Optimización de Planificación de Rutas por Lotes

Antes de la optimización diaria de rutas, un "Gestor de Rutas" en Route Manager puede necesitar clasificar grandes volúmenes de puntos de entrega o tareas según criterios específicos, como la fecha límite de entrega, la proximidad geográfica o el peso total del paquete para la carga del vehículo. El algoritmo HeapSort ofrece una garantía de rendimiento de O(N log N) para la ordenación, lo que lo convierte en una opción fiable para esta fase de preparación de datos.

Historia de usuario Nro.	1	**Título:**	Ordenar Tareas para Planificación de Rutas
Descripción	COMO:	Gestor de Rutas	
	QUIERO:	Poder ordenar una lista grande de tareas de entrega por una métrica específica (ej. fecha límite, volumen de carga)	
	PARA:	Preparar los datos de manera eficiente para los algoritmos de optimización de rutas y consolidar cargas de forma lógica.	
Criterios de aceptación	• La entrada será una lista de tareas, cada una con un ID (string) y una métrica numérica (entero). • El programa debe ordenar la lista de tareas en orden ascendente basándose en la métrica numérica utilizando el algoritmo HeapSort. • La salida debe ser la lista de IDs de tareas ordenadas. • Si la lista de entrada está vacía, la salida debe ser una línea vacía.		

Historia de Usuario Nro. 1: Clasificación de Tareas por Lotes

HeapSort garantiza un rendimiento de O(N log N) en todos los casos (mejor, promedio y peor), a diferencia de QuickSort, que puede degradarse a O(N^2) en el peor escenario. Para Route Manager, donde grandes lotes de datos pueden procesarse para la planificación de rutas, esta garantía proporciona una predictibilidad valiosa en sistemas empresariales. Sin embargo, en la práctica, QuickSort a menudo se ejecuta más rápido debido a una mejor localidad de caché y factores constantes más pequeños. Este ejercicio subraya la importancia de comprender tanto las garantías teóricas como las implicaciones prácticas al seleccionar algoritmos para un sistema de producción.

Programa para Clasificación por Lotes con HeapSort

La estructura conceptual del programa en Java implicaría una clase DeliveryTask con atributos id y metric. Una clase RoutePlannerSorter contendría un método estático heapSort(List<DeliveryTask> tasks), junto con métodos auxiliares como _heapify y _buildMaxHeap.

Entrada	Una única línea con tareas, cada una formateada como <ID>-<Métrica>, separadas por espacios (ej. "ORDEN_A-10 ORDEN_B-5 ORDEN_C-12")..
Salida	Una única línea con los IDs de las tareas ordenadas, separados por espacios. Si la entrada está vacía, la salida debe ser una línea vacía.

Instrucciones para Calificación Automática:

- La clase principal debe llamarse HeapSort.
- Dentro de la clase HeapSort, debe existir un método llamado ejecutar.
- Únicamente se debe imprimir la salida final ordenada.

Casos de prueba (visibles)

Entradas de ejemplo 1	`A-10 B-5 C-15 D-2 E-8`
Salida de ejemplo 1	`D B E A C`

Entradas de ejemplo 2	`Tarea1-0 Tarea2--5 Tarea3-10 Tarea4-0`
Salida de ejemplo 2	`Tarea2 Tarea1 Tarea4 Tarea3`

Entradas de ejemplo 3	`ItemX-7 ItemY-7 ItemZ-7`
Salida de ejemplo 3	`ItemX ItemY ItemZ`

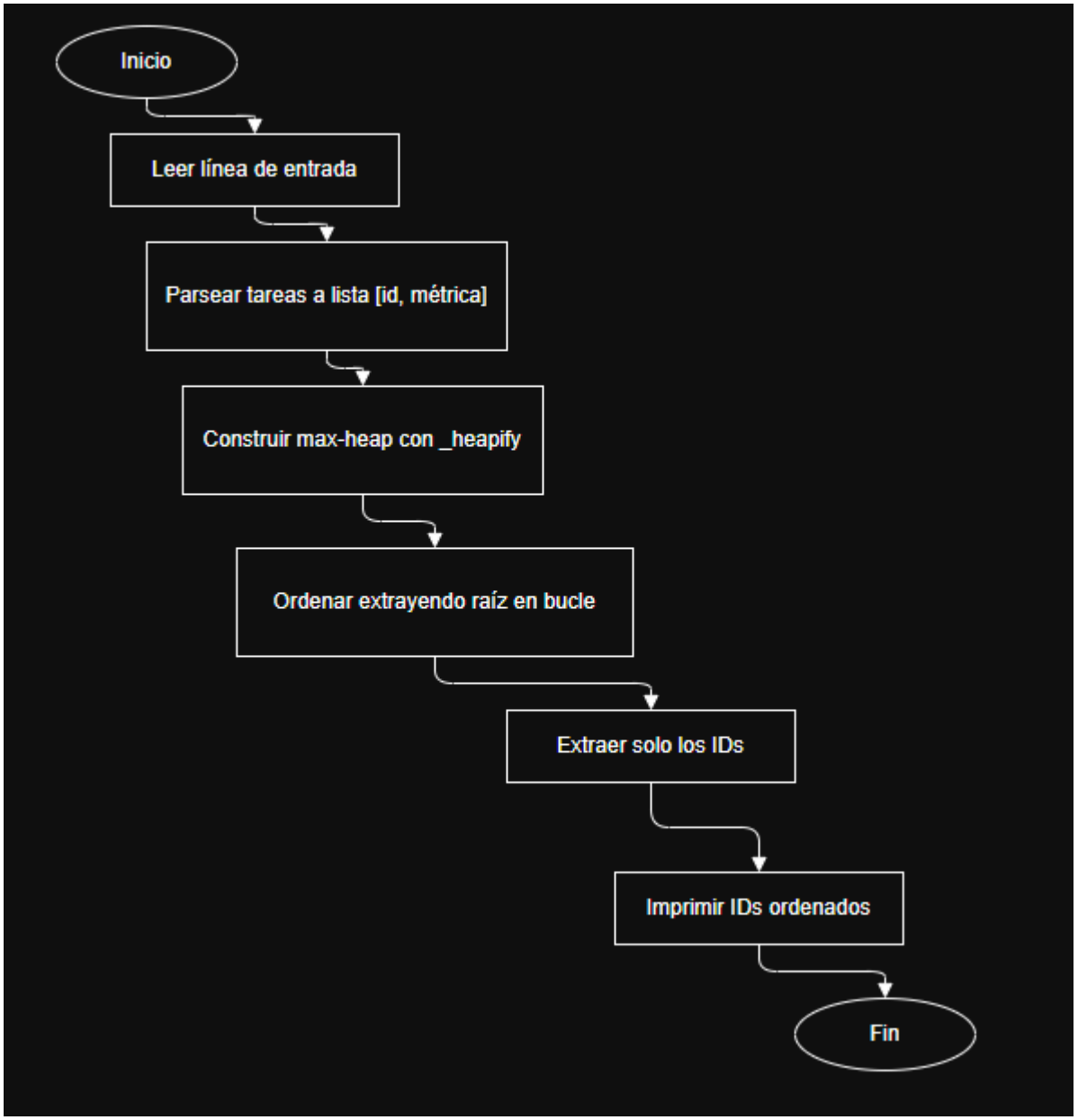
Entradas de ejemplo 4	`SingleTask-42`
Salida de ejemplo 4	`SingleTask`

Casos de prueba (ocultos)

Caso prueba	Entrada	Salida esperada
1	`Task9-1 Task8-2 Task7-3 Task6-4 Task5-5`	`Task9 Task8 Task7 Task6 Task5`
2	`Correcto-5 Incorrecto`	`Correcto OtraCorrecta`

	SinMetrica- TaskSoloID OtraCorrecta-10	

Diseño: diagrama de flujo



Desarrollo: Implementación

Código Python :

```
class HeapSort:

    def _heapify(self, arr, n, i):

        # n es el tamaño del heap

        # i es el índice de la raíz

        largest = i  # Inicializa largest como la raíz

        left = 2 * i + 1

        right = 2 * i + 2



        # Si el hijo izquierdo es mayor que la raíz

        if left < n and arr[i][1] < arr[left][1]: # Comparar métricas (índice 1)

            largest = left



        # Si el hijo derecho es mayor que el actual largest

        if right < n and arr[largest][1] < arr[right][1]: # Comparar métricas
(índice 1)

            largest = right



        # Si largest no es la raíz

        if largest != i:

            arr[i], arr[largest] = arr[largest], arr[i]  # Intercambio

            self._heapify(arr, n, largest)



    def ejecutar(self):

        try:

            line = input().strip()
```

```
            if not line:
                print("")
                return

            # Parsear la entrada: "ID-Métrica ID-Métrica..."
            tasks_str = line.split()
            tasks = []
            for task_s in tasks_str:
                parts = task_s.split('-')
                if len(parts) == 2:
                    task_id = parts[0]
                    try:
                        metric = int(parts[1])
                        tasks.append([task_id, metric])
                    except ValueError:
                        # Ignorar elementos con métricas no numéricas
                        pass

            n = len(tasks)

            # Construir un max-heap
            # Reorganizar elementos desde el último nodo no hoja hacia arriba
            for i in range(n // 2 - 1, -1, -1):
                self._heapify(tasks, n, i)

            # Extraer elementos uno por uno
            # El heapify construye un max-heap basado en la métrica (mayor métrica en la raíz).
```

```
            # Para obtener un orden ascendente, extraemos el elemento más grande
(raíz)
            # y lo ponemos al final, luego reconstruimos el heap con los elementos
restantes.
            for i in range(n - 1, 0, -1):
                tasks[i], tasks[0] = tasks[0], tasks[i]  # Mover la raíz (más
grande) al final
                self._heapify(tasks, i, 0) # Llamar heapify en el heap reducido
(tamaño i)

            # La lista 'tasks' ahora está ordenada por métrica en orden ascendente
            # Extraer solo los IDs para la salida
            sorted_ids = [task[0] for task in tasks]
            print(" ".join(sorted_ids))

        except EOFError:
            pass
        except Exception as e:
            # print(f"Error: {e}") # For debugging
            pass # Ignorar otras excepciones para la calificación automática

if __name__ == "__main__":
    HeapSort().ejecutar()
```