**Estrategias o técnicas algorítmicas**

**Fuerza bruta**

* Resuelve un problema siguiendo la estrategia más obvia de solución
* Sencillos de implementar, pero coste bastante alto. Se pueden usar para tallas pequeñas.
* **Ejemplos:**
  + Búsqueda lineal: Recorre la lista completa comparando con el elemento que se busca. **O(n)**
  + Ordenación por inserción: Recorre la lista completa, busca el mínimo y lo coloca al principio. Luego, repite el proceso empezando desde el segundo elemento. **O(n²).**

**Divide y vencerás**

* Consiste en descomponer el problema en varios subproblemas del mismo tipo pero más sencillos, resolverlos y combinar sus soluciones para obtener la solución del problema inicial.
* Es **necesario** que el problema admita una formulación recursiva, y el problema original debe poder resolverse mediante combinaciones de subproblemas, de tamaño estrictamente menor.
* Es **deseable** que el tamaño de los subproblemas sea similar y decreciente con progresión geométrica (n/c con c constante y lo más grande posible), y el coste de descomponer/combinar debe ser poco.
  + También es importante elegir bien el **umbral n0** a partir del cual la función deja de ser recursiva, y se debe evitar resolver el mismo subproblema más de una vez.
  + Por ejemplo, Fibonacci crece de forma aditiva y no geométrica, por lo que es posible resolverlo recursivamente pero no ideal.
* El **coste total** de un algoritmo divide y vencerás, que origina **k** suproblemas de tamaño **n/c** es .
  + Si el tiempo de descomposición td,y combinación tc no son excesivos, se considera que tienen un coste polinómico: .
  + Entonces, la utilización de divide y vencerás no garantiza nada sobre la eficiencia del algoritmo obtenido. En comparación a un algoritmo iterativo, puede ser peor o mejor.
    - Si k<ci, t(n) ∈ O(ni)
    - Si k=ci, t(n) ∈ O(ni logn)
    - Si k>ci, t(n) ∈ O(nlogc k)

**Ejemplos de divide y vencerás**

* **Búsqueda binaria:** Permite ordenar una lista (palabras o números). Se procede comparando el elemento que se quiere localizar con el central en la lista, para dividirla en dos sublistas.
  + Para buscar x en una lista [a0, a1, a2… an], comparamos x con am, donde m = ⌊(n/2)⌋
  + Si x > am, la búsqueda se restringe a la 2ª mitad. Si x≤ am, se restringe a la 1ª mitad, donde se incluye am.
  + El **coste total** es de t(n) = t(n/2) + O(1). Resolviendo la ecuación recursiva tenemos t(n) = 1 + log2n \* O(1) → **t(n)∈O(log n)** (más eficiente que b. lineal)
* **Multiplicación de enteros:** Supongamos que queremos multiplicar dos enteros x e y de n dígitos, expresados en una base i. La solución tradicional es **O(n2).**
  + Dividimos los enteros en dos partes: x = a\*in/2 + b, y = cin/2+d
  + Se comprueba que **xy = ac in + (ad+bc) in/2 + bd**. De esta forma se puede calcular xy mediante 4 productos de tamaño n/2. Sin embargo, esta solución sigue siendo **O(n2).**
* La solución es el algoritmo de **Karatsuba y Ofman**, basándose en la propiedad **xy = ac in + (ac + bd + (a-b)(d-c)) in/2 + bd.**
  + De esta forma, se realizan 3 productos de tamaño n/2. La complejidad es **O(n1.57)**

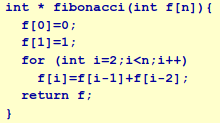
**Técnica voraz**

* Consiste en fijar, en cada paso, el valor de una de las variables mediante una función de selección que siempre toma la decisión **localmente óptima**.
* Un sistema voraz nunca reconsidera una decisión tomada.
* Es **necesario** que exista una **función objetivo** que sea la que deseamos maximizar/minimizar.
  + Debe ser conocido el dominio de la función objetivo.
  + Debe existir también una **función solución** que compruebe si unos determinados valores son una solución del problema.
  + Debe existir una función denominada **factible**, que compruebe si las decisiones tomadas hasta el momento satisfacen las restricciones.
* El **coste** depende del número de iteraciones del bucle y el coste de las funciones seleccion y factible.
  + Suelen ser eficientes, pero no garantizan que se obtengan una solución óptima.
* Los problemas resolubles mediante algoritmo voraz deben cumplir el **principio de optimalidad:** para una secuencia óptima de decisiones, toda subsecuencia ha de ser también óptima.

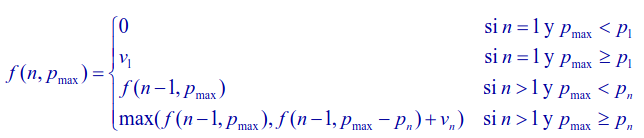
**Ejemplo: Cambio de monedas (Voraz)**

* Forma de obtener una cantidad de x centimos con monedas {e1, …, ek}
  + **Función objetivo:** nº de monedas, que hay que minimizar
  + **Dominio:** Conjunto C de todas las monedas
  + **Solución:** S = {e1, …, ek} es solución si su suma es x
  + **Factible**: S = {e1, …, ek} es solución si su suma es <=x
  + **Función selección:** Elige la moneda de mayor valor que, al sumarla a la solución en curso, no supera x.
* No devuelve siempre una solución óptima, dependiendo del conjunto de monedas que tengamos.
  + Para que el algoritmo alcance la solución óptima, C debe estar formado por tipos de moneda que sean potencia de un tipo básico t. Ej: C={125,25,5,1}
* **Coste: O(max(nlog n, m))** donde n es el nº de tipos distintos de monedas y m es el nº de iteraciones que realiza el bucle

**Programación dinámica**

* Similar a divide y vencerás, pero los subproblemas idénticos se resuelven solo una vez, por lo que se guarda su solución para que, si vuelve a surgir, se devuelve de inmediato.
* Sacrifica espacio para ganar tiempo de ejecución.
* Es una técnica **ascendiente:** requiere resolver primero los suproblemas más pequeños.

**Ejemplo: problema de la mochila**

* Se dispone de n objetos o1, …, on, cada uno con un peso pi y un valor vi asociados.
* Se deben guardar en una mochila que soporta un peso pmax, maximizando el valor.Existen dos variantes del problema:
* **Mochila fraccionada:** Se pueden guardar partes de objetos. Para cada objeto, xi es la fracción que guardamos en la mochila
  + **Objetivo:** maximizar Σi xi vi
  + **Solución:** S es solución si Σi xi pi = pmax
  + **Factible:** S es factible si Σi xi pi <= pmax
  + **Selección:** Seleccionar los objetos por orden decreciente de relación valor/peso.
  + **Coste:** La parte más costosa del algoritmo es la ordenación de los elementos según su relacion valor/peso. El coste será igual al del algoritmo de ordenación usado, en el mejor caso, **O(n log n)**
* **Mochila entera:** Los objetos no se pueden dividir. El algoritmo voraz no devuelve una solución óptima, pero la programación dinámica si.
  + Llamamos f(n, pmax) al valor de la solución óptima. Se tiene: 

(si queda un objeto y no cabe)

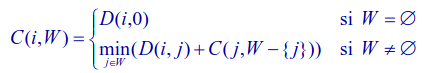
(si queda un objeto y cabe)

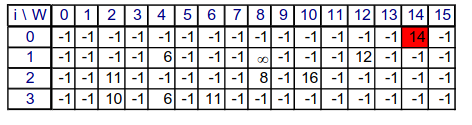
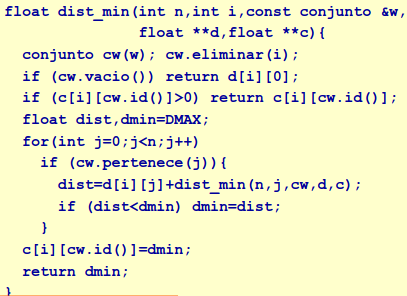
(queda más de un objeto, y el siguiente no cabe)

(queda más de un objeto y el siguiente cabe: la función se divide en dos posibilidades, no incluir el objeto o incluirlo. de estas opciones se selecciona la que dea un valor mayor)

* + De esta forma, la función crea un árbol de posibilidades y algunos de los suproblemas se repetirán, de forma similar a fibonacci.
  + **Coste:** O(n\*pmax). Si pmax es muy grande y mayor que n, sería un coste cuadrático o superior.

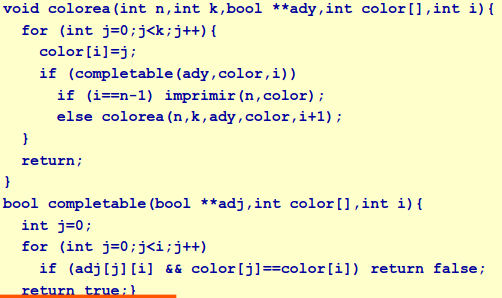
**Problema del viajante**

* Un viajante tiene que recorrer n ciudades y volver al inicio sin pasar dos veces por la misma ciudad. Encontrar el camino mínimo.
* Se representa cada ciudad como un nodo de un grafo, creando un grafo dirigido. Sea 0 el punto de inicio
* Para dos puntos, sea D(i,j) la longitud del camino entre i y j. Siendo W un subconjunto de los nodos, y C(i,W) la longitud del camino mínimo de i a 0 que pasa por todos los vértices de W. Entonces:
* La solución al problema será C(0, V-{0}). Se cumple el principio de optimalidad.
* Se puede resolver mediante **programación dinámica:** guardamos los valores de C(i,W) en una tabla para evitar repetir su cálculo.
  + **Coste: O(n22n)**, mejor que n! en fuerza bruta



**Vuelta atrás**

* Método de algoritmo voraz pero que permite reconsiderar decisiones tomadas si la solución alcanzada no es óptima.
* Se dice que una solución en curso es **completable** si a partir de ella se puede alcanzar la solución óptima a partir de ella. SI se llega a una solución no completable, se da marcha atrás y se reconsideran las decisiones tomadas.
  + Si se alcanza una solución no completable, no se construye el subárbol correspondiente. Esto se conoce como **poda** del espacio e búsqueda.
* Tipos de algoritmo:
  + Vuelta atrás para una solución: Recorre todo el espacio de búsqueda hasta encontrar la primera solución
  + Vuelta atrás para todas las soluciones: Recorre el espacio de búsqueda guardando todas las soluciones
  + Vuelta atrás para la mejor solución: Recorre el espacio de búsqueda comparando cada solución con la previa mejor, y quedándose con la mejor.
* Coste:
  + Depende del nº de nodos del espacio d búsqueda que se visitan, v(n), y del coste de las funciones solución y completable, p(n). **O(p(n)v(n))**
  + El tamaño de v(n) dependerá de cuántos nodos e poden. Puede ser entre **O(p(n)n)** y **O(p(n)kn)**
* Ejemplo: Coloreado de Grafos con Vuelta Atrás



**Ramificación y Poda**

* La técnica de vuelta atrás normal realiza un recorrido ciego, tras cada nodo se avanza al siguiente.
* La técnica de ramificación y poda avanza de cada nodo al próximo más **prometedor** en base a la información de que se disponga. Se generan primero todos los hijos de un nodo y después se avanza al más prometedor de ellos
  + Diremos que un nodo es prometedor si expandiéndolo se puede conseguir una solución mejor que la actual.
* El coste sigue siendo**O(p(n)v(n)).** En este caso, v(n) suele ser dn con d=max|Ck|
  + Ck es el conjunto de hijos en la última capa del árbol (?)