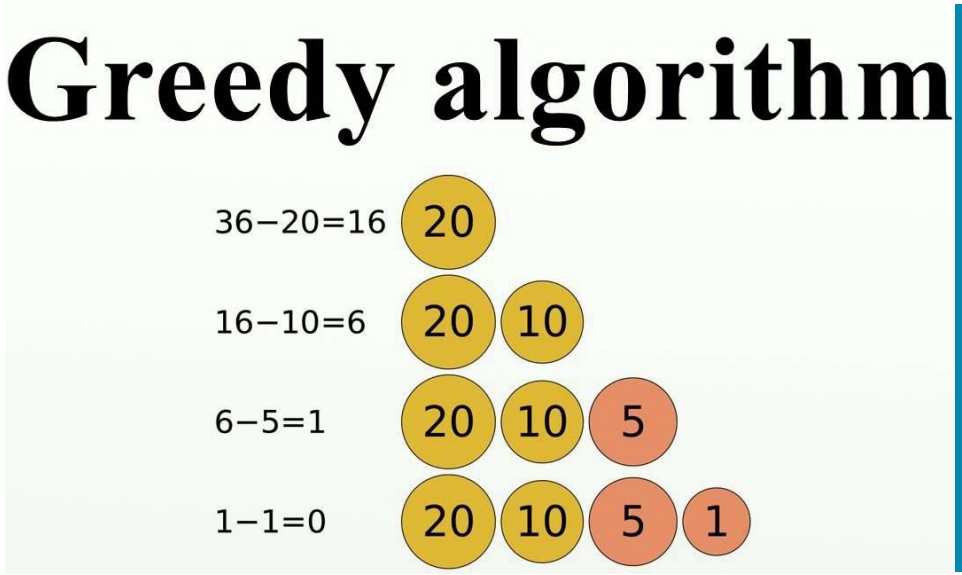


Complejidad Algorítmica

Unidad 2: Algoritmos voraces, programación dinámica y problemas P-NP

Módulo 12: Algoritmos Voraces y Programación Dinámica

Complejidad Algorítmica

Semana 12

Objetivos

- Entender como trabajan los Algoritmos Voraces
- Manejar y desarrollar los principios del paradigma de programación dinámica.

MÓDULO 12: Algoritmos Voraces y Programación Dinámica

Contenido

1. Conceptos básicos – Algoritmos Voraces
2. Ventajas y desventajas del enfoque codicioso
3. El enfoque codicioso en un ejemplo
4. Pasos de un algoritmo voraz
5. Tipos de Algoritmos Voraces
6. Programación Dinámica
7. Diferencia entre Programación Dinámica y Codiciosa

Preguntas / Conclusiones

1. Conceptos básicos – Algoritmos Voraces

¿Por qué un **algoritmo** se considera **voraz o codicioso**?

Algoritmo: secuencia de pasos que conducen a la solución de un problema.

Voraz o codicioso: un enfoque que para resolver un problema selecciona la mejor opción disponible en ese momento.

Características de los Algoritmos Codiciosos

- No se preocupan si el mejor resultado actual traerá luego el resultado óptimo general.
- Un algoritmo voraz nunca revierte la decisión anterior, incluso si la elección es incorrecta. Funciona en un enfoque de arriba hacia abajo.
- Es posible que este algoritmo no produzca el mejor resultado para todos los problemas. Esto sucede porque siempre busca la mejor opción local para producir el mejor resultado global.

¿Es posible utilizar un Algoritmo Voraz para solucionar cualquier tipo de problema?

1. Conceptos básicos – Algoritmos Voraces

¿Cuándo aplicar un **algoritmo voraz o codicioso** en la solución de un problema?

Podremos determinar si un algoritmo codicioso se puede usar con cualquier problema si el problema tiene las siguientes dos propiedades:

1. **Propiedad de elección codiciosa**

Si se puede encontrar una solución óptima al problema eligiendo la mejor opción en cada paso sin reconsiderar los pasos anteriores una vez elegidos, **entonces** el problema se puede resolver utilizando un enfoque codicioso.

2. **Subestructura óptima**

Si **la solución general óptima del problema corresponde a la solución óptima de sus subproblemas**, **entonces** el problema se puede resolver utilizando un enfoque codicioso.

2. Ventajas y desventajas del enfoque codicioso

VENTAJAS

- El algoritmo es más fácil de describir.
- Este algoritmo puede funcionar mejor que otros algoritmos (pero no en todos los casos).

DESVENTAJAS

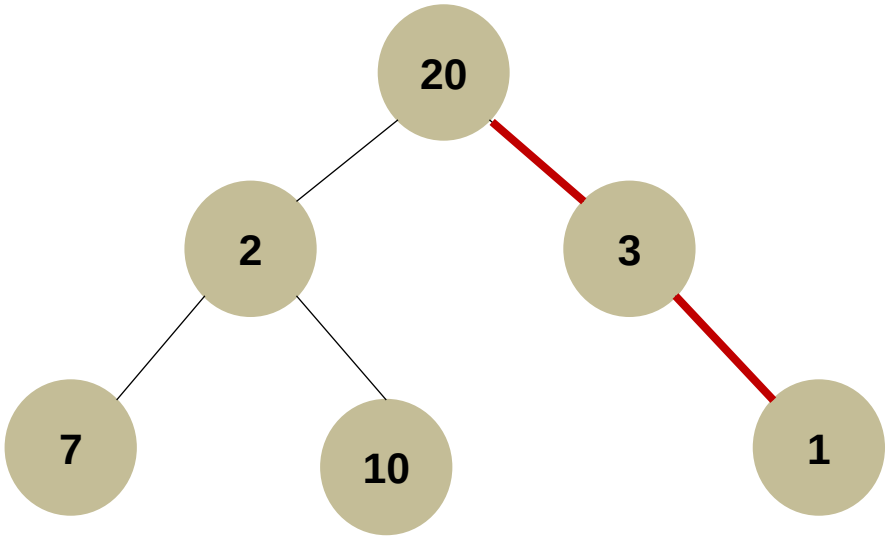
- El algoritmo codicioso no siempre produce la solución óptima. Esta es la principal desventaja del algoritmo.

Veamos el enfoque codicioso en uno ejemplos

3. El enfoque codicioso en un ejemplo

Ejemplo #1

- Aplicar el enfoque codicioso al siguiente árbol desde la raíz hasta la hoja para encontrar la ruta mas larga.



Enfoque codicioso:

- Comencemos con el nodo raíz 20 . El peso del nodo derecho es 3 y el peso del nodo izquierdo es 2.
- Nuestro problema es encontrar el **camino más largo**. Y, la solución óptima en este momento es 3 . Entonces, el algoritmo codicioso elegirá 3 .
- Finalmente el peso del nodo/hijo único de 3 es 1 . Esto nos da un resultado final 20 + 3 + 1 = 24.

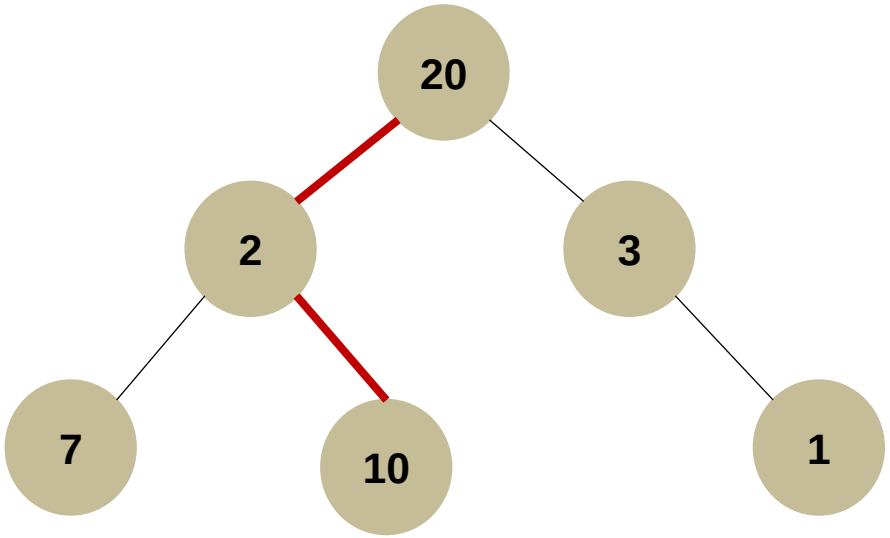
Sin embargo, no es la solución óptima... ¿Verdad?

Hay otro camino que tiene más peso, ¿Cuál es?

3. El enfoque codicioso en un ejemplo

Ejemplo #1

- Aplicar el enfoque codicioso al siguiente árbol desde la raíz hasta la hoja para encontrar la ruta mas larga.



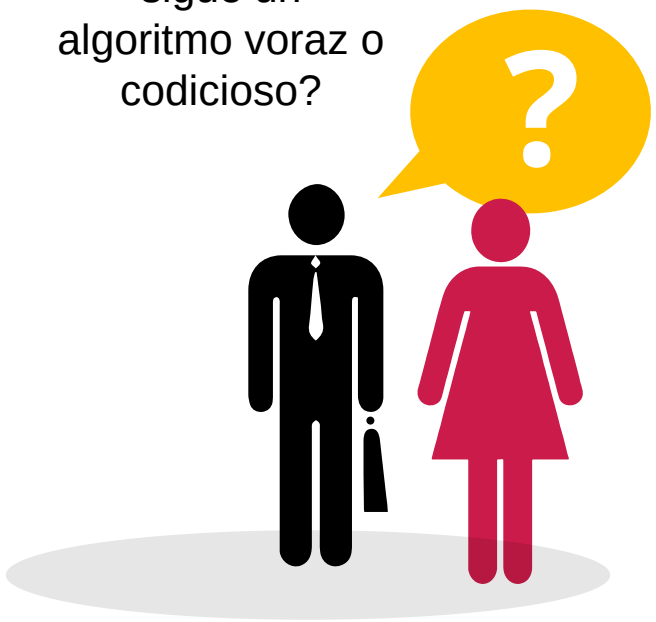
Existe un camino con mayor peso

- 20 + 2 + 10 = 32 como se muestra en la imagen.

Por lo tanto, los algoritmos codiciosos no siempre dan una solución óptima/factible.

4. Pasos de un algoritmo voraz

¿Qué pasos sigue un algoritmo voraz o codicioso?



Algoritmo codicioso

1. Para empezar, el conjunto de soluciones (que contiene las respuestas) está vacío.
2. En cada paso, se agrega un elemento al conjunto de soluciones hasta que se llega a una solución.
3. Si el conjunto de soluciones es factible, se mantiene el elemento actual.
4. De lo contrario, el elemento se rechaza y no se vuelve a considerar nunca más.

Ahora usemos este algoritmo para resolver un problema!

4. Pasos de un algoritmo voraz

PROBLEMA:

- ❑ Hacer un cambio de una cantidad utilizando el menor número de monedas posible.

Monto: S/.18

- ❑ Las monedas disponibles son monedas de:
  - S/. 5
  - S/. 2
  - S/. 1

- ❑ No hay límite para el número de cada moneda que puede usar.

4. Pasos de un algoritmo voraz

SOLUCION:

1. Se crea un conjunto-solución vacío **solution-set = { }**. Y las monedas disponibles son **{5, 2, 1}**.
2. Debemos encontrar el sum = 18. Entonces, comencemos con sum = 0.
3. Se selecciona siempre la moneda con el mayor valor (es decir, 5) hasta cuya suma no sea sum > 18. Cuando seleccionamos el valor más grande en cada paso, esperamos llegar al destino más rápido. Recordemos que este concepto se llama **propiedad de elección codiciosa**.
   - En la primera iteración, **solution-set = {5}** y sum = 5.
   - En la segunda iteración, **solution-set = {5, 5}** y sum = 10.
   - En la tercera iteración, **solution-set = {5, 5, 5}** y sum = 15.
   - En la cuarta iteración, **solution-set = {5, 5, 5, 2}** y sum = 17
     (no podemos seleccionar 5 aquí porque si lo hacemos, sum = 20 y seria mayor que 18, entonces, seleccionamos el segundo elemento más grande que es 2).
   - De manera similar, en la quinta iteración, se selecciona 1.
4. Ahora sum = 18 y **solution-set = {5, 5, 5, 2, 1}**.

5. Tipos de Algoritmos Voraces

¿Qué algoritmos son de tipo voraz o codicioso?

Existen diferentes tipos de algoritmos codiciosos. Entre ellos se pueden señalar los siguientes:

Algoritmos codiciosos estándar:

- Clasificación de selección.
- Problema de programación de trabajos.
- Codificación de Huffman.

Algoritmos codiciosos en grafos:

- **Árbol de expansión mínimo (MST)**
- **Algoritmo de árbol de expansión mínimo de Prim**
- **Algoritmo de árbol de expansión mínimo de Kruskal**
- Algoritmo de árbol de expansión mínimo de Dijkstra
- **Algoritmo de Ford-Fulkerson (Flujo Máximo en Redes)**

Algoritmos codiciosos para casos especiales de problemas de Programación Dinámica:

- Problema de mochila fraccionada.
- Número mínimo de monedas requeridas.

6. Programación Dinámica

¿De qué se trata la **Programación Dinámica** o DP?

Programación Dinámica (DP) como Divide y Vencerás, combinan soluciones a subproblemas para hallar una solución.

La programación dinámica es el algoritmo que divide el problema en varios subproblemas para encontrar la solución óptima.

Divide y Vencerás:

- Divide un problema en partes independientes.
- Resuelve cada parte.
- Combina las soluciones para resolver el problema original.

Algoritmo Divide y Vencerás

- **Divide el problema en subproblemas independientes** para resolver los subproblemas recursivamente y luego combinan sus soluciones para resolver el problema original.

Algoritmo de Programación Dinámica

- Es **aplicable cuando el subproblema no es independiente**, es decir, cuando los subproblemas comparten subproblemas.
- Por lo tanto, **un algoritmo de programación dinámica resuelve cada subproblema solo una vez** y luego guarda sus respuestas en una tabla, evitando así el trabajo de volver a calcular la respuesta cada vez que se encuentra el subproblema.

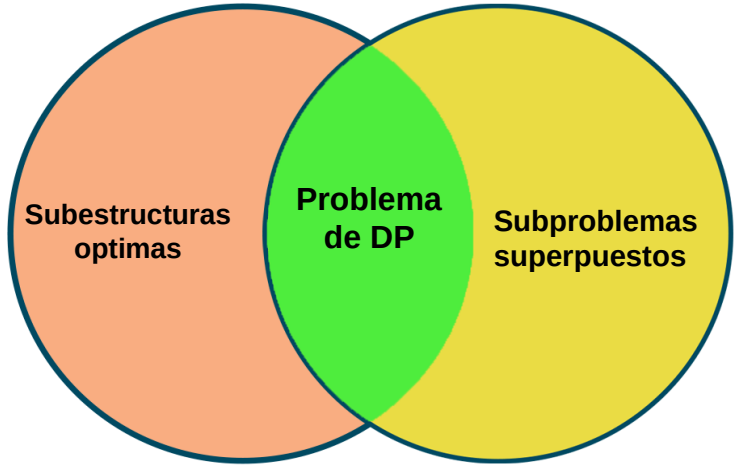
6. Programación Dinámica

¿De qué se trata la **Programación Dinámica** o DP?

A continuación, entenderemos sus características y elementos.

CARACTERÍSTICAS DE LA PROGRAMACIÓN DINÁMICA

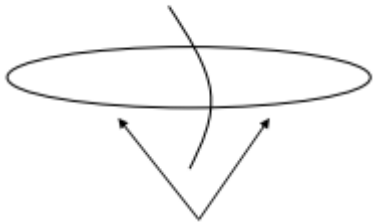
- **Utiliza dos enfoques para resolver un problema de optimización**:



6. Programación Dinámica

CARACTERÍSTICA DP #1

1. **Subestructuras optimas**



Cada subestructura es óptima
(Principio de optimalidad)

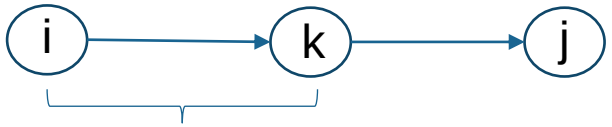
El **algoritmo de programación dinámica** obtiene la solución utilizando el principio de optimalidad.

¿Qué es el Principio de Optimalidad?

- El **principio de optimalidad** establece que **"en una secuencia óptima de decisiones o elecciones, cada subsecuencia también debe ser óptima"**.
- Cuando no es posible aplicar el principio de optimización, es casi imposible obtener la solución utilizando el enfoque de programación dinámica.

El **principio de optimización**:

"Si **k** es un nodo en el camino más corto para ir de **i** a **j**, entonces la parte del camino **de i a k**, y la parte **de k a j**, también deben ser óptimas".

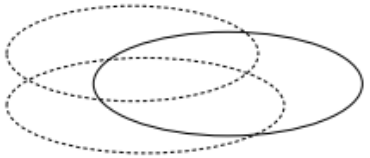


Parte del camino optimo

6. Programación Dinámica

CARACTERÍSTICA DP #2

2. Problemas que se sobreponen



Los subproblemas son dependientes o Subproblemas superpuestos
(Si no fuera así, se trata de Divide y Vencerás - D&C porque los subproblemas que resuelve son independientes)

¿Qué son los subproblemas superpuestos?

- El problema se llama **subproblemas superpuestos** si el algoritmo recursivo visita repetidamente los mismos subproblemas.

- Si algún problema tiene **subproblemas superpuestos**, podemos mejorar la implementación repetitiva de los subproblemas calculándolos solo una vez.

- Si el problema no tiene subproblemas superpuestos, entonces no es factible usar un algoritmo de programación dinámica para resolver el problema.

6. Programación Dinámica

ELEMENTOS DE LA PROGRAMACION DINAMICA

- Como ya mencionamos, el algoritmo DP divide el problema en varios subproblemas para encontrar la solución óptima.
- En DP, el **problema se divide en 3 elementos en total para obtener el resultado final**. Estos elementos son:

1. Subestructura

- La sub estructuración es el proceso de dividir el enunciado del problema dado en **subproblemas más pequeños**.
- Aquí logramos identificar la solución del problema original en términos de la solución de subproblemas.

2. Estructura de la tabla

- Es necesario almacenar la solución del subproblema en una tabla después de resolverlo.
- La programación dinámica **reutiliza** las soluciones de los subproblemas muchas veces para que no tengamos que resolver repetidamente el mismo problema, una y otra vez.

3. Enfoque de abajo hacia arriba

- Es el proceso de combinar las soluciones de los subproblemas para lograr el resultado final utilizando la tabla.
- El proceso comienza resolviendo el subproblema más pequeño y luego combinando su solución con los subproblemas de tamaño creciente hasta obtener la solución final del problema original.

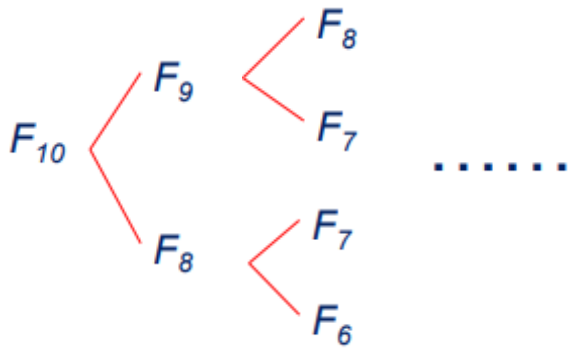
6. Programación Dinámica

EJEMPLO: PROBLEMA NÚMEROS FIBONACCI

- Se define a los números Fibonacci como:

$$\mathbf{F_0 = 0}$$
$$\mathbf{F_1 = 1}$$
$$\mathbf{F_i = F_{i-1} + F_{i-2}} \quad \boldsymbol{para\ i > 0}$$

- ¿Cómo se calcula $\mathbf{F_{10}}$?



¿Se puede aplicar DP en este problema?

- ✓ DP es aplicable cuando los problemas **NO SON INDEPENDIENTES**.
  - Los problemas comparten subproblemas.
- ✓ Números Fibonacci:
  - Recurrencia: $F_i = F_{i-1} + F_{i-2} \quad para\ i > 0$
  - Límites: $F_0 = 0\ F_1 = 1$

- ❑ Un acercamiento D&C resolvería repetidamente los subproblemas comunes.
- ✓ **DP resuelve cada problema 1 vez y lo guarda en una tabla**
- ✓ El uso de la tabla evita el tener que ejecutar re - computaciones.

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
0	1	1	2	3	5	8	13	21	34	55

6. Programación Dinámica

¿Diferencias entre **Programación Dinámica** y Programación Codiciosa?

Programación dinámica	Programación codiciosa
Tome una decisión en cada paso considerando el problema actual y la solución al problema resuelto previamente para calcular la solución óptima	Hacer la elección que sea mejor en un momento determinado con la esperanza de que conduzca a soluciones óptimas.
Garantía de conseguir la solución óptima	No existe tal garantía de obtener una solución óptima
Más lento en comparación con la programación codiciosa	Más rápido en comparación con la programación dinámica
Basado en la naturaleza recursiva para usar los resultados calculados previamente	Hacer uso de opciones localmente óptimas en cada paso

6. Programación Dinámica

VENTAJAS

- Mediante la programación dinámica se puede obtener una solución óptima tanto local como total (global).
- La programación dinámica reduce las líneas del código.

DESVENTAJAS

- La programación dinámica hace que la utilización de la memoria sea necesaria.
- A medida que utiliza la tabla mientras resuelve el problema de programación dinámica, necesita mucho espacio de memoria para fines de almacenamiento.

6. Programación Dinámica

APLICACIONES

1. En el problema de la mochila 0/1, que se plantea con más frecuencia en las entrevistas técnicas.
2. Como ayuda para almacenar el problema de la ruta más corta.
3. Se usa ampliamente al resolver un problema de optimización matemática.
4. Se utiliza en control de vuelo y robótica.
5. Se utiliza en algoritmos de enrutamiento.
6. Se utiliza principalmente en redes informáticas y problemas gráficos.

6. Programación Dinámica

CONCLUSIONES

1. La programación dinámica es más importante para optimizar las soluciones del problema en comparación con el enfoque recursivo.
2. Ayuda a reducir las llamadas a funciones repetidas y, por lo tanto, demuestra ser más rápido y efectivo que el enfoque recursivo y divide y vencerás.
3. La idea de simplemente almacenar los resultados de los subproblemas y utilizarlos para calcular el resto del problema hace que el algoritmo de programación dinámica sea un tema diferente e importante en el dominio de la estructura de datos y el algoritmo.
4. Es usual, que en una entrevista técnica se pregunte sobre las ventajas y desventajas de la DP, por ello es muy recomendable aprender y comprender.

PREGUNTAS

Dudas y opiniones