

Técnicas de diseño de algoritmos

Programación dinámica

Dividir un problema en subproblemas más sencillos para poder resolverlos y luego combinar sus soluciones para obtener la solución general es una técnica que se usa para resolver problemas, como, por ejemplo, en los recursivos. Pero, a veces, los subproblemas se solapan o se calcula lo mismo en varios subproblemas, restando eficiencia a la ejecución de la solución final. Otras veces, los algoritmos de optimización, como los algoritmos voraces, no logran el resultado final óptimo para algunos conjuntos de datos.

La **programación dinámica** intenta resolver esta falta de eficiencia o eficacia, planteándose como una técnica para evitar calcular lo mismo varias veces, siempre de una forma óptima. Usa una **tabla de resultados intermedios** mientras se resuelven los subproblemas, resultados óptimos que luego se utilizarán para alcanzar la solución óptima final.

Para resolverlo, la programación dinámica se basa en el **principio de optimalidad**, que dice que en una secuencia de decisiones óptimas, toda subsecuencia también es óptima.

Intenta conservar la solución a cada subproblema, que ya se ha resuelto en una tabla (puede ser un arreglo o una matriz), para tomarla cuando ésta se requiera. Utiliza tablas como una estructura auxiliar para evitar el solapamiento, es decir, evitar calcular varias veces un mismo resultado.

Pasos para el diseño del algoritmo

1. Plantear un algoritmo con solución recursiva.
2. Organizar las soluciones parciales en una tabla.
3. Modificar el algoritmo recursivo de manera que, antes de hacer el llamado recursivo, se consulte la tabla y, en caso de hallarse la solución en ésta, ya no se haga el llamado recursivo.

Backtracking

Cuando las técnicas “divide y vencerás”, “algoritmos voraces” y “programación dinámica” no son aplicables para resolver un problema, entonces no queda más remedio que utilizar

la fuerza bruta. A estos algoritmos también se los conoce como **exhaustivos**, y son aquellos que analizan todo el espacio de búsqueda para encontrar una o todas las soluciones, es decir, agotan todas las posibilidades hasta hallar la solución. Estos algoritmos garantizan que pueden encontrar una solución óptima; sin embargo, son bastante inefficientes y, cuando el conjunto de soluciones posibles es muy grande, resultan no viables debido a que su tiempo de ejecución es enorme.

El **backtracking** (búsqueda con retroceso) se aplica a problemas que requieren de una búsqueda exhaustiva dentro del conjunto de todas las soluciones potenciales. Es una técnica de resolución de problemas que realiza una búsqueda exhaustiva, sistemática y organizada sobre el espacio de búsqueda del problema y es aplicable a problemas de optimización, juegos y búsqueda, entre otros.

Características de estos algoritmos

- Funciona en problemas cuyas soluciones se pueden construir por etapas.
- La solución maximiza, minimiza o satisface una **función criterio**.
- Sea S el conjunto de todas las soluciones posibles de un problema, y Si una de éstas soluciones. Una solución parcial se expresa mediante una n-tupla (x_1, x_2, \dots, x_n) . El *backtracking* trabaja tratando de extender una solución parcial continuamente, de tal forma que cada $x_i \in Si$ representa la decisión que se tomó en la i -ésima etapa, dentro de un conjunto finito de alternativas.
- El espacio de posibles soluciones S se estructura como un árbol de exploración, en el que, en cada nivel, se toma la decisión de la etapa correspondiente, y se le llama **nodo estado** a cualquier nodo del árbol de exploración.

Algoritmo general de esta técnica

```
Función Retroceso( etapa )
    Inicializar_Opciones();
    Hacer
        opcion←SeleccionarOpcion
        Si aceptable(opcion) Entonces
            Guardar (opcion)
            Si incompleta(solucion) Entonces
                exito ←Retroceso( siguiente( etapa ) ) // recursión
                Si ( exito = FALSO ) Entonces
                    retirar( opcion )
                Fin Si
                Si no // solución completa u hoja del árbol
                    exito ← VERDADERO
                Fin Si
                Fin Si
        mientras (exito = FALSO ) AND ( opcion != UltimaOpcion)

            regresar exito

    Fin Retroceso
```