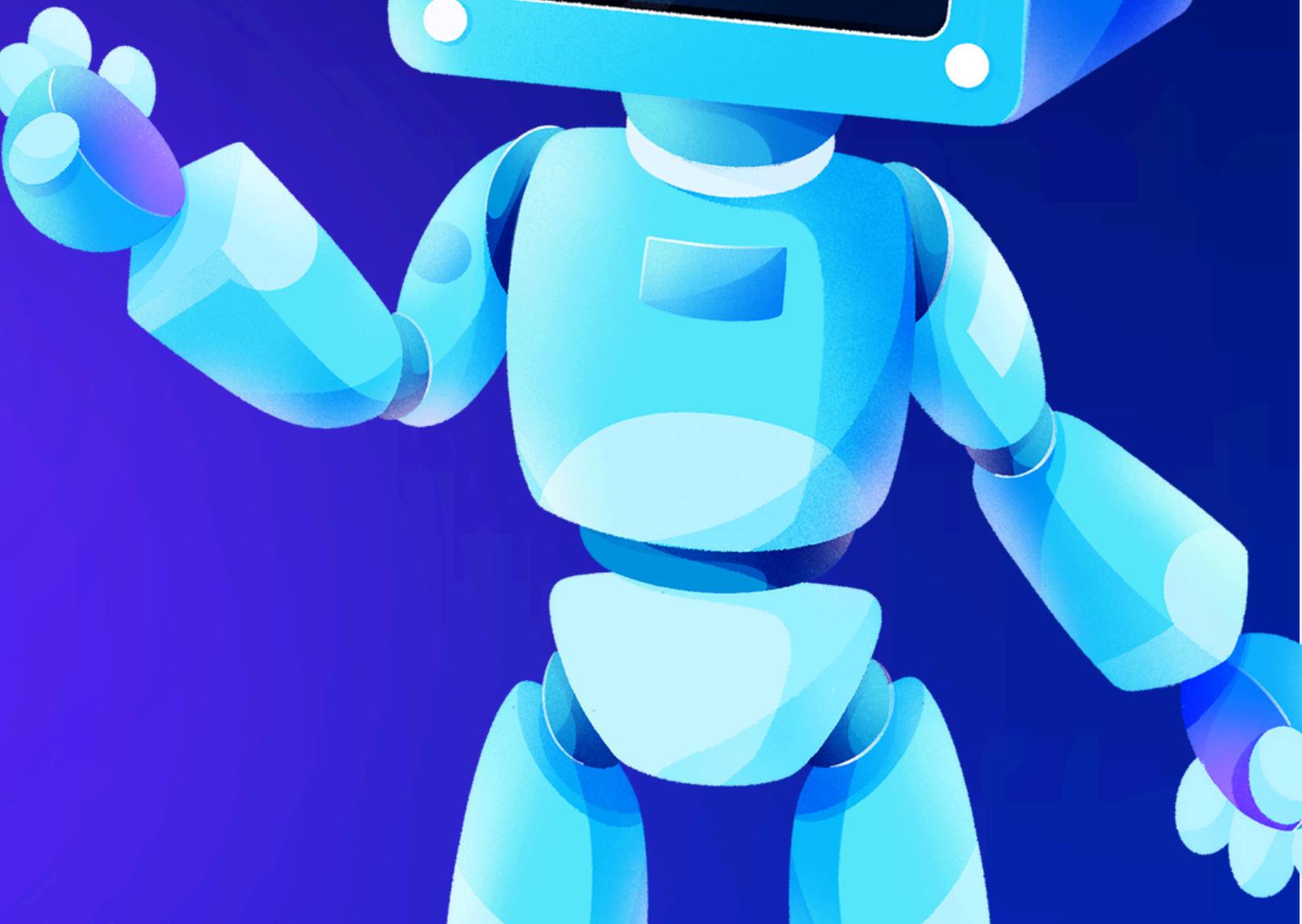
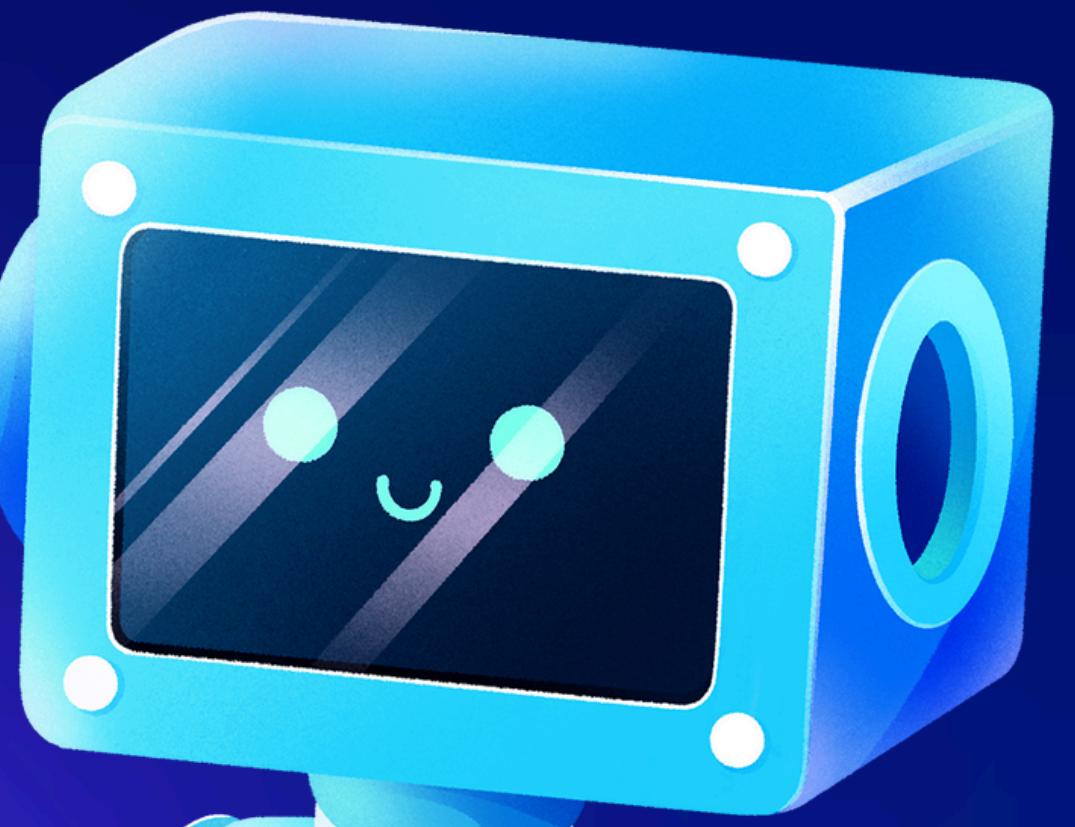


# ESTRUCTURAS DE DATOS MAPAS HASH Y DICIONARIOS





# INTEGRANTES

- Carlos Noguez Juarez
- Gael de Jesus Posada Hernandez
- Rene Tellez Carmona



# TABLAS HASH



# ¿QUÉ SON?

Una estructura de datos que implementa un tipo abstracto de datos llamado diccionario. Esta estructura asocia llaves o claves con valores.

---



# FUNCIONAMIENTO

01

Almacenan información en posiciones pseudo-aleatorias, lo que significa que el acceso ordenado a su contenido es relativamente lento.

02

La operación principal que soportan de manera eficiente es la búsqueda. Permite acceder a los elementos almacenados a partir de una clave generada (usando el nombre, número de cuenta o identificador).

03

Funcionan transformando la clave con una función hash en un número que identifica la posición (casilla o cubeta) donde la tabla hash localiza el valor deseado.

04

Aunque se pueden implementar en vectores de una dimensión, también existen implementaciones multidimensionales basadas en varias claves.

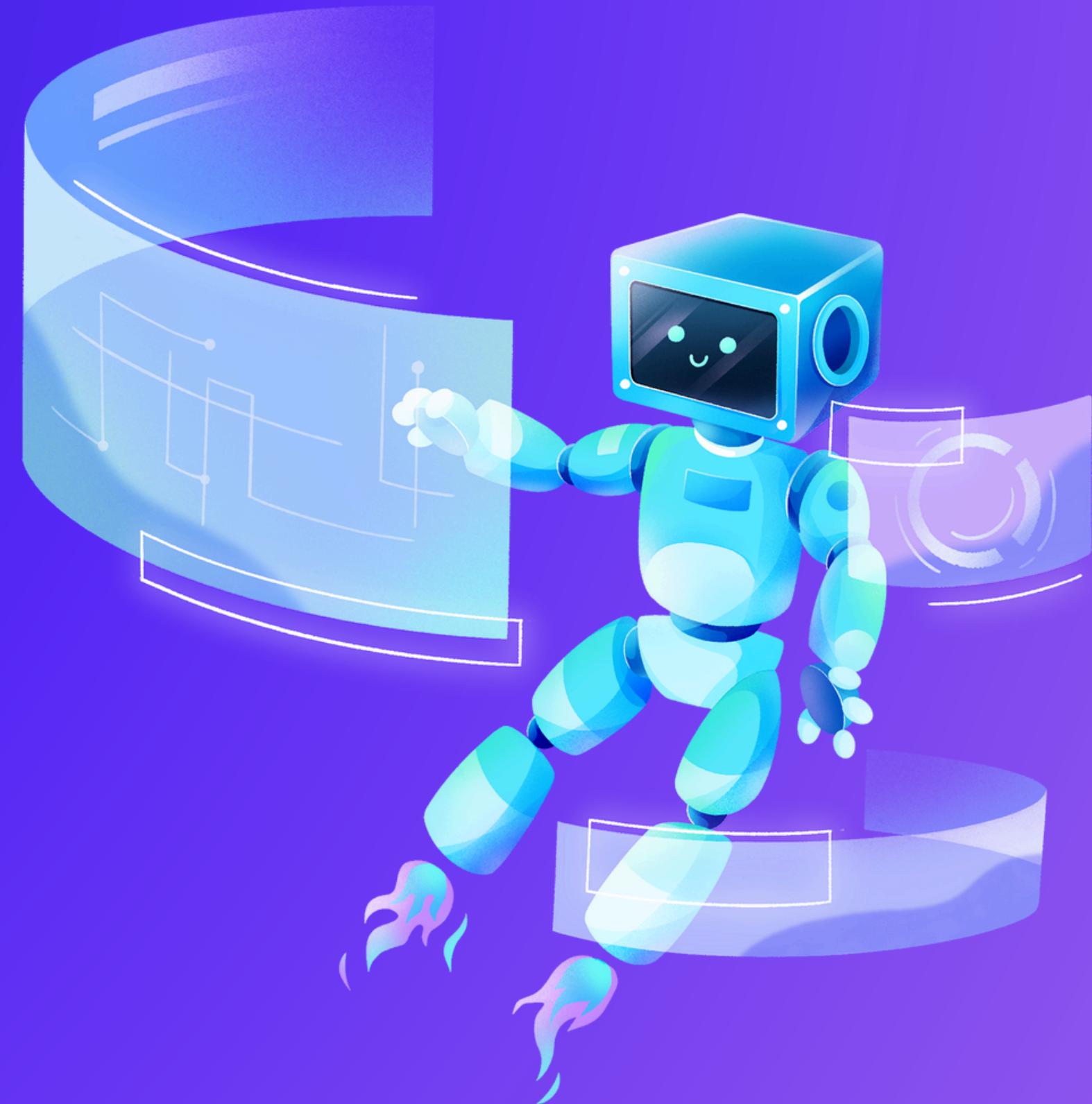
05

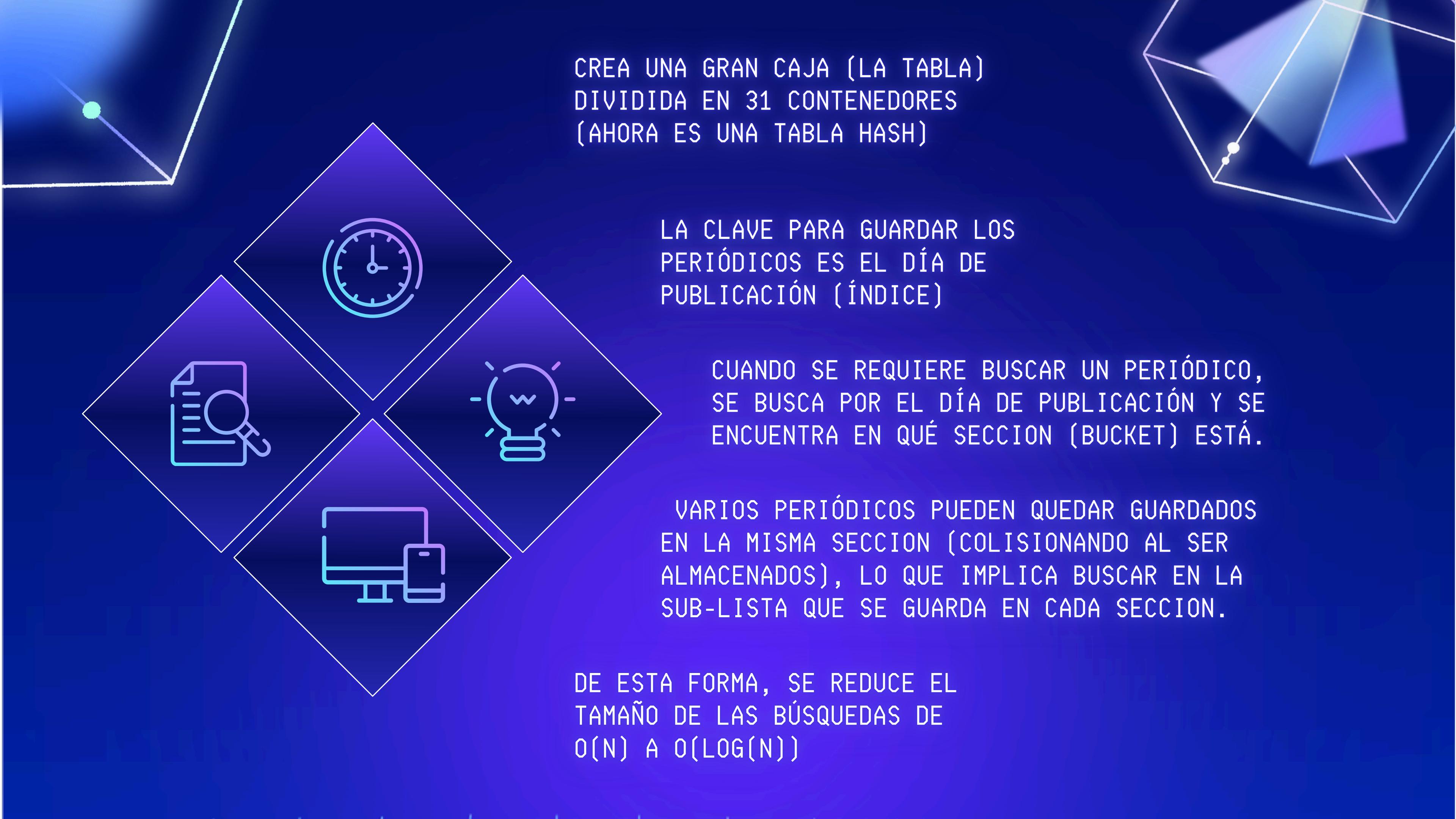
Comparadas con otras estructuras de arrays asociadas, las tablas hash son más útiles cuando se almacenan grandes cantidades de información.

# EJEMPLO

Imagina que necesitas organizar los periódicos que llegan diariamente para ubicarlos rápidamente. Aquí está cómo funciona una tabla hash

---





CREA UNA GRAN CAJA (LA TABLA)  
DIVIDIDA EN 31 CONTENEDORES  
(AHORA ES UNA TABLA HASH)



LA CLAVE PARA GUARDAR LOS  
PERIÓDICOS ES EL DÍA DE  
PUBLICACIÓN (ÍNDICE)



CUANDO SE REQUIERE BUSCAR UN PERIÓDICO,  
SE BUSCA POR EL DÍA DE PUBLICACIÓN Y SE  
ENCUENTRA EN QUÉ SECCION (BUCKET) ESTÁ.



VARIOS PERIÓDICOS PUEDEN QUEDAR GUARDADOS  
EN LA MISMA SECCION (COLISIONANDO AL SER  
ALMACENADOS), LO QUE IMPLICA BUSCAR EN LA  
SUB-LISTA QUE SE GUARDA EN CADA SECCION.



DE ESTA FORMA, SE REDUCE EL  
TAMAÑO DE LAS BÚSQUEDAS DE  
 $O(N)$  A  $O(\log(N))$

# COLISIONES

Ocurren cuando dos o más claves diferentes se asignan a la misma posición en la tabla hash mediante la función hash. Esto puede suceder debido a la limitada cantidad de posiciones disponibles en la tabla y a la naturaleza de la función hash.

- 
- Separate chaining (encadenamiento separado): Cada posición de la tabla hash contiene una lista enlazada (o algún otro tipo de estructura de datos) que almacena todos los valores asociados a esa posición
  - Open addressing (direcciónamiento abierto): En lugar de usar una estructura de datos separada para manejar colisiones, se busca la siguiente posición disponible en la tabla hash.
  - Rehashing: Si la tabla hash está llena o tiene demasiadas colisiones, se puede aumentar su tamaño y luego reorganizar los elementos existentes utilizando una nueva función hash.

# DICCIONARIOS



# ¿QUÉ SON?



Son una estructura de datos fundamental que almacena datos en forma de pares clave-valor.

Cada elemento en un diccionario consta de una clave única que se utiliza para acceder al valor correspondiente. Esto hace que los diccionarios sean ideales para organizar y recuperar datos de manera eficiente

# ¿CÓMO UTILIZARLOS?

En C++, un diccionario no es un tipo de dato incorporado, pero se implementa utilizando Map y Unordered\_map

Ambas almacenan elementos en pares de clave-valor, donde cada clave está asociada con un valor

Unordered\_map se implementa como una tabla de hash y no mantiene ningún orden específico

- Usa map cuando quieras mantener el orden de los elementos.
- Si el orden no es importante y deseas un acceso más rápido en promedio, utiliza unordered\_map





Para acceder al valor de una clave específica en un map o unordered\_map, utiliza la notación de corchetes: map[key].



Si intentas acceder a una clave que no existe, devolverá un valor predeterminado (cero para claves numéricas y cadena vacía para claves de cadena)

```
std::map<int, std::string> map1 = {  
    {1, "Apple"},  
    {2, "Banana"},  
    {3, "Mango"},  
    {4, "Raspberry"},  
    {5, "Blackberry"},  
    {6, "Cocoa"}  
};
```

# APLICACIONES DE LOS DICCIONARIOS

Algunas de las aplicaciones de los diccionarios son:

- **Búsqueda eficiente:** Buscar valores asociados a una clave sin recorrer toda la estructura.
- **Conteo de elementos:** Contar la frecuencia de aparición de elementos en un conjunto de datos.
- **Configuración:** Almacenar configuraciones o parámetros con nombres descriptivos.



# VENTAJAS DE LOS DICCIONARIOS

- **Acceso rápido:** Permiten un acceso rápido a los valores a través de sus claves, ideal para operaciones de búsqueda y recuperación de datos.
- **Almacenamiento flexible:** Puedes almacenar diferentes tipos de datos como valores en un diccionario.
- **Facilidad de uso:** La sintaxis para agregar, eliminar y acceder a elementos en un diccionario es sencilla.
- **Evitan duplicados:** Las claves deben ser únicas, lo que evita la duplicación de claves.
- **Operaciones eficientes:** Las operaciones como la verificación de claves, agregar nuevos pares clave-valor o eliminar elementos son eficientes.



# DESVENTAJAS DE LOS DICCIONARIOS

- **Claves únicas:** Cada clave debe ser única, lo que puede ser limitante en ciertos casos.
- **Orden no garantizado:** Los diccionarios no mantienen un orden específico de sus elementos.
- **Iteración más compleja:** Iterar sobre un diccionario puede ser más complejo que sobre listas o tuplas.
- **Uso de memoria:** Pueden consumir más memoria, especialmente con muchos elementos o claves largas.
- **Rendimiento variable:** Algunas operaciones, como la copia de diccionarios grandes, pueden ser lentas.
- **No permiten claves mutables:** Las claves deben ser inmutables, excluyendo el uso de listas u otros objetos mutables.

# REFERENCIAS BIBLIOGRÁFICAS

- Diccionarios — Programación. (s. f.).  
<http://progra.usm.cl/apunte/materia/diccionarios.html>
- Sedgewick, R., & Wayne, K. (2011). Algorithms. Addison-Wesley Professional.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press.
- Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). Data Structures and Algorithms in Python. Wiley.