

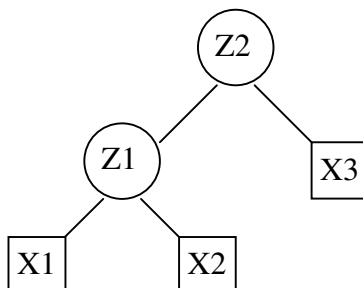
## Metodología y Tecnología de la Programación

### Hoja 3 de problemas – Avance Rápido

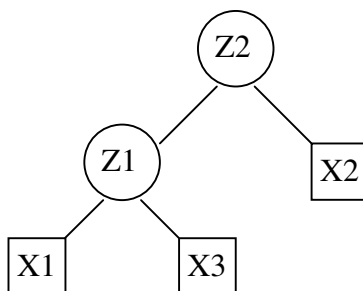
1. Proponer un ejemplo de problema de la mochila de modo que tenga varias soluciones óptimas y describir todas las que se obtienen en ese caso.
2. Escribir y estudiar la complejidad de un procedimiento que reconstruye los caminos mínimos desde el vértice 1 a todos los demás a partir del array *anterior*[2..n] obtenido en el algoritmo de *Dijkstra*. El procedimiento posee la siguiente especificación:

```
procedimiento caminos_mínimos (anterior[2..n], (var) caminos:  
tipo_a_elegir)  
  
{El array anterior[2..n] contiene los caminos mínimos desde 1 a cada  
vértice, codificados de modo que anterior[i] contiene el predecesor del  
vértice i en el camino desde 1 hasta i. El "tipo a decidir" es una  
estructura que debe elegir razonadamente el diseñador }
```

3. Sea un conjunto de  $n$  archivos de longitudes  $l_1, \dots, l_n$  (número de datos), que se desean mezclar dos a dos progresivamente hasta obtener un solo archivo. Proponer una estrategia para construir el árbol óptimo de mezcla, que es aquél que en cada nodo interior tiene un archivo intermedio y en cada hoja un archivo original, de modo que la mezcla guiada por un recorrido del árbol minimiza el número de datos recorridos. Por ejemplo, sean los archivos X1, X2 y X3 con longitudes 10, 30 y 20 respectivamente. Si mezclamos X1 y X2, obtenemos Z1 habiendo recorrido 40 datos. Si luego mezclamos Z1 y X3, recorreremos 60 datos más (total, recorreremos 100 datos). El árbol de mezcla es en este caso:



En cambio, si mezclamos primero X1 y X3 y el resultado con X2, se recorren sólo 60 datos, dando lugar al árbol (óptimo) siguiente:



Codificar el algoritmo de avance rápido que construye el árbol óptimo de mezcla y estudiar su complejidad.

4. Dado el árbol óptimo de mezcla para  $n$  archivos, escribir y estudiar la complejidad de un algoritmo que muestre por pantalla una lista de mensajes que comunican al usuario cómo debe ir mezclando los archivos sucesivamente, identificándolos, por ejemplo, por su longitud.
5. Dado un conjunto de clientes a ser atendidos en un servidor, cada uno con un tiempo de proceso conocido de antemano  $t_i$ , y llegando todos en el mismo instante, proponer una estrategia que permita decidir en qué orden se atenderán los clientes si se desea minimizar el tiempo medio que un cliente pasa en el sistema. Un cliente pasa en el sistema todo el tiempo que espera a que atiendan a los clientes anteriores más el tiempo de atenderle a él. Es decir, que si hemos ordenado los cliente de la forma  $i_1, \dots, i_n$ , entonces el cliente  $i_j$  pasa en el sistema  $\sum_{k=1}^j t_{i_k}$  y por tanto, el tiempo medio de espera es  $\frac{\sum_{j=1}^n \sum_{k=1}^j t_{i_k}}{n}$ . Por ejemplo, si hay tres clientes con tiempos 10,15 y 5, y se ordenan así directamente (1,2,3), el tiempo medio de espera es de 21,6, pero si se ordenan (3,1,2) se obtiene un tiempo (óptimo) de 18,3. Escribir el pseudocódigo de un algoritmo de avance rápido que siga la estrategia propuesta y estudiar su complejidad.
6. Dado un grafo  $G=(V,A)$  y dados dos vértices  $i,j \in V$ , demostrar que la estrategia devoradora más obvia consistente en tomar en cada paso el vértice más cercano (que va por el camino correcto, es decir, tal que desde el se puede llegar a  $j$ ) no lleva a una solución óptima necesariamente.
7. (Problema del recubrimiento de conjuntos) Sea  $S$  una familia de  $m$  conjuntos  $S_i$ , es decir,  $S=\{S_1, \dots, S_m\}$  es un conjunto de conjuntos. Un recubrimiento de  $S$  es un subconjunto de conjuntos  $T=\{T_1, \dots, T_n\}$  de  $S$  que lo cubren, es decir, tal que  $\cup T_i = \cup S_j$ . Los conjuntos  $T_i$  pertenecen a la familia  $S$ , es decir, todo  $T_i = S_j$  para algún  $1 \leq j \leq m$ . Se desea encontrar un recubrimiento de  $S$  con un número mínimo de conjuntos  $n$ .  
  
Sea la estrategia devoradora que selecciona en cada paso  $k$  el conjunto de  $S$  que no haya sido seleccionado previamente y tal que posea el máximo número de elementos que aún no están en los  $T_s$  seleccionados previamente. Se detiene cuando  $\cup T_i = \cup S_j$ . Si se supone que, para facilitar la implementación,  $\cup S_j = \{1, \dots, r\}$  (es decir, los  $S_j$  son conjuntos de enteros entre 1 y  $r$ ), escribir el algoritmo que implementa esta estrategia y estudiar su complejidad. Demostrar que el algoritmo anterior no siempre calcula un recubrimiento mínimo (dar un contraejemplo).  
  
Si se define un recubrimiento mínimo como aquel cuyos conjuntos tienen un mínimo número de elementos, es decir, tal que  $\sum |T_i|$  es mínimo, ¿lleva la anterior estrategia a un recubrimiento mínimo?
8. (Problema del recubrimiento de vértices) Dado un grafo no dirigido  $G=(V,A)$ , un recubrimiento de  $G$  es un conjunto de vértices  $U \subseteq V$  tal que todas arista de  $A$  es incidente (toca) algún vértice de  $U$ . Un recubrimiento mínimo es aquel que posee el número mínimo de vértices. Estudiar si el algoritmo que a continuación se propone resuelve el problema del recubrimiento de vértices, y estudiar su complejidad.

```

procedimiento Cubrir(G: grafo, U: conjunto de vértices)
U ← ∅
RA ← G.A
RV ← G.V
repetir

```

```

v ← máximo_grado(G, RV)
{Selecciona el vértice de RV con mayor número de aristas
inicidentes}
U ← U ∪ {v}
RV ← RV - {v}
RA ← RA - {a=(a1, a2) tal que a1=v ó a2=v}
hasta RA = ∅

```

9. (Problema de la asignación óptima de tareas) Dados  $n$  trabajadores y  $n$  trabajos, y suponiendo que cada trabajador  $i$  esta cualificado con adecuación  $v_{ij} > 0$  para realizar la tarea  $j$ , se desea asignar cada trabajo a un trabajador (es decir, la variable  $x_{ij}$  vale 0 si no se asigna el trabajador  $i$  a la tarea  $j$ , y 1 si se asigna) de modo que cada trabajador realice una sola tarea ( $x_{i1} + \dots + x_{in} = 1$ ) y cada tarea sea realizada por un solo trabajador ( $x_{1j} + \dots + x_{nj} = 1$ ).

Se desea maximizar la cualificación de los trabajadores, esto es,  $\sum_i \sum_j v_{ij} x_{ij}$ . Las dos posibles estrategias devoradoras obvias son, asignar cada trabajador a su mejor posible trabajo, y seleccionar para cada tarea a su mejor trabajador. Escribir el pseudocódigo de ambos esquemas, estudiar sus complejidades respectivas y demostrar que ambos fallan.

10. Sean  $n$  programas de longitudes  $l_1, \dots, l_n$  a almacenar en una cinta en la que caben todos ellos. Supongamos que el programa  $i$  va a ser recuperado con frecuencia  $f_i$ . Si los programas se almacenan en un orden  $i_1, \dots, i_n$ , el tiempo esperado de recuperación (TER) es

$$\frac{\sum_j \left( f_j \sum_{k=1}^j l_{i_k} \right)}{\sum_i f_i}$$

- (a) Demostrar que si se almacenan los programas por orden no decreciente de  $l_i$ , no se minimiza necesariamente el TER.
- (b) Demostrar que si se almacenan los programas por orden no creciente de  $f_i$ , no se minimiza necesariamente el TER.
- (c) TER se minimiza si se almacenan los programas por orden no creciente de  $f_i/l_i$ . Escribir el pseudocódigo de esta idea.