ALGORITMOS VORACES

Los algoritmos voraces (Greedy Algorithms) son algoritmos que toman decisiones de corto alcance, basadas en información inmediatamente disponible, sin importar consecuencias futuras. Suelen ser bastante simples y se emplean sobre todo para resolver problemas de optimización, como por ejemplo, encontrar la secuencia óptima para procesar un conjunto de tareas por un computador.

Habitualmente, los elementos que intervienen son:

· un conjunto o lista de candidatos (tareas a procesar, por ejemplo);

· un conjunto de decisiones ya tomadas.

· una función que determina si un conjunto de candidatos es una solución al problema (aunque no tiene por qué ser la óptima);

· una función que determina si un conjunto de candidatos es una solución óptima al problema

Debemos identificar:

- Una función “es_solucion” que decide si la solución es valida

- Una función “elegir_candidato” que nos indique como elegir un candidato de forma adecuada

- Una función “es_factible” que nos indique si la solución propuesta tiene sentido

Ventajas:

- Este tipo de algoritmos es útil en problemas de optimización, es decir, cuando hay una variable que maximizar o minimizar.

Desventajas:

- Hay que tener cuidado. No siempre la solución que encontremos será óptima

- Si necesitamos una solución óptima, se puede intentar demostrar matemáticamente que el algoritmo voraz es correcto, o se puede intentar con una técnica más avanzada, como la programación dinámica

BUSQUEDA EXHAUSTIVA

La Búsqueda Exhaustiva se utiliza cuando parece que no hay mejor manera de resolver un problema que probando todas las posibles soluciones. Casi siempre es lenta, pero a veces es lo único que se puede hacer

También a veces es útil plantear un problema como Búsqueda Exhaustiva y a partir de ahí encontrar una mejor solución. Otro uso práctico de la Búsqueda Exhaustiva es resolver un problema con un tamaño de datos de entrada lo suficientemente pequeño.

Para obtener un algoritmo de búsqueda exhaustiva necesitamos:

- Una función “es_solucion” que verifique si un intento es correcto

- Un generador “candidatos” que devuelva uno a uno de los candidatos con los que hay que probar

Ventajas:

- Es una técnica fácil de implementar y de leer, por lo que se suele utilizar cuando se quiere tener implementaciones fáciles de probar y depurar.

- Hay problemas que admiten muchas soluciones. Cuando se usa una técnica de búsqueda exhaustiva, es fácil adaptar el programa para encontrar todas las soluciones del problema, o una cantidad K de soluciones, o buscar soluciones hasta haber consumido cierta cantidad de recursos (tiempo, memoria, CPU, etc.).

Desventajas:

- La cantidad de posibles soluciones a explorar por lo general crece exponencialmente a medida que crece el tamaño del problema

- Depende mucho del poder de cómputo de la máquina para resolver el problema.

Pasos para resolver un problema con Búsqueda Exhaustiva:

- Identificar el conjunto de candidatos a solución.

- Pensar que tipo de datos tienen nuestros candidatos a solución.

- Dar un orden lógico a los elementos de nuestro conjunto de candidatos

- Establecer nuestro objetivo. ¿Cuándo podemos parar de buscar?

- Basado en los puntos 2 y 4, escribir una función es solución en Python

- Basados en los puntos 1, 2 y 3, escribir un generador que provea uno a uno los candidatos a solución en el orden establecido.

- Iterar sobre los candidatos hasta en encontrar una solución.

ITERABLES

Una clase iterable es una clase que puede ser iterada, es decir, podemos recorrer sus elementos utilizando bucles while o for. Dentro de Python hay gran cantidad de clases iterables como las listas, strings o diccionarios

Se podría explicar la diferencia entre iteradores e iterables usando un libro como analogía. El libro sería nuestra clase iterable, ya que tiene diferentes páginas a las que podemos acceder. El libro podría ser una lista, y cada página un elemento de la lista. Por otro lado, el iterador sería

un marca páginas, es decir, una referencia que nos indica en qué posición estamos del libro, y que puede ser usado para “navegar” por él.

GENERADORES

Los generadores introducen una nueva forma de devolver valores en Python: la palabra clave yield.

Cualquier función que utilice al menos una instrucción yield sería un generador.

Un generador se diferencia de una función normal en que tras ejecutar el yield, la función devuelve el control a quien la llamo, pero la función es pausada y el estado (valor de las variables) es guardado. Esto permite que su ejecución pueda ser reanudada más adelante.

ALGORITMOS

Un algoritmo es una serie finita de pasos para resolver un problema.

Hay que hacer énfasis en dos aspectos para que un algoritmo exista:

1. El número de pasos debe ser finito. De esta manera el algoritmo debe terminar en un tiempo finito con la solución del problema,

2. El algoritmo debe ser capaz de determinar la solución del problema.

Características de un algoritmo:

1. Entrada: definir lo que necesita el algoritmo

2. Salida: definir lo que produce.

3. No ambiguo: explicito, siempre sabe qué comando ejecutar.

4. Finito: El algoritmo termina en un número finito de pasos.

5. Correcto: Hace lo que se supone que debe hacer. La solución es correcta

6. Efectividad: Cada instrucción debe ser básica y se completa en tiempo finito.

7. General: Debe ser lo suficientemente general como para contemplar todos los casos de entrada.

DIVIDE Y VENCERAS

La técnica Divide y Vencerás permite generar algoritmos recursivos para resolver una amplia variedad de problemas.

Consiste en:

- Dividir: Descomponer un problema en un conjunto de sub problemas más pequeños

- Conquistar: Resolver los sub problemas más pequeños

- Combinar: Combinar los sub problemas más pequeños para obtener la solución general

Necesitamos identificar:

- Una función “es_caso_base” que verifique si estamos en un caso base

- Una función “resolver_caso_base” que le da solución a los casos base

- Una función “dividir” que nos indique como dividir el problema en varios sub problemas relacionados, pero de menor tamaño

- Una función “combinar” que nos permita juntar las soluciones parciales del caso anterior para obtener la solución.

Ventajas:

- Por lo general, da como resultado algoritmos elegantes y eficientes

- Como los sub problemas son independientes, si poseemos una máquina con más de un procesador podemos paralelizar la resolución de los sub problemas, acortando así el tiempo de ejecución de la solución.

Desventajas:

- Los sub problemas en los que dividimos el problema original deben ser independientes, sino, no funcionará

- Se necesita cierta intuición para ver cuál es la mejor manera de descomponer un problema en partes

- Algunas veces es mejor utilizar la Programación Dinámica

- Si bien los algoritmos que genera utilizar esta técnica son eficientes, demostrar tal eficiencia requiere matemáticas avanzadas.