App:

Se quiere realizar una app que resuelva el puzzle-8. Este juego consiste en una matriz de números mezclados al azar, en el cuál se deben ir moviendo las piezas hasta tener una matriz ordenada (por ej.: 1,2,3,4,5,6,7,8,0).

Para la solución de este juego se utiliza la búsqueda bidireccional: Esta búsqueda consiste en 2 piezas principales:

* Una matriz ordenada: 1,2,3,

4,5,6,

7,8,0

* Una matriz desordenada: 1,2,3,

4,0,5,

7,8,6

Se comienza moviendo el cero en ambas matrices. Este movimiento se realiza hasta que ambas matrices coincidan, esto quiere decir que ambas deben ser iguales. En el caso anterior haciendo un solo movimiento en ambas matrices se encuentra el nodo en común:

* Una matriz ordenada: 1,2,3, 1,2,3

4,5,6, -> 4,5,0

7,8,0 7,8,6

* Una matriz desordenada: 1,2,3, 1,2,3

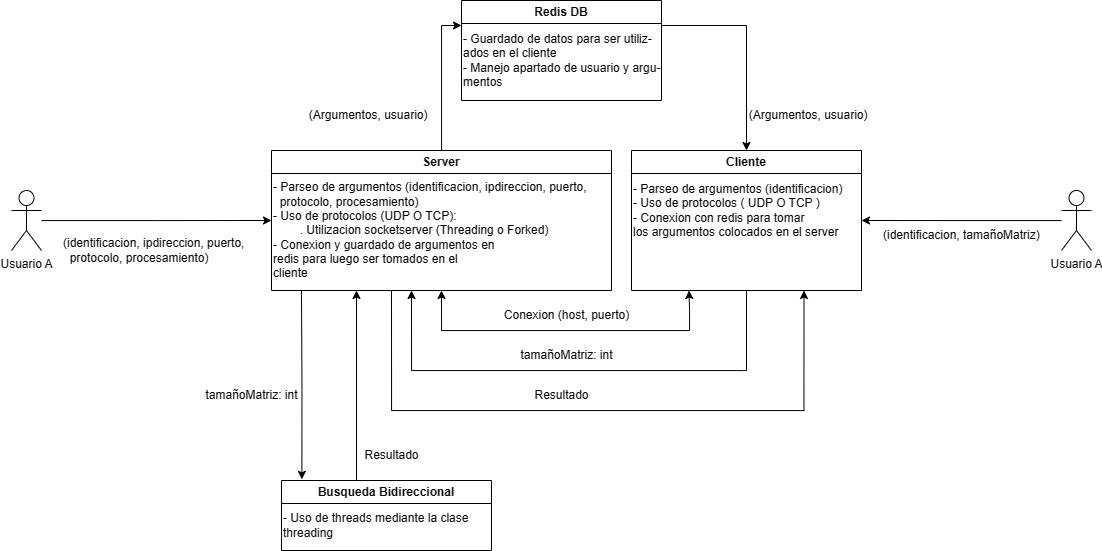
4,0,5, -> 4,5,0 -**>** Nodo en común

7,8,6 7,8,6

Una vez que se encuentra el nodo en común se puede hacer el chequeo de movimientos para atrás de la matriz ordenada para encontrar el mejor camino.

Una consideración a tener en cuenta es que el movimiento del cero en el algoritmo de la búsqueda bidireccional es gracias a un grafo en cuál se detalla los vecinos que tiene el cero dependiendo su posición. Esto nos da la posibilidad de saber en qué direcciones se puede mover.

Arquitectura:



Como se ve en el presente diagrama, la actividad comienza cuando el **usuario** ejecuta el **servidor**, en el se piden argumentos como:

* Identificación: para identificar al **usuario**, puede contener cualquier caractér
* Ipdireccion: host al cuál se quiere conectar
* Puerto: puerto al cuál se quiere conectar
* Protocolo: que protocolo se quiere utilizar (TCP O UDP)
* Procesamiento: si se quiere ejecutar al server mediante procesos o hilos

Una vez parseados los argumentos son guardados en **redis** (DB), esto para facilitar que el **usuario** no vuelva a colocar tantos argumentos en la parte del **cliente** como ya lo hizo en el **servidor.**

Cuando es pedido el tipo de procesamiento y el protocolo es cuando entra en acción **socketserver.** Se verifica el tipo de procesamiento ingresado y se deriva a ‘ForkedProcess’ (para crear procesos) o ‘Threading’ (para crear hilos), esto con el fin de que se puedan conectar tantos clientes como se quiera (y pueda el pc). Por último se sobreescribe la función handle para que este pueda procesar los datos que llegan del **cliente.**

Cuando el **usuario** quiere ejecutar el **cliente** solo necesita lo siguiente:

* Identificación: nombre de **usuario** que utilizó en el **servidor**

Entonces una vez hecha la conexión con el **cliente**,solo con indicar su nombre de **usuario** se tomarán los demás **argumentos** (de la base de datos **redis**)necesarios para realizar la ejecución. Cuando es identificado el **usuario**, se verifica que tipo de protocolo se quiere utilizar (TCP O UDP) y se le pide el tamaño de la matriz tantas veces como quiera.

El tamaño de matriz es enviado al **servidor**, este hace el llamado a la **Búsqueda bidireccional.** Esta clase hace uso de:

* Threads : Para poder realizar la ejecución de 2 hilos. Uno que se encarga de hacer la búsqueda por el inicio (matriz ordenada) y otro que se encarga de hacer la búsqueda por el final (matriz desordenada)
* Barreras:
  + Lock: Se utiliza en la función ‘check’ para la verificar que se encontró un nodo en común. Se utiliza para evitar que los hilos accedan al mismo recurso al mismo tiempo, haciendo que el primer hilo tome la posta y el segundo deba espera
  + Barriers: Se utilizan 2 en la función ‘busquedaBid’, que es la función encargada de realizar el movimiento del cero hasta que la función check encuentra el punto en común entre ambas búsquedas (nodos en común). Volviendo al uso de los hilos en la función ‘busquedaBid’, el primero es el encargado de hacer que un solo hilo entre en la sección core de la función, en la cuál se tiene la utilización de una ‘cola’ en donde se van guardando diferentes combinaciones, esta cola es usada por 2 colas que son variables globales, por lo tanto se debe asegurar que solo una entra a la vez. El segundo barrier es utilizado para bloquear al primer hilo que llegue a el, este deberá esperar a que el segundo hilo llegue al segundo barrier para ser liberado
  + Funciones no prioritarias: Las demás funciones se encargan de:
    - Crear un puzzle (que representaria al puzzle inicial/cola inicial)
    - Crear el grafo (que represente los nodos vecinos)
    - Crear un puzzle random (que representaria al puzzle final/cola final)

Una vez finalizada la búsqueda el **resultado** es enviado al **servidor** y por el ultimo al **cliente** para que el **usuario** pueda verlo