Diseño de Algoritmos

May 31, 2024

Contenido

Algoritmos Voraces

Los algoritmos ávidos o voraces (*Greedy Algorithms*) son algoritmos que toman decisiones de corto alcance, basadas en información inmediatamente disponible, sin importar consecuencias futuras.

Algoritmos Voraces

Los algoritmos ávidos o 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.

Algoritmos Voraces

Habitualmente, los elementos que intervienen son:

Algoritmos Voraces

Habitualmente, los elementos que intervienen son:

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

Algoritmos Voraces

Habitualmente, los elementos que intervienen son:

- un conjunto o lista de **candidatos** (tareas a procesar, por ejemplo);
- un conjunto de decisiones ya tomadas.

Algoritmos Voraces

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);

Algoritmos Voraces

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

Algoritmos Voraces

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

Algortimos Voraces

Para resolver el problema de optimización hay que encontrar un conjunto de candidatos que optimiza la función objetivo. Los algoritmos voraces proceden por pasos, eligiendo un candidato por vez

Algortimos Voraces

Para resolver el problema de optimización hay que encontrar un conjunto de candidatos que optimiza la función objetivo. Los algoritmos voraces proceden por pasos, eligiendo un candidato por vez
. Inicialmente nuestra elección de solución es vacía. A continuación, en cada paso, se intenta añadir al conjunto el mejor candidato de los aún no escogidos, utilizando la función de selección.

Algortimos Voraces

Para resolver el problema de optimización hay que encontrar un conjunto de candidatos que optimiza la función objetivo. Los algoritmos voraces proceden por pasos, eligiendo un candidato por vez
. Inicialmente nuestra elección de solución es vacía. A continuación, en cada paso, se intenta añadir al conjunto el mejor candidato de los aún no escogidos, utilizando la función de selección.
Si el conjunto resultante no es plausible, se rechaza el candidato y no se le vuelve a considerar en el futuro.

Algortimos Voraces

Para resolver el problema de optimización hay que encontrar un conjunto de candidatos que optimiza la función objetivo. Los algoritmos voraces proceden por pasos, eligiendo un candidato por vez
. Inicialmente nuestra elección de solución es vacía. A continuación, en cada paso, se intenta añadir al conjunto el mejor candidato de los aún no escogidos, utilizando la función de selección.
Si el conjunto resultante no es plausible, se rechaza el candidato y no se le vuelve a considerar en el futuro.
En caso contrario, se incorpora al conjunto de candidatos escogidos y permanece siempre en él. Tras cada incorporación se comprueba si el conjunto resultante es una solución del problema.

Algortimos Voraces

Para resolver el problema de optimización hay que encontrar un conjunto de candidatos que optimiza la función objetivo. Los algoritmos voraces proceden por pasos, eligiendo un candidato por vez
. Inicialmente nuestra elección de solución es vacía. A continuación, en cada paso, se intenta añadir al conjunto el mejor candidato de los aún no escogidos, utilizando la función de selección.
Si el conjunto resultante no es plausible, se rechaza el candidato y no se le vuelve a considerar en el futuro.
En caso contrario, se incorpora al conjunto de candidatos escogidos y permanece siempre en él. Tras cada incorporación se comprueba si el conjunto resultante es una solución del problema.
Un algoritmo voraz es correcto si la solución así encontrada es siempre óptima.

Algoritmos Voraces

Problema Tenemos billetes de 1000, 500, 200 100, 50, 20 y 10 pesos. Si un cliente gastó 960 pesos, pagó con 1000 pesos y suponiendo que tengo cantidad suficiente de todos los billetes, ¿cual es la mejor forma de darle vuelto, minimizando la cantidad de billetes que entrego?

Algoritmos Voraces

Problema Tenemos billetes de 1000, 500, 200 100, 50, 20 y 10 pesos. Si un cliente gastó 960 pesos, pagó con 1000 pesos y suponiendo que tengo cantidad suficiente de todos los billetes, ¿cual es la mejor forma de darle vuelto, minimizando la cantidad de billetes que entrego?

Lo mas sensato, es dar dos billetes de 20 pesos. Así, minimizamos la cantidad de billetes que debemos entregar

Algoritmos Voraces

Observemos que siempre me conviene entregar un billete de denominación lo más alta posible, sin pasarme del valor que debo devolver.

Algoritmos Voraces

Observemos que siempre me conviene entregar un billete de denominación lo más alta posible, sin pasarme del valor que debo devolver.

Si luego de elegir esta billete sigo debiendo , vuelvo a repetir con lo que me quede.

```
total = 20
candidatos =[1000, 500, 200, 100, 50, 20, 10] * 2

def es_solucion(eleccion_actual):
    return sum(eleccion_actual) == total

def elegir_candidato():
    return max(candidatos)

def es_factible(eleccion):
    return sum(eleccion) <= total

eleccion_actual = []

while not es_solucion(eleccion_actual):
    x = elegir_candidato()
    candidatos.remove(x)
    if es_factible(eleccion_actual + [x]):
        eleccion_actual.append(x)

print(eleccion_actual)
```

Algoritmos Voraces

El fragmento

```
eleccion_actual = []

while not es_solucion(eleccion_actual):
    x = elegir_candidato()
    candidatos.remove(x)
    if es_factible(eleccion_actual + [x]):
        eleccion_actual.append(x)

print(eleccion_actual)
```

es común a todos los algoritmos voraces, también conocidos como *greedy*.

Solo debemos identificar:

Solo debemos identificar:

- Un función ‘es_solucion‘ que decida si la solución es válida.
- 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.

Algoritmos Voraces

Problema En el país Albariocoque la moneda oficial es el fiji. Tienen billetes de 100 fijis y monedas de 4 fijis, 3 fijis y 1 fiji. Si un cliente gastó 94 fijis, pagó con 100 fijis y suponiendo que tengo cantidad infinita de todas las monedas, ¿cual es la mejor forma de darle vuelto, minimizando la cantidad de monedas que entrego?

Algunas formas posibles de darle el vuelto, serian:

- seis monedas de un fiji.
- tres monedas de un fiji y una moneda de tres fijis.
- dos monedas de tres fijis
- una moneda de cuatro fijis y dos monedas de un fiji.

Este proceso, de buscar todas las posibilidades y ver cual me conviene utilizar, es una búsqueda exhaustiva. Podemos ver que la solución óptima es elegir entregar dos monedas de 3 fijis. Sin embargo, nuestro algoritmo greedy hubiera entregado una moneda de 4 fijis y dos monedas de 1 fiji, dando una solución subóptima.

Ventajas

Ventajas

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

Algoritmos Voraces

Ventajas

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

Desventajas

Algoritmos Voraces

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.

Algoritmos Voraces

Ventajas

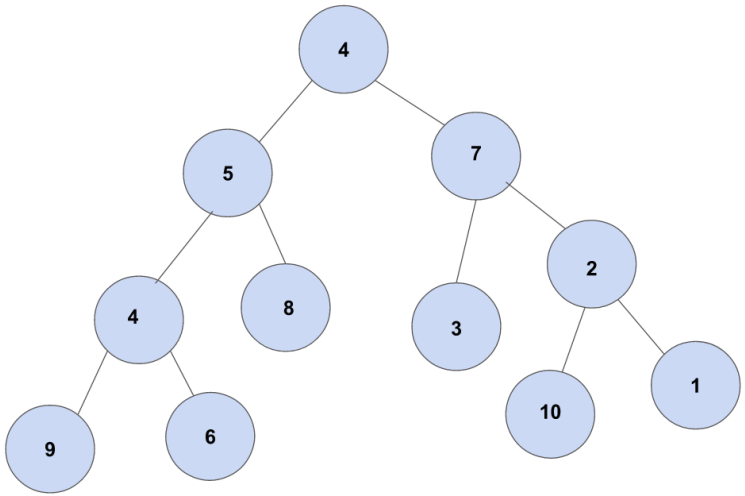
- Este tipo de algoritmos es útil en problemas de optimización, es decir, cuando hay una variable que maximizar o minimizar.

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.

Algoritmos Voraces

Problema 1 Un algoritmo voraz recorre el siguiente árbol, comenzando desde la raíz y eligiendo en cada paso el nodo de más valor, intentando encontrar el camino desde la raíz de mayor suma. ¿Qué camino elegirá el algoritmo? ¿Es óptima la respuesta?



Contenido

Búsqueda exhaustiva

A veces parece que no hay mejor manera de resolver un problema que probando todas las posibles soluciones. Esta aproximación es llamada Búsqueda Exhaustiva, casi siempre es lenta, pero a veces es lo único que se puede hacer.

Búsqueda exhaustiva

A veces parece que no hay mejor manera de resolver un problema que probando todas las posibles soluciones. Esta aproximación es llamada Búsqueda Exhaustiva, 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.

Búsqueda exhaustiva

A veces parece que no hay mejor manera de resolver un problema que probando todas las posibles soluciones. Esta aproximación es llamada Búsqueda Exhaustiva, 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.

La mayoría de los problemas de Búsqueda Exhaustiva pueden ser reducidos a generar objetos de combinatoria, como por ejemplo cadenas de caracteres, permutaciones (reordenaciones de objetos) y subconjuntos.

Iterables

Una clase **iterable** es una clase que puede ser iterada, es decir, podemos recorrer sus elementos utilizando bucles `while` o `for`.

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.

Iteradores

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.

Iteradores

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 marcapáginas, es decir, una referencia que nos indica en qué posición estamos del libro, y que puede ser usado para "navegar" por él.

Iteradores

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 marcapáginas, es decir, una referencia que nos indica en qué posición estamos del libro, y que puede ser usado para "navegar" por él.

Dada una clase iterable, podemos obtener un iterador sobre ella utilizando la función incluida `iter`.

Iteradores

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 marcapáginas, es decir, una referencia que nos indica en qué posición estamos del libro, y que puede ser usado para "navegar" por él.

Dada una clase iterable, podemos obtener un iterador sobre ella utilizando la función incluida `iter`.

Cuando estamos trabajando con iteradores puede ser de utilidad la función `next` la cuál avanza el iterador una posición, devolviendo lo que se encontraba en la posición actual.

Generadores

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

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` será un generador.

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` será un generador.

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

Búsqueda exhaustiva

```
N = 2047

def es_solucion(intento):
  if N % intento == 0: return True
  return False

def candidatos():
    for i in range(2, N):
      yield i

def resolver():
  for candidato in candidatos():
      if es_solucion(candidato):
        return candidato
  return -1 # Error

solucion = resolver()
print(f"{solucion} divide a {N}")
```

Para obtener un algoritmo de búsqueda exhaustiva necesitamos:

- Una función `es_solución` que verifique si un intento es correcto.
- Un generador `candidatos` que devuelva uno a uno los candidatos con los que hay que probar.

Búsqueda Exhaustiva

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.).

Búsqueda Exhaustiva

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.

Búsqueda Exhaustiva

Pasos para resolver un problema mediante búsqueda exhaustiva:

1. Identificar el conjunto de candidatos a solución.

Búsqueda Exhaustiva

Pasos para resolver un problema mediante búsqueda exhaustiva:

1. Identificar el conjunto de candidatos a solución.
2. Representar formalmente cada candidato, pensar que tipo de datos tienen nuestros candidatos a solución. ¿Estamos buscando un número, una lista, una tupla de dos elementos?

Búsqueda Exhaustiva

Pasos para resolver un problema mediante búsqueda exhaustiva:

1. Identificar el conjunto de candidatos a solución.
2. Representar formalmente cada candidato, pensar que tipo de datos tienen nuestros candidatos a solución. ¿Estamos buscando un número, una lista, una tupla de dos elementos?
3. Dar un orden lógico a los elementos de nuestro conjunto de candidatos.

Búsqueda Exhaustiva

Pasos para resolver un problema mediante búsqueda exhaustiva:

1. Identificar el conjunto de candidatos a solución.
2. Representar formalmente cada candidato, pensar que tipo de datos tienen nuestros candidatos a solución. ¿Estamos buscando un número, una lista, una tupla de dos elementos?
3. Dar un orden lógico a los elementos de nuestro conjunto de candidatos.
4. Establecer nuestro objetivo. ¿Cuando podemos parar de buscar?

Búsqueda Exhaustiva

Pasos para resolver un problema mediante búsqueda exhaustiva:

1. Identificar el conjunto de candidatos a solución.
2. Representar formalmente cada candidato, pensar que tipo de datos tienen nuestros candidatos a solución. ¿Estamos buscando un número, una lista, una tupla de dos elementos?
3. Dar un orden lógico a los elementos de nuestro conjunto de candidatos.
4. Establecer nuestro objetivo. ¿Cuando podemos parar de buscar?
5. Basado en los puntos 2 y 4, escribir una función `es_solución` en Python.

Búsqueda Exhaustiva

Pasos para resolver un problema mediante búsqueda exhaustiva:

1. Identificar el conjunto de candidatos a solución.
2. Representar formalmente cada candidato, pensar que tipo de datos tienen nuestros candidatos a solución. ¿Estamos buscando un número, una lista, una tupla de dos elementos?
3. Dar un orden lógico a los elementos de nuestro conjunto de candidatos.
4. Establecer nuestro objetivo. ¿Cuando podemos parar de buscar?
5. Basado en los puntos 2 y 4, escribir una función `es_solución` en Python.
6. Basados en los puntos 1, 2 y 3, escribir un generador que provea uno a uno los candidatos a solución en el orden establecido.

Búsqueda Exhaustiva

Pasos para resolver un problema mediante búsqueda exhaustiva:

1. Identificar el conjunto de candidatos a solución.
2. Representar formalmente cada candidato, pensar que tipo de datos tienen nuestros candidatos a solución. ¿Estamos buscando un número, una lista, una tupla de dos elementos?
3. Dar un orden lógico a los elementos de nuestro conjunto de candidatos.
4. Establecer nuestro objetivo. ¿Cuando podemos parar de buscar?
5. Basado en los puntos 2 y 4, escribir una función `es_solución` en Python.
6. Basados en los puntos 1, 2 y 3, escribir un generador que provea uno a uno los candidatos a solución en el orden establecido.
7. Iterar sobre los candidatos hasta en encontrar una solución.

Búsqueda Exhaustiva

Pasos para resolver un problema mediante búsqueda exhaustiva:

1. Identificar el conjunto de candidatos a solución.
2. Representar formalmente cada candidato, pensar que tipo de datos tienen nuestros candidatos a solución. ¿Estamos buscando un número, una lista, una tupla de dos elementos?
3. Dar un orden lógico a los elementos de nuestro conjunto de candidatos.
4. Establecer nuestro objetivo. ¿Cuando podemos parar de buscar?
5. Basado en los puntos 2 y 4, escribir una función `es_solución` en Python.
6. Basados en los puntos 1, 2 y 3, escribir un generador que provea uno a uno los candidatos a solución en el orden establecido.
7. Iterar sobre los candidatos hasta en encontrar una solución.

Búsqueda Exhaustiva

En nuestro ejemplo:

1. El conjunto de candidatos es $\{x \in \mathbb{N} | 2 \leq x \leq n-1\}$.

Búsqueda Exhaustiva

En nuestro ejemplo:

1. El conjunto de candidatos es $\{x \in \mathbb{N} | 2 \leq x \leq n-1\}$.
2. Cada candidato a solución puede representarse en Python como un `int`.

Búsqueda Exhaustiva

En nuestro ejemplo:

1. El conjunto de candidatos es $\{x \in \mathbb{N} | 2 \leq x \leq n-1\}$.
2. Cada candidato a solución puede representarse en Python como un `int`.
3. Una posible forma lógica de recorrer nuestro conjunto de candidatos es recorrerlos de menor a mayor.
4. Podemos parar de buscar encontremos un candidato i tal que la división entre n e i sea exacta.
5. Programamos...
6. Programamos...
7. Programamos...

Búsqueda Exhaustiva

Ejercicio 1

Escriba un algoritmo de Búsqueda Exhaustiva que ordene una lista de números.

Búsqueda Exhaustiva

Ejercicio 1

1. El conjunto de candidatos es ...

Búsqueda Exhaustiva

Ejercicio 1

1. El conjunto de candidatos es ... el conjunto de todas las posibles permutaciones de la lista.

Búsqueda Exhaustiva

Ejercicio 1

1. El conjunto de candidatos es ... el conjunto de todas las posibles permutaciones de la lista.
2. Cada candidato puede representarse como...

Búsqueda Exhaustiva

Ejercicio 1

1. El conjunto de candidatos es ... el conjunto de todas las posibles permutaciones de la lista.
2. Cada candidato puede representarse como... una lista!

Búsqueda Exhaustiva

Ejercicio 1

1. El conjunto de candidatos es ... el conjunto de todas las posibles permutaciones de la lista.
2. Cada candidato puede representarse como... una lista!
3. ¿Como recorremos todas las posibles permutaciones de una lista?

Búsqueda Exhaustiva

Ejercicio 1

1. El conjunto de candidatos es ... el conjunto de todas las posibles permutaciones de la lista.
2. Cada candidato puede representarse como... una lista!
3. ¿Como recorremos todas las posibles permutaciones de una lista? ...
4. ¿Cuando podemos parar de buscar?

Búsqueda Exhaustiva

Ejercicio 1

1. El conjunto de candidatos es ... el conjunto de todas las posibles permutaciones de la lista.
2. Cada candidato puede representarse como... una lista!
3. ¿Como recorremos todas las posibles permutaciones de una lista? ...
4. ¿Cuando podemos parar de buscar? Cuando demos con una lista que esta ordenada!

Búsqueda Exhaustiva

Ejercicio 1

1. El conjunto de candidatos es ... el conjunto de todas las posibles permutaciones de la lista.
2. Cada candidato puede representarse como... una lista!
3. ¿Como recorremos todas las posibles permutaciones de una lista? ...
4. ¿Cuando podemos parar de buscar? Cuando demos con una lista que esta ordenada!
5. Programar...

Búsqueda Exhaustiva

Adaptación para que el algoritmo de Búsqueda Exhaustiva encuentre todas las soluciones

Muchas veces encontramos problemas que admiten varias soluciones y estamos interesados en encontrar **todas** las soluciones del problema.

Es muy fácil adaptar el algoritmo de búsqueda exhaustiva para que encuentre todas las soluciones posibles a un problema dado - simplemente las vamos acumulando en una lista.

```
soluciones = []
for candidato in candidatos():
    if es_solucion(candidato):
        soluciones.append(candidato)

print(soluciones)
```

Búsqueda exhaustiva

Adaptación para que el algoritmo de Búsqueda Exhaustiva encuentre la mejor solución

Muchos problemas admiten varias soluciones. Sin embargo, muchas veces cada uno de estas posibles soluciones tiene un costo asociado - no da lo mismo elegir cualquiera.

En dichos casos, se puede adaptar el algoritmo de búsqueda exhaustiva para encontrar la **mejor** solución posible.

Búsqueda exhaustiva

Adaptación para que el algoritmo de Búsqueda Exhaustiva encuentre la mejor solución

Muchos problemas admiten varias soluciones. Sin embargo, muchas veces cada uno de estas posibles soluciones tiene un costo asociado - no da lo mismo elegir cualquiera.

En dichos casos, se puede adaptar el algoritmo de búsqueda exhaustiva para encontrar la **mejor** solución posible. Lo que hacemos es lo siguiente:

Búsqueda exhaustiva

Adaptación para que el algoritmo de Búsqueda Exhaustiva encuentre la mejor solución

Muchos problemas admiten varias soluciones. Sin embargo, muchas veces cada uno de estas posibles soluciones tiene un costo asociado - no da lo mismo elegir cualquiera.

En dichos casos, se puede adaptar el algoritmo de búsqueda exhaustiva para encontrar la **mejor** solución posible. Lo que hacemos es lo siguiente:

1. Agregamos una función `calcular_costo` que dado un candidato a solución, nos devuelva un número que cuantifique que tan buena es nuestra solución de alguna manera.

Búsqueda exhaustiva

Adaptación para que el algoritmo de Búsqueda Exhaustiva encuentre la mejor solución

Muchos problemas admiten varias soluciones. Sin embargo, muchas veces cada uno de estas posibles soluciones tiene un costo asociado - no da lo mismo elegir cualquiera.

En dichos casos, se puede adaptar el algoritmo de búsqueda exhaustiva para encontrar la **mejor** solución posible. Lo que hacemos es lo siguiente:

1. Agregamos una función `calcular_costo` que dado un candidato a solución, nos devuelva un número que cuantifique que tan buena es nuestra solución de alguna manera.
2. Cuando iteramos por los candidatos a solución, lo hacemos de forma que, si tenemos solución, calculamos el costo de la solución y comparamos con la mejor solución encontrada al momento.

Búsqueda exhaustiva

```
mejor_solucion = None
costo_mejor_solucion = float('inf')
for candidato in candidatos():
    if es_solucion(candidato):
        costo = medir(candidato)
        if costo < costo_mejor_solucion:
          mejor_solucion = candidato
          costo_mejor_solucion = costo

print(mejor_solucion)
```

El módulo `itertools`

Python trae incluido un módulo llamado `itertools`, el cual estandariza una serie de herramientas que se pueden utilizar para escribir generadores de forma más sencilla.

Tarea Investigar la documentación oficial de este modulo: https://docs.python.org/3/library/itertools.html