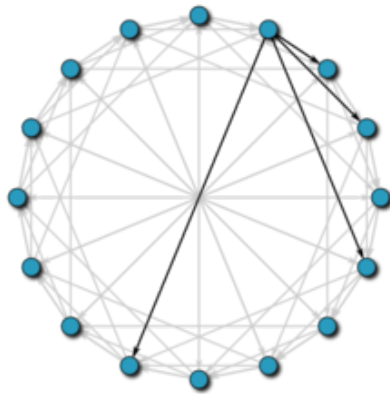


**IES Fernando Aguilar Quignon**

ADMINISTRACIÓN DE SISTEMAS OPERATIVOS

## P2P01. TABLAS DHT Y OVERLAYS



Pedro Peralta Guerrero  
José María Riol Sánchez  
José Manuel Rodríguez Guerrero  
Juan María Sánchez Ureba  
Carlos Montesino Fernando  
6 de febrero de 2023

# Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Acerca de Chord</b>	<b>2</b>
2.1. Funcionamiento general . . . . .	2
2.2. Procedimiento de búsqueda . . . . .	2
<b>3. Ejercicio P2P01.</b>	<b>3</b>
3.1. Montando la red overlay. . . . .	3
3.2. Forma de guardar la información. . . . .	3
3.3. Recursos del código . . . . .	4
3.4. Iniciar el programa. . . . .	4
3.4.1. Requerimientos del programa. . . . .	4
3.4.2. Cómo ejecutar el programa. . . . .	4
3.5. Código python realizado. . . . .	4
3.6. Demostración de uso del código. . . . .	6
3.7. Log de cabezazos. . . . .	10
<b>4. Alternativas para la resolución del ejercicio.</b>	<b>12</b>
4.1. Alternativa 1. . . . .	12
4.2. Alternativa 2. . . . .	15
<b>Referencias.</b>	<b>16</b>

## 1. Introducción

Una red superpuesta es una red virtual o lógica que se crea sobre una red física existente. Internet, que conecta muchos nodos a través de la conmutación de circuitos, es un ejemplo de una red superpuesta.

Una red superpuesta es cualquier capa virtual sobre la infraestructura de red física. Esto puede ser tan simple como una red de área local virtual (VLAN), pero generalmente se refiere a capas virtuales más complejas de redes definidas por software (SDN) o una red de área amplia definida por software (SD-WAN).

La superposición crea una nueva capa donde el tráfico se puede dirigir mediante programación a través de nuevas rutas de red virtual en lugar de requerir vínculos físicos. Las superposiciones permiten a los administradores definir y administrar flujos de tráfico, independientemente de la infraestructura física subyacente.

## 2. Acerca de Chord

Chord es un protocolo y algoritmo para una tabla hash distribuida peer-to-peer. Una tabla hash distribuida almacena pares clave-valor mediante la asignación de claves a diferentes equipos (conocidos como "nodos"); Un nodo almacenará los valores de todas las claves de las que es responsable. Chord especifica cómo se asignan las claves a los nodos y cómo un nodo puede descubrir el valor de una clave determinada localizando primero el nodo responsable de esa clave.

Un proyecto para poder ver el funcionamiento de chord a la hora de añadir nodos a la red lo podemos encontrar en GitHub con el nombre [chordial](#). Se trata de un proyecto realizado en java, vue, scala y otros lenguajes.

### 2.1. Funcionamiento general

Chord es uno de los cuatro protocolos de tabla hash distribuidos originales, junto con CAN, Tapestry y Pastry.

Se considera una red formada por  $2^m$  nodos, que pueden estar activos o ausentes de la red. Dado un conjunto de claves en esa red, el protocolo Chord se encarga de asignar las claves existentes a los nodos activos y mantener esas asignaciones dinámicamente, es decir, a medida que los nodos van entrando y saliendo de la red. El resto de tareas vinculadas a la red -autenticación, almacenamiento de datos, interfaz, etc.- son responsabilidad de los niveles superiores de la arquitectura.

La asignación de identificadores a nodos y claves se realiza mediante una función de hash consistente.

Una clave de id  $k$  se asigna al nodo de id  $k$  si éste está activo en la red. Si  $k$  no está, se busca el primer nodo posterior a  $k$  que esté activo y se le asigna a la clave: este nodo sustitutivo se denomina sucesor de  $k$  ( $\text{successor}(k)$ ).

Cuando  $k$  se conecte a la red, su nodo sucesor le transferirá las claves que estuvieran destinadas a él. Cuando un nodo abandona la red, transfiere las claves de las que se hacía cargo a su sucesor, es decir, al nodo con id siguiente a la suya. Con estos mecanismos, se garantiza la autorregulación de la red y el mantenimiento de las claves aun en un contexto de movilidad de los nodos participantes.

### 2.2. Procedimiento de búsqueda

La forma en la que se mantienen los nodos sucesores garantiza que las búsquedas se procesen de forma exhaustiva, además, cada nodo almacena información adicional sobre la red permitiendo acelerar las búsquedas.

Esta información adicional se reduce a unos pocos nodos activos de la red reflejándose en una tabla de rutas interna (finger table).

Cuando un nodo pide una clave  $j$ , primero mira su propia tabla de rutas; si encuentra el nodo responsable de  $j$ , envía la petición directamente al nodo afectado. Si no, pregunta al nodo con id más cercana a  $j$  y éste devolverá la id del nodo más cercano a  $j$  (menor) que encuentre en su tabla de rutas. Así, el nodo remitente obtiene en cada nueva iteración un nodo más cercano a  $k$ , hasta llegar a  $k$  o a su sucesor.

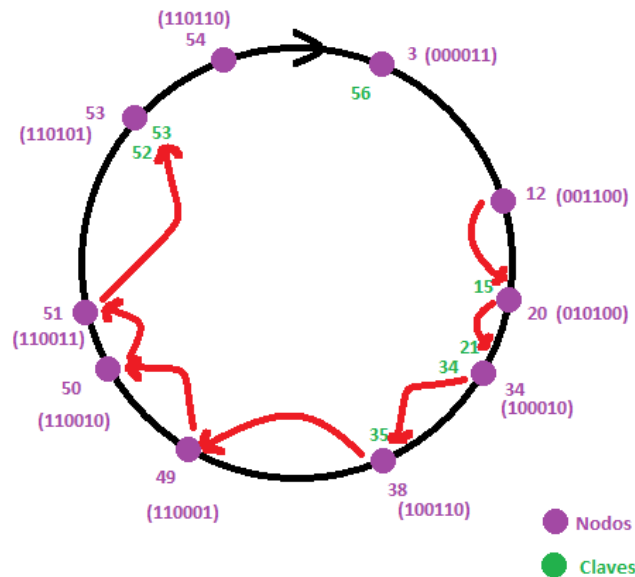


Figura 1: Búsqueda secuencial

### 3. Ejercicio P2P01.

#### 3.1. Montando la red overlay.

Una red overlay no es más que una red superpuesta. Esto quiere decir que partiendo de una topología física concreta, que pueden ser los clientes distribuidos en diferentes puntos de la red tenemos una topología lógica en anillo basándonos en el principio de que cada nodo solo puede ver directamente a otros dos nodos, que son sus vecinos, el antecesor y el sucesor.

Nuestra red overlay se componen de esos nodos que forman el anillo y son éstos nodos los que realmente mantienen nuestra red. La forma en que mantienen la red no es más que informando al tracker de que quieren unirse o desertar e informando a los nodos vecinos (en caso de desertar) para que actualicen sus respectivos antecesores y sucesores. Así los nodos van actualizando la información del tracker que será necesario para mantener parte de la información de la red, pero no es el encargado de mantener la red, como comentamos antes, la red se mantiene gracias a los propios nodos, agilizando la red.

#### 3.2. Forma de guardar la información.

Para este tipo de red (overlay) tenemos una forma de guardar la información adaptado a las necesidades que tenemos y es usar Redis. De esta forma vamos a tratar la información como diccionarios de clave valor, información más que suficiente con la que mantener la red y poder encontrar la información que queremos, unirnos a la red e incluso desertar.

### 3.3. Recursos del código

Para realizar este ejercicio hemos optado por usar el [12]módulo **py2p** y crear un pequeño programa en python3 a partir de este.

En este programa realizamos una conexión a redis el cual es el servidor que almacena los recursos y los nodos que se encuentran compartiendolo.

Además en la clase **Node** hemos creado los metodos para unirnos a la red, actualizar los datos del nodo(sucesor,predecesor), añadir datos, eliminarlos y buscarlos.

Podremos añadir datos ya que cada nodo no almacenará el recurso en su totalidad sino que almacenará partes de este.

### 3.4. Iniciar el programa.

#### 3.4.1. Requerimientos del programa.

Para el correcto funcionamiento del programa hay que cumplir una serie de requisitos:

- Tener instalado Redis, la base de datos clave-valor.
- Tener instalado los módulos de python correspondientes para usar el programa.

Los módulos podremos instalarlos ejecutando el siguiente comando<sup>1</sup>:

```
pip3 install -r requirements.txt
```

#### 3.4.2. Cómo ejecutar el programa.

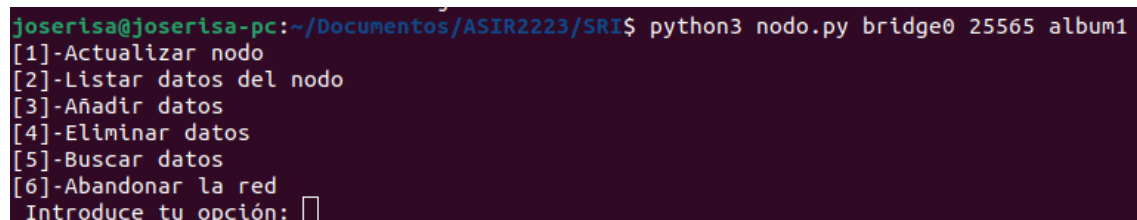
Para iniciar el programa hay que seguir la siguiente forma de ejecutarlo:

```
user@user-pc:~$ python3 nodo.py nombre_interfaz puerto recurso
```

Siendo:

- **nombre\_interfaz** la interfaz a usar para la conexión siendo nuestro caso bridge0 (usar lo en caso de querer realizarlo en local).
- **puerto** un puerto a elegir de los comunes.
- **recurso** el recurso al que queremos conectarnos y buscar los pares clave valor.

Añadir que para ejecutar el código se debe cambiar antes la dirección del servidor de redis.



```
joserisa@joserisa-pc:~/Documentos/ASIR2223/SRI$ python3 nodo.py bridge0 25565 album1
[1]-Actualizar nodo
[2]-Listar datos del nodo
[3]-Añadir datos
[4]-Eliminar datos
[5]-Buscar datos
[6]-Abandonar la red
Introduce tu opción: █
```

Figura 2: Muestra de la ejecución del programa.

### 3.5. Código python realizado.

---

<sup>1</sup>El archivo requirements.txt podemos encontrarlo en el [GitHub del ejercicio](#)

```

1
2 import redis
3 import py2p
4 import sys,os
5 import netifaces as ni
6
7 #OBTENER LA DIRECCION IP DE LA INTERFAZ INDICADA EN EL PRIMER ARGUMENTO
8 ip = ni.ifaddresses(sys.argv[1])[ni.AF_INET][0]['addr']
9
10 ip_address = socket.gethostbyname(socket.gethostname())
11 #CONEXION A REDIS (TRACKER)
12 r = redis.Redis(host="10.1.2.203", port=6380)
13
14
15 #CONSTRUCTOR
16 class Node():
17
18     def __init__(self,ip,port,recurso):
19
20         self.socket = f"{ip}:{port}"
21         self.recurso = recurso
22         self.data = {}
23         self.sock = None
24         self.predecessor = None
25         self.successor = None
26
27     def join(self):
28
29         self.sock = py2p.ChordSocket(self.socket.split(':')[0],int(self.socket.
split(':')[1]))
30         self.sock.join()
31
32         lista = self.get_tracker_list()
33         if lista:
34             ip = [r.hget(self.recurso,x.decode("utf8")).decode("utf8") for x in
lista][0]
35             self.sock.connect(ip.split(':')[0],int(ip.split(':')[1]))
36             r.hset(self.recurso,self.sock.id_10,self.socket)
37
38         else:
39             r.hset(self.recurso,self.sock.id_10,self.socket)
40
41     def leave(self):
42
43         r.hdel(self.recurso,self.sock.id_10)
44         self.sock.unjoin()
45         return "Bye"
46
47     def get_tracker_list(self):
48         tracker_list = list(r.hgetall(self.recurso))
49         return tracker_list
50
51
52     def update(self):
53         if self.sock:
54             if self.sock.routing_table == {}:
55                 nodos = self.get_tracker_list()
56                 nodos_ip = [r.hget(self.recurso,x.decode("utf8")).decode("utf8")
for x in nodos]
57                 for nid in nodos_ip:
58                     self.sock.connect(nid.split(':')[0],int(nid.split(':')[1]))
59                     self.successor = self.sock.next.id
60                     self.predecessor = self.sock.prev.id
61                 return f"ID: {self.sock.id}\nSucesor: {self.sock.prev.id}\nPredecesor: {
self.sock.next.id}"
62
63 def main():
64     nodo = Node(ip,int(sys.argv[1]),sys.argv[2])
65     nodo.join()
66     option = input("[1]-Actualizar nodo\n[2]-Listar datos del nodo\n[3]-Anadir
datos\n[4]-Eliminar datos\n[5]-Buscar datos\n[6]-Abandonar la red\n Introduce
tu opcion: ")
67     while True:

```

```

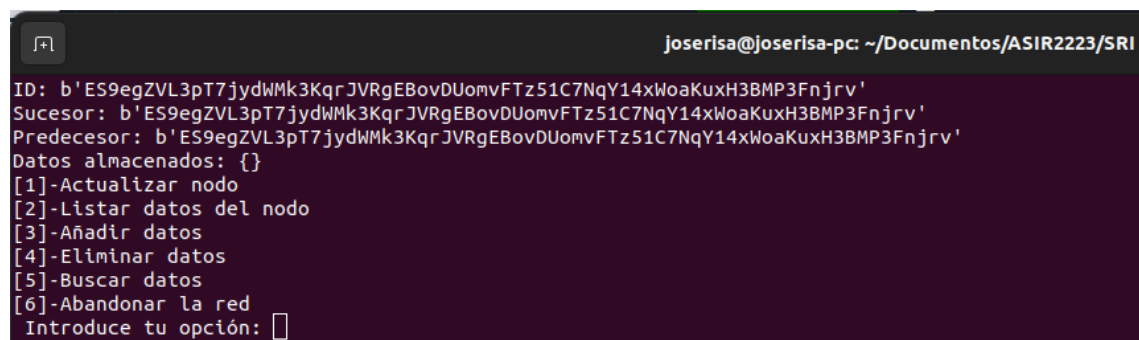
68     match option:
69         case "1":
70             os.system("clear")
71             nodo.update()
72             print(f"ID: {nodo.sock.id}\nSucesor: {nodo.sock.prev.id}\nPredecesor: {nodo.sock.next.id}\nDatos almacenados: {nodo.data}")
73         case "2":
74             os.system("clear")
75             print(f"Datos del nodo {nodo.data}")
76         case "3":
77             os.system("clear")
78             clave = input("Introduce la clave: ")
79             valor = input("Introduce el valor: ")
80             nodo.data[clave]=valor
81             nodo.sock[clave]=valor
82         case "4":
83             os.system("clear")
84             clave = input("Introduce la clave: ")
85             del nodo.data[clave]
86             del nodo.sock[clave]
87         case "5":
88             os.system("clear")
89             clave = input("Introduce la clave: ")
90             if clave in nodo.data:
91                 print(f"El valor de la clave {clave} es {nodo.data[clave]}\n")
92             else:
93                 try:
94                     print(f"El valor de la clave {clave} es {nodo.sock[clave]}\n")
95                 except:
96                     print("ERROR: No existe valor para la clave indicada.")
97         case "6":
98             nodo.leave()
99             return "Abandonando la red..."
100        case _:
101            print("ERROR: El valor introducido no es valido.")
102
103    option = input("[1]-Actualizar nodo\n[2]-Listar datos del nodo\n[3]-Anadir datos\n[4]-Eliminar datos\n[5]-Buscar datos\n[6]-Abandonar la red\n Introduce tu opcion: ")
104 if __name__ == '__main__':
105     main()

```

### 3.6. Demostración de uso del código.

#### Actualización e inserción de nodos.

Si seleccionamos actualizar nodo podremos ver arriba del todo la ID de nuestro nodo y cual es su antecesor y su sucesor.



```

joserisa@joserisa-pc: ~/Documentos/ASIR2223/SRI
ID: b'ES9egZVL3pT7jydWMk3KqrJVRgEBovDUomvFTz51C7NqY14xWoaKuxH3BMP3Fnjrv'
Sucesor: b'ES9egZVL3pT7jydWMk3KqrJVRgEBovDUomvFTz51C7NqY14xWoaKuxH3BMP3Fnjrv'
Predecesor: b'ES9egZVL3pT7jydWMk3KqrJVRgEBovDUomvFTz51C7NqY14xWoaKuxH3BMP3Fnjrv'
Datos almacenados: {}
[1]-Actualizar nodo
[2]-Listar datos del nodo
[3]-Añadir datos
[4]-Eliminar datos
[5]-Buscar datos
[6]-Abandonar la red
Introduce tu opción: 

```

Figura 3: Actualización del nodo.

En el momento que entre otro nodo, si volvemos a actualizar, al haber 2 nodos el sucesor y predecesor serán el mismo

```
joserisa@joserisa-pc: ~/Documentos/ASIR2223/SRI
ID: b'ES9egZVL3pT7jydWMk3KqrJVRgEBovDUomvFTz51C7NqY14xWoaKuxH3BMP3Fnjrv'
Sucesor: b'9iQdjtA7hjojggqJPFB9wuF4ZGujcqfHbminjPpGxpjSXQQynimdyUMa8Et1WZ1Jei'
Predecesor: b'9iQdjtA7hjojggqJPFB9wuF4ZGujcqfHbminjPpGxpjSXQQynimdyUMa8Et1WZ1Jei'
Datos almacenados: {}
[1]-Actualizar nodo
[2]-Listar datos del nodo
[3]-Añadir datos
[4]-Eliminar datos
[5]-Buscar datos
[6]-Abandonar la red
Introduce tu opción: 
```

Figura 4: Actualización del nodo.

Si tenemos un tercer nodo ya tendríamos un sucesor distinto al predecesor.

```
joserisa@joserisa-pc: ~/Documentos/ASIR2223/SRI
ID: b'ES9egZVL3pT7jydWMk3KqrJVRgEBovDUomvFTz51C7NqY14xWoaKuxH3BMP3Fnjrv'
Sucesor: b'smCn9wKu1EDf6Y7UGfcXaGxsovec7nUUneK7ab7dQUsdhV7f7tsXAVn4CiGJT6NrE'
Predecesor: b'9iQdjtA7hjojggqJPFB9wuF4ZGujcqfHbminjPpGxpjSXQQynimdyUMa8Et1WZ1Jei'
Datos almacenados: {}
[1]-Actualizar nodo
[2]-Listar datos del nodo
[3]-Añadir datos
[4]-Eliminar datos
[5]-Buscar datos
[6]-Abandonar la red
Introduce tu opción: 
```

Figura 5: Actualización del nodo.

### Búsqueda de información.

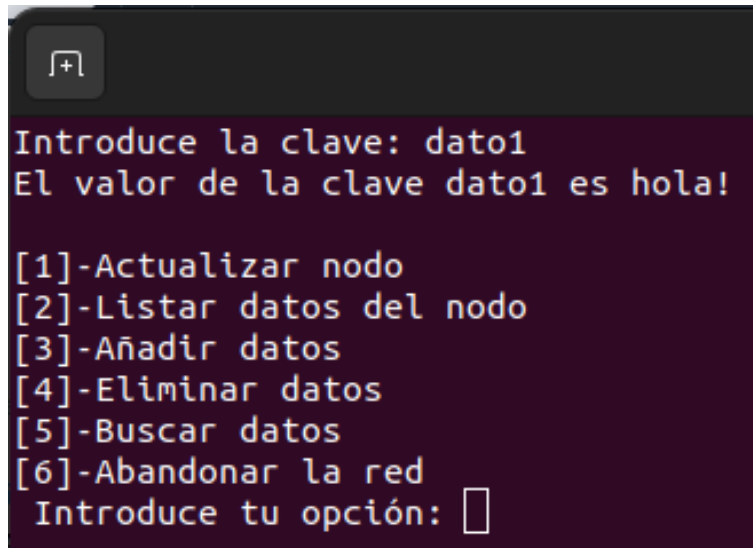
Un nodo procede a subir unos datos, y nosotros vamos a buscar esos datos.

```
/bin/bash 104x26
ID: b'9iQdjtA7hjojggqJPFB9wuF4ZGujcqfHbminjPpGxpjSXQQynimdyUMa8Et1WZ1Jei'
Sucesor: b'ES9egZVL3pT7jydWMk3KqrJVRgEBovDUomvFTz51C7NqY14xWoaKuxH3BMP3Fnjrv'
Predecesor: b'smCn9wKu1EDf6Y7UGfcXaGxsovec7nUUneK7ab7dQUsdhV7f7tsXAVn4CiGJT6NrE'
Datos almacenados: {'dato1': 'hola!'}
[1]-Actualizar nodo
[2]-Listar datos del nodo
[3]-Añadir datos
[4]-Eliminar datos
[5]-Buscar datos
[6]-Abandonar la red
Introduce tu opción: 
```

Figura 6: Datos añadidos.

Una vez le damos a la opción 3 y añadimos una clave y un valor y actualizamos, podemos ver que tenemos el diccionario actualizado con los datos insertados. Si ahora nosotros queremos buscar ese dato tan solo que tenemos usar la opción 5 y buscar por la clave.



A terminal window with a dark background and a light-colored border. It contains text in a monospaced font. The text shows a search operation where a key 'dato1' is entered, and the value 'hola!' is returned. Below this, a menu of options is displayed, and the prompt 'Introduce tu opción:' is followed by a cursor.

```
Introduce la clave: dato1
El valor de la clave dato1 es hola!

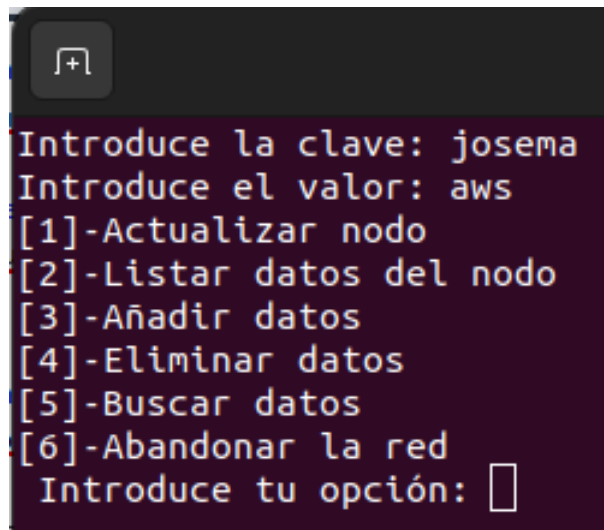
[1]-Actualizar nodo
[2]-Listar datos del nodo
[3]-Añadir datos
[4]-Eliminar datos
[5]-Buscar datos
[6]-Abandonar la red
Introduce tu opción: 
```

Figura 7: Dato buscado.

Nuestro nodo ha logrado obtener el valor de la clave insertada, preguntando a su sucesor, y este a su sucesor y así hasta encontrar el nodo que lo contiene y éste se lo devuelve a nuestro nodo directamente.

#### Añadir y borrar datos.

Añadir y borrar datos de nuestro diccionario es tan simple como entrar en las opciones 3 y 4 e insertar, en el caso de añadir datos la clave y el valor y en el caso de borrar tan solo escribimos la clave.

A terminal window with a dark background and a light-colored border. It contains text in a monospaced font. The text shows the process of adding a new key-value pair: 'josema' as the key and 'aws' as the value. Below this, the same menu of options is displayed, and the prompt 'Introduce tu opción:' is followed by a cursor.

```
Introduce la clave: josema
Introduce el valor: aws
[1]-Actualizar nodo
[2]-Listar datos del nodo
[3]-Añadir datos
[4]-Eliminar datos
[5]-Buscar datos
[6]-Abandonar la red
Introduce tu opción: 
```

Figura 8: Añadir datos clave josema valor aws.

```
Joserisa@Joserisa-pc: ~/Documentos
ID: b'ES9egZVL3pT7jydWmk3KqrJVRgEBovDUomvFTz51C7NqY14xWoaKuxH3BMP3Fnjrv'
Sucesor: b'ES9egZVL3pT7jydWmk3KqrJVRgEBovDUomvFTz51C7NqY14xWoaKuxH3BMP3Fnjrv'
Predecesor: b'ES9egZVL3pT7jydWmk3KqrJVRgEBovDUomvFTz51C7NqY14xWoaKuxH3BMP3Fnjrv'
Datos almacenados: {'josema': 'aws'}
[1]-Actualizar nodo
[2]-Listar datos del nodo
[3]-Añadir datos
[4]-Eliminar datos
[5]-Buscar datos
[6]-Abandonar la red
Introduce tu opción: 
```

Figura 9: Nodo actualizado y vista de los datos añadidos en la figura anterior.

```
Introduce la clave: josema
[1]-Actualizar nodo
[2]-Listar datos del nodo
[3]-Añadir datos
[4]-Eliminar datos
[5]-Buscar datos
[6]-Abandonar la red
Introduce tu opción: 
```

Figura 10: Borrado de los datos con clave josema.

```
Joserisa@Joserisa-pc: ~/Documentos/ASIR2223/SRI
ID: b'ES9egZVL3pT7jydWmk3KqrJVRgEBovDUomvFTz51C7NqY14xWoaKuxH3BMP3Fnjrv'
Sucesor: b'ES9egZVL3pT7jydWmk3KqrJVRgEBovDUomvFTz51C7NqY14xWoaKuxH3BMP3Fnjrv'
Predecesor: b'ES9egZVL3pT7jydWmk3KqrJVRgEBovDUomvFTz51C7NqY14xWoaKuxH3BMP3Fnjrv'
Datos almacenados: {}
[1]-Actualizar nodo
[2]-Listar datos del nodo
[3]-Añadir datos
[4]-Eliminar datos
[5]-Buscar datos
[6]-Abandonar la red
Introduce tu opción: 
```

Figura 11: Nodo actualizado y vista de los datos borrados en la figura anterior.

### Listado de los datos.

Si tenemos varios datos añadidos y quisiéramos verlos, podemos hacer uso de la opción 2, la cual nos mostrará arriba el diccionario de datos que tenga el nodo en el instante de introducir la opción.

```
joserisa@
Datos del nodo {'josema': 'aws', 'queen': 'Freddie Mercury'}
[1]-Actualizar nodo
[2]-Listar datos del nodo
[3]-Añadir datos
[4]-Eliminar datos
[5]-Buscar datos
[6]-Abandonar la red
Introduce tu opción: 
```

Figura 12: Listado de datos del nodo.

### Deserción del nodo.

En algún momento el nodo querrá desertar de la red, en ese caso el nodo informará de ello y abandonará la red, haciendo que sus vecinos sean entre sí los nuevos vecinos, actualizando la información de la red y del tracker.

```
joserisa@
Datos del nodo {'josema': 'aws', 'queen': 'Freddie Mercury'}
[1]-Actualizar nodo
[2]-Listar datos del nodo
[3]-Añadir datos
[4]-Eliminar datos
[5]-Buscar datos
[6]-Abandonar la red
Introduce tu opción: 6
joserisa@joserisa-pc:~/Documentos/ASIR2223/SRI$ 
```

Figura 13: Deserción.

### 3.7. Log de cabezazos.

El primer error que nos encontramos es cuando ejecutabamos el programa y creabamos la red con varios nodos, si intentabamos buscar una clave la cual no existe.

```
Introduce la clave: hola
Traceback (most recent call last):
  File "/home/pedro/Clases/Modulos/SRI/Tablas-DHT/TAREA_OF/node.py", line 103, in <module>
    main()
  File "/home/pedro/Clases/Modulos/SRI/Tablas-DHT/TAREA_OF/node.py", line 93, in main
    print(f"El valor de la clave {clave} es {nodo.sock[clave]}\n")
  File "/home/pedro/.local/lib/python3.10/site-packages/py2p/chord.py", line 568, in __getitem__
    return self.__getitem(key)
  File "/home/pedro/.local/lib/python3.10/site-packages/py2p/chord.py", line 539, in __getitem
    raise TimeoutException()
TimeoutError
```

Figura 14: Error.

Este error se produce por la siguiente linea en el código:

```
    print( "ERROR: Esa clave no se encuentra almacenada." )
case "5":
    os.system("clear")
    clave = input("Introduce la clave: ")
    if clave in nodo.data:
        print(f"El valor de la clave {clave} es {nodo.data[clave]}\n")
    else:
        print(f"El valor de la clave {clave} es {nodo.sock[clave]}\n")
```

Figura 15: Linea.

Para solucionar esto, he añadido una excepción:

```
case "5":
    os.system("clear")
    clave = input("Introduce la clave: ")
    if clave in nodo.data:
        print(f"El valor de la clave {clave} es {nodo.data[clave]}\n")
    else:
        try:
            print(f"El valor de la clave {clave} es {nodo.sock[clave]}\n")
        except:
            print("ERROR: No existe valor para la clave indicada.")
```

Figura 16: Solución al error.

Gracias a esto, si introducimos una clave que no existe, nos devolverá un comentario indicando el error pero no parará la ejecución del programa.

Otro error con el que nos encontramos es la conexión a redis. Cuando intentamos conectar a redis desde otro dispositivo de la red, la respuesta del servidor redis es la siguiente:

(error) *DENIED* Redis is running in protected mode because protected mode is enabled, no bind address was specified, no authentication password is requested to clients. In this mode connections are only accepted from the loopback interface. If you want to connect from external computers to Redis you may adopt one of the following solutions: 1) Just disable protected mode sending the command 'CONFIG SET protected-mode no' from the loopback interface by connecting to Redis from the same host the server is running, however MAKE SURE Redis is not publicly accessible from internet if you do so. Use CONFIG REWRITE to make this change permanent. 2) Alternatively you can just disable the protected mode by editing the Redis configuration file, and setting the protected mode option to 'no', and then restarting the server. 3) If you started the server manually just for testing, restart it with the '--protected-mode no' option. 4) Setup a bind address or an authentication password. NOTE: You only need to do one of the above things in order for the server to start accepting connections from the outside.

Redis nos devuelve varias soluciones a este error, optamos por usar la opción **protected-mode no**, es decir, usamos la siguiente instrucción:

```
pedro@MSI-BRAVO-07:/C/Classes/Modulos/SRI/Tareas/BHT/TAREA_0F]: redis-server --port 6380 --protected-mode no  
7924:C 06 Feb 2023 22:38:11.286 # oOoOoOoOoOoO RedIs Is starting oOoOoOoOoOoOoO  
7924:C 06 Feb 2023 22:38:11.286 # Redis version=6.0.16, bits=64, commit=00000000, modified=0, pid=7924, just started  
7924:C 06 Feb 2023 22:38:11.286 # Configuration loaded  
7924:M 06 Feb 2023 22:38:11.287 * Increased maximum number of open files to 10032 (it was originally set to 1024).  
  
Redis 6.0.16 (00000000/0) 64 bit  
  
Running in standalone mode  
Port: 6380  
PID: 7924  
  
http://redis.io
```

Figura 17: Solución al error de redis.

El último error con el que nos hemos encontrado ha sido cuando un nodo abandona la red ya que la tabla de enrutamiento del resto de nodos se ve afectada y no es capaz de actualizarse.

```

/bin/bash 138x53
ID: b'77APEHw4DXdFhvtSG3FuJdDyq1UFRG8cH41QeAgoLy2DNJm7A6lHzge1XFdDvpoAJ'
Sucesor: b'8KCNTolizWUqq2eD3X5S5YsqCMDqA2SmwmyVtzzyvqbTQY1uqnp9TSMn4C75FHsxHjRb'
Predecesor: b'3DwFDY3Nf98cfbGRkExLSpGsdFkkyFSNksm9afZLhbktczxRufSnnojCybNKtgrCj1'
Datos almacenados: {}
[1]-Actualizar nodo
[2]-Listar datos del nodo
[3]-Añadir datos
[4]-Eliminar datos
[5]-Buscar datos
[6]-Abandonar la red
Introduce tu opción: 25565 There was an unhandled exception with peer id b'3DwFDY3Nf98cfbGRkExLSpGsdFkkyFSNksm9afZLhbktczxRufSnnojCybNKtgrCj1'. This peer is being disconnected, and the relevant exception is added to the debug queue. If you'd like to report this, please post a copy of your MeshSocket.status to git.p2p.today/issues.

```

Figura 18: Error al abandonar la red.

Como solución, hemos decido que si la tabla de enrutamiento desaparece el nodo vuelva a preguntar a redis(tracker) quien esta en la red para volver a conectarse entre ellos.

```

def update(self):
    if self.sock:
        if self.sock.routing_table == {}:
            nodos = self.get_tracker_list()
            nodos_ip = [r.hget(self.recurso,x.decode("utf8")).decode("utf8") for x in nodos]
            for nid in nodos_ip:
                self.sock.connect(nid.split(':')[0],int(nid.split(':')[1]))
            self.successor = self.sock.next.id
            self.predecessor = self.sock.prev.id
            return f"ID: {self.sock.id}\nSucesor: {self.sock.prev.id}\nPredecesor: {self.sock.next.id}"

```

Figura 19: Solución.

## 4. Alternativas para la resolución del ejercicio.

### 4.1. Alternativa 1.

La primera alternativa puede ser hacer uso del siguiente proyecto el cual se encuentra en [GitHub](#).

Este proyecto es muy parecido a nuestra resolución, la diferencia es que este utiliza directamente el [módulo sockets](#) además de otros como [threading](#), [hashlib](#) implementación del protocolo sin utilizar otros módulos que lo faciliten como [py2p](#).

Para iniciar el anillo usaremos:

```
user@user-pc:~$ python3 Node_DHT.py puerto
```

```

pedro@MSI-BRAVO: [~/Clases/Modulos/SRI/Tablas-DHT/chord_protocol]: python3 Node_DHT.py 25565
CREATING RING
=====
STABILIZING
=====
ID: 2
Successor ID: 2
predecessor ID: 2
=====
===== FINGER TABLE =====
Entry: 0 Interval start: 3 Successor: 2
Entry: 1 Interval start: 4 Successor: None
Entry: 2 Interval start: 6 Successor: None
Entry: 3 Interval start: 10 Successor: 2
Entry: 4 Interval start: 18 Successor: None
Entry: 5 Interval start: 34 Successor: None
Entry: 6 Interval start: 66 Successor: 2
=====
DATA STORE
=====
{}
=====
+++++ END +++++

```

Figura 20: Creación el anillo.

Si queremos añadir más nodos al anillo debemos de llamar al programa de la siguiente manera:

```
user@user-pc:~$ python3 Node_DHT.py puerto puerto
```

Donde:

- **puerto** Nuevo puerto para el nodo que estamos creando
- **puerto** Puerto de un nodo existente en el anillo.

```
pedro@MSI-BRAVO:~/Clases/Modulos/SRI/Tablas-DHT/chord_protocol]: python3 Node_DHT.py 25566 25565
JOINING RING
=====
STABILIZING
=====
ID: 105
Successor ID: 2
=====
===== FINGER TABLE =====
Entry: 0 Interval start: 106 Successor: 2
Entry: 1 Interval start: 107 Successor: None
Entry: 2 Interval start: 109 Successor: 2
Entry: 3 Interval start: 113 Successor: None
Entry: 4 Interval start: 121 Successor: None
Entry: 5 Interval start: 9 Successor: None
Entry: 6 Interval start: 41 Successor: None
=====
DATA STORE
=====
{}
=====
+++++ END +++++
```

Figura 21: Añadir nodos al anillo.

Vemos que la id del nodo que creó el anillo es 2 y la del segundo nodo es la 105, también podemos ver que el sucesor del nodo 105 es el nodo 2.

Automáticamente, cada x segundos, se actualiza el sucesor y predecesor de cada nodo además de los datos almacenados:

```
=====
STABILIZING
=====
ID: 2
Successor ID: 105
predecessor ID: 105
=====
===== FINGER TABLE =====
Entry: 0 Interval start: 3 Successor: 105
Entry: 1 Interval start: 4 Successor: 105
Entry: 2 Interval start: 6 Successor: 105
Entry: 3 Interval start: 10 Successor: 105
Entry: 4 Interval start: 18 Successor: 2
Entry: 5 Interval start: 34 Successor: 105
Entry: 6 Interval start: 66 Successor: 105
=====
DATA STORE
=====
{}
=====
+++++ END +++++
```

Figura 22: Nodo 2

```

=====
STABILIZING
=====
ID: 105
Successor ID: 2
predecessor ID: 2
=====
===== FINGER TABLE =====
Entry: 0 Interval start: 106 Successor: 2
Entry: 1 Interval start: 107 Successor: 2
Entry: 2 Interval start: 109 Successor: 2
Entry: 3 Interval start: 113 Successor: 2
Entry: 4 Interval start: 121 Successor: 2
Entry: 5 Interval start: 9 Successor: 105
Entry: 6 Interval start: 41 Successor: 105
=====
DATA STORE
=====
{}
=====
+++++ END +++++

```

Figura 23: Nodo 105.

Para poder almacenar, consultar o eliminar datos nos ofrece un cliente con el cual podemos conectar a cualquier nodo de la red.

Para ejecutar este programa simplemente usamos la siguiente instrucción:

```
user@user-pc:~$ python3 Client.py
```

Nos pedirá el puerto del nodo con el que queremos conectar y seguidamente nos ofrece una interfaz por consola para realizar las diferentes operaciones.

```

pedro@MSI-BRAVO:~/Clases/Modulos/SRI/Tablas-DHT/chord_protocol]: python3 Client.py
Give the port number of a node25565
*****MENU*****
PRESS *****
1. TO ENTER *****
2. TO SHOW *****
3. TO DELTE *****
4. TO EXIT *****
*****

```

Figura 24: Conexión del cliente al nodo 2.

Si insertamos un valor:

```

pedro@MSI-BRAVO:~/Clases/Modulos/SRI/Tablas-DHT/chord_protocol]: python3 Client.py
Give the port number of a node25565
*****MENU*****
PRESS *****
1. TO ENTER *****
2. TO SHOW *****
3. TO DELTE *****
4. TO EXIT *****
*****
1
ENTER THE KEY : clave1
ENTER THE VALUE : valor1
Inserted at node id 2 key was clave1 key hash was 106
*****MENU*****
PRESS *****
1. TO ENTER *****
2. TO SHOW *****
3. TO DELTE *****
4. TO EXIT *****
*****

```

Figura 25: Conexión del cliente al nodo 2.

Ahora consultamos el nodo, podremos ver que ha aparecido la clave y el valor en el apartado **DATA STORAGE**.

```

=====
STABILIZING
=====
ID: 2
Successor ID: 105
predecessor ID: 105
=====
----- FINGER TABLE -----
Entry: 0 Interval start: 3 Successor: 105
Entry: 1 Interval start: 4 Successor: 105
Entry: 2 Interval start: 6 Successor: 105
Entry: 3 Interval start: 10 Successor: 105
Entry: 4 Interval start: 18 Successor: 105
Entry: 5 Interval start: 34 Successor: 105
Entry: 6 Interval start: 66 Successor: 105
=====
DATA STORE
=====
{'clave1': 'valor1'}
=====
+++++ END +++++

```

Figura 26: Consultamos el nodo 2.

## 4.2. Alternativa 2.

Otra alternativa fue hacer uso del [módulo kademlia](#).

[16]Kademlia está destinado a sistemas P2P, ya que las Tablas Hash Distribuidas son un tipo de tablas hash cuyos rangos de registros clave-valor quedan dispuestos de una forma más o menos equitativa entre los nodos participantes en el sistema, sin necesidad de tener un servidor central.

De esta forma se produce una equiparación de responsabilidades entre todos los nodos participantes dentro del propio sistema, eliminando así, la dependencia de un nodo central que mantenga toda la responsabilidad y toda la información que pudieran requerir los nodos y por esto se elimina un cuello de botella del sistema, aumentando el rendimiento de este.

Cada nodo almacena dos tablas, una que será la tabla con los pares clave valor relacionados con la información a compartir, y otra que se considera una tabla de rutas o encaminamiento, que contendrá registros con información sobre otros nodos de la red.

En Kademlia, cada nodo participante, al entrar en el sistema, adquiere un identificador mediante un proceso pseudo-aleatorio, este identificador tiene una longitud total de 160 bits. El proceso debe de ser pseudo-aleatorio, ya que se debe de garantizar una distribución más o menos uniforme.

La forma en que Kademlia trata a su red es de árbol binario, siendo las hojas de este árbol los nodos participantes en el sistema.

[15]El módulo de kademlia en python se trata de una implementación asíncrona de Python de la tabla hash distribuida. Utiliza asyncio para proporcionar comunicación asíncrona. Los nodos se comunican mediante RPC sobre UDP para comunicarse, lo que significa que es capaz de trabajar detrás de un NAT.

[14]RPC es un programa que utiliza una computadora para ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambas, de forma que parezca que se ejecuta en local. El protocolo que se utiliza para esta llamada es un gran avance sobre los sockets de Internet usados hasta el momento. De esta manera el programador no tenía que estar pendiente de las comunicaciones, estando estas encapsuladas dentro de las RPC.

El motivo por el cual no lo hemos utilizado es debido a que, como explico en la definición, la red es un árbol binario y necesitamos una red circular, por ello hemos optado por el protocolo chord.



## Referencias

- [1] "GitHub chordial", *tristanpenman*, [En línea]. Disponible en: <https://github.com/tristanpenman/chordial>. [Accedido: 4-feb-2023]
- [2] "Overlay network", *Fruehe. J.*, [En línea]. Disponible en: <https://www.techtarget.com/searchnetworking/definition/overlay-network>. [Accedido: 4-feb-2023]
- [3] "Distributed Hash Tables and Chord", *Balakrishnan .H.*, [En línea]. Disponible en: <http://web.mit.edu/6.829/www/currentsemester/materials/chord.pdf>. [Accedido: 4-feb-2023]
- [4] "Distributed Hash Tables", *Pang .J.*, [En línea]. Disponible en: <https://www.cs.cmu.edu/~dga/15-744/S07/lectures/16-dht.pdf>. [Accedido: 4-feb-2023]
- [5] "Chord", *Morris .R and co.*, [En línea]. Disponible en: [https://resources.mpi-inf.mpg.de/d5/teaching/ws03\\_04/p2p-data/11-18-writeup1.pdf](https://resources.mpi-inf.mpg.de/d5/teaching/ws03_04/p2p-data/11-18-writeup1.pdf). [Accedido: 4-feb-2023]
- [6] "UNDERSTANDING AND IMPLEMENTING CHORD", *Zave .P.*, [En línea]. Disponible en: <https://www.cs.princeton.edu/courses/archive/fall18/cos561/docs/ChordClass.pdf>. [Accedido: 4-feb-2023]
- [7] "GitHubPython Chord", *gingaramo*, [En línea]. Disponible en: <https://github.com/gingaramo/python-chord>. [Accedido: 4-feb-2023]
- [8] "Chord (DHT) in Python", *Yu .F.*, [En línea]. Disponible en: <https://medium.com/princeton-systems-course/chord-dht-in-python-b8b8985cb80e>. [Accedido: 4-feb-2023]
- [9] "Bitbucket COS518\_Project", *Yu .F.*, [En línea]. Disponible en: [https://bitbucket.org/felixy12/cos518\\_project/src/master/Chord\\_Python/src\\_2/](https://bitbucket.org/felixy12/cos518_project/src/master/Chord_Python/src_2/). [Accedido: 4-feb-2023]
- [10] "Chord (peer-to-peer)", *Wikipedia*, [En línea]. Disponible en: [https://en.wikipedia.org/wiki/Chord\\_\(peer-to-peer\)](https://en.wikipedia.org/wiki/Chord_(peer-to-peer)). [Accedido: 4-feb-2023]
- [11] "The implementation and performance of Chord", *Kaija .K.*, [En línea]. Disponible en: <https://core.ac.uk/download/pdf/157587916.pdf>. [Accedido: 4-feb-2023]
- [12] "Python P2P networking library", *pyp*, [En línea]. Disponible en: <https://pypi.org/project/pyp2p/>. [Accedido: 4-feb-2023]
- [13] "Protocolo Chord", *Wikipedia*, [En línea]. Disponible en: <https://es.wikipedia.org/wiki/Chord>. [Accedido: 6-feb-2023]
- [14] "Llamada a procedimiento remoto", *Wikipedia*, [En línea]. Disponible en: [https://es.wikipedia.org/wiki/Llamada\\_a\\_procedimiento\\_remoto](https://es.wikipedia.org/wiki/Llamada_a_procedimiento_remoto). [Accedido: 6-feb-2023]
- [15] "Kademlia", *Muller .B.*, [En línea]. Disponible en: <https://kademlia.readthedocs.io/en/latest/>. [Accedido: 6-feb-2023]
- [16] "Kademlia", *Wikipedia*, [En línea]. Disponible en: <https://es.wikipedia.org/wiki/Kademlia>. [Accedido: 6-feb-2023]