Examen de Diseño de Algoritmos

Contexto: Simulación y Supervivencia

Generado por Aris

Diciembre 2025

Instrucciones

- Responda con precisión. La ambigüedad se penalizará.
- Para el ejercicio de Divide y Vencerás, es obligatorio plantear la recurrencia $T(n)$.
- Para el ejercicio de Voraz/Backtracking, complete la lógica basada en el diagrama UML proporcionado.

---

1. Ejercicio 1: Divide y Vencerás (The Matrix)

Contexto: Eres el Operador de la nave Nebuchadnezzar. Estás analizando el código fuente de Matrix en busca de "glitches" o anomalías. El código se visualiza como una secuencia de n números enteros distintos que representan firmas de energía.

Una secuencia ordenada de menor a mayor representa estabilidad perfecta. Sin embargo, cada vez que un número mayor aparece *antes* que un número menor en la secuencia (es decir, una pareja de índices (i, j) tal que $i < j$ pero $A[i] > A[j]$), se considera una **inversión** o fallo de realidad.

Cuantas más inversiones tenga la secuencia, más inestable es el sector y más urgente es enviar a Neo.

Problema: Diseña un algoritmo eficiente basado en **Divide y Vencerás** que calcule el número total de inversiones en una secuencia de entrada de tamaño n.

Se pide:

1. **Diseño:** Explica detalladamente cómo divides el problema, cómo resuelves los casos base y, crucialmente, cómo combinas las soluciones parciales para contar las inversiones que cruzan la mitad izquierda y derecha (pista: piensa en una modificación de *MergeSort*).
2. **Recurrencia y Complejidad:** Plantea la ecuación de recurrencia $T(n)$ de tu algoritmo. Resuélvela usando el Teorema Maestro (o el método del árbol) para obtener la complejidad asintótica $O(\cdot)$. Compara esta complejidad con la de un enfoque de fuerza bruta.
3. **Ejemplo:** Traza la ejecución brevemente para el vector de entrada: $[2, 4, 1, 3, 5]$.

2. Ejercicio 2: Backtracking y Voraz (Fallout: New Vegas)

Contexto: El Mensajero (The Courier) ha encontrado un búnker de la preguerra lleno de suministros valiosos (armas, medicinas, tecnología). Sin embargo, su *Pip-Boy* indica que su capacidad de carga es limitada. Si excede el peso máximo, no podrá escapar de los Mutantes que se aproximan.

Cada objeto en el búnker tiene:

- Un **Peso** (w_i) en kg.
- Un **Valor** (v_i) en chapas (moneda del juego).

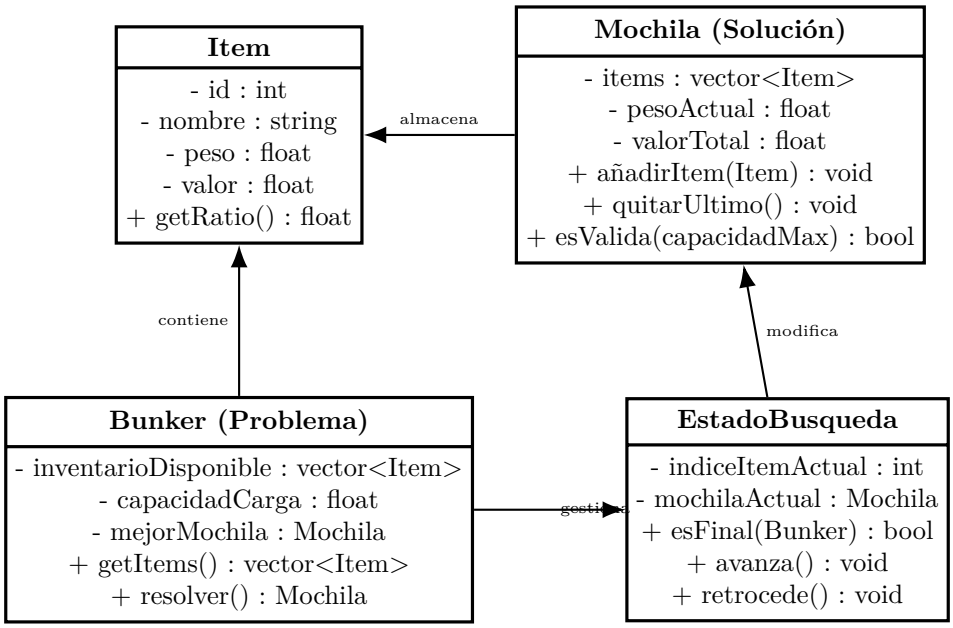
El objetivo es maximizar el valor total de los objetos cargados en la mochila sin superar la capacidad máxima W.

Parte A: Estategia Voraz (Greedy)

1. Diseña una función de selección voraz para este problema. ¿Qué criterio usas para elegir el siguiente objeto? (Ej: ¿El más valioso? ¿El más ligero? ¿Alguna relación entre ambos?).
2. ¿Garantiza tu algoritmo voraz la solución óptima en todos los casos para el problema de la mochila 0/1 (donde no se pueden fraccionar objetos)? Si la respuesta es no, proporciona un contraejemplo numérico.

Parte B: Diseño Orientado a Objetos y Backtracking Como la estrategia voraz no garantiza la optimización perfecta, debes implementar un sistema de **Backtracking** para encontrar la combinación ideal.

A diferencia de los códigos monolíticos anteriores, el sistema del Pip-Boy está modularizado. Se te proporciona el siguiente diagrama UML. Tu tarea es implementar la lógica interna basándote en esta estructura.



Se pide implementar (pseudocódigo o C++):

1. La función `resolver()` de la clase `Bunker`: Debe inicializar el backtracking y devolver la `mejorMochila`.

2. La función recursiva de backtracking (puede ser un método privado o función auxiliar) que utiliza `EstadoBusqueda`. Debes detallar explícitamente:

   - **Marcaje:** Cómo añades un ítem a la solución parcial.
   - **Llamada Recursiva:** Cómo exploras las ramas (incluir el ítem vs no incluirlo).
   - **Desmarcaje (Backtracking):** Cómo restauras el estado para explorar otras ramas.
   - **Poda (Opcional pero recomendada):** ¿Cómo usarías el valor de la solución voraz o una cota superior para dejar de explorar ramas inútiles?