Registrar y consultar el estado de paquetes

Un hospital necesita organizar la atención de pacientes en orden de llegada.

Para ello, utiliza una cola. Cada paciente que llega se agrega al final, y el siguiente en ser atendido es siempre el que ha esperado más tiempo.

Historia de usuario Nro.	1	**Título:**	Registro y consulta de estado de paquetes
Descripción	COMO:	Agente de atención al cliente	
	QUIERO:	Registrar el estado de un paquete por su número de seguimiento, consultarlo, actualizarlo y eliminarlo con una tabla hash que maneje colisiones	
	PARA:	Asegurar que múltiples paquetes con hashes iguales sigan siendo accesibles y actualizables	

Criterios de aceptación	• La tabla tiene un número fijo de cubetas (por ejemplo, 16).
	• Cada cubeta es una lista de pares (clave, valor).
	• Al INSERTAR (REGISTER):
	  ○ Calcula idx = hash(tracking_id) % num_cubetas.
	  ○ Si tracking_id ya está en la lista de esa cubeta, actualiza su status.
	  ○ Si no, añade (tracking_id, status) al final de la lista.
	• Al CONSULTAR (STATUS):
	  ○ Mira en la cubeta idx.
	  ○ Si encuentra la clave, imprime status; si no, "NO ENCONTRADO".
	• Al ELIMINAR (CANCEL):
	  ○ Busca y borra el par de la cubeta; si no existe, nada.
	• COUNT: suma las longitudes de todas las cubetas.
	• END: termina la ejecución.		

Usted es contratado por el hospital para construir un programa en Python que cumpla las funcionalidades requeridas por el personal del hospital teniendo como referencia las historias de usuario presentadas previamente.

Entrada	Varias líneas con comandos: • REGISTER A1 En camino • REGISTER B2 Entregado • STATUS A1 • REGISTER C3 Retrasado • REGISTER A1 Entregado • STATUS A1 • CANCEL B2 • STATUS B2 • COUNT • END

Salida	En camino Entregado NO ENCONTRADO 2

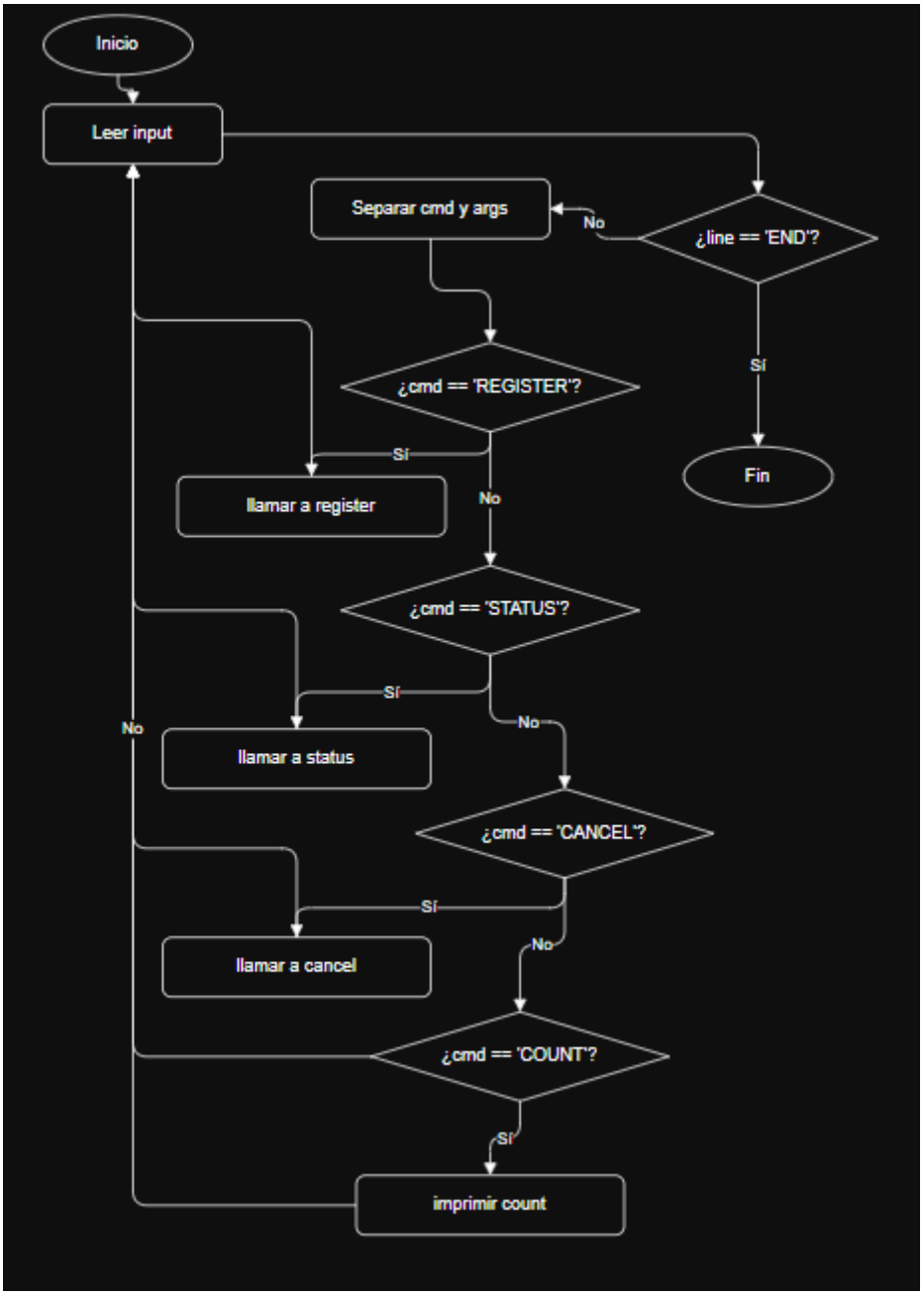
Casos de prueba (visibles)

Entradas de ejemplo	REGISTER PKG001 En_camino REGISTER PKG002 Entregado STATUS PKG001 STATUS PKG003 COUNT REGISTER PKG001 Retrasado STATUS PKG001 CANCEL PKG002 COUNT END
Salida de ejemplo	En_camino NO ENCONTRADO 2 Retrasado 1

Casos de prueba (ocultos)

Caso prueba	Entrada	Salida esperada
1	REGISTER A Entregado REGISTER AA En_bodega STATUS A STATUS AA COUNT CANCEL A STATUS A STATUS AA COUNT END	Entregado En_bodega 2 NO ENCONTRADO En_bodega 1

Diagrama de flujo:



Código Python :

```
class HashTable:

    def __init__(self, num_buckets=16):

        # Creamos un arreglo fijo de cubetas (listas)

        self.num_buckets = num_buckets

        self.buckets = [[] for _ in range(self.num_buckets)]
```

```
    def _bucket_index(self, key: str) -> int:
        # Usamos la función hash de Python para distribuir, luego módulo
        return (sum(ord(c) for c in key) * 31) % self.num_buckets

    def _find_in_bucket(self, bucket: list, key: str):
        """Devuelve el índice en la cubeta o None si no existe."""
        for i, (k, v) in enumerate(bucket):
            if k == key:
                return i
        return None

    def put(self, key: str, value: str):
        idx = self._bucket_index(key)
        bucket = self.buckets[idx]
        pos = self._find_in_bucket(bucket, key)
        if pos is not None:
            # Actualizar valor existente
            bucket[pos] = (key, value)
        else:
            # Insertar nuevo par al final
            bucket.append((key, value))

    def get(self, key: str) -> str:
        idx = self._bucket_index(key)
        bucket = self.buckets[idx]
```

```
        pos = self._find_in_bucket(bucket, key)
        return bucket[pos][1] if pos is not None else "NO ENCONTRADO"

    def remove(self, key: str):
        idx = self._bucket_index(key)
        bucket = self.buckets[idx]
        pos = self._find_in_bucket(bucket, key)
        if pos is not None:
            bucket.pop(pos)

    def count(self) -> int:
        # Suma todos los pares en cada cubeta
        total = 0
        for bucket in self.buckets:
            total += len(bucket)
        return total



class RetoHashEnvios:
    def __init__(self):
        self.table = HashTable(num_buckets=16)

    def run(self):
        while True:
            try:
```

```
            line = input().strip()
            if line == "END":
                break

            parts = line.split(maxsplit=2)
            cmd = parts[0]

            if cmd == "REGISTER" and len(parts) == 3:
                tracking_id, status = parts[1], parts[2]
                self.table.put(tracking_id, status)

            elif cmd == "STATUS" and len(parts) == 2:
                print(self.table.get(parts[1]))

            elif cmd == "CANCEL" and len(parts) == 2:
                self.table.remove(parts[1])

            elif cmd == "COUNT":
                print(self.table.count())

            # Líneas inválidas se ignoran
        except EOFError:
            break
        except Exception:
            pass
```

```
if __name__ == "__main__":
    RetoHashEnvios().run()
```

```python
if __name__ == "__main__":
    RetoHashEnvios().run()
```

```
REGISTER A Entregado
REGISTER AA En_bodega
STATUS A
Entregado
STATUS AA
En_bodega
COUNT
2
CANCEL A
STATUS A
NO ENCONTRADO
STATUS AA
En_bodega
COUNT
1
END
```