ESTRUCTURA DE DATOS Y ALGORITMOS I **2020-2**

Solución de Ejercicios y Dudas frecuentes

TEMA 3. ESTRATEGIAS PARA CONSTRUIR ALGORITMOS

Ejercicio de la mochila

- Problema de la mochila

Peso	10	20	30	45	50	60	70
Costo	30	35	55	45	65	80	60
Valor X unidad	3.0	1.75	1.83	1.0	1.30	1.33	0.85

- Primero el más ligero

  Peso: 10 + 20 + 30 + 45 = 105
  Valor: 30 + 35 + 55 + 45 = 165

- Primero el más valioso

  Peso: 60 + 50 + 30 + 10 = 150
  Valor: 80 + 65 + 55 + 30 = 230

- Primero el de mayor valor por unidad

  Peso: 10 + 30 + 20 + 60 = 120
  Valor: 30 + 55 + 35 + 80 = 200

- Problema de la mochila fraccional

Peso	10	20	30	45	50	60	70
Costo	30	35	55	45	65	80	60
Valor X unidad	3.0	1.75	1.83	1.0	1.30	1.33	0.85

- Primero el más ligero

  Peso: 10 + 20 + 30 + 45 + 45 (de 50) = 150
  Valor: 30 + 35 + 55 + 45 + 58.5 = 223.5

- Primero el más valioso

  Peso: 60 + 50 + 40 (de 70) = 150
  Valor: 80 + 65 + 34 = 179

- Primero el de mayor valor por unidad

  Peso: 10 + 30 + 20 + 60 + 30 (de 50) = 150
  Valor: 30 + 55 + 35 + 80 + 39 = 239

Ejercicios de recursividad

- Escribe una función recursiva que imprima los números impares entre 0 y n de manera descendente

  Si n = 18 >>> 17,15,13,11,9,7,5,3,1

```
void impares(int num) {
    if(num%2==0)
        num--;
    if (num > 0){
        printf("%d \n",num);
        impares(num-2);
    }
}
```

- Escribe una función recursiva que imprima los números pares entre 0 y n de manera ascendente

  Si n= 23 >> 0,2,4,6,8,10,12,14,16,18,20,22

```
void pares(int num) {
    if(num%2!=0)
        num--;
    if (num >= 0){
        pares(num-2);
        printf("%d \n",num);
    }
}
```

Aclaración: Recuerden que no se dice "más óptimo" porque la definición de óptimo se refiere "al mejor de todos", entonces cuando dices más óptimo, estas diciendo "más mejor"

¿A qué se refiere la dependencia de datos entre sub problemas en los algoritmos Greedy?.
Que una tarea requiera datos de otra, suponiendo que un problema se divide en tarea "a" "b" y "c", la tarea b haga cálculos y procese datos que A necesite en alguna parte de su procedimiento.
Que no haya dependencia implica que cada uno (a b y c) pueden hacer su secuencia de operaciones de manera independiente. Y después combinar sus resultados finales

¿Los algoritmos de fuerza bruta (o greedy) pueden trabajar con las dos estrategias de diseño? (top down y bottom up)
Si, se pueden combinar las técnicas de diseño con cualquier tipo de estrategia de construcción de algoritmos (greedy, fuerza bruta, divide y vencerás, etc). Aunque algunas combinaciones son más frecuentes que otras

¿Todos los algoritmos fuerza bruta se pueden resolver con una estrategia greedy?
No, porque no cualquier algoritmo se puede expresar como greedy

¿Se pueden combinar las dos estrategias de diseño en una misma solución?
Si, eso ocurre frecuentemente en soluciones complejas.

¿Se puede optimizar más de una variable en un problema de optimización?
Si es posible, aunque requiere otro tipo de técnicas.

¿Por qué los algoritmos greedy una vez tomada la decisión no se puede revertir?
Así se define su forma de ejecución, y en realidad esta forma es eficiente en muchos casos, en problemas donde se requiere "regresar" por la naturaleza del problema, tal vez es mas apropiado usar Backtrack o incluso fuerza bruta

¿Si en la mochila hay dos artículos con el mismo costo y peso, cual se elige?
Cualquiera de los dos, y si hay espacio, el siguiente en elegir es el segundo artículo.
Si los objetos tienen el mismo valor por unidad, pero diferente peso, se tomaría el más ligero, porque eso daría una mayor posibilidad de que quepa otro objeto

¿Las funciones recursivas siempre tienen contraparte iterativa?
No se si exista alguna demostración matemática al respecto, (sería bueno que alguien lo buscara de tarea moral) pero se toma como verdadera esta afirmación

¿Funciones recursivas pueden ahorrar más memoria que las iterativas?
En algunos casos sí pero no siempre ocurre.

¿Qué ocurre en la memoria cuando se usan funciones recursivas?
Cada llamada tiene su propio espacio de memoria, por ejemplo, si se llega a declarar una variable dentro de la función, en cada llamada la variable es local. (Se aplican las mismas reglas de alcance de variables)
Las funciones recursivas se pueden ver como una pila, cada vez que se llama de nuevo, la anterior se queda en la pila de ejecución y cuando termina o llega al caso base, las que estaban en espera van saliendo en orden inverso al que ingresaron, y si hubiera variables locales, en cada caso van desapareciendo

¿Cómo se sabe para un problema, si es más eficiente la versión iterativa o la recursiva?
Se puede hacer con el análisis de complejidad o si no es del todo claro, la ejecución de los algoritmos en un contexto controlado

¿Se pueden usar ciclos adentro de las funciones recursivas?

La gran mayoría de las funciones recursivas no tienen ciclos, y esto es porque la recursividad se pretende que reemplace al uso de estos.

Si un problema se puede resolver por medio del algoritmo Greedy eso implica que también se puede resolver por Backtrack

Existen diversos problemas como por ejemplo el de la mochila que se pueden resolver desde diferentes estrategias, pero no hay una regla como tal. En cada problema se tiene que verificar con cual estrategia se puede resolver

¿Cuál es la estrategia más eficiente de todas?

Las estrategias no son eficientes o ineficientes por sí mismas, pudiera pensarse por ejemplo qué fuerza bruta es ineficiente, pero recuerden que la eficiencia radica en la dificultad para resolver los problemas, no es lo mismo adivinar una contraseña de 50 caracteres por fuerza bruta, que hacer una búsqueda lineal en un arreglo de 10 elementos.

En el estudio de algoritmos es difícil encontrar reglas generales que apliquen para todo, en la mayoría de los casos se tiene que revisar caso por caso y saber cuál es la técnica más apropiada muchas veces depende de la experiencia y del conocimiento que se tenga para encontrar lo más apropiado.