Documentación del método “Descolocar_Elemento”

Código completo:

```
# Método para descolocar/eliminar un elemento según su prioridad
    def Descolocar_Elemento(self):
        if self.final is None: #si la cola está vacía
            print("No hay elementos, cola vacía.")
            return None
        #inicializamos punteros para recorrer la cola
        actual = self.frente
        anterior = None

        #guardamos el nodo con mayor prioridad(menor número)**nos ayudara a
        nodo_mayor_prioridad = actual
        nodo_anterior_mayor_prioridad = None

        #reorremos la cola en búsqueda de la menor prioridad(mayor prioridad s
        while actual is not None:
            if actual.dato.prioridad < nodo_mayor_prioridad.dato.prioridad:
                nodo_mayor_prioridad = actual #nuevo nodo de mayor prioridad
                nodo_anterior_mayor_prioridad = anterior #nodo anterior al que se e
            anterior = actual
            actual = actual.siguiente

        # Si el nodo a eliminar está al inicio
        if nodo_anterior_mayor_prioridad is None:
            self.frente = nodo_mayor_prioridad.siguiente #sse mueve el frente
            if self.frente is None:
                self.final = None #si ya no hay más elementos
        else:
            # Si el nodo está en el medio o al final
            nodo_anterior_mayor_prioridad.siguiente = nodo_anterior_mayor_priori
            if nodo_mayor_prioridad == self.final:
                self.final = nodo_anterior_mayor_prioridad #actualizamos el final, si e
```

```
    # Mostramos el elemento eliminado
    print(f"Elemento '{nodo_mayor_prioridad.dato.elemento}' con prioridad {
    return nodo_mayor_prioridad.dato
```

```
actual = self.frente
anterior = None
```

✅ Se **preparan punteros** para recorrer la cola:

- `actual`: empieza en el primer nodo de la cola (`frente`).
- `anterior`: inicialmente `None`, irá siguiendo al nodo anterior durante el recorrido.

```
nodo_mayor_prioridad = actual
nodo_anterior_mayor_prioridad = None
```

✅ Se asume que el primer nodo (`actual`) es el de mayor prioridad hasta que se demuestre lo contrario.

- Se guarda en `nodo_mayor_prioridad` .
- Su anterior (al principio) también es `None`.

```
while actual is not None:
```

✅ Comienza recorrer la cola nodo por nodo, hasta que `actual` llegué a `None` (el final).

```
    if actual.dato.prioridad < nodo_mayor_prioridad.dato.prioridad:
```

✅ Compara que la prioridad del nodo actual con la del nodo que, hasta el momento, tiene la prioridad más alta (es decir, el número más pequeño).

```
        nodo_mayor_prioridad = actual
        nodo_anterior_mayor_prioridad = anterior
```

✅ Si el nodo actual tiene mayor prioridad lógica, se actualizan:

- `nodo_mayor_prioridad`: el nuevo nodo a eliminar.
- `nodo_anterior_mayor_prioridad`: su nodo anterior.

```
    anterior = actual
    actual = actual.siguiente
```

✅ Se mueven los punteros al siguiente nodo:

- `anterior` se convierte en el nodo actual.
- `actual` pasa al siguiente nodo (`siguiente`).

Ahí termina el recorrido de la cola.

Ahora se elimina el nodo de mayor prioridad que se identificó.

```
if nodo_anterior_mayor_prioridad is None:
```

✅ Verifica si nodo a eliminar está en el frente de la cola.

Si es así, su nodo anterior es `None`.

```
    self.frente = nodo_mayor_prioridad.siguiente
```

✅ Se actualiza el `frente` de la cola para que apunte al siguiente nodo, eliminando efectivamente el nodo anterior.

```
    if self.frente is None:
        self.final = None
```

✅ Si después de eliminar ya no queda ningún nodo, se actualiza también `final` a `None` (cola vacía).

```
else:
```

✅ Si el nodo a eliminar **no es el primero**, está en medio o al final de la cola.

```
    nodo_anterior_mayor_prioridad.siguiente = nodo_mayor_prioridad.siguie
nte
```

✅ Se **salta el nodo a eliminar** uniendo el nodo anterior con el siguiente del nodo a eliminar.

Esto rompe la conexión con el nodo que se quiere sacar.

```
    if nodo_mayor_prioridad == self.final:
        self.final = nodo_anterior_mayor_prioridad
```

✅ Si el nodo que se eliminó era el último (`final`), se actualiza `self.final` para que sea ahora el nodo anterior.

```
print(f"Elemento '{nodo_mayor_prioridad.dato.elemento}' con prioridad {no
do_mayor_prioridad.dato.prioridad} eliminado.")
```

✅ Muestra en consola qué elemento fue eliminado y cuál era su prioridad.

```
return nodo_mayor_prioridad.dato  # Se retorna el dato eliminado
```

✅ Devuelve el objeto `Elemento` que se eliminó (su contenido, no el nodo).

Resumen del funcionamiento:

1. **Recorre toda la cola** para encontrar el nodo con menor número de prioridad.
2. **Elimina ese nodo,** ya sea que esté al principio, medio o final.
3. **Ajusta los punteros** `frente` y `final` si es necesario.
4. **Devuelve** el elemento eliminado para usarlo si quiere.