UNIVERSIDAD TORCUATO DI TELLA

Tecnología Digital V: Diseño de algoritmos
Licenciatura en Tecnología Digital
Primer Semestre 2025

Guía de ejercicios 2: Técnicas de diseño de algoritmos

---

1. Dado un grafo $G = (V, E)$, el *problema de clique máxima* consiste en encontrar la clique de tamaño máximo de un grafo.

   a) Proponer un algoritmo basado en *backtracking* para este problema. Pista: Si el grafo tiene vértices $V = \{1, \ldots, n\}$, representar una solución como una secuencia de decisiones $(a_1, \ldots, a_n)$, de modo tal que $a_i \in \{0, 1\}$ represente si se selecciona el vértice i para la clique o no, para $i \in V$.

   b) Si además tenemos una función de peso $w : V \to \mathbb{R}$ que asigna un peso a cada vértice, entonces podemos considerar el *problema de clique de peso máximo* sobre el grafo. Mostrar un grafo G cuya clique de peso máximo no tenga tamaño $\omega(G)$.

   c) ¿Cómo se modifica el algoritmo propuesto en el punto a) si ahora necesitamos resolver el problema de clique de peso máximo?

2. Dado un grafo $G = (V, E)$ y un conjunto C de colores, queremos saber si existe un coloreo de G utilizando los colores de C (esto es equivalente a determinar si $\chi(G) \geq |C|$ o no). Proponer un algoritmo basado en *backtracking* para este problema. Pista: Si el grafo tiene vértices $V = \{1, \ldots, n\}$, representar una solución como una secuencia de decisiones $(a_1, \ldots, a_n)$, de modo tal que $a_i \in C$ represente el color asignado al vértice i, para $i \in V$.

3. Proponer un algoritmo basado en *backtracking* para encontrar el menor conjunto dominante de un grafo (¿cómo conviene representar una solución como una secuencia de decisiones en este caso?).

4. Una empresa utiliza una flota de camiones para realizar el reparto de su mercadería. Antes de cada viaje, se debe decidir qué paquetes de mercadería cargar en cada camión, de modo de aprovechar los viajes al máximo. Cada camión tiene una capacidad máxima de m metros cúbicos, y cada paquete a transportar tiene un volumen (en metros cúbicos). El problema de *bin packing* consiste en determinar en qué camión se ubica cada paquete, de modo tal de utilizar la menor cantidad de camiones. Escribir un algoritmo basado en *backtracking* para este problema (¿cómo conviene representar una solución como una secuencia de decisiones en este caso?).

5. Dado un multiconjunto (es decir, un conjunto con repeticiones) de números enteros $C = \{a_1, a_2, \ldots, a_n\}$ y un número k, el SUBSET-SUM PROBLEM busca determinar si existe un subconjunto $S \subseteq C$ tal que la suma de los elementos en S sea k. Formalmente, buscamos un subconjunto $S \subseteq C$ tal que

   $$\sum_{a_j \in S} a_j = k$$

   A modo de ejemplo, dado el conjunto $C = \{3, 5, 8, 10, 10\}$ el subconjunto $S = \{3, 5, 10\}$ suma $k = 18$, y no existe subconjunto de C que sume $k = 17$.

   a) Plantear una función recursiva para resolver este problema.

   b) Escribir un algoritmo de programación dinámica para este problema.

   c) Escribir una rutina que reconstruya la solución (si es que existe).

6. Tenemos un multiconjunto con B de valores de billetes y queremos comprar un producto de costo c de una máquina que no da vuelto. Para poder adquirir el producto debemos cubrir su costo usando un subconjunto de nuestros billetes. El objetivo es pagar con el mínimo exceso posible a fin de minimizar nuestra pérdida. Por ejemplo, si $c = 14$ y $B = \{2, 3, 5, 10, 10, 20, 20\}$ (notar que tenemos dos billetes de \$10 y dos billetes de \$20), la solución es pagar \$15, con exceso \$1. Escribir un algoritmo de programación dinámica para este problema.

7. Consideremos el problema de, dado un número $n \in \mathbb{N}$, determinar de cuántas formas puede escribirse como suma de los números 1, 3 y 4. Por ejemplo, para $n = 5$, la respuesta es 6:

   - $5 = 1 + 1 + 1 + 1 + 1$
   - $5 = 1 + 4$
   - $5 = 4 + 1$
   - $5 = 1 + 1 + 3$
   - $5 = 1 + 3 + 1$
   - $5 = 3 + 1 + 1$

   Se cuenta con la siguiente función, que resuelve el problema pero en tiempo exponencial:

```
1  int sumasDistintas(int n) {
2     if n == 1
3        return 1;      // hay una forma de escribir 1: (1)
4     else if n == 2
5        return 1;      // hay una forma de escribir 2: (1+1)
6     else if n == 3
7        return 2;      // hay dos formas de escribir 3: (1+1+1),(3)
8     else if n == 4
9        return 4;      // hay cuatro formas de escribir 4: (1+1+1+1),(1+3)
             ,(3+1),(4)
10    else
11       return sumasDistintas(n-1) + sumasDistintas(n-3) + sumasDistintas(n
             -4);
12 }
```

   a) Plantear una función recursiva para resolver este problema, sabiendo que hay una forma de escribir
   b) Explicar por qué este algoritmo resuelve el problema y por qué es exponencial.
   c) Escribir una función en Python basada en programación dinámica que resuelva el mismo problema en $O(n)$.
   d) Comparar los tiempos de ejecución de ambas funciones (la dada y la propuesta) tomando $n = 30$, $n = 35$, $n = 40$ y $n = 45$. ¿A qué se deben las diferencias observadas?

8. *[Opcional]* Tenemos una vara de acero de n metros de longitud, y la podemos cortar en varas más pequeñas de longitudes $\ell_1, \ell_2, \ldots, \ell_k$. Cada longitud ℓ_i tiene un precio de venta p_i, para $i = 1, \ldots, k$. El problema consiste en determinar qué longitudes cortar, de modo de maximizar el precio total de venta (suponiendo que vamos a poder vender todas las varas que cortemos). Por ejemplo, si $n = 10$ y las longitudes posibles son las siguientes:

i	ℓ_i	p_i
1	2	\$80
2	3	\$140
3	5	\$170

entonces la solución óptima consiste en cortar dos varas de dos metros y dos varas de tres metros, obteniendo un beneficio total de $440. Escribir un algoritmo de programación dinámica para este problema.

9. *[Opcional]* Consideremos una empresa dedicada a la compraventa de un único producto. Cada día puede comprar una unidad del producto, puede vender una unidad del producto, o puede no hacer nada. Tenemos una secuencia de n días, y conocemos el precio p_i de una unidad del producto en el día i, para $i = 1, \ldots, n$. Si la empresa compra/vende una unidad en el día i, entonces paga/recibe p_i pesos en ese día. La empresa comienza sin stock en su depósito (con lo cual, la primera operación que realice tiene que ser una compra), y solamente puede vender productos que estén en stock (es decir, que hayan sido comprados previamente). El problema que queremos resolver consiste en determinar qué días comprar y qué días vender para maximizar la ganancia total. Observar que en una solución óptima terminamos el día n sin stock en el depósito. Escribir un algoritmo de programación dinámica para este problema, planteando primero una función recursiva `ganancia(j,c)` que represente la máxima ganancia que se puede obtener el día j teniendo c unidades en stock.

10. *[Opcional]* Sea $v = (v_1, v_2, \ldots v_n)$ un vector de números naturales, y sea $w \in \mathbb{N}$. Se desea intercalar entre los elementos de v las operaciones $+$ (suma), $\times$ (multiplicación) y $\uparrow$ (potenciación) de tal manera que al evaluar la expresión obtenida el resultado sea w. Para evaluar la expresión se opera de izquierda a derecha ignorando la precedencia de los operadores. Por ejemplo, si $v = (3, 1, 5, 2, 1)$, y las operaciones elegidas son $+$, $\times$, $\uparrow$ y $\times$ (en ese orden), la expresión obtenida es $3 + 1 \times 5 \uparrow 2 \times 1$, que se evalúa como $(((3 + 1) \times 5) \uparrow 2) \times 1 = 400$.

    a) Escribir una formulación recursiva que sea la base de un algoritmo de PD que, dados v y w, encuentre una secuencia de operaciones como la deseada, en caso de que tal secuencia exista. Explicar su semántica e indicar cuáles serían los parámetros para resolver el problema.

    b) Diseñar un algoritmo basado en PD con la fórmula del ítem anterior, y dar su complejidad temporal y espacial auxiliar. Comparar cómo resultaría un enfoque *top-down* con uno *bottom-up*.