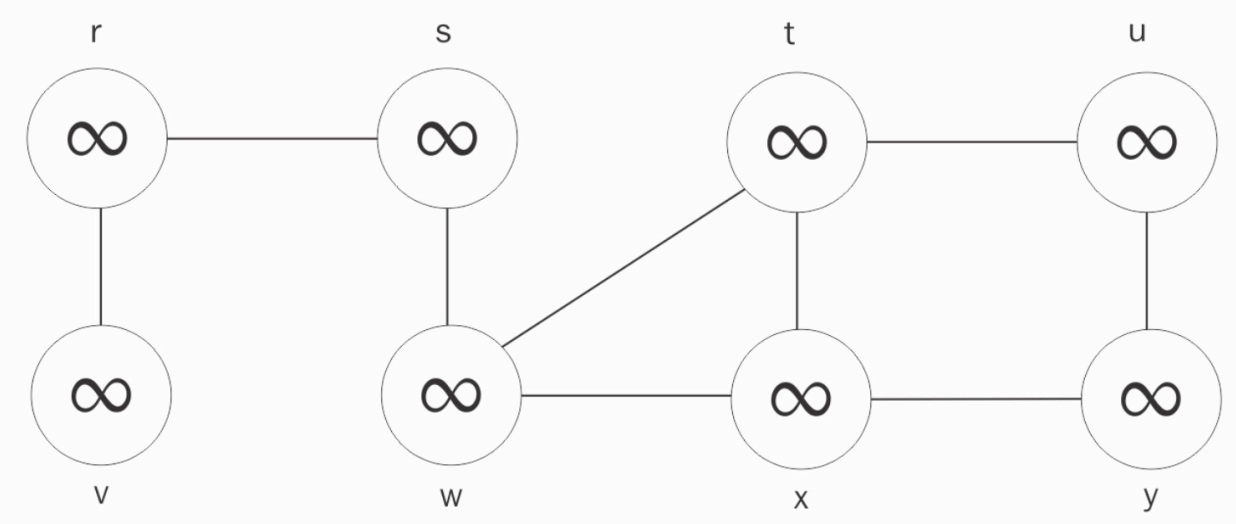
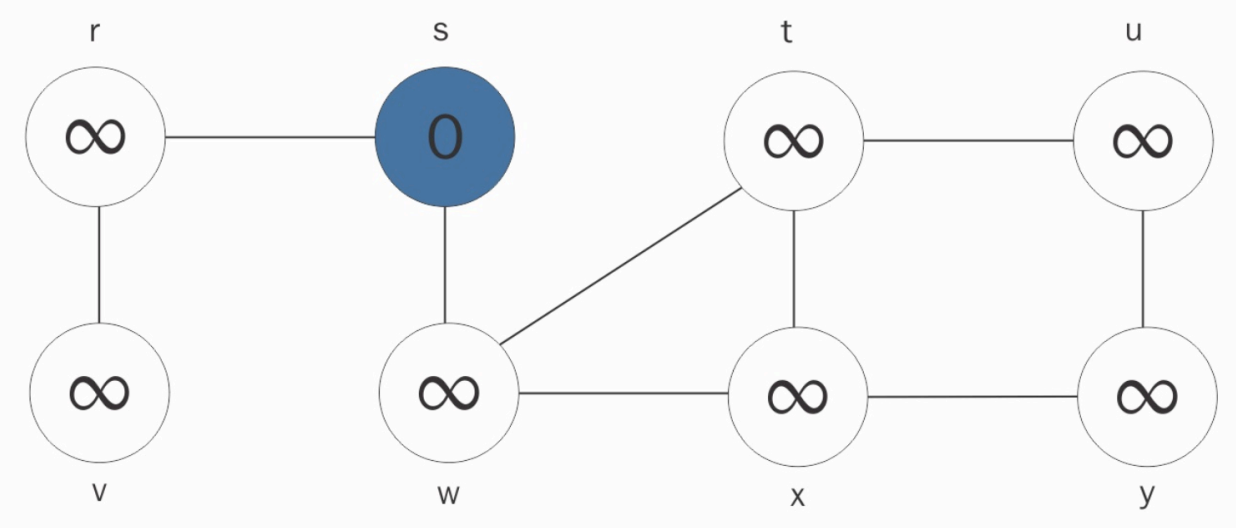
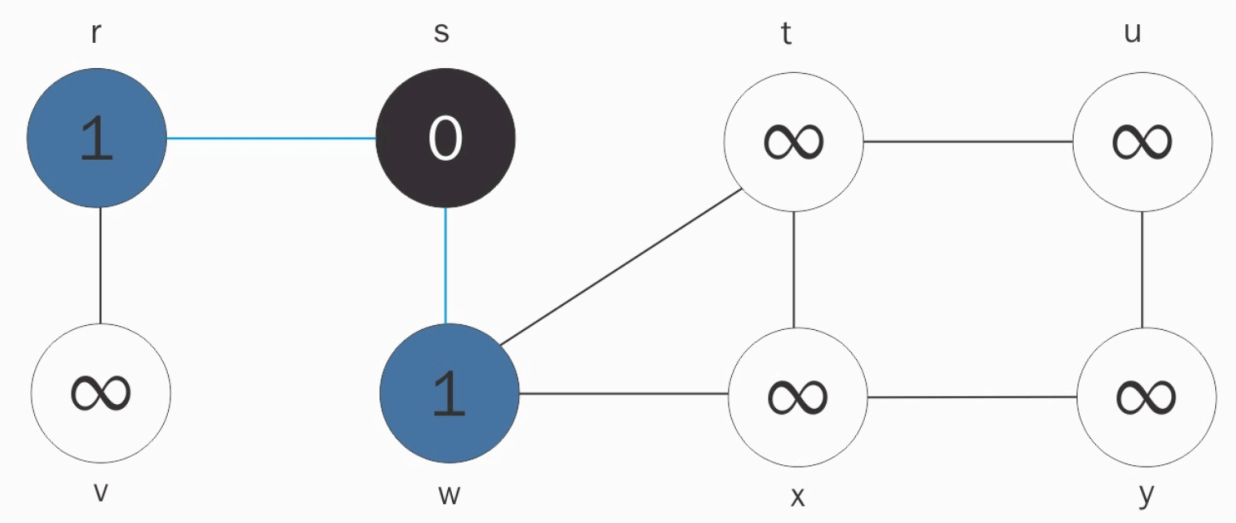
grados-de-separacion/grados-de-separacion.py

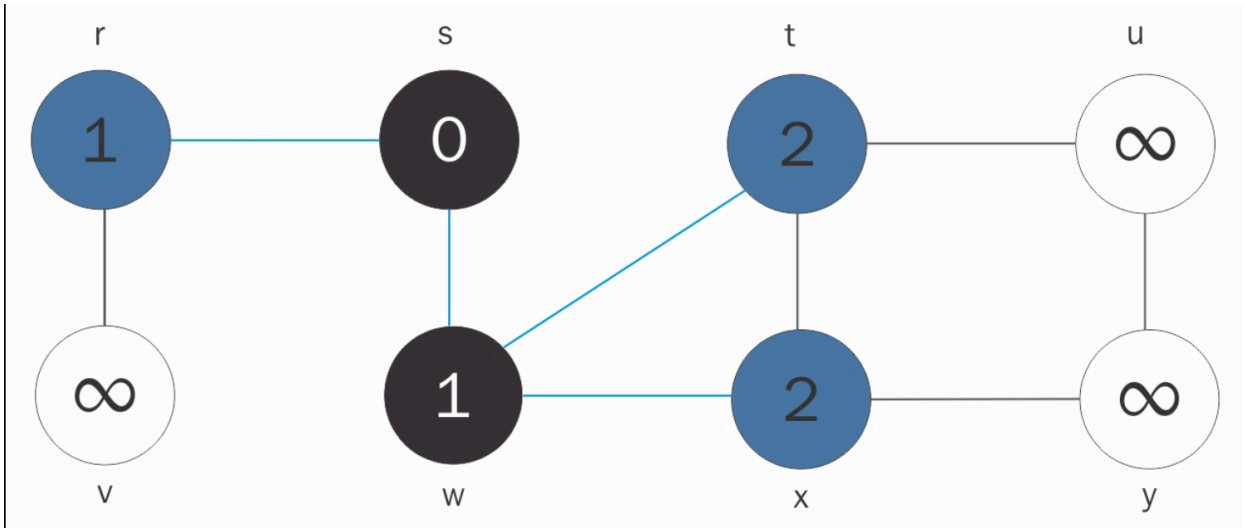
Dado un par de héroes de Marvel, hay que hallar los grados de separación entre ellos. Es decir, cuántos pasos hay que dar para llegar de uno al otro.

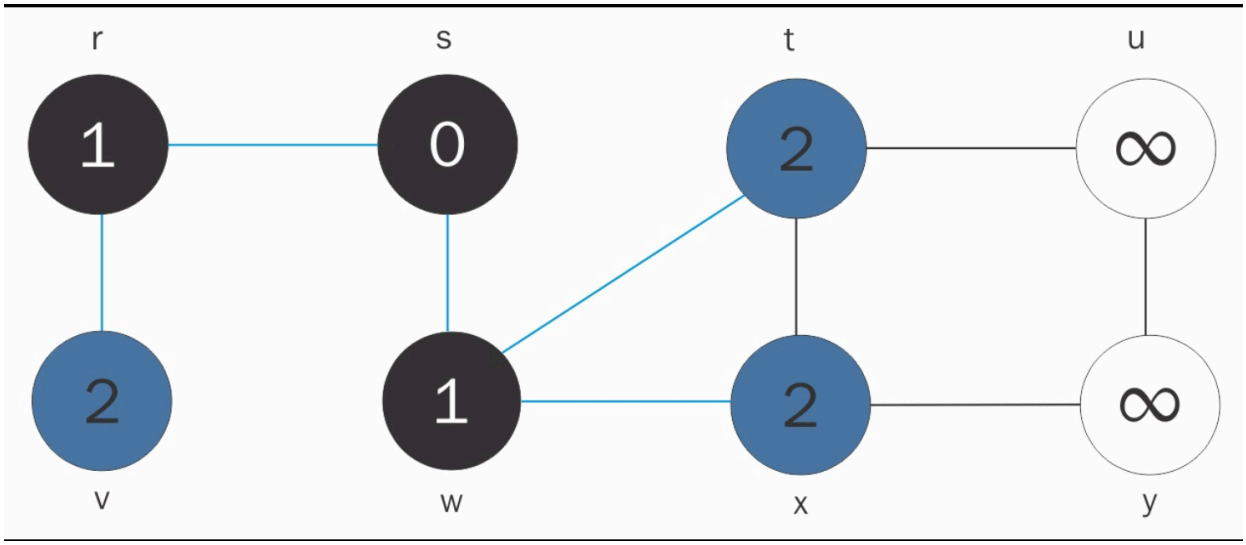
Se usa un algoritmo de breadth-first search.

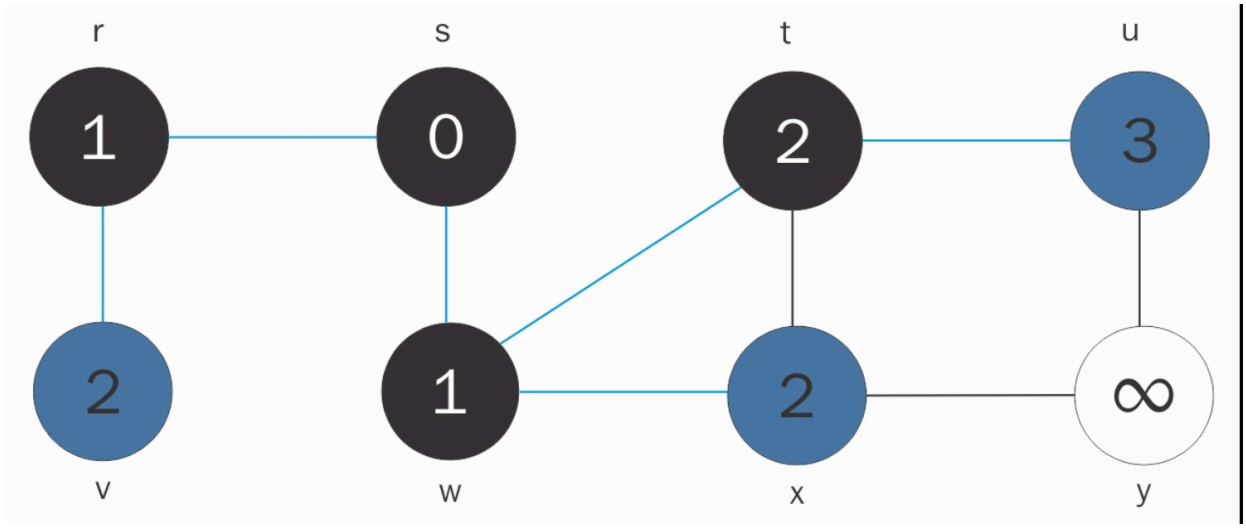


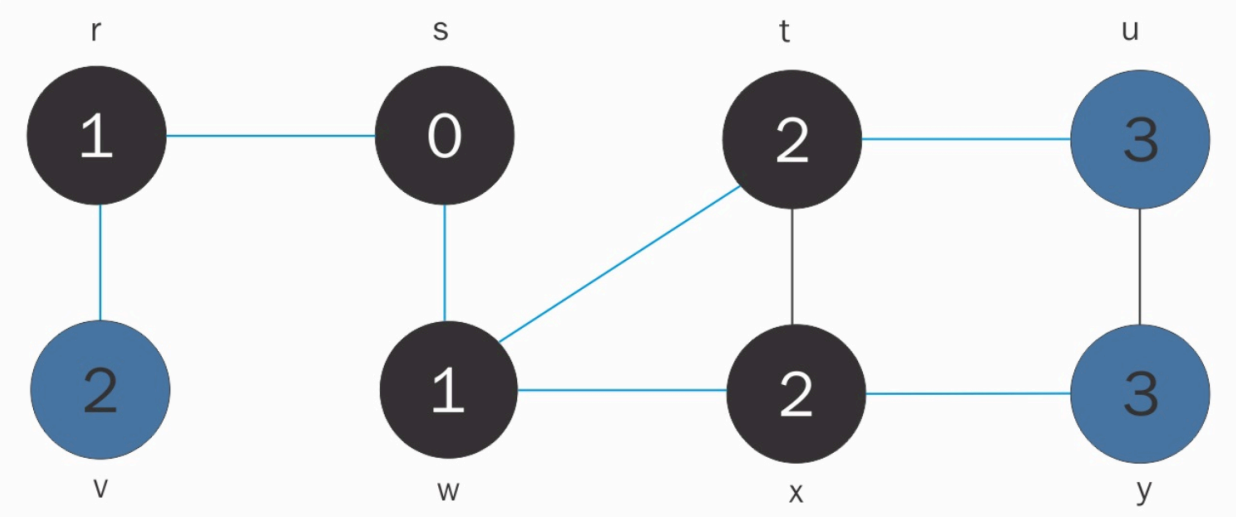


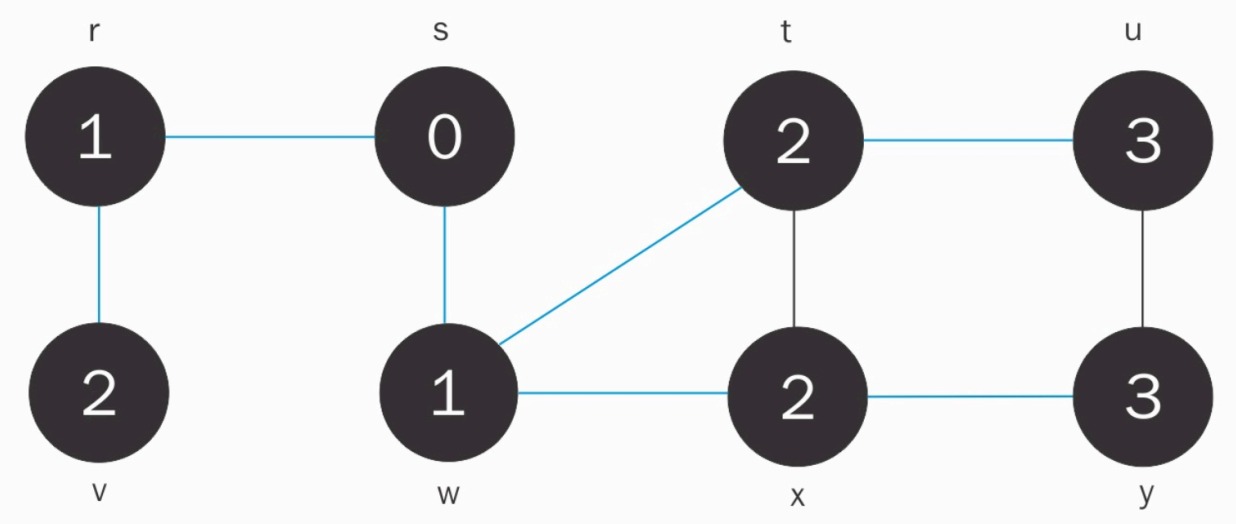



r
s
t
u
1
0
2
∞
∞
1
2
∞
v
w
x
y


r
s
t
u
1
0
2
∞
2
1
2
∞
v
w
x
y


r
s
t
u
1
0
2
3
2
1
2
∞
v
w
x
y





input (Marvel-graph.txt)

```
5988 748 1722 3752 4655 5743 1872 3413 5527 6368 6085 4319 4728 1636 ...
3878 2429 1334 4595 2767 3956 3877 4776 4946 3407 128 269 5775 5121 481
```

Es el mismo que el de *heroe-mas-popular* pero en este caso hay que pasarlo como argumento al programa.

El primer elemento es el héroe y el resto son los héroes que han aparecido con él en al menos una película.

Esquema de la implementación

Bucle en el que hacemos pasadas incrementando los valores de los nodos vecinos a los grises y pasando los grises a negros:

1. Pasar de 3878 2429 1334 4595 2767 3956 a *(3878, (2429, 1334, 4595, 2767, 3956), 9999, 'BLANCO')* en donde 9999 representa el infinito. Esto se hace con todos excepto con el origen que se pondría así: *(3878, (2429, 1334, 4595, 2767, 3956), 0, 'GRIS')*.
2. `flatMap` de todos los elementos de esta manera:

a. Los blancos y negros se quedan como están.

b. Los grises se mapean así: de *(8888, (2429, 1334, 4595, 2767, 3956), 4, 'GRIS')* a

```
(2429, ([]), 5, 'GRIS')
(1334, ([]), 5, 'GRIS')
(4595, ([]), 5, 'GRIS')
(2767, ([]), 5, 'GRIS')
(3956, ([]), 5, 'GRIS')
(8888, (2429, 1334, 4595, 2767, 3956), 4, 'NEGRO')
```

c. Si uno de los mapeados es el destino cambiamos el valor del acumulador a la distancia respectiva. Esto hará que el main termine.

3. Después de esto, se hace un `reduceByKey` para juntar las claves repetidas. Por ejemplo, el id *2429* que se crea nuevo se tiene que juntar con el *2429* original que estaba en blanco. A la hora de juntar, se suman los vecinos, se elige la distancia más pequeña, y se pone el color más oscuro.
4. Es posible que las entradas originales no fueran únicas. Por ejemplo puede haber dos entradas asociadas al 1334 y por lo tanto dos elementos de RDD con 1334 como clave. La primera vez que se hace un `reduceByKey` se juntan correctamente.
5. El proceso se termina cuando aparece el nodo final que se estaba buscando. Cuando el executor lo detecta (2. c.) pone el acumulador al valor de la distancia. El driver tiene que comprobar el acumulador cada vez que se ejecuta el bucle. Cuando lo detecta tiene que dar por terminado el programa.
6. Si no hay enlace entre el nodo de inicio y el nodo final, el programa se termina después de 10 bucles.

Explicar al final

código

grados-de-separacion/grados-de-separacion2a.py

En el anterior damos un origen y tenemos que calcular la distancia a otro id. En éste tenemos que calcular la distancia de todos los puntos. Es decir, damos un origen y calculamos la distancia de todos los puntos al origen.

Hay que hacer iteraciones hasta que no quede ningún vértice de color gris. Al final por cada distancia, hay que imprimir la cantidad de ids que están a esa distancia del origen.

Después de cada reduce, hay que usar la función `noHayGrises` que devuelve `True` si no queda más grises en el RDD.

entrada (Marvel-graph-corto.txt)

```
1 2 3
2 1 4
3 1 4 5
4 2 3
5 3 6
```

```
6 5
100 101
101 100
102 100
```

Salida

```
Al Comienzo de todo
(1, ([2, 3], 0, 'GRIS'))
(2, ([1, 4], 9999, 'BLANCO'))
(3, ([1, 4, 5], 9999, 'BLANCO'))
(4, ([2, 3], 9999, 'BLANCO'))
(5, ([3, 6], 9999, 'BLANCO'))
(6, ([5], 9999, 'BLANCO'))
(100, ([101], 9999, 'BLANCO'))
(101, ([100], 9999, 'BLANCO'))
(102, ([100], 9999, 'BLANCO'))

Después de flatMap
(1, ([2, 3], 0, 'NEGRO'))
(2, ([], 1, 'GRIS'))
(2, ([1, 4], 9999, 'BLANCO'))
(3, ([], 1, 'GRIS'))
(3, ([1, 4, 5], 9999, 'BLANCO'))
(4, ([2, 3], 9999, 'BLANCO'))
(5, ([3, 6], 9999, 'BLANCO'))
(6, ([5], 9999, 'BLANCO'))
(100, ([101], 9999, 'BLANCO'))
(101, ([100], 9999, 'BLANCO'))
(102, ([100], 9999, 'BLANCO'))

Después de reduceByKey
(1, ([2, 3], 0, 'NEGRO'))
(2, ([1, 4], 1, 'GRIS'))
(3, ([1, 4, 5], 1, 'GRIS'))
(4, ([2, 3], 9999, 'BLANCO'))
(5, ([3, 6], 9999, 'BLANCO'))
(6, ([5], 9999, 'BLANCO'))
(100, ([101], 9999, 'BLANCO'))
(101, ([100], 9999, 'BLANCO'))
(102, ([100], 9999, 'BLANCO'))

Después de flatMap
(1, ([], 2, 'GRIS'))
(1, ([2, 3], 0, 'NEGRO'))
(1, ([], 2, 'GRIS'))
(2, ([1, 4], 1, 'NEGRO'))
(3, ([1, 4, 5], 1, 'NEGRO'))
(4, ([], 2, 'GRIS'))
(4, ([2, 3], 9999, 'BLANCO'))
(4, ([], 2, 'GRIS'))
(5, ([], 2, 'GRIS'))
(5, ([3, 6], 9999, 'BLANCO'))
```

```
(6, ([5], 9999, 'BLANCO'))
(100, ([101], 9999, 'BLANCO'))
(101, ([100], 9999, 'BLANCO'))
(102, ([100], 9999, 'BLANCO'))

Después de reduceByKey
(1, ([2, 3], 0, 'NEGRO'))
(2, ([1, 4], 1, 'NEGRO'))
(3, ([1, 4, 5], 1, 'NEGRO'))
(4, ([2, 3], 2, 'GRIS'))
(5, ([3, 6], 2, 'GRIS'))
(6, ([5], 9999, 'BLANCO'))
(100, ([101], 9999, 'BLANCO'))
(101, ([100], 9999, 'BLANCO'))
(102, ([100], 9999, 'BLANCO'))

Después de flatMap
(1, ([2, 3], 0, 'NEGRO'))
(2, ([], 3, 'GRIS'))
(2, ([1, 4], 1, 'NEGRO'))
(3, ([], 3, 'GRIS'))
(3, ([1, 4, 5], 1, 'NEGRO'))
(3, ([], 3, 'GRIS'))
(4, ([2, 3], 2, 'NEGRO'))
(5, ([3, 6], 2, 'NEGRO'))
(6, ([5], 9999, 'BLANCO'))
(6, ([], 3, 'GRIS'))
(100, ([101], 9999, 'BLANCO'))
(101, ([100], 9999, 'BLANCO'))
(102, ([100], 9999, 'BLANCO'))

Después de reduceByKey
(1, ([2, 3], 0, 'NEGRO'))
(2, ([1, 4], 1, 'NEGRO'))
(3, ([1, 4, 5], 1, 'NEGRO'))
(4, ([2, 3], 2, 'NEGRO'))
(5, ([3, 6], 2, 'NEGRO'))
(6, ([5], 3, 'GRIS'))
(100, ([101], 9999, 'BLANCO'))
(101, ([100], 9999, 'BLANCO'))
(102, ([100], 9999, 'BLANCO'))

Después de flatMap
(1, ([2, 3], 0, 'NEGRO'))
(2, ([1, 4], 1, 'NEGRO'))
(3, ([1, 4, 5], 1, 'NEGRO'))
(4, ([2, 3], 2, 'NEGRO'))
(5, ([], 4, 'GRIS'))
(5, ([3, 6], 2, 'NEGRO'))
(6, ([5], 3, 'NEGRO'))
(100, ([101], 9999, 'BLANCO'))
(101, ([100], 9999, 'BLANCO'))
```

```
(102, ([100], 9999, 'BLANCO'))

Después de reduceByKey
(1, ([2, 3], 0, 'NEGRO'))
(2, ([1, 4], 1, 'NEGRO'))
(3, ([1, 4, 5], 1, 'NEGRO'))
(4, ([2, 3], 2, 'NEGRO'))
(5, ([3, 6], 2, 'NEGRO'))
(6, ([5], 3, 'NEGRO'))
(100, ([101], 9999, 'BLANCO'))
(101, ([100], 9999, 'BLANCO'))
(102, ([100], 9999, 'BLANCO'))

0: 1
1: 2
2: 2
3: 1
9999: 3
```

código

grados-de-separacion/grados-de-separacion2b.py

Lo mismo que el 2a excepto que ahora para detectar el final del ciclo no usamos la función `noHayGrises`. Tenemos un acumulador, `numGrises`, que incializamos a cero. Luego cuando la función `bfsMap` crea un nuevo nodo de color gris incrementa el acumulador.

Después de hacer el map/reduce en el main comprobamos el valor del acumulador. Si es mayor que 0, lo ponemos a 0 y seguimos en el bucle. Si es 0 el bucle ha terminado.

En el main, después de hacer el reduce, ejecutamos `take(1)` para provcar una acción. Si no lo hacemos, no se ejecuta el map ni el reduce ya que Spark trabaja en modo lazy. Y entonces el programa se termina.

código

grados-de-separacion/grados-de-separacion3.py

Como grados-de-separación.py pero para calcular el final comprobramos si ya lo ha encontrado o si no hay más grises. Ambas cosas las hacemos con un acumulador.

código