Algoritmos y Estructuras de Datos II - 24 de julio de 2013

Examen Final Teórico-Práctico

Docentes: Daniel Fridlender, Silvia Pelozo y Alejandro Tiraboschi

Alumno: ..... Email: .....

Incluir SIEMPRE justificaciones de sus respuestas con la mayor claridad posible.

1. Sean $r(n) \in \mathcal{O}(f(n))$ y $s(n) \in \mathcal{O}(g(n))$. Determiná cuáles de las siguientes afirmaciones son verdaderas y cuáles falsas. Justificá apropiadamente.

   a) $r(n) * s(n) \in \mathcal{O}(f(n) * g(n))$

   b) $r(n) * g(n) \in \mathcal{O}(f(n) * s(n))$

   c) $r(n)/s(n) \in \mathcal{O}(f(n)/g(n))$

   d) $r(n)/g(n) \in \mathcal{O}(f(n)/s(n))$

2. Se desea realizar un viaje en un automóvil con autonomía a, desde la localidad l_1 hasta la localidad l_n pasando por las localidades $l_2, \ldots l_{n-1}$ en ese orden. La autonomía a y las distancias d_i de la localidad l_i a la siguiente (l_{i+1}) están dadas en kilómetros. Se tiene $d_i \leq a$ para todo i, esto garantiza que el viaje se puede realizar.

   En cada localidad es posible conseguir combustible, pero en todas ellas existen filas de vehículos para cargar. Cada una tiene asociado un tiempo t_i de demora estimada para poder cargar combustible en esa localidad. Se desea completar el recorrido minimizando el tiempo total de espera para cargar combustible.

   Cada vez que se carga combustible se llena el tanque, es decir, el vehículo podría de inmediato recorrer a kilómetros sin volver a cargar. Pero no está obligado a esperar que se vacíe para hacerlo, puede convenir hacerlo antes, incluso tal vez mucho antes. Por ejemplo si uno sabe que las siguientes estaciones tendrán mucha espera.

   Para encontrar el menor tiempo de espera posible para llegar a la localidad de destino (la localidad l_n) es necesario utilizar backtracking. Se pide que dés un algoritmo que resuelva dicho problema.

   Una posibilidad es definir $m(i, j) =$ "mínimo tiempo de espera necesario para llegar de la localidad l_i a la localidad l_n si se dispone de combustible suficiente en el tanque para recorrer j kilómetros sin cargar". Puede resultarte conveniente contemplar los siguientes casos:

   $$m(i,j) = \begin{cases} \ldots & i = n \\ \ldots & i < n \wedge j < d_i \\ \ldots & i < n \wedge j \geq d_i \end{cases}$$

   Si lo resolvés de esta manera, justificá detalladamente los valores elegidos en cada uno de esos tres casos. Si lo resolvés de otra manera, justificá también de manera similar.

   ¿Cuál sería la llamada principal a este algoritmo para que resuelva el problema planteado (calcular el menor tiempo total de espera posible)?

3. El algoritmo usual para encontrar el mínimo valor de un arreglo a : **array**$[1..N]$ **of int** utilizando la técnica de búsqueda lineal, realiza exactamente $N - 1$ comparaciones entre elementos del arreglo. De la misma manera, se puede computar el máximo con $N - 1$ comparaciones entre elementos del arreglo. Si quisieramos computar ambos valores, entonces simplemente podríamos utilizar ambos programas, obteniendo una solución que realiza $2N - 2$ comparaciones entre elementos del arreglo.

   Valiéndote de la técnica divide y vencerás, podés encontrar una solución levemente más eficiente, que realiza menos de $\frac{3}{2}N$ comparaciones entre elementos del arreglo. Es decir, es también lineal, pero las constantes son mejores.

   Escribí dicho algoritmo, cuyo encabezado es **proc** $minmax$(**in** a : **array of int**; **out** min, max : **int**) y debe computar simultáneamente el mínimo y el máximo valor del arreglo a en los parámetros min y max utilizando divide y vencerás. La clave para obtener la solución que realiza menos de $\frac{3}{2}N$ comparaciones entre elementos del arreglo está en considerar el caso de dos celdas como uno de los casos base.

   Contá el número de comparaciones que realiza tu algoritmo para N igual a 0, 1, 2, 4, 8, 16, 32 y 64. Y contrastalo con el algoritmo basado en dos búsquedas lineales mencionado más arriba.

   ¿Cuántas comparaciones entre elementos del arreglo realiza tu algoritmo para N potencia de 2?

4. Se define el TAD silueta, que representa la silueta que dejan los edificios de una ciudad en el horizonte.

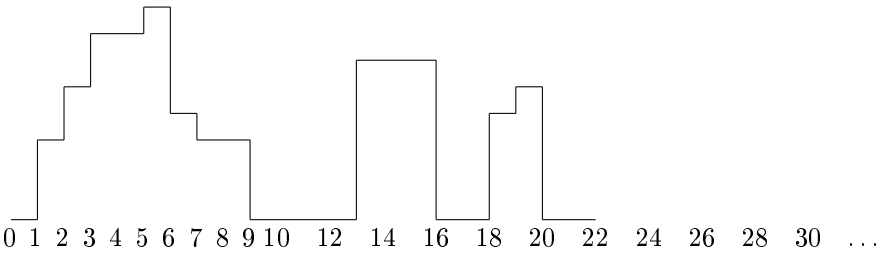


Figura 1:

El tipo abstracto tiene 2 constructores: uno para crear la silueta vacía, es decir, sin edificios y otro para agregar un edificio a la izquierda de una silueta preexistente. Este último constructor tiene por argumento dos naturales a y h y una silueta s y agrega a la izquierda de s un edificio de ancho a y altura h.

TAD silueta
constructores
vacía : silueta
ag_edif : natural × natural × silueta → silueta
operaciones
un_edif : natural × natural × natural → silueta
levantar : silueta → silueta
hundir : silueta → silueta
... *otras operaciones*
ecuaciones
ag_edif(a,h,ag_edif(b,h,s)) = ag_edif(a+b,h,s)
ag_edif(0,h,s) = s

un_edif(d,a,h) = ag_edif(d,0,ag_edif(a,h,vacia))

levantar(vacía) = vacía
levantar(ag_edif(a,h,s)) = ag_edif(a,h+1,levantar(s))

hundir(vacía) = vacía
hundir(ag_edif(a,0,s)) = ag_edif(a,0,hundir(s))
h > 0 ⟹ hundir(ag_edif(a,h,s)) = ag_edif(a,h-1,hundir(s))

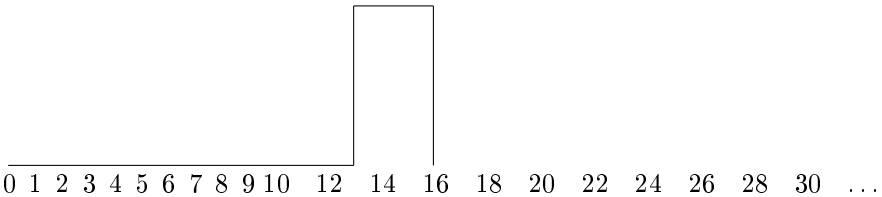
... *ecuaciones de otras operaciones*

Así, la silueta de la Figura 1 puede ser construida como
s = ag_edif(1,0,ag_edif(1,3,ag_edif(1,5,ag_edif(2,7,ag_edif(1,8,ag_edif(1,4,ag_edif(2,3,s')))))))
donde s' = ag_edif(4,0,ag_edif(3,6,ag_edif(2,0,ag_edif(1,4,ag_edif(1,5,ag_edif(2,0,vacía)))))). La utilización de s' se debe únicamente a que no hay suficiente espacio en una sola línea para toda la definición de s.

Una misma silueta puede construirse de diferentes maneras, por ejemplo
s' = ag_edif(4,0,ag_edif(1,6,ag_edif(2,6,ag_edif(2,0,ag_edif(0,8,ag_edif(1,4,ag_edif(1,5,ag_edif(2,0,vacía)))))))).

La primera ecuación del TAD dice justamente que dos edificios adyacentes de igual altura producen el mismo efecto que un solo edificio suficientemente ancho. Y la segunda ecuación dice que un edificio de ancho 0 no afecta la silueta. No así un edificio de altura 0 pero ancho positivo, ya que puede funcionar como separador de dos edificios.

La operación un_edif se usa para obtener la silueta de un edificio solo. Por ejemplo, un_edif(13,3,6) es la silueta



a) Implementá el TAD silueta utilizando arreglos según la siguiente definición. Se utiliza el nombre skyline para la representación de la silueta.

```
type building = tuple
                    width: nat
                    height: nat
                end
type skyline = tuple
                    buildings: array[1..N] of building
                    size: nat
                end
```

Con esta implementación, la silueta s de la Figura 1 se representaría de la siguiente manera:

s.buildings	2	1	1	2	3	4	2	1	1	2	1	1	1																
	0	5	4	0	6	0	3	4	8	7	5	3	0																
s.size	13																												

width
height

Ésta es la representación simplificada. Otra representación (no simplificada) es posible reemplazando cualquier celda con campo width mayor que dos, por dos celdas contiguas con campo width apropiado e igual altura, y/o intercalando un número cualquiera de celdas con campo width igual a 0. Por ejemplo:

s.buildings	0	2	1	1	2	1	2	4	2	0	1	1	0	2	1	1	1	0											
	3	0	5	4	0	6	6	0	3	9	4	8	4	7	5	3	0	1											
s.size	18																												

width
height

donde se han agregado cuatro celdas con campo width igual a 0 y se ha reemplazado una con campo width igual a 3 y height igual a 6 por dos contiguas con campo height igual a 6 y width igual a 1 y 2 respectivamente. **Implementá** un procedimiento **auxiliar simplify**, que modifica su argumento transformándolo en su representación simplificada: sin celdas con width igual a 0 ni celdas consecutivas de igual height. Recordá que el campo size final debe consignar el número de celdas ocupada por la representación simplificada.

Una manera de hacerlo es recorriendo dos veces el arreglo: la primera, removiendo celdas con width igual a 0 y la segunda, convirtiendo celdas contiguas de igual height en una sola celda.

Implementá todas las operaciones del TAD silueta. Se asume que las operaciones que reciben una silueta la reciben simplificada, y las que devuelven siluetas, deben devolverlas también simplificadas, por ejemplo, valiéndose del procedimiento simplify.

b) Agregá a la especificación una operación que devuelva la altura máxima de la silueta, otra que devuelva la superficie bajo la silueta, otra que permita agregar un edificio a la derecha de una silueta.

5. (para alumnos libres) Da la forma general de los algoritmos voraces. Explica detalladamente la forma general y cada uno de sus componentes.