Técnicas de diseño de algoritmos

El diseño de un algoritmo es una labor creativa. Existen diversos patrones de diseño en los que el programador se puede basar para resolver ciertos problemas. Las técnicas más conocidas son:

- Divide y vencerás.
- Algoritmos ávidos o voraces.
- Programación dinámica.
- Algoritmos de búsqueda con retroceso (*backtracking*).

Divide y vencerás

Es una técnica de diseño de algoritmos que consiste en resolver un problema a partir de la solución de subproblemas del mismo tipo, pero de menor tamaño. Posteriormente, se combinan los resultados parciales para obtener la solución del problema original. Para aplicar esta técnica se requiere que los subproblemas sean del mismo tipo que el problema original.

Pasos

1. Plantear el problema de forma que pueda descomponerse en *k* subproblemas del mismo tipo, pero de menor tamaño.
2. Resolver independientemente todos los subproblemas (ya sea directamente, si son elementales, o bien de forma recursiva).
3. Combinar las soluciones obtenidas en el paso anterior para construir la solución del problema original.

```
Función Dividir_y_Conquistar (Problema x)
    Si x es sencillo o conocido entonces
        regresar (solución(x))
    Sino
```

```
        Descomponer x en problemas más pequeños x1,x2,... xn
    Para i desde 1 hasta n hacer
            yi ← Dividir_y_Conquistar(xi)
        Combinar los yi para obtener una solución y de x
    Regresar y
Fin si
Fin
```

Algoritmos voraces

Los algoritmos *voraces*, también llamados *ávidos* o *glotones*, se aplican principalmente a problemas de optimización; es decir, problemas en los que hay que maximizar o minimizar algo. Estos algoritmos toman decisiones basándose en la información que tienen disponible de forma inmediata, sin tomar en cuenta los efectos que pudieran tener estas decisiones en el futuro.

Los algoritmos voraces intentan optimizar una *función objetivo*. La *función objetivo* proporciona el valor de una solución al problema, por ejemplo: la cantidad de monedas que se utilizarán para dar cambio, la longitud de la ruta construida, el tiempo necesario para procesar todas las tareas de la planificación.

1. En cada paso se considera al candidato indicado por una *función de selección voraz*, que indica cuál es el candidato más prometedor de entre los que aún no se han considerado (es decir, los que aún no han sido seleccionados ni rechazados).
2. Conforme avanza el algoritmo, los elementos del conjunto inicial de candidatos pasan a formar parte de uno de dos conjuntos: el de los candidatos que ya se consideraron y que forman parte de la solución, o el conjunto de los candidatos que ya se consideraron y se rechazaron.
3. Mediante una *función de factibilidad* se determina si es posible o no completar un conjunto de candidatos que constituya una solución al problema. Es decir, se analizan si existe un candidato compatible con la solución parcial construida hasta el momento.

Pseudocódigo

```
funcion voraz(C:conjunto, S:conjunto solución)
        { C es el conjunto de todos los candidatos }
  S <= vacio    { S es el conjunto en el que se construye la solución}
  mientras No solución(S) y C <> vacio hacer
      x=seleccionar(C)
      C = C - { x}
      si es solución(S U {x})
          entonces S = S U {x}
  findelmientras
  si solución(S)
     entonces devolver solución
    si no devolver no hay solución
```