Examen de Programación 3
6 de febrero de 2025

En recuadros con este formato aparecerán aclaraciones que cumplen una función explicativa pero que no eran requeridos como parte de la solución.

Ejercicio 1 (50 puntos)

En una conferencia una sala está destinada a n sesiones y se debe decidir cómo planificar las asignaciones. De cada sesión se conoce su duración medida en minutos y su cantidad de participantes. La asignación de la sala a una sesión es por un único intervalo de tiempo igual a la duración de la sesión. Los intervalos asignados a diferentes sesiones no deben solaparse. No debe haber tiempos ociosos, por lo que el final de una sesión debe coincidir con el inicio de la siguiente. La planificación debe hacerse de modo que sea mínima la suma de los tiempos de espera de todos los participantes. O sea, si para la sesión que se asigna en la $i-$ésima posición, denotamos su duración con d_i y su cantidad de participantes con c_i, se debe minimizar $\sum_{i=2}^{n} \left(c_i \cdot \sum_{j=1}^{i-1} d_j \right)$ (los participantes de la primera sesión no tienen que esperar).

(a) Describa el criterio con el que se deben ordenar las sesiones.

Muestre que se cumple para los siguientes casos particulares de dos sesiones:

A:

Sesión	Duración	Cantidad
1	40	10
2	60	20

B:

Sesión	Duración	Cantidad
1	40	10
2	100	20

Note que en el caso `A` se debe asignar primero la sesión 2, a pesar de ser la de mayor duración, para un tiempo de espera de 600 (10 participantes deben esperar 60 minutos cada uno), mientras que si se asignara primero la sesión 1 el tiempo de espera sería 800 (20 participantes deberían esperar 40 minutos cada uno).

En el caso `B` se debe asignar primero la sesión 1, a pesar de ser la de menor cantidad de partipantes, para un tiempo de espera de 800 (20 participantes deben esperar 40 minutos cada uno), mientras que si se asignara primero la sesión 2 el tiempo de espera sería 1000 (10 participantes deberían esperar 100 minutos cada uno).

(b) Sea A una planifcación obtenida mediante el criterio de ordenación propuesto. Demuestre, mediante la técnica *exchange argument* vista en el curso que A es óptima. Asuma que se renumeran las sesiones de forma que $A = (1, 2, \ldots, n)$.

Solución:

En este ejercicio se espera:

(a)
- Proponer un criterio correcto.
- Mostrar que con el criterio propuesto se obtiene el resultado correcto en los dos ejemplos.

(b) Demostrar, mediante la técnica *exchange argument*, la corrección del criterio propuesto. Para esto

- se tranforma una planificación O, diferente a A, en una planificación O',
- se muestra que O' es válida, o sea, que incluye una vez cada sesión,
- se demuestra que el tiempo de espera de O' no es mayor que el de O,
- se argumenta que según algún criterio se acerca a A,
- se argumenta que si O' sigue siendo diferene a A, entonces mediante la misma transformación en una cantidad finita de pasos se llegaría a A,
- se concluye que ninguna planificación puede tener menor tiempo de espera que A.

Lo anterior, en particular, se puede demostrar mediante el concepto de inversión. Para esto hay que

- definir inversión,
- recordar que si una planificación O es diferente a A entonces existe al menos un par de posiciones consecutivas, h y $h+1$, que constituyen una inversión,
- proponer la planificación O', idéntica a O excepto porque se invierten (quedan según el criterio con que se construyó A) las sesiones de las posiciones h y $h+1$,
- hacer notar que las sesiones que están en las posiciones $1 \cdots h-1, h+1 \cdots n$ tienen el mismo tiempo de inico que en O, por lo que no se modifica el sumando correspondiente en el cálculo del tiempo de espera.
- demostrar que la suma de los sumandos correspondientes a las posiciones h y $h+1$ en O' es menor o igual que en O,
- concluir de los dos puntos anteriores que el tiempo de espera en O' no es mayor que el de O,
- hacer notar que O' tiene una inversión menos que O y que la cantidad de inversiones es finita,
- argumentar que el tiempo de espera de A no puede ser mayor que el de O y concluir que A es una planificación óptima.

(a) Las sesiones se deben ordenar de manera decreciente (no estricta) según el cociente entre la cantidad de participantes y la duración, c_i/d_i.

La intuición, que debe demostrarse, con este criterio, es lograr un compromiso entre

- poner primero la sesión con muchos participantes para reducir el tiempo de espera de ellos,
- poner primero la sesión con menor duración para reducir el tiempo de espera de los participantes de las otras sesiones.

A El cociente es 0.25 para la sesión 1, y 0.33 para la sesión 2. Por lo tanto se debe asignar primero la sesión 2.

B El cociente es 0.25 para la sesión 1, y 0.2 para la sesión 2. Por lo tanto se debe asignar primero la sesión 1.

Obviamente, también se puede ordenar de manera creciente (no estricta) según el cociente entre la duración y la cantidad de participantes, d_i/c_i. En este caso los cocientes serían

A 4 para la sesión 1 y 3 para la sesión 2.

B 4 para la sesión 1 y 5 para la sesión 2.

El siguiente es un ejemplo de soluciones incorrecta. En cada paso se elige como primera sesión la que minimice el producto de su duración por la suma de las cantidades de participantes de las otras sesiones. En símbolos, se elige la sesión j que minimice $(C - c_j) \cdot d_j$, donde $C = \sum_{i=1}^{n} c_i$.

Como contraejemplo, sea $c_1 = c_2 = 100, d_1 = d_2 = 10, c_3 = 1, d_3 = 1$.

Eligiendo las sesiones en orden 1, 2, 3, que es una solución óptima, la suma total de tiempos de espera es $100 \cdot 10 + 1 \cdot 20 = 1020$.

Según el criterio propuesto se debe elegir primero la sesión 3, porque $200 \cdot 1 = 200$ es menor que $101 \cdot 10 = 1010$, que es el valor que correspondería a elegir la sesión 1 o 2. Pero el tiempo de espera es $100 \cdot 1 + 100 \cdot (1 + 10) = 1200$.

En otra propuesta se elige primero la sesión que maximice el producto de su cantidad de participantes por la suma de las duraciones de las otras sesiones. O sea, se elige la sesión j que maximice $(D - d_j) \cdot c_j$, donde $D = \sum_{i=1^n} d_i$.

Como contraejemplo, sea $c_1 = c_2 = 10, d_1 = d_2 = 1, c_3 = 600, d_3 = 100$.

Eligiendo las sesiones en orden 1, 2, 3 que es una solución óptima, la suma total de tiempos de espera es $10 \cdot 1 + 600 \cdot (1 + 1) = 1210$.

Según el criterio propuesto se debe elegir primero la sesión 3, porque $600 \cdot (1 + 1) = 1200$ es mayor que $10 \cdot (1 + 100) = 1010$, que es el valor que correspondería a elegir la sesión 1 o 2. Pero el tiempo de espera es $10 \cdot 100 + 10 \cdot (100 + 1) = 2010$.

Otra solución incorrecta. Se elige primero la sesión que maximice el producto de su cantidad por el mínimo de las duraciones de las otras sesiones. O sea la sesión j que maximice $c_j \cdot \text{mín}\{d_i : i \neq j\}$. La motivación es que no elegir esa sesión implica por lo menos un tiempo de espera igual a ese producto.

Como contraejemplo, sea $c_1 = c_2 = 100, d_1 = d_2 = 10, c_3 = 200, d_3 = 25$.

Eligiendo las sesiones en orden 1, 2, 3, que es la solución óptima, la suma total de tiempos de espera es $100 \cdot 10 + 200 \cdot 20 = 5000$.

Según el criterio propuesto se debe elegir primero la sesión 3, porque $200 \cdot 10 = 200$ es mayor que $100 \cdot 10 = 1000$, que es el valor que correspondería a elegir la sesión 1 o 2. Pero esta solución tiene un tiempo de espera total $100 \cdot 25 + 100 \cdot 35 = 6000$.

Otras dos soluciones incorrectas similares. Se calculan todos los productos $c_i \cdot d_j, i \neq j$.

En la primera propuesta se busca el máximo de esos productos y se elige i como primera sesión (la que corresponde al factor cantidad).

Como contraejemplo, sea $c_1 = 10, d_1 = 1, c_2 = 100, d_2 = 20, c_3 = 200, d_3 = 100$.

Eligiendo las sesiones en orden 1, 2, 3, que es la solución óptima, la suma total de tiempos de espera es $100 \cdot 1 + 200 \cdot 21 = 4300$.

Según el criterio propuesto se debe elegir primero la sesión 2, porque $100 \cdot 100 = 1000$ que es el máximo de los productos y correspone a $c_2 \cdot d_3$. Luego se elige la sesión 1 y finalmente la 3. En esta solución el tiempo de espera total $10 \cdot 20 + 200 \cdot 21 = 4400$.

En la propuesta se busca el mínimo de esos productos y se elige j como primera sesión (la que corresponde al factor duración).

Como contraejemplo, sea $c_1 = 100, d_1 = 10, c_2 = 10, d_2 = 5, c_3 = 1, d_3 = 1$.

Eligiendo las sesiones en orden 1, 2, 3, que es la solución óptima, la suma total de tiempos de espera es $10 \cdot 10 + 1 \cdot 15 = 115$.

Según el criterio propuesto se debe elegir primero la sesión 2, porque $1 \cdot 5 = 1000$ es el mínimo de los productos y correspone a $c_3 \cdot d_2$. Luego se elige la sesión 1 y finalmente la 3. En esta solución el tiempo de espera total $100 \cdot 5 + 1 \cdot 15 = 515$.

(b) Sea $O = (o_1, \ldots, o_n)$ una planificación diferente a A. Según lo visto en el curso esto implica que exsite un par de posiciones consecutivas, $h, h+1$, en las que hay una inversión, que significa $o_h > o_{h+1}$.

Consideremos la planificación $O' = (o_1, \ldots, o_{h-1}, o_{h+1}, o_h, o_{h+2}, \ldots, o_n)$, sin tiempos ociosos y que incluye todas las sesiones, que se obtiene a partir de O invirtiendo las sesiones de las posiciones h e $h+1$ y dejando las demás en las mismas posiciones. Las sesiones de las posiciones $1, \ldots, h-1$, $h+2, \ldots, n$ no cambian su tiempo de inicio, por lo que no cambia el tiempo de espera de sus participantes, y por lo tanto no cambia la suma $c_{o_i} \cdot \sum_{j=1}^{i-1} d_{o_j}$.

Con respecto a O, en O' la sesión o_h empieza $d_{o_{h+1}}$ minutos más tarde, por lo que la suma del tiempo de espera de sus participantes aumenta $d_{o_{h+1}} \cdot c_{o_h}$.

Y la sesión o_{h+1} empieza d_{o_h} minutos más temprano, por lo que la suma del tiempo de espera de sus participantes disminuye $d_{o_h} \cdot c_{o_{h+1}}$.

Por lo tanto el tiempo de espera para la planificaión O' es igual al de la planificación óptima más $d_{o_{h+1}} \cdot c_{o_h} - d_{o_h} \cdot c_{o_{h+1}}$.

La inversión entre las sesiones o_h y o_{h+1} implica que $c_{o_h}/d_{o_h} \leq c_{o_{h+1}}/d_{o_{h+1}}$, de donde se obtiene $d_{o_{h+1}} \cdot c_{o_h} - d_{o_h} \cdot c_{o_{h+1}} \leq 0$, por lo que en la planificación O' el tiempo de espera no es mayor que en la planificación O.

La cantidad de inversiones de O' es una menos que la cantidad de inversiones de O. Si O' es diferente a A se puede aplicar el mismo razonamiento que se aplicó con O, para encontrar otra planificación con aún menos inversiones respecto a A. Como la cantidad de inversiones es a lo sumo $\binom{n}{2}$, y por lo tanto finita, cualquier planificación puede transformarse en A sin que aumente el tiempo de espera. Por lo tanto ninguna planificación tiene tiempo de espera menor que A, por lo que A es óptima.

También se puede definir inversión en O como un par de índices $i < j$ para los que se cumple $c_{o_i}/d_{d_i} < c_{o_j}/d_{d_j}$. El desarrollo es el mismo que antes y se va a llegar a $d_{o_{h+1}} \cdot c_{o_h} - d_{o_h} \cdot c_{o_{h+1}} < 0$, lo que demuestra que O' tiene un tiempo de espera estrictamente menor que el de O.

Con esta definición de inversión, una solución O sin inversiones puede ser diferente de A, por lo que se debe agregar a la demostración que una solución sin inversiones tiene el mismo tiempo de espera que A. Para esto, veamos que una solución sin inversiones difiere de A en que tiene segmentos de posiciones, $i, i+1, \ldots, j$, en los que los elementos tienen el mismo valor c/d. Con el mismo procedimiento que antes, podemos intercambiar posiciones consecutivas hasta que los elementos queden en el mismo orden que en A. En cada intercambio se cumple $c_{o_h}/d_{o_h} = c_{o_{h+1}}/d_{o_{h+1}}$, por lo que $d_{o_{h+1}} \cdot c_{o_h} - d_{o_h} \cdot c_{o_{h+1}} = 0$, lo que demuestra que el tiempo de espera no cambia.

Ejercicio 2 (50 puntos)

Sea O un arreglo no vacío, **ordenado de menor a mayor**. Sea F un conjunto (**no ordenado**) que, excepto por un elemento que falta, contiene todos los elementos de O. Nos interesa saber cuál es el elemento de O faltante en F.

Para simplificar las expresiones suponemos que O no tiene elementos repetidos, y que la cantidad de elementos de O es uno menos que una potencia de 2. O sea, $|O| = n - 1$ (y por lo tanto $|F| = n - 2$), donde n es potencia de 2. Note que $m = n/2$ es el punto medio del rango $1\,..\,n-1$ ($m = (1 + n - 1)/2$), por lo que la longitud de los rangos $1\,..\,m-1$ y $m+1\,..\,n-1$ es $n/2 - 1$, que también es uno menos que una potencia de 2.

(a) Escriba un algoritmo que siga una estrategia de tipo *Dividir y Conquistar* para encontrar el elemento de O faltante en F. El tiempo de ejecución debe ser $O(n)$.

(b) Justifique que el tiempo de ejecución es $O(n)$ escribiendo una relación de recurrencia, donde $T(n)$ es el tiempo de ejecución de las instancias en las que $|O| = n - 1$ (no hace falta desarrollar la relación de recurrencia). Explique el origen de cada término.

Sugerencia: Use una estrategia similar a la que emplea *QuickSort* para dividir las instancias. Puede asumir que existe un algoritmo `separar(A,s)`, que dados un conjunto A y un elemento separador s, devuelve un par de subconjuntos de A, $(A_<, A_>)$, tal que, en $A_<$ están todos los elementos de A estrictamente menores a s, y en $A_>$ están todos los elementos de A estrictamente mayores a s. El tiempo de ejecución de `separar(A,s)` es $\Theta(|A|)$. Note que s **no** es un elemento de A si y solo si $|A| = |A_<| + |A_>|$.

Solución:

En este ejercicio se espera:

(a) Aunque se puede diseñar un algoritmo iterativo, lo siguiente se plantea para la versión recursiva.

Se debe escribir un algoritmo correcto y con tiempo de ejecución es $O(n)$ que

- tiene un parámetro F' que es un subconjunto de F, y otro que es un subarreglo de O consistente con F' (o sea, tiene los mismos elementos que F', y uno más),
- tiene algún caso base,
- en las llamadas recursivas los parámetros se acercan a algún caso base,
- en todos los casos devuelve un elemento de O,
- es inicialmente invocado con los conjuntos F y O.

(b) Escribir una relación de recurrencia, que debe incluir la condición inicial, consistente con el algoritmo, y

- hacer notar (sin hacer el desarrollo) que la solución de esa recurrencia cumple con lo que se pide,
- explicar el valor en la condición inicial y el término correspondiente,
- explicar la cantidad de términos recursivos, y justificar el valor de sus parámetros,
- justificar el término independiente en la relación genérica.

(a) En este algoritmo se considera que el tamaño de la instancia es n, donde n es potencia de 2. Se usa la mediana de O como elemento separador para F. Si los dos subconjuntos resultado de la separación de F tienen el mismo tamaño, entonces el elemento faltante es la mediana de O. En otro caso, el elemento faltante está en el subarreglo de O correspondiene al subconjunto de F que tiene menor tamaño.

```
1  Algoritmo DC-faltante (O, F, n)
       Precondición:
         O es un arreglo ordenado, sin elementos repetidos, definido en el rango 1..n − 1.
         F es un conjunto que, excepto uno, tiene los mismos elementos que O.
2      si n = 2 entonces
           /* Caso base. En este algoritmo, podría no incluirse este caso base. */
3          devolver O[1]
4      m ← n/2
       /* m es el punto medio de 1..n − 1, O[m] es la mediana de O[1..n − 1] */
5      (F<, F>) ← separar(F, O[m])
6      si |F<| = |F>| entonces
7          devolver O[m]
       /* Casos recursivos */
8      si |F<| < |F>| entonces
9          Copiar O[1..m − 1] en O'
10         devolver DC-faltante (O', F<, n/2)
11     en otro caso
12         Copiar O[m + 1..n − 1] en O'
13         devolver DC-faltante (O', F>, n/2)
```

Algunas observaciones.

- El tiempo de ejecución de ordenar es $\Theta(n \log n)$, por lo que ordenar F para, al obtenerlo como una secuencia, poder recorrerlo buscando el elemento faltante, impide cumplir el orden $O(n)$.
- Los elementos de O y de F son genéricos de algún dominio en el que está definido un orden. Pero no está definida la suma entre sus elementos.
- F es un conjunto, no un arreglo. No se puede acceder a sus elementos mediante índices.
- El segundo parámetro de `separar`, s, es un elemento del mismo dominio que los de F, no es un entero, y en particular no es un índice del arreglo O.
- Aunque los elementos de O y F fueran enteros, en general m (un índice que puede ser el punto medio de un segmento) no tiene el mismo valor que $O[m]$. El parámetro a pasar a `separar` debe ser $O[m]$, no m.
- El primer parámetro de `separar` es un conjunto, no un arreglo. Además hay que recordar que O ya está ordenado, por lo que sería no se necesita `separar` para obtener los menores y mayores que $O[m]$.

(b) La relación de recurrencia es:

$$T(2) \leq O(1),$$
$$T(n) \leq T(n/2) + O(n).$$

En el libro de referencia del curso se muestra, en (5.5), que $T(n)$ es $O(n)$.

El algoritmo tiene un caso base para $n = 2$ (que implica que O tiene solo un elemento y F es vacío) si se cumple la condición de la línea 2 (podría no incluirse, porque si $n = 2$ el conjunto F es vacío, y entonces también lo son $F_>$ y $F_>$, por lo que se cumple la condición de la línea 6, y se devuelve $O[1]$). Esto se resuelve con operaciones elementales (una comparación y un acceso al arreglo O) por lo que es $O(1)$. Este caso base corresponde a la condición inicial de la relación de recurrencia. .

En el caso general hay una cantidad fija de operaciones elementales y una invocación a *separar* en la línea 5. El orden de este fragmento del algoritmo es $O(n)$ debido a que F tiene $n - 2$ elementos, y el orden de *separar* es lineal. Si se cumple la condición de la línea 6 el algoritmo termina.

En otro caso se hace exactamente una invocación recursiva. Como se hace noter en la letra, el tamaño del arreglo en esta invocación es $n/2 - 1$ lo que según lo definido corresponde a una

instancia de tamaño $n/2$. Junto a la invocación recursiva se agregan una operación elemental (comparación) y la copia de un segmento de largo $n/2 - 1$. El orden de estas operaciones es, entonces, $O(n)$.

Por lo tanto, en el caso general del algortimo hay

- una invocación recursiva de tamaño $n/2$ y
- un cantidad fija de operaciones cada una de las cuales tiene orden acotado superiormente por $O(n)$.

Los términos del caso general de la relación de recurrencia se corresponden con estos dos items.

Aunque no se pide, veamos que el algoritmo es correcto.

En la invocación inicial, el tamaño de la instancia es, por definición, potencia de dos. Como se ha visto, en las invocaciones recursivas el tamaño también es potencia de dos. Por lo tanto, si el elemento faltante no se encontró antes se va a llegar al caso base $n = 2$ en el que el algoritmo termina.

En el caso base F es vacío y O tiene un único elemento, que, por lo tanto, es el elemento faltante. Y es el que devuelve el algoritmo.

En el caso genérico, $O[m]$ es la mediana de O, que está ordenado, por lo que en $O[1\,..\,m-1]$ están los elementos estrictamente menores que $O[m]$, y en $O[m+1\,..\,n-1]$ los estrictamente mayores, y los dos subarreglos tienen $m-1$ elementos. Como F es un subconjunto de los elementos de O, y por definición de `separar`, en $F_<$ quedan los elementos de F estrictamente menores a $O[m]$, $F_<$ es un subconjunto de los elementos de $O[1\,..\,m-1]$. Además, como en F solo falta un elemento de O, $|F_<|$ es $m-2$ si el elemento faltante está en $O[1\,..\,m-1]$, o $m-1$ en otro caso. Por argumentos similares $F_>$ es un subconjunto de los elementos de $O[m+1\,..\,n-1]$ y $|F_<|$ es $m-2$ si el faltante está en $O[m+1\,..\,n-1]$, o $m-1$ en otro caso.

El, único, elemento faltante puede ser $O[m]$. Si es así $|F_<| = |F_>|$, se cumple la condición de la línea 6 y el algoritmo termina devolviendo $O[m]$.

En otro caso el elemento faltante está en $O[1\,..\,m-1]$ o en $O[m+1\,..\,n-1]$. Si está en $O[1\,..\,m-1]$ entonces $|F_<| = m-2 < m-1 = |F_>|$ y se hace la invocación recursiva con el subarreglo $O[1\,..\,m-1]$ y $F_<$. Se debe notar que se cumplen las precondiciones del algoritmo, siendo m potencia de 2, el arreglo O' tiene $m-1$ elementos y el conjunto $F_<$ tiene todos, excepto uno, todos los elementos de O'. Valen argumentos similares si el elemento faltante está en $O[m+1\,..\,n-1]$.

Otra versión En esta versión no se copian segmentos de O, sino que se indican mediante el inicio y el final. Esto no cambia el orden del tiempo de ejecución porque la separación de F, que se mantiene, tiene es del mismo orden que la copia.

(a)

```
1  Algoritmo Faltante (n, O, F)
       Precondición:
         n es potencia de 2,
         O es un arreglo ordenado, sin elementos repetidos, definido en el rango 1 .. n − 1,
         F es un conjunto que, excepto uno, tiene los mismos elementos que O[1 .. n − 1].
2      devolver DC-faltante (O,1,n-1,F)
3  Subalgoritmo DC-faltante (O, ínf, sup, F)
       Precondición:
         sup − ínf +2 es potencia de 2,
         O es un arreglo ordenado, sin elementos repetidos, definido en el rango ínf .. sup,
         F es un conjunto que, excepto uno, tiene los mismos elementos que O[ínf .. sup].
4      si inf = sup entonces
           /* Caso base */
5          devolver O[inf]
6      m ← (ínf + sup)/2
7      (F<, F>) ← separar(F, O[m])
8      si |F<| = |F>| entonces
9          devolver O[m]
       /* Casos recursivos */
10     si |F<| < |F>| entonces
11         devolver DC-faltante (O[ínf .. m − 1],F<)
12     en otro caso
13         devolver DC-faltante (O[m + 1 .. sup],F>)
```

(b) Similar a la otra versión.