

⚡ Algoritmos Voraces (Greedy) 1

⌚ Breve historia de los algoritmos Greedy

Los **algoritmos voraces (greedy)** son una de las estrategias más antiguas y elegantes en el diseño de algoritmos.

El término "greedy" se popularizó en la década de **1950 y 1960**, especialmente con el trabajo de **Edsger Dijkstra**, quien en 1956 formuló el famoso **algoritmo de Dijkstra** para encontrar caminos más cortos en grafos —un ejemplo clásico de enfoque voraz.

El principio fundamental es **tomar siempre la mejor decisión local posible**, esperando que al final conduzca a la **solución global óptima**.

No siempre funciona, pero en muchos problemas —como el cambio de monedas, la mochila fraccionaria o la selección de actividades— esta estrategia produce soluciones exactas o muy eficientes.

La belleza de los algoritmos voraces radica en su **simplicidad y eficiencia**, ya que suelen tener complejidad **O(n log n)** o **O(n)** y son fáciles de implementar.

💡 Introducción a los Algoritmos Voraces

- Los **algoritmos voraces** (o *greedy algorithms*) son una técnica de diseño que consiste en **tomar en cada paso la mejor decisión local**, con la esperanza de alcanzar una **solución global óptima**.
- Se aplican a muchos problemas de **optimización**, donde se busca maximizar o minimizar una cantidad (por ejemplo, valor, peso, tiempo o coste).
- Su principal ventaja es la **simplicidad y eficiencia**: no requieren explorar todas las combinaciones posibles, sino que se guían por una regla local.

❖ Estructura general de un algoritmo voraz

1. Inicializar una **solución vacía**.
2. Seleccionar el **mejor candidato disponible** según un criterio local.
3. Comprobar si la solución parcial sigue siendo **factible**.
4. Repetir hasta completar la solución o no queden candidatos válidos.

⚠️ No todos los problemas pueden resolverse con esta estrategia.

Algunos requieren **programación dinámica** o **búsqueda exhaustiva** para garantizar la optimalidad.

ⓧ Encontrar el mínimo de una lista

Dificultad: ★

Descripción del problema:

Dada una lista de números, queremos encontrar el número más pequeño.

Ejemplo:

- Entrada: [5, 2, 9, 1, 7]
- Salida: 1

Explicación de la solución (greedy):

En cada paso, comparamos el número actual con el mínimo encontrado hasta ahora y actualizamos el mínimo si encontramos uno más pequeño. La idea greedy consiste en **elección localmente el valor más pequeño en cada comparación**, garantizando que al final se obtiene el mínimo global.

1 Problema del candado (Minimum Rotations Lock Problem)

Dificultad: ★

Descripción del problema:

Tenemos un candado con **n** anillos, cada uno con los dígitos del 0 al 9. El candado muestra un número inicial y queremos llegar a un número objetivo. Podemos girar cada anillo hacia arriba o hacia abajo y queremos **minimizar el número total de rotaciones**.

Ejemplo:

- Entrada: `inicio = "1234", objetivo = "3456"`
- Salida: 10 (rotaciones mínimas)

Explicación de la solución (greedy):

Para cada anillo, la elección voraz consiste en **girar en la dirección que requiera menos pasos** (ya sea hacia arriba o hacia abajo). Esta decisión localmente óptima para cada anillo asegura el número mínimo total de rotaciones.

2 Dar cambio con el menor número de monedas

Dificultad: ★ ★

Descripción del problema:

Dado un valor **N** y un conjunto de monedas disponibles, queremos formar el valor exacto usando **la menor cantidad posible de monedas**.

Ejemplo:

- Entrada: `monedas = [50, 20, 10, 5, 2, 1], N = 93`
- Salida: [50, 20, 20, 2, 1] (5 monedas)

Explicación de la solución (greedy):

Se selecciona siempre la **moneda de mayor valor que no exceda el monto restante**. Este proceso se repite hasta llegar al total. Funciona correctamente si el sistema de monedas es **canónico**.

3 Suma mayor que la mitad

Dificultad: ★ ★

Descripción del problema:

Dado un arreglo de números no negativos, queremos seleccionar el **mínimo número de elementos** cuya suma sea **mayor que la suma del resto**.

Ejemplo:

- Entrada: [3, 1, 7, 1, 2, 4]
- Salida: 2 (elementos [7, 4] suman 11 > 10)

Explicación de la solución (greedy):

Se ordenan los números de mayor a menor y se van sumando hasta superar la mitad de la suma total. El enfoque greedy consiste en **tomar siempre los elementos más grandes primero**, minimizando el número de elementos necesarios.

4 Assign Maximum Cookies **Dificultad:** ★ ★ ★**Descripción del problema:**

Dadas dos listas:

- `greed[]`: nivel mínimo de hambre de cada niño
- `cookie[]`: tamaño de cada cookie

Cada niño puede recibir a lo sumo una cookie. Queremos **maximizar el número de niños satisfechos**, donde un niño se considera satisfecho si recibe una cookie de tamaño al menos igual a su hambre.

Ejemplo:

- Entrada: `greed = [1, 2, 3], cookie = [1, 1]`
- Salida: 1 (solo el niño con hambre 1 puede recibir una cookie)

Explicación de la solución (greedy):

Ordenamos ambas listas. Luego, **asignamos a cada niño la cookie más pequeña que lo satisfaga**. Así, se maximiza la cantidad de niños que pueden recibir una cookie, dejando las más grandes para los más hambrientos.

5 Reducir un arreglo a un solo elemento con costo mínimo **Dificultad:** ★ ★ ★**Descripción del problema:**

Dado un arreglo de enteros, podemos elegir pares de elementos y eliminar el **mayor** de los dos. El **costo de cada operación** es el **valor del menor** del par. Queremos reducir el arreglo a un solo elemento con **costo total mínimo**.

Ejemplo:

- Entrada: [4, 2, 1, 3]
- Salida: 6 (eliminando en orden de menor a mayor: 1+2+3)

Explicación de la solución (greedy):

La observación clave es que **cada elemento excepto el máximo se sumará una vez al costo**. Por tanto, el enfoque óptimo es **sumar todos los elementos menos el mayor**, minimizando así el costo total.

6 Formar el número más grande (Largest Number Problem)

Dificultad: ★ ★ ★

Descripción del problema:

Dada una lista de números, queremos combinarlos para formar el **número más grande posible**.

Ejemplo:

- Entrada: [54, 546, 548, 60]
- Salida: "6054854654"

Explicación de la solución (greedy):

Se compara cada par de números considerando su concatenación en ambos órdenes. La elección voraz consiste en **colocar primero el número que genere la mayor combinación**. Repetido para todos los números, se obtiene el número globalmente más grande.

7 Seleccionar objetos para obtener el mayor valor (Fractional Knapsack)

Dificultad: ★ ★ ★ ★

Descripción del problema:

Tenemos **n** objetos, cada uno con un **valor** y un **peso**. La capacidad total que podemos transportar es **W**. Podemos tomar **fracciones** de objetos. Queremos **maximizar el valor total transportado**.

Ejemplo:

- Entrada: `objetos = [(60,10),(100,20),(120,30)], W = 50`
- Salida: 240 (tomando los objetos de mayor valor/peso hasta llenar la capacidad)

Explicación de la solución (greedy):

Se calcula el **valor por unidad de peso** de cada objeto y se ordenan de mayor a menor. Luego, se toman objetos empezando por el de **mayor valor relativo**, hasta llenar la capacidad. Si un objeto no cabe completo, se toma la fracción que quepa. La decisión local (tomar el mejor valor/peso disponible) lleva a la solución global óptima.

Conclusión

Los algoritmos greedy muestran que **la simplicidad puede ser poderosa**.

Aunque no siempre conducen a la solución óptima, en muchos problemas reales ofrecen soluciones rápidas y sorprendentemente efectivas.

El secreto está en **identificar correctamente la elección local óptima** que garantice un resultado globalmente bueno.