Clases a Definir

Modularizaremos la solución en las siguientes clases:

- **EdR**: Clase principal que implementa la interfaz del TP. Contendrá las estructuras de datos principales.
- **Estudiante**: Clase auxiliar que almacena la información individual de cada estudiante.
- **MinHeap**: Nuestra implementación de un Heap de Mínimos que almacenará tuplas (puntaje, idEstudiante) .
- **HeapHandle**: Clase auxiliar para el MinHeap que permite la operación actualizarPrioridad en O(log E).
- **NotaFinal**: Clase auxiliar para la operación corregir(), conteniendo (nota, id) para facilitar el ordenamiento.

Estructuras de Datos Principales (Atributos de EdR)

- **Estudiante[] _estudiantes**: Un arreglo de tamaño E (cantidad total de estudiantes).
  - **Representa:** La base de datos principal de estudiantes.
  - **Justificación:** Permite acceso O(1) a cualquier estudiante por su id.
- **MinHeap _puntajes**: Un Heap de Mínimos que almacena tuplas de (puntaje, id_estudiante)[8].
  - **Representa:** El ranking de puntajes de los estudiantes.
  - **Justificación:** Permite encontrar al estudiante con el peor puntaje en O(1) y extraerlo en O(\log E). Es crucial para consultarDarkWeb.
- **int[][] _aula**: Una matriz (grilla) de tamaño Lado x Lado.
  - **Representa:** El mapa de asientos del aula. _aula[f][c] contiene el id del estudiante en ese asiento, o un valor (ej. -1) si está vacío.
  - **Justificación:** Permite encontrar a los vecinos en copiarse en O(1).
- **int[] _solucionCanoninca**: Un arreglo de tamaño R (cantidad de respuestas).
  - **Representa:** La plantilla de respuestas correctas.
- **boolean[] _yaEntregaron**: Arreglo de booleanos de tamaño E.
  - **Representa:** El estado de entrega de cada estudiante.
  - **Justificación:** Permite a entregar operar en O(1).
- **boolean[] _esSospechoso**: Arreglo de booleanos de tamaño E.
  - **Representa:** El resultado del chequeo de copias.
  - **Justificación:** Permite a corregir filtrar estudiantes en O(1).

Clase Auxiliar: Estudiante

Esta clase se almacena en el arreglo _estudiantes y contiene:

- **int[] _examen**: Arreglo de tamaño R que almacena las respuestas (o -1 si está en blanco).
- **int _puntaje**: El puntaje actual del estudiante.

- **HeapHandle _posicionEnHeap**: El "enlace" al heap. Es un puntero a la posición del estudiante en el _puntajes Min-Heap, permitiendo actualizaciones en O(log E).
- **Posicion _pos**: (fila, col) del estudiante, para calcular vecinos en copiarse.

Estructura Auxiliar: Mapa de Frecuencias (para chequearCopias)

Esta estructura se crea **localmente** dentro de la función chequearCopias.

- **int[][] _conteoRespuestas**: Una matriz de R x C (donde C es la cantidad de opciones). _conteoRespuestas[i][j] almacena cuántos estudiantes respondieron la opción j a la pregunta i.
- **Justificación:** Permite pre-calcular los conteos de todas las respuestas en O(E . R) , para luego verificar a cada estudiante en O(1) por respuesta, cumpliendo la complejidad total de O(E . R) .

Cómo se Cumplen las Complejidades (Operación por Operación)

nuevoEdR(ladoAula, E, examenCanonico): O(E . R)

1. Crear arreglos _estudiantes[E], _yaEntregaron[E], _esSospechoso[E] y la grilla _aula[L][L]: O(E + L^2).
2. Guardar _solucionCanoninca: O(R).
3. Iterar i de 0 a E-1 (Bucle O(E)):
    - Crear new Estudiante() (O(1)).
    - Crear _examen[R] (inicializado en -1): O(R).
    - Calcular (fila, col) y guardar en _aula[f][c] = i y e._pos = (f,c) (O(1)).
    - Guardar _estudiantes[i] = e (O(1)).
4. Construir el _puntajes (Min-Heap) con los E estudiantes (todos con puntaje 0) usando BuildHeap: O(E).

    Total: El paso dominante es el bucle de O(E . R).

entregar(estudiante): O(1)

5. Accedemos al arreglo de entregas: _yaEntregaron[estudiante] = true.
6. Esto es una asignación en un arreglo, lo cual es **O(1)**.

notas(): O(E)

7. Creamos un ArrayList<Integer> notas.
8. Iteramos desde i = 0 hasta E-1 (el tamaño del arreglo _estudiantes).
9. En cada paso, accedemos a _estudiantes[i]._puntaje (acceso O(1)) y lo añadimos a la lista .

    Total: O(E).

resolver(estudiante, nroEjercicio, respuesta): O(log E)

10. Accedemos al estudiante: Estudiante e = _estudiantes[estudiante] (O(1)).
11. Actualizamos el examen: e._examen[nroEjercicio] = respuesta (O(1)).
12. Recalculamos el puntaje: Comparamos con _solucionCanoninca[nroEjercicio] y ajustamos e._puntaje (O(1)) .
13. Actualizamos el Heap: _puntajes.actualizarPrioridad(e._posicionEnHeap, e._puntaje). Esta operación es O(log E) .
14. **Total:** O(log E).

copiarse(estudiante): O(R + log E)

15. Accedemos al estudiante: Estudiante e = _estudiantes[estudiante] (O(1)).
16. Buscamos al vecino: Usamos e._pos y la grilla _aula para encontrar los 3 ids de los vecinos (izquierda, derecha, adelante) en O(1).
17. Comparamos exámenes: Iteramos por las R respuestas para encontrar la primera que el vecino (con más respuestas) tenga y e no. Esto es O(R) .
18. Actualizamos el examen y puntaje: e._examen[...] = ..., e._puntaje += ... (O(1)).
19. Actualizamos el Heap: _puntajes.actualizarPrioridad(e._posicionEnHeap, e._puntaje) (O(log E)).
20. **Total:** O(R + log E).

consultarDarkWeb(k, examenDW): O(k . (R + log E))

21. Iniciamos un bucle que se repite k veces.
22. Extraemos al peor: (puntaje, id) = _puntajes.extraerMin() (O(log E)).
23. Accedemos al estudiante: Estudiante e = _estudiantes[id] (O(1)).
24. Reemplazamos el examen: e._examen = examenDW (copia de arreglo, O(R)) .
25. Recalculamos el puntaje: Comparamos el nuevo examen con _solucionCanoninca (O(R)).
26. Re-insertamos en el heap: e._posicionEnHeap = _puntajes.insertar(e._puntaje, id) (O(log E)).
27. **Total:** O(k . (R + \log E)).

chequearCopias(): O(E . R)

28. Construir el mapa de frecuencias: Creamos la matriz _conteoRespuestas. Iteramos por los E estudiantes y sus R respuestas, poblando el mapa . O(E . R).
29. Verificar sospechosos: Iteramos por los E estudiantes. Para cada uno, iteramos por sus R respuestas (no en blanco) y consultamos el mapa (O(1)) para verificar si supera el 25%. Si todas sus respuestas son sospechosas, marcamos _esSospechoso[id] = true. O(E . R).
30. **Total:** O(E . R).

corregir(): O(E . log E)

31. Creamos un ArrayList<NotaFinal> notasParaOrdenar.

32. **Filtramos:** Iteramos de id = 0 a E-1 (O(E)). Si !_esSospechoso[id] , obtenemos su puntaje de _estudiantes[id]._puntaje (O(1)) y añadimos new NotaFinal(puntaje, id) a la lista.
33. **Ordenamos:** Usamos Collections.sort() (o un MergeSort implementado) en notasParaOrdenar con el comparador personalizado (desc por nota, mayor id por empate).

    Total: O(E) (filtrado) + O(E . log E) (ordenamiento) = O(E . log E).