Algoritmos Voraces

GREEDY ALGORITHMS

Algoritmos Voraces

- Los algoritmos voraces encuentran la solución del problema evaluando cada uno de los elementos de un conjunto que llamaremos “conjunto de candidatos”.
- El algoritmo consiste en una serie de iteraciones, en cada una de las cuales se evalúa un elemento del conjunto de candidatos para determinar si corresponde incluirlo en el conjunto solución o no.
- Luego, aquellos elementos no incluidos serán descartados definitivamente.
- Un algoritmo voraz no reconsidera una opción. Simplemente la toma o deja.

- En este tipo de algoritmos podemos identificar cuatro funciones:
  - La **función de selección** se ocupa de seleccionar el elemento mas apropiado del conjunto de candidatos.
  - La **función de factibilidad** que, una vez seleccionado un elemento, evalúa si es factible incluirlo en el conjunto solución.
  - La **función solución** evalúa si el conjunto solución efectivamente constituye una solución para el problema.
  - La **función objetivo** que busca optimizar la solución encontrada.

Características generales de los algoritmos voraces

- Tenemos que resolver algún problema de forma óptima.
- Para construir la solución de nuestro problema, disponemos de un conjunto (o una lista) de candidatos:
  - Las monedas disponibles.
  - Las aristas de un grafo que se pueden utilizar para construir una ruta.
  - El conjunto de tareas que hay que planificar.
  - Etc.

- A medida que avanza el algoritmo, vamos acumulando dos conjuntos.
  - Uno contiene candidatos que ya han sido considerados y seleccionados
  - Mientras que el otro contiene candidatos que han sido considerados y rechazados.
- Existe una función que comprueba si un cierto conjunto de candidatos constituye una ***solución*** de nuestro problema, ignorando si es o no óptima por el momento.
- Hay una segunda función que comprueba si un cierto conjunto de candidatos es ***factible***, esto es, si es posible o no completar el conjunto añadiendo otros candidatos para obtener al menos una solución de nuestro problema.

- Una vez más, no nos preocupa aquí si esto es óptimo o no.
- Normalmente, se espera que el problema tenga al menos una solución que sea posible obtener empleando candidatos del conjunto que estaba disponible inicialmente.
- Hay otra función más, la ***función de selección***, que indica en cualquier momento cuál es el más prometedor de los candidatos restantes, que no han sido seleccionados, ni rechazados.
- Por último, existe una ***función objetivo*** que da el valor de la solución que hemos hallado.
- A diferencia de las anteriores, la función objetiva no aparece explícitamente en el algoritmo voraz.

- Para resolver nuestro problema, buscamos un conjunto de candidatos que constituya una solución, y que optimice (minimice o maximice, según los casos) el valor de la *función objetivo*.
- Los algoritmos voraces avanzan paso a paso.
- Inicialmente, el conjunto de elementos seleccionados esta vacío.
- Entonces, en cada paso se considera añadir a este conjunto el mejor candidatos sin considerar los restantes, estando guiada nuestra elección por la función de selección.
- Si el conjunto ampliado de candidatos seleccionados ya no fuera factible, rechazamos el candidato que estamos considerando en ese momento.

- Sin embargo, si el conjunto aumentado sigue siendo factible, entonces añadimos el candidato actual al conjunto de candidatos seleccionados, en donde pasará a estar desde ahora en adelante.

- Cada vez que se amplia el conjunto de candidatos seleccionados, comprobamos si éste constituye ahora una solución para nuestro problema.

- Cuando el algoritmo voraz funciona correctamente, la primera solución que se encuentre de esta manera es siempre óptima.

Función voraz (C: conjunto) : devuelve conjunto

{C es el conjunto de todos los candidatos}
S ← ∅ {S es el conjunto en el que se construye la solución}

Mientras C ≠ ∅ **y no** solución(s) **hacer**
    x ← seleccionar(C)
    C ← C - {x}
    si factible(S ∪ {x}) **entonces** S ← S ∪ {x}

si solución(S) **entonces** devolver S
si no devolver "No hay soluciones"

- Esta clara la razón por la cual tales algoritmos se denominan “voraces”.
- En cada paso, el procedimiento selecciona el mejor bocado que pueda tragar, sin preocuparse por el futuro.
- Nunca cambia de opinión: una vez que un candidato se ha incluido en la solución, queda allí para siempre; una vez que se excluye un candidato de la solución, nunca vuelve a ser considerado.
- La función selección suele estar relacionada con la función objetivo.
- Por ejemplo, si estamos intentando maximizar nuestros beneficios, es probable que seleccionemos aquel candidato restante que posea un mayor valor individual. Si intentamos minimizar el coste, entonces quizá seleccionemos el más barato de los candidatos disponibles.

Cambio de monedas

Cambio de monedas

- Supongamos que vivimos en un país en el que están disponibles las siguientes monedas:
  - $ 100
  - $ 25
  - $ 10
  - $ 5
  - $ 1
- Nuestro problema consiste en diseñar un algoritmo para pagar una cierta cantidad a un cliente, utilizando el menor número posible de monedas.

- Por ejemplo, si tenemos que pagar 289, la mejor solución consiste en dar al cliente 10 monedas.
  - 2 de $100 3 de $25 1 de $10 4 de $1

- Casi todos nosotros resolvemos este tipo de problema todos los días sin pensarlo dos veces, empleando de forma inconsciente un algoritmo evidentemente voraz.

- Empezamos por nada, y en cada fase vamos añadiendo a las monedas que ya estén seleccionadas una moneda de la mayor denominación posible, pero que no debe de llevarnos más allá de la cantidad que haya que pagar.

- El algoritmo se puede formalizar de la manera siguiente:

Función devolver cambio (n) : conjunto de monedas
{Da el cambio de n unidades utilizando el menor número posible de monedas.}
{La constante C especifica las monedas disponibles}
const C = {100, 25, 10, 5, 1}
S ← ∅ {S es un conjunto que contendrá la solución}
s ← 0 {s es la suma de los elementos de S}
Mientras s ≠ n **hacer**
x ← el mayor elemento de C tal que s + x ≤ n
si no existe ese elemento **entonces**
devolver "No encuentro la solución"
S ← S ∪ {Una moneda de valor x}
s ← s + x
devolver S

- Es fácil convencerse (pero sorprendentemente difícil de probar formalmente) de que con los valores dados para las monedas, y disponiendo de un suministro adecuado de cada denominación, este algoritmo siempre produce una solución óptima para nuestro problema.

- Sin embargo, una serie de valores diferente, o si el suministro de alguna de las monedas esta limitado, el algoritmo voraz puede no funcionar.

- En algunos casos, puede seleccionar un conjunto de monedas que no sea óptimo (esto es, el conjunto contiene más monedas que las necesarias), mientras que quizá en otros casos no llegue a encontrar ninguna solución aún cuando ésta exista. (Aunque esto no puede suceder si disponemos de un suministro ilimitado de monedas de una unidad).

- El algoritmo es “voraz” porque en cada paso selecciona la mayor de las monedas que pueda encontrar, sin preocuparse por lo correcto de esta decisión a la larga.

- Además nunca cambia de opinión: una vez que una moneda se ha incluido en la solución, la moneda queda allí para siempre.

- De acuerdo a las características generales de los algoritmos voraces.
  - Los **candidatos** son un conjunto de monedas (100, 25, 10, 5, 1)
  - Con tantas monedas de cada valor que nunca las agotamos.
  - Sin embargo, el conjunto de candidatos debe de ser finito.
  - La **función solución** comprueba si el valor de las monedas seleccionadas hasta el momento es exactamente el valor que hay que pagar.
  - Un conjunto de monedas será **factible** si su valor total no sobrepasa la cantidad que haya que pagar.
  - La **función de selección** toma la moneda de valor más alto que quede en el conjunto de candidatos.
  - La **función objetivo** cuenta el número de monedas utilizadas en la solución.