
print obtener_multa(catalogo, prestamos,(2017,6,1))
{'14025935-K': 400, '12422532-2': 19000}
print obtener_multa(catalogo, prestamos,(2017,5,26))
{'12422532-2': 14000}
```

[40 %] La empresa frutícola Pythonfrut se dedica a la producción, procesamiento y exportación de distintos tipos de frutas. Ellos tienen registradas las frutas cosechadas en cada mes en una lista de conjuntos, donde el índice en la lista indica el mes y el conjunto contiene las frutas que se cosechan en éste.

```
frutas = [
    set(['manzana', 'frutilla', 'platano', 'mora']),
    set(['manzana', 'frutilla', 'mora']),
    set(['manzana', 'pera', 'durazno', 'mora']), # ...
]
```

Ahora usted debe:

Desarrollar la función `cantidades(frutas)` que reciba como parámetro la lista de conjuntos de `frutas` y retorne una lista de tuplas, donde cada tupla almacena una fruta y la cantidad de meses donde se cosecha.

```
>>> cantidades(frutas)
[('manzana', 3), ('frutilla', 2), ('platano', 1), ('pera', 1), ('
   durazno', 1), ('mora', 3)]
```

Desarrollar la función `frutas_mas_repetidas(frutas)` que reciba como parámetro la lista de conjuntos de `frutas` y retorne una lista con los nombres de las frutas que se producen en la mayor cantidad de meses.

```
>>> frutas_mas_repetida(frutas)
['manzana', 'mora']
```

Desarrolle la función `fruta_exclusiva(frutas, mes)` que reciba como parámetro la lista de conjuntos de `frutas` y un entero correspondiente al `mes`, los que están enumerados desde cero. La función debe retornar un conjunto con las frutas que sólo se producen en dicho mes. En caso de no haber, debe retornar un conjunto vacío.

```
>>> fruta_exclusiva(frutas, 0)
set(['platano'])
```

[40 %] En una tierra lejana, de una época desconocida, se necesita determinar al rey entre los nobles de 5 poderosas familias. Cada noble (vivo o muerto) con descendencia es almacenada en un diccionario, donde la llave es el nombre del sujeto y como valor el conjunto de nombres de sus hijos. Un **ejemplo reducido** sería el siguiente diccionario:

```
hijos = {
  'Tywin': set(['Tyron', 'Jaime', 'Cersei']),
  'Jaime': set(['Joffrey', 'Myrcella', 'Tommen']),
  'Cersei': set(['Joffrey', 'Myrcella', 'Tommen']),
  'Eddard': set(['Robb', 'Sansa', 'Arya', 'Brandon', 'Rickon', 'Jon']),
  'Catelyn': set(['Robb', 'Sansa', 'Arya', 'Brandon', 'Rickon']),
  # ...
}
```

Cada individuo es hijo de un padre y una madre, si uno de estos no se encuentra en el diccionario es porque: o no era parte de la nobleza o no hay registros. Todo hijo de padre y/o madre noble es por herencia noble.

Además se tiene un conjunto con los nombres de todos los nobles muertos desde que existe registro:

```
muertos = set(['Tywin', 'Tommen', #... ])
```

Ahora usted debe ayudar a determinar al siguiente rey entregando una lista con todos los candidatos de linaje más puro que aún están con vida. Un linaje es más puro mientras más cantidad de generaciones nobles tiene. Como **ejemplo**, el linaje de Joffrey tiene 3 generaciones; la de él, la de sus padres (Jaime y Cersei) y la de su abuelo (Tywin).

Cree la función `candidatos(hijos, muertos)` que recibe las dos estructuras anteriormente descritas y entrega como resultado una lista con todos los candidatos que aún están vivos. Para el **ejemplo** esta sería:

```
['Joffrey', 'Myrcella']
```

Además recuerde que puede crear todas las funciones auxiliares que estime conveniente.