Greedy algorithm

(Algoritmo voraz)

y

Divide y vencerás

**Integrantes:**

Juan Manuel Garcia

Mauro Szuchman

Ariel Gerardo Ríos

David Tagliaferr

Ramiro Machado

Laureano Gabriel Clausi

# Indice

[Greedy algorithm (algoritmo voraz)](#h.whp2tbnuvx3j)

[Características y propiedades de los problemas solucionados mediante esta estrategia](#h.dstuaayestzh)

[Estructura de un algoritmo Greedy](#h.lep7etiwit1y)

[Definición de feasibilidad](#h.hvav2ofc3i0r)

[Ejemplo práctico](#h.8lm0bi824ft5)

[Otros ejemplos](#h.18zl609b0p45)

[Aplicación en la práctica](#h.gm4unx6b1uf3)

[Divide y vencerás](#h.mneami25wuem)

[Problema ejemplo:](#h.ecup6ymer5v3)

[Bibliografía](#h.k9oks76q717r)

# Greedy algorithm (algoritmo voraz)

Un algoritmo voraz, al cual también se lo refiere como ávido, devorador o goloso, es aquel que, para resolver un problema en particular, sigue una heurística (método para encontrar una solución aproximada) consistente en elegir la opción óptima en cada paso. De esta manera se intenta, aunque no siempre se logra, llegar a una solución general óptima.

De este esquema se desprende que los algoritmos ávidos son muy fáciles de

implementar y producen soluciones muy eficientes.

Aunque pueda pensarse que este tipo de algoritmos pueden usarse en cualquier situación, no todos los problemas admiten esta estrategia de solución. De hecho, la búsqueda de óptimos locales no tiene por qué conducir siempre a un óptimo global. La estrategia de los algoritmos ávidos consiste en tratar de ganar todas las batallas sin pensar que, como bien saben los estrategas militares y los jugadores de ajedrez, para ganar la guerra muchas veces es necesario perder alguna batalla.

Además, este algoritmo no tiene en consideración posibles consecuencias de las decisiones que toma en cada paso y, al seguir una heurística, el mismo no asegura que el problema se resuelva de la manera más óptima.

Vale aclarar que una de las características fundamentales de las heurísticas es resignar precisión, optimalidad o exhaustividad en pos de ganar velocidad. En cierta forma las heurísticas pueden ser consideradas un “shortcut” hacia la búsqueda de una solución.

# 

## Características y propiedades de los problemas solucionados mediante esta estrategia

Para construir una solución de forma óptima, el algoritmo mantiene 2 conjuntos: uno de elementos elegidos y otro de rechazados.

El algoritmo voraz poses 4 funciones componentes:

1. Una función que verifica si un conjunto de elementos elegidos proveen una solución.
2. Una función que verifica si un conjunto es feasible.
3. Una función de selección la cual indica cuál de los candidatos es el más prometedor.
4. Una función objetivo (la que por lo general no aparece explicitamente) que da el valor de una solución.

## Estructura de un algoritmo Greedy

* Inicialmente el conjunto de elementos elegidos (el conjunto solución) está vacío.
* En casa paso
* Un ítem será agregado en un conjunto solución usando una función de selección.
* Si el conjunto ya no es más feasible:
  + Descartar los elementos considerados (y no se considerarán más).
* Si el conjunto es todavía feasible entonces:
  + agregar el elemento actual.

#### 

## Definición de feasibilidad

Un conjunto feasible (de candidatos) es prometedor si puede ser extendido para producir no sólo una solución, sino una solución óptima al problema; en particular, el conjunto vacío es siempre prometedor porque una solución óptima siempre existe.

De manera distinta a la programación dinámica, la cual resuelve los subproblemas con un enfoque de abajo hacia arriba (bottom-up), una estrategia greedy usualmente progresa mediante el enfoque arriba hacia abajo (top-down), haciendo una elección luego de otra, reduciendo cada problema a uno más pequeño.

## Ejemplo práctico

Como ejemplo práctico, y clásico, de un algoritmo voraz, se puede plantear el problema de dar cambio de billetes y monedas:

Supongamos que tenemos entregar cambio a una persona que tiene $6,35.

En este caso la definición del algoritmo voraz para resolver este problemas sería: “En cada paso encontrar el billete (o moneda) más grande que, sumado a lo anteriormente acumulado, no sobrepase el valor total ($6.35)”

Con lo cual, dicho algoritmo ejecutará los siguientes pasos:

- Tomar un billete de $5

- Tomar una moneda de $1

- Tomar una moneda de $0.25

- Tomar una moneda de $0.10

Como se puede observar, en este caso en particular (sistema monetario argentino) el algoritmo encontró la solución óptima.

Sin embargo, pueden existir casos en los cuales la misma no sea hallada:

Supongamos que ahora tenemos la restricción de no usar monedas de $1 ni de $0.50.

En este caso el algoritmo mencionado anteriormente ejecutará:

- Tomar un billete de $5

- Tomar 5 monedas de $0.25

- Tomar una moneda de $0.10

Total: 1 billete y 6 monedas = 7

Cuando lo ideal hubiese sido:

- Tomar 3 billetes de $2

- Tomar 1 moneda de $0.25

- Tomar una moneda de $0.10

Total: 3 billetes y 2 monedas = 5

Nótese que el error cometido por el algoritmo voraz es no tomar en cuenta que la decisión de seleccionar el billete de $5 en primera instancia no es la mejor. De allí se deduce que encontrar la mejor solución para cada etapa no siempre determina encontrar la mejor solución global.

Para lograr soluciones más eficientes a este tipo de problemas se puede emplear un técnica más compleja denominada programación dinámica.

## Otros ejemplos

Algunos casos particulares de algoritmos voraces conocidos son:

- Algoritmo de Kruskal

- Algoritmo de Prim

- Algoritmo de Dijkstra

## Aplicación en la práctica

Los algoritmos voraces son utilizados generalmente en problemas de optimización o para resolver problemas cuya solución óptima es imposible o muy complicada de lograr.

Por ejemplo en un programa de Inteligencia de artificial para jugar al ajedrez, donde la estrategia global es inmensamente compleja debido a la cantidad de posibilidades de juego e imposible de alcanzar para las computadoras de hoy en día.

# Divide y vencerás

En las [ciencias de la computación](http://es.wikipedia.org/wiki/Ciencias_de_la_computaci%C3%B3n), el término divide y vencerás (DYV) hace referencia a uno de los más importantes paradigmas de diseño [algorítmico](http://es.wikipedia.org/wiki/Algoritmo). El método está basado en la resolución [recursiva](http://es.wikipedia.org/wiki/Recursividad) de un problema dividiéndolo en dos o más subproblemas de igual tipo o similar. El proceso continúa hasta que éstos llegan a ser lo suficientemente sencillos como para que se resuelvan directamente. Al final, las soluciones a cada uno de los subproblemas se combinan para dar una solución al problema original.

Esta técnica es la base de los algoritmos eficientes para casi cualquier tipo de problema como, por ejemplo, [algoritmos de ordenamiento](http://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento) ([quicksort](http://es.wikipedia.org/wiki/Quicksort), [mergesort](http://es.wikipedia.org/wiki/Mergesort), entre muchos otros), [multiplicar números grandes](http://es.wikipedia.org/wiki/Algoritmo_de_multiplicaci%C3%B3n) ([Karatsuba](http://es.wikipedia.org/wiki/Algoritmo_de_Karatsuba)), [análisis sintácticos](http://es.wikipedia.org/wiki/Analizador_sint%C3%A1ctico) ([análisis sintáctico top-down](http://es.wikipedia.org/w/index.php?title=An%C3%A1lisis_sint%C3%A1ctico_top-down&action=edit&redlink=1)) y la [transformada discreta de Fourier](http://es.wikipedia.org/wiki/Transformada_de_Fourier).

Al igual que en la inducción, a veces es necesario sustituir el problema original por uno más complejo para conseguir realizar la recursión, y no hay un método sistemático de generalización.

El nombre divide y vencerás también se aplica a veces a algoritmos que reducen cada problema a un único subproblema, como la [búsqueda binaria](http://es.wikipedia.org/wiki/B%C3%BAsqueda_binaria) para encontrar un elemento en una lista ordena. Estos algoritmos pueden ser implementados más eficientemente que los algoritmos generales de “divide y vencerás”; en particular, si es usando una serie de recursiones que lo convierten en simples bucles. Bajo esta amplia definición, sin embargo, cada algoritmo que usa recursión o bucles puede ser tomado como un algoritmo de “divide y vencerás”. El nombre decrementa y vencerás ha sido propuesta para la subclase simple de problemas.

La corrección de un algoritmo de “divide y vencerás”, está habitualmente probada una inducción matemática, y su coste computacional se determina resolviendo relaciones de recurrencia.

### 

### Problema ejemplo:

Como ejemplo sencillo de aplicación de esta estrategia puede considerarse la búsqueda de una palabra en un diccionario de acuerdo con el siguiente criterio.

Se abre el diccionario por la pagina centrar (quedando dividido en dos mitades) y se comprueba si la palabra aparece allí o si es anterior o posterior. Si no ha encontrado y es anterior se procede a buscarla en la primera mitad, si es posterior, se buscara en la segunda mitad. El procedimiento se repite sucesivamente hasta encontrar la palabra o decidir que no aparece.

# Bibliografía

<http://es.wikipedia.org/wiki/Algoritmo_divide_y_vencer%C3%A1s#Dise.C3.B1o_e_implementaci.C3.B3n>

<http://www.lcc.uma.es/~av/Libro/CAP3.pdf>

<http://www.lcc.uma.es/~av/Libro/CAP4.pdf>

<http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Greedy/greedyIntro.htm>

<http://es.wikipedia.org/wiki/Algoritmo_divide_y_vencer%C3%A1s#Dise.C3.B1o_e_implementaci.C3.B3n>

[http://www.monografias.com/trabajos11/alcom/alcom.shtml#ixzz35KgFShG](http://www.monografias.com/trabajos11/alcom/alcom.shtml#ixzz35KgFShGB)