SumaGolosa

Enunciado

Queremos encontrar la suma de los elementos de un multiconjunto de n números naturales. Cada suma se realiza exactamente entre dos números, x e y, y tiene costo $x + y$.

Por ejemplo, si queremos encontrar la suma de $\{1, 2, 5\}$, tenemos 3 opciones:

- 1 + 2 (con costo 3) y luego 3 + 5 (con costo 8), resultando en un costo total de 11;
- 1 + 5 (con costo 6) y luego 6 + 2 (con costo 8), resultando en un costo total de 14;
- 2 + 5 (con costo 7) y luego 7 + 1 (con costo 8), resultando en un costo total de 15.

Queremos encontrar la forma de sumar que tenga costo mínimo, por lo que en nuestro ejemplo la mejor forma sería la primera.

a. Explicitar una estrategia golosa para resolver el problema.
b. Demostrar que la estrategia propuesta resuelve el problema.
c. Implementar esta estrategia en un algoritmo iterativo. Nota: el mejor algoritmo simple que conocemos tiene complejidad $O(n \log n)$ y utiliza una estructura de datos que implementa una secuencia ordenada.

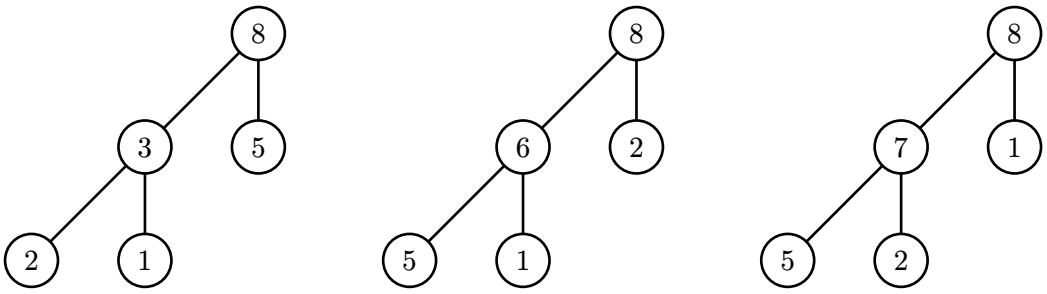
Resolución

a. Vamos a considerar al multiconjunto que nos pasan como una lista de enteros. Una estrategia para resolver esto es mantener una lista de sumandos. $n - 1$ veces vamos a tomar los dos sumandos más pequeños que haya en nuestra lista, x e y, removerlos de la lista, y agregar $x + y$ a la lista. Cada vez que hacemos esto, sumamos $x + y$ a nuestro costo total (que comienza en cero), que vamos a devolver. Asimismo, cada vez que hacemos esto la longitud de la lista se reduce en 1, y luego de las $n - 1$ repeticiones, vamos a tener un sólo elemento. Devolvemos entonces el costo total.

   En pseudocódigo:

```
1:  procedure Suma(m ∈ ℤ^n)
2:      ▷ Inicializamos q teniendo todos los elementos de m.
3:      q ← Min-Heap-Create(m)
4:      c ← 0
5:      for 0 ≤ i < n − 1 do
6:          ▷ Sacamos los dos elementos más pequeños de q, x e y.
7:          x ← Min-Heap-Pop(q)
8:          y ← Min-Heap-Pop(q)
9:          z ← x + y
10:         ▷ Agregamos a q la suma de x e y.
11:         Min-Heap-Push(Q, z)
12:         c ← c + z
13:     end
14:     return c
15: end
```

b. Podemos entender lo que hace nuestro algoritmo como construir un árbol binario, donde cada hoja es un elemento de la lista original, y un padre con dos hijos x e y tiene valor $x + y$. El costo del árbol es la suma de los valores de los vértices internos. Por ejemplo, para el multiconjunto que nos dan en el enunciado, $\{1, 2, 5\}$, las tres formas de sumarlo se pueden ver como estos tres árboles:



   El costo del primer árbol es $3 + 8 = 11$, el del segundo es $6 + 8 = 14$, y el del tercero es $7 + 8 = 15$. En general, podemos ver que el costo del árbol va a ser la suma de los valores todos sus vértices, menos la suma de la lista original. Para el primer árbol, esto es $2 + 1 + 3 + 5 + 8 - 2 + 1 + 5 = 11$. Como queremos minimizar la suma del costo del árbol, pero *todos* los árboles van a tener el mismo multiconjunto de valores de hojas, este último es una constante. Luego podemos minimizar la suma de los valores de todo el árbol, y va a ser lo mismo que minimizar la suma de todos los valores *internos* del árbol.

   Vamos a probar, entonces, que nuestro algoritmo crea un árbol de sumas, de mínima suma de costos entre todos los árboles de suma para ese multiconjunto.

   Lema 1. El algoritmo es correcto para $n \leq 1$.
   Demostración. Esto es obvio, puesto que no entramos nunca al ciclo, y devolvemos cero. Cero es

efectivamente el mínimo número de sumas que hay que hacer, si nos dan cero o un número como entrada.

En lo que sigue, entonces, vamos a asumir que $n \geq 2$.

Definición 1. Dado un árbol de sumas T, $\mathbf{costo(T)}$ se define como la suma de los valores de los vértices de T que no son hojas.

Definición 2. Dado un multiconjunto de números M, llamamos a un árbol binario T **árbol de sumas para M** cuando los valores de las hojas de T son los elementos de M, y los valores de los vértices internos de T son la suma de los valores de sus hijos.

Definición 3. Dado un multiconjunto de números M, y un árbol de sumas T para M, decimos que T es **óptimo para M** cuando T tiene el mínimo $\text{costo}(T)$ entre todos los árboles de sumas para M.

Lema 2. Sea un M multiconjunto de números. Existe un árbol de sumas óptimo para M.
Demostración. Primero vemos que si $\mathcal{T}(M)$ es el conuunto el conjunto de árboles de sumas para M, entonces $\mathcal{T}(M)$ no es vacío. Podemos, por ejemplo, crear un árbol T que representa $(...((((M_1 + M_2) + M_3) + M_4) + M_5) + ...)$, donde M_i son los elementos de M. Este árbol tiene como hojas a los elementos de M, y cada vértice tiene como valor la suma de sus dos hijos (por ser una suma de dos sub-árboles). Por ende, es un árbol de sumas para M. Como existe al menos un árbol de sumas para M, vemos que $\mathcal{T}(M) \neq \emptyset$. También vemos que $\mathcal{T}(M)$ es finito, puesto que hay sólo finitas formas de poner paréntesis anidados en la expresión $M_1 + M_2 + M_3 + ... + M_n$. Luego, la función costo alcanza su máximo en $\mathcal{T}(M)$, y luego existe al menos un árbol de sumas óptimo para M.

Lema 3. Sea M un multiconjunto de números, y T un árbol de sumas para M. Para cada $x \in M$, sea $d_T(x)$ la longitud del único camino en T entre la raíz de T y x. Es decir, $d_T(x)$ es la altura de x en T. Entonces

$$\text{costo}(T) = \sum_{x \in M} x \cdot d_T(x)$$

Demostración. Consideremos el camino en T entre la raíz y x. Cada vértice interno va a tener, como valor, la suma de sus dos hijos. Luego, en este camino, x va a ser sumado una vez por cada vértice que aparezca, puesto que cada vértice acumula en su valor los valores de todos sus hijos. Luego, si sumamos $x \cdot d(x)$ por cada $x \in M$, vamos a tener la suma de los valores de todos los vértices internos de T, que es precísamente el costo de T.

Lema 4. Sea M un multiconjunto de números con $|M| \geq 2$. Sean x e y dos elementos de mínimo valor en M. Entonces existe un árbol de sumas óptimo para M donde x e y son hojas hermanas, y están a distancia máxima de la raíz, entre todas las hojas.
Demostración. Consideremos cualquier árbol de sumas óptimo T para M.

Este árbol tiene hojas hermanas, porque en algún momento va a tener que tener una suma que no tiene sumas como hijos, siendo el árbol finito. Tomemos entonces dos hojas hermanas, y de todas las hojas hermanas, elegimos el par que estén a la máxima distancia de la raíz. Sean a, b estas hojas.

Si $(a, b) = (x, y)$, o $(a, b) = (y, x)$, terminamos. Si no, sin pérdida de generalidad asumo que $a \leq b$ (si no, intercambiamos sus nombres). Igualmente sin pérdida de generalidad asumimos que $x \leq y$ (si no, intercambiamos sus nombres). Como x e y eran los dos elementos más pequeños de M, y a y b son elementos de M, tenemos que $x \leq a$, y que $y \leq b$.

Consideremos ahora T', igual a T, pero cambiando de lugar la hoja x, y la hoja a. Como T es óptimo para M, y T' es un árbol de sumas para M, entonces $\text{costo}(T) \leq \text{costo}(T')$. Usando el **Lema 3**, tenemos $\text{costo}(T') = \text{costo}(T) - x \cdot d_T(x) - a \cdot d_T(a) + x \cdot d_T(a) + a \cdot d_T(x) = \text{costo}(T) - (a - x)(d_T(a) - d_T(x))$. Como $x \leq a$, $a - x \geq 0$. Como a era una hoja de máxima distancia hasta la raíz, $d_T(a) \geq d_T(x)$. Luego $(a - x)(d_T(a) - d_T(x)) \geq 0$, y luego $\text{costo}(T') \leq \text{costo}(T)$. Como sabíamos que $\text{costo}(T) \leq \text{costo}(T')$, tenemos que $\text{costo}(T) = \text{costo}(T')$. Luego T' es un árbol de sumas óptimo para M.

Por el mismo argumento, podemos construir T'', que es T', cambiando la hoja b y la hoja y de lugar, y obtenemos $\text{costo}(T'') = \text{costo}(T)$, luego T'' es un árbol de paréntesis óptimo para M, donde x e y son hojas a la máxima distancia desde la raíz.

Teorema del invariante

Como siempre que tenemos un ciclo, vamos a usar el teorema del invariante para probar que es correcto. Para esto tenemos que definir 5 cosas:

1. Una precondición. Esto es algo que vale antes de correr el ciclo. En nuestro caso, la precondición es que $i = 0$, que $c = 0$, y que $q = m$. Por $q = m$ nos referimos a que los elementos de q son los mismos elementos de m.
2. Una postcondición. Esto es algo que vale luego de finalizar el ciclo. En nuestro caso, la postcondición es que existe un árbol de sumas T óptimo para m, cuyo costo es c. Como devolvemos c, si probamos que la postcondición vale al terminar el ciclo, estamos probando que nuestro algoritmo es correcto.
3. Un invariante del ciclo. Esto es algo que vale antes y después de cada iteración del ciclo. En nuestro caso, el invariante es que $0 \leq i < n$, que $|q| = n - i$, y que existe un árbol de sumas T, óptimo para m, y un árbol de sumas T', óptimo para q, tal que $\text{costo}(T) = \text{costo}(T') + c$.:
4. Una guarda. Esto es algo que vale cada vez que entramos al ciclo, y está dado sintácticamente por la condición del ciclo. En nuestro caso esto es $i < n - 1$.
5. Una función variante. Esto es un número que decrece con cada iteración del ciclo. En nuestro caso esto es $n - i$.

Probemos, entonces, las cosas que hay que probar para usar el teorema del invariante.

La precondición vale

En nuestro caso esto es simple, dado que $i = 0$ es el valor inicial de i, que 0 es el valor inicial de c, y que construímos q teniendo exactamente los elementos de m.

La precondición implica el invariante

Como sabemos que $i = 0$ al comenzar el ciclo, y asumimos por el **Lema 1** que $n \geq 2$, claramente vale $0 \leq i < n$. Asimismo, como $q = m$, tenemos que $|q| = n$, y como $i = 0$, esto implica que $|q| = n - i$. Por último, por el **Lema 2** sabemos que existe al menos un árbol de sumas óptimo para M. Sea T un tal árbol. Podemos tomar $T' = T$, que al ser óptimo para m, y $m = q$, resulta ser también óptimo para q. Finalmente, como $c = 0$, tenemos que $\text{costo}(T) = \text{costo}(T') + c$.

La función variante decrece en cada iteración

Esto es obvio por cómo funcionan los ciclos. En cada iteración aumenta i, y por lo tanto la función variante, $n - i$, decrece.

Si la función variante es cero, la guarda no se cumple

Como la función variante es $n - i$, si la función variante es cero entonces $n - i = 0$, y luego $i = n$. La guarda es $i < n - 1$. Efectivamente, vemos que $i < n - 1$ no es cierto cuando $i = n$.

El invariante es preservado por las iteraciones
Saquémonos de encima la parte simple del invariante. Como sabemos que vale el invariante al comenzar el ciclo, tenemos que $0 \leq i < n$. También sabemos que, al haber entrado en el cuerpo del ciclo, vale la guarda, y luego $i < n - 1$. Llamemos i' al estado de i al terminar el cuerpo del ciclo. Sabemos que $i' = i + 1$, por cómo funcionan los ciclos. Como teníamos que $i < n - 1$, debemos tener que $0 \leq i' < n$. También, si llamamos q' al estado de q al terminar el cuerpo del ciclo, tenemos que $|q'| = |q| - 1$, puesto que estamos sacando dos cosas de q, y agregando una. Luego, Pero como $|q| = n - i$, esto es lo mismo que $|q'| = (n - i) - 1 = n - (i + 1) = n - i'$, y por lo tanto las dos partes sencillas del invariante se cumplen.

Ahora viene la parte principal del teorema, realmente la única que no es trivial, que es ver que el valor de c sigue siendo el que queremos.

Sabemos que existe un árbol de sumas T, óptimo para m, y un árbol de sumas T', óptimo para q, tal que $\text{costo}(T) = \text{costo}(T') + c$. Sean x e y los dos elementos más pequeños de q. El cuerpo del ciclo construye $q' = q \setminus \{x, y\} \cup \{x + y\}$, y le asigna q' a q, y también construye $c' = c + x + y$, y le asigna c' a c.

Vamos a ver que existe un árbol de sumas T'', óptimo para q', tal que $\text{costo}(T'') = \text{costo}(T') - (x + y)$. Con un poco de aritmética vemos que:

$$\begin{aligned}
\text{costo}(T'') + c' &= \text{costo}(T') - (x + y) + c' \\
&= \text{costo}(T) - c - (x + y) + c' \\
&= \text{costo}(T) - c - (x + y) + (c + (x + y)) \\
&= \text{costo}(T)
\end{aligned}$$

Luego, si probamos que existe tal T'', vemos que el invariante queda reestablecido, puesto que T'' es óptimo para q', y $\text{costo}(T) = \text{costo}(T'') + c'$, que es lo que pide el invariante al terminar el cuerpo del ciclo.

Por el **Lema 4**, sabemos que existe un árbol de sumas T^*, óptimo para q, donde x e y son hojas hermanas, a distancia máxima de la raíz. Sea z el padre de x e y. Sea entonces T'' igual a T^*, excepto que reemplazamos a z por una hoja de valor $x + y$. Tenemos que $\text{costo}(T'') = \text{costo}(T^*) - (x + y)$, porque ahora z es una hoja, y el costo de un árbol no incluye los valores de las hojas.

Veamos que T'' es óptimo para q'. Sea S un árbol de sumas óptimo para q'. Como $(x + y)$ es un elemento de q', hay una hoja con valor $x + y$ en S. Si expandimos esa hoja a un padre con valor $(x + y)$ y dos hijos, x e y, tenemos un árbol S' de sumas para q. Tenemos que $\text{costo}(S') = \text{costo}(S) + (x + y)$, por haber agregado un vértice interno $x + y$ al construir S'. Recordemos que T^* es óptimo para q. Luego, $\text{costo}(S') = \text{costo}(S) + (x + y) \geq \text{costo}(T^*)$. Luego:

$$\begin{aligned}
\text{costo}(S) + (x + y) &\geq \text{costo}(T^*) \\
\text{costo}(S) &\geq \text{costo}(T^*) - (x + y) \\
&= \text{costo}(T'')
\end{aligned}$$

Como S es óptimo para q', y T'' es un árbol de sumas para q', luego T'' también es óptimo para q'.

Finalmente, sabíamos que $\text{costo}(T'') = \text{costo}(T^*) - (x + y)$. Sabemos que T' es óptimo para q, y T^* también es óptimo para q, luego $\text{costo}(T') = \text{costo}(T^*)$. Entonces, $\text{costo}(T'') = \text{costo}(T') - (x + y)$.

Luego, encontramos un árbol de sumas T'', óptimo para q', tal que $\text{costo}(T'') = \text{costo}(T') - (x + y)$. Esto muestra que el invariante se mantiene al finalizar el cuerpo del ciclo.

El invariante y la negación de la guarda implican la postcondición
Como vale el invariante, sabemos que $i < n$. Como vale la negación de la guarda, sabemos que $i \geq n - 1$. Luego, sabemos que $i = n - 1$. Como vale el invariante, sabemos que existe un árbol de sumas T, óptimo para m, y un árbol de sumas T', óptimo para q, tal que $\text{costo}(T) = \text{costo}(T') + c$. Pero como vale el invariante, $|q| = n - i = n - (n - 1) = 1$, y por ende T' tiene un sólo elemento, la raíz, que es una hoja por no tener hijos. Por ende, $\text{costo}(T') = 0$, y tenemos que existe un árbol de sumas T, óptimo para m, tal que $\text{costo}(T) = c$. Esto es precísamente la postcondición.

c. El algoritmo en pseudocódigo es el que mostramos en a). Usamos la estructura *heap* para obtener el mínimo de un conjunto de k elementos en $O(\log k)$ operaciones, e insertar algo en el *heap* en $O(\log k)$ operaciones. Nuestro heap q tiene siempre a lo sumo n elementos, y hacemos tres operaciones en cada iteración del ciclo. Luego, en cada iteración hacemos $O(\log n)$ operaciones, y en total haremos $O(n \log n)$ operaciones.