UNIVERSIDAD SIMÓN BOLIVAR **21 de octubre de 2009**
CI-7621 - Teoría de Algoritmos **Lic. Alejandro Claro Mosqueda**

Tarea #1

Problema #1: Muestre como implementar una cola (queue) con dos pilas (stack) ordinarias de tal manera que el costo amortizado de cada operación ENCOLAR (ENQUEUE) y cada operación DESENCOLAR (DEQUEUE) es O(1). (Ejercicio 17.3-6 del Cormen)

Solución:

Denotemos las dos pilas como S_1 y S_2. La operación ENCOLAR es simplemente insertar el objeto en la pila S_1.

```
ENCOLAR(Q, x)
1  PUSH(Q.S_1, x)
```

La operación DESENCOLAR consiste simplemente en retirar el objeto superior de la pila S_2 si dicha pila no esta vacía. Si la pila S_2 esta vacía, primero se retira de forma sucesiva el objeto superior de S_1 y se inserta en S_2 hasta que S_1 este vacía. Esto invierte el orden de los objetos de S_1 dentro de S_2.

```
DESENCOLAR(Q)
1  si Q.S_2 está vacía entonces
2          mientras Q.S_1 no esté vacía hacer
3                  x ← POP(Q.S_1)
4                  PUSH(Q.S_2, x)
5  devolver POP(Q.S_2)
```

Es bastante claro que el costo amortizado de una operación ENCOLAR es $O(1)$ debido a que la operación es simplemente insertar el objeto en la pila S_1.

En cambio, el costo amortizado de la operación DESENCOLAR, no es tan evidente. Asignemos un costo de 4 créditos a cada operacion ENCOLAR y 0 créditos a cada operación DESENCOLAR. Cuando se encola un objeto se cancela 1 crédito al insertar el objeto en S_1 y se guardan los otros 3 junto al objeto en la pila. Ahora, al desencolar un objeto se debe cancelar 1 crédito por la operación POP en S_1 y 1 crédito por la operación PUSH en S_2 por cada objeto durante el proceso que invierte el orden de los objetos; y finalmente 1 crédito por la operación POP en S_2 del objeto a desencolar. Con este razonamiento se observa que la cantidad de créditos nunca se hace negativo y que efectivamente el costo amortizado de la operación DESENCOLAR es $O(1)$.

Problema #2: Mostrar que la altura de un árbol AVL con n nodos es $O(\log n)$. (ayuda: use inducción sobre n, ó establezca una recurrencia para T_n = número mínimo de nodos de un árbol AVL de altura h, resuélvala y utilícela). Un árbol AVL es un árbol binario de búsqueda donde para cada nodo la diferencia de las alturas de sus dos sub-árboles es a lo sumo 1. Considere que el árbol vacío posee altura -1.

Solución:

Proposición. Sea A_h un árbol AVL de altura h que tiene el número mínimo de nodos, y A_I y A_D los sub-árboles de A_h. (Figura 1)

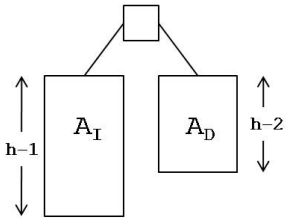

Figura 1: Árbol AVL

Entonces, el sub-árbol A_I o el sub-árbol A_D debe tener altura $h-2$, puesto que por definición se debe cumplir que las alturas del sub-árbol izquierdo y el sub-árbol derecho no deben diferir en más de 1 en todo árbol AVL. De esta manera es bastante claro que la fórmula de recurrencia para el número de nodos en función de la altura es,

$$n_h = \begin{cases} 1 & \text{si } h = 0, \\ 2 & \text{si } h = 1, \\ n_{h-1} + n_{h-2} + 1 & \text{si } h > 1 \end{cases} \tag{1}$$

donde n_h denota el número de nodos en el árbol de altura h. Obsérvese que al sumar 1 en ambos lados a la ecuación (1), se obtiene la fórmula de recurrencia de los números de Fibonacci con el índice desplazado 3 posiciones.

$$n_h + 1 = \begin{cases} 2 & \text{si } h = 0, \\ 3 & \text{si } h = 1, \\ (n_{h-1} + 1) + (n_{h-2} + 1) & \text{si } h > 1 \end{cases} \tag{2}$$

Utilizando la fórmula de De Moivre (Brassard pp. 32) se obtiene que,

$$n_h + 1 \approx \frac{\left(\frac{1+\sqrt{5}}{2}\right)^{h+3}}{\sqrt{5}} \qquad h \gg 1 \tag{3}$$

$$h \approx 1{,}4 \log n_h \tag{4}$$

Con lo que queda demostrado que la altura de un árbol AVL con n_h nodos es $O(\log n_h)$. □

Problema #3a: Dar una instancia sencilla (4 objetos y números enteros) del problema de la mochila 0-1 donde al aplicar el algoritmo Greedy considerando los elementos por orden no creciente, o no decreciente, de peso; por orden no creciente, o no decreciente, de beneficio; ó por orden no creciente, o no decreciente, de beneficio por peso unitario (v_i/w_i), ninguno produce la solución óptima.

Solución:

En el siguiente cuadro se muestra una instancia de 4 objetos donde ninguna de las estrategias mencionadas produce la solución óptima si la mochila tiene capacidad 100.

w_i	v_i	v_i/w_i	Max v_i	Min v_i	Max w_i	Min w_i	Max v_i/w_i	Min v_i/w_i	óptimo
100	400	4	1	0	1	0	0	1 (0)	0
50	350	7	0	0	0	0	1	0 (0)	1
45	180	4	0	1	0	1	0	0 (1)	1
20	100	5	0	1	0	1	1	0 (1)	0
Peso total			100	65	65	100	70	100 (65)	95
Beneficio total			400	280	280	400	450	400 (280)	530

Cuadro 1: Ejemplo del problema de la mochila 0-1 ($C = 100$)

Problema #3b: En la sección 6.5 del Bassard se suponía que estaban disponibles n objetos, numerados de 1 a n. Supongamos en cambio que tenemos n tipos de objetos disponibles, con un suministro adecuado de objetos de cada tipo. Formalmente, esto sólo cambia la restricción anterior $0 \leq x_i \leq 1$ por la restricción más permisiva $x_i \geq 0$. ¿Sigue funcionando el algoritmo voraz de la sección 6.5? ¿sigue siendo necesario? (Ejercicio 6.18 del Brassard)

Solución:

En esta situación el algoritmo de la sección 6.5 no funciona sin modificaciones, o sin insertar todo el suministro "adecuado" de cada tipo de objeto a la lista de objetos, puesto que el bucle voraz impone la restricción $0 \leq x_i \leq 1$ entre las líneas 6 y 11 (ver algoritmo MOCHILA al final de problema). Modificar el algoritmo carece de sentido debido a que ya no es necesario. Es posible llenar la mochila únicamente con el tipo de objeto (k) con mayor beneficio por unidad de peso,

$$v_{\text{MOCHILA}} = W \frac{v_k}{w_k} \tag{5}$$

lo cual es la solución óptima del problema.

Demostración. Supongamos, sin pérdida de generalidad, que los objetos disponibles están ordenados por valor decreciente de beneficio por unidad de peso, esto es, que

$$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n} \tag{6}$$

Sabemos que el beneficio total en la mochila es,

$$v_{\text{MOCHILA}} = \sum_{i=1}^{n} x_i v_i \tag{7}$$

donde $v_i > 0$, $w_i > 0$, $x_i \geq 0$. Además, se debe cumplir que

$$\sum_{i=1}^{n} x_i w_i \leq W \tag{8}$$

donde $W > 0$ es la capacidad de la mochila. Reacomodando la ecuación (7) se obtiene que

$$v_{\text{MOCHILA}} = \sum_{i=1}^{n} (x_i w_i) \frac{v_i}{w_i} \tag{9}$$

Como es cierto que, $\frac{v_j}{w_j} \leq \frac{v_1}{w_1} \ \forall\, j > 1$, entonces

$$\sum_{i=1}^{n} (x_i w_i) \frac{v_i}{w_i} \leq \frac{v_1}{w_1} \sum_{i=1}^{n} (x_i w_i) \tag{10}$$

Finalmente, haciendo uso de la restricción (8), se concluye que

$$\sum_{i=1}^{n} (x_i w_i) \frac{v_i}{w_i} \leq W \frac{v_1}{w_1} \tag{11}$$

$$v_{\text{MOCHILA}} \leq \frac{W}{w_1} v_1 \tag{12}$$

Como la única restricción sobre x_i es $x_i \geq 0$, queda demostrado que la solución óptima es

$$x_i = \begin{cases} W/w_i & \text{si } i = 1, \\ 0 & \text{si } i > 1 \end{cases} \tag{13}$$

$$v_{\text{MOCHILA}} = W \frac{v_1}{w_1} \tag{14}$$

□

```
MOCHILA(w[1...n], v[1...n], W)
    ▷ inicialización
 1  para i ← 1 hasta n hacer
 2          x_i ← 0
 3  peso ← 0

    ▷ bucle voraz
 4  mientras peso < W hacer
 5          i ← mejor objeto restante
 6          si peso + w[i] ≤ W entonces
 7                  x[i] ← 1
 8                  peso ← peso + w[i]
 9          sino
10                  x[i] ← (W − peso) / w[i]
11                  peso ← W
12  devolver x
```

Problema #4: Dado un grafo no dirigido simple (sin aristas múltiples) $\mathbf{G} = (\mathbb{V}, \mathbb{E})$, con $\mathbb{E}$ no vacío, la matroide de cociclos $\mathbf{M} = (\mathbb{E}, \mathbb{F})$ de $\mathbf{G}$ tiene como conjunto base a $\mathbb{E}$ y un independiente es un conjunto de aristas que al eliminarlas simultáneamente de $\mathbf{G}$ no aumenta el número de componentes conexas.

a) Muestre que $\mathbf{M} = (\mathbb{E}, \mathbb{F})$ es una matroide.

Solución:

Para demostrar que $M = (\mathbb{E}, \mathbb{F})$ es una matroide basta con demostrar que el conjunto $\mathbb{E}$ es no vacío (suposición hecha en el enunciado del problema) y que el conjunto $\mathbb{F}$ cumple las siguiente propiedades:

1. $\emptyset \in \mathbb{F}$
2. Si $F \in \mathbb{F} \wedge F' \subseteq F$ entonces $F' \in \mathbb{F}$
3. Si $F, F' \in \mathbb{F} \wedge |F'| < |F|$ entonces $|F'| \cup \{x\} \in \mathbb{F}$ para algún $x \in F \backslash F'$

Es bastante claro que la primera propiedad se cumple. No eliminar aristas de $\mathbf{G}$ no aumenta el número de componentes conexas del grafo.

Por otro lado, el conjunto $\mathbb{F}$ es la colección de todos los sub-conjuntos E de $\mathbb{E}$ tal que el grafo $(\mathbb{V}, \mathbb{E} \backslash E)$ tiene el mismo número de componentes conexas que $\mathbf{G}$. Ahora, como $F' \subseteq F$, se observa que el grafo $(\mathbb{V}, \mathbb{E} \backslash F)$ tiene igual numero de componente conexas que el grafo $(\mathbb{V}, \mathbb{E} \backslash F')$. Entonces, por definición, $F' \in \mathbb{F}$. Quedando demostrada así la segunda propiedad.

Para demostrar la tercera propiedad, digamos, sin perdida de generalidad, que el grafo $\mathbf{G}$ tiene n componentes conexas y sean F y F' dos sub-conjuntos de $\mathbb{E}$ tales que los grafos $(\mathbb{V}, \mathbb{E} \backslash F)$ y $(\mathbb{V}, \mathbb{E} \backslash F')$ tienen n componentes; y ademas $|F'| < |F|$. Se quiere mostrar que el grafo $(\mathbb{V}, \mathbb{E} \backslash (F' \cup \{x\}))$ tiene n componentes conexas para algun $x \in F \backslash F'$.

Supongase por un momento lo contrario, esto es, $(\mathbb{V}, \mathbb{E} \backslash (F' \cup \{x\}))$ tiene mas de n componentes conexas para todo $x \in F \backslash F'$. Entonces, $(\mathbb{V}, \mathbb{E} \backslash (F' \cup F))$ tiene $n + |F \backslash F'|$ componentes. Ademas, como $\mathbb{E} \backslash F = (\mathbb{V}, \mathbb{E} \backslash (F' \cup F)) \cup (F' \backslash F)$, se obtiene que $(\mathbb{V}, \mathbb{E} \backslash F')$ tiene al menos $n + |F \backslash F'| - |F' \backslash F| = n + |F| - |F'| > n$ componentes. Pero esto contradice el hecho de que $F \in \mathbb{F}$. Por lo tanto debe ser cierto lo contrario, que era lo que se deseaba demostrar. □

b) Un subconjunto de $\mathbb{E}$ que no esté en $\mathbb{F}$ se dice que es dependiente. Un conjunto dependiente minimal es uno donde todo subconjunto propio es independiente. Caracterice en términos de grafos un dependiente minimal. Cómo serían los dependientes minimales de un ciclo elemental con n-vértices.

Solución:

Tomemos por ejemplo, un grafo formado por 3 vértices y 3 aristas que conectan a los vértices entre si (Figura 2a). Un dependiente minimal es el subconjunto $F = \{(a, b), (a, c)\}$. Si se eliminan simultáneamente las dos aristas (Figura 2b), el número de componentes conexas aumenta en 1. Pero los sub-conjuntos , $\{(a, b)\}$ y $\{(a, c)\}$, son independientes, ya que al retirar la arista en cada conjunto no se altera el numero de componentes conexas del grafo (Figura 2c).

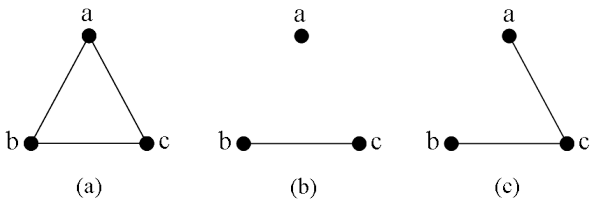

Figura 2: Ejemplo de un grafo con 3 vértices y 3 aristas

Para un ciclo elemental con n-vértices, todos los subconjuntos de aristas conformados por exactamente dos aristas distintas del ciclo son dependientes minimales. En este caso, no existen dependiente minimales de cardinalidad mayor a 2, puesto que cualquier conjunto de aristas con mas de dos aristas contiene al menos un sub-conujunto de dos aristas que es un dependiente minimal.

Problema #5a: Tiene usted que organizar un torneo con n participantes. Cada participante tiene que competir exactamente una vez con todos los posibles oponentes. Además, cada participante tiene que jugar exactamente un partido cada día, con la posible excepción de un solo día en el cual no juega. (Ejercicio 7.38 del Brassard)

a) Si n es una potencia de 2, dar un algoritmo para construir un horario que permita que el torneo concluya en $n - 1$ días.

Solución:

En este caso (n potencia de 2) es posible diseñar un algoritmo divide y vencerás. Este algoritmo consiste en dividir al conjunto de n jugadores en dos partes iguales (digamos A y B) y se resuelven los partidos internos de cada grupo de forma recursiva, entonces sólo falta programar los partidos entre los jugadores del sub-grupo A y el sub-grupo B. Esto es sencillamente, emparejar los jugadores de A con los jugadores de B en un cierto orden el primer día, y luego en los $n/2 - 1$ días rotando cíclicamente (o anti-cíclicamente) los jugadores de B (Figura 3).

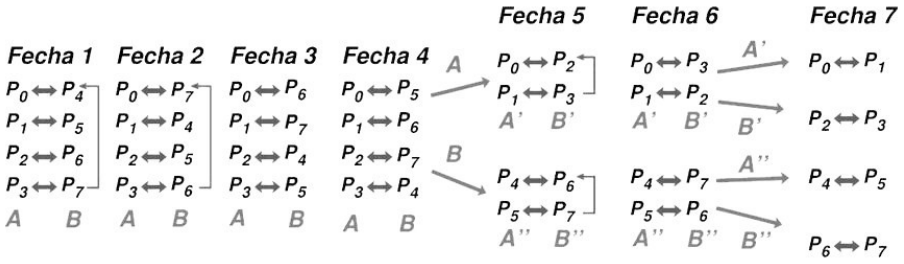

Figura 3: Solución divide y vencerás para $n = 8$

El pseudo-código de dicho algoritmo, PLANIFICAR-TORNEO, se muestra a continuación. Los parámetros del procedimiento son una matriz T de $(n-1) \times n$ donde se almacena el oponente del par (día, jugador); el día inicial (d_0) de emparejamiento de los jugadores j_0 hasta j_f; y los índices (j_0, j_f) que definen el conjunto de jugadores a emparejar. Los resultados de un ejemplo sencillo para $n = 8$ [llamada PLANIFICAR-TORNEO$(T, 1, 1, 8)$] se muestra en el cuadro 2.

```
PLANIFICAR-TORNEO(T[1 ... n − 1][1 ... n], d_0, j_0, j_f)
    ▷ El algoritmo supone que n es potencia de 2.
    ▷  1 ≤ d_0 < (n − 1)   ∧   1 ≤ j_0 < n   ∧   1 < j_f ≤ n

 1  m ← (j_f − j_0 + 1)/2   ▷ Divide el conjunto de jugadores en dos partes iguales.

 2  para d ← 0 hasta m − 1 hacer
 3          para i ← 0 hasta m − 1 hacer
 4                  j_1 ← j_0 + i
 5                  j_2 ← j_0 + m + (d + i) mod m
 6                  T[d_0 + d][j_1] ← j_2
 7                  T[d_0 + d][j_2] ← j_1

 8  si m > 1 entonces
 9          PLANIFICAR-TORNEO(T, d_0 + m, j_0, j_0 + m − 1)
10          PLANIFICAR-TORNEO(T, d_0 + m, j_0 + m, j_f)
```

Día \ Jugador	1	2	3	4	5	6	7	8
1	5	6	7	8	1	2	3	4
2	6	7	8	5	4	1	2	3
3	7	8	5	6	3	4	1	2
4	8	5	6	7	2	3	4	1
5	3	4	1	2	7	8	5	6
6	4	3	2	1	8	7	6	5
7	2	1	4	3	6	5	8	7

Cuadro 2: Una solución del problema de planificación para $n = 8$

b) Para todo entero $n > 1$, dar un algoritmo para construir un horario que permita que finalice el torneo en $n - 1$ días si n es par, o en n días si n es impar.

Solución:

Un algoritmo que da una solución al caso en que n es par, consiste en emparejar el primer día, a los primeros $n/2$ jugadores con los últimos $n/2$ en orden inverso. Luego para el siguiente día se fija el primer jugador y se permutan los $n - 1$ jugadores restantes. Al finalizar la permutación, se emparejan los jugadores de la misma manera que en la primera fecha. Este proceso se repite hasta que se llega de nuevo a la secuencia de jugadores original. Por ejemplo, para $n = 4$,

día	1	2	3
secuencia	1 2 3 4	1 3 4 2	1 4 2 3
encuentros	1 vs. 4 2 vs. 3	1 vs. 2 3 vs. 4	1 vs. 3 4 vs. 2

Si el número de jugadores es impar, entonces sólo hace falta insertar un jugador más, que marca

el día libre de los demas jugadores. A continuación se muestra el pseudo-código del algoritmo. Obsérvese que el algoritmo coloca -1 para indicar el día libre en el caso de n impar, y que en las líneas 12 y 13 se calcula directamente las operaciones de permutación e inversión, mencionadas anteriormente, de los últimos $n-1$ jugadores.

```
PLANIFICAR-TORNEO(T[1...n][1...n])
 1  si n mod 2 = 0 entonces
 2          jugadores ← n
 3  sino
 4          jugadores ← n + 1

 5  dias ← jugadores − 1
 6  m ← jugadores / 2

 7  para d ← 0 hasta dias − 1 hacer
 8          para i ← 0 hasta m − 1 hacer
 9                  si i = 0 entonces
10                          ip ← 0
11                  sino
12                          ip ← 1 + (i + d − 1) mod (jugadores − 1)

13                  jp ← 1 + (jugadores − i + d − 2) mod (jugadores − 1)

14                  si jp = n entonces
15                          T[d + 1][ip + 1] ← −1
16                  sino
17                          T[d + 1][ip + 1] ← jp + 1

18                  si ip = n entonces
19                          T[d + 1][jp + 1] ← −1
20                  sino
21                          T[d + 1][jp + 1] ← ip + 1
```

Problema #5b: Considere una matriz $T[1\ldots n]$ y un entero k entre 1 y n. Utilice una simplificación para diseñar un algoritmo eficiente que intercambie los k primeros elementos de T con los $n-k$ últimos, sin hacer uso de una matriz auxiliar. Analice el tiempo de ejecución de su algoritmo. (Ejercicio 7.40 del Brassard)

Salvo el caso $k = n/2$ y n par, no es posible intercambiar directamente a su posición final los elementos del arreglo. Una posible solución a este problema es el uso de una simplificación que funciona de la siguiente manera:

1. Si $k \leq n-k$ se intercambian los primeros k elementos de arreglo con los siguiente k elementos, sino, se intercambian los primeros $n-k$ con los últimos $n-k$ elementos.

2. Luego se repite el paso 1, para el sub-arreglo de los últimos $n-\Delta$ elementos, donde Δ es el número de elementos intercambiados anteriormente, hasta culminar en un sub-arreglo de un solo elemento.

A continuación se ilustra un ejemplo sencillo ($n = 8$; $k = 3$), seguido del pseudo-código del algoritmo.

Arreglo	1	2	3	4	5	6	7	8
Intercambio	4	5	6	1	2	3	7	8

Arreglo	1	2	3	7	8
Intercambio	7	8	3	1	2

Arreglo	3	1	2
Intercambio	1	3	2

Arreglo	3	2
Intercambio	2	3

Resultado	4	5	6	7	8	1	2	3

Cuadro 3: Ejemplo del algoritmo de intercambio. Llamada INTERCAMBIO($T[1 \dots 8], 3, 1$)

```
INTERCAMBIO(T[1...n], k, i)
    ▷  1 ≤ k < n   ∧   1 ≤ i ≤ k

 1  si (N − k) < (k − i + 1) entonces
 2          Δ ← N − k
 3  sino
 4          Δ ← k − i + 1

 5  para j ← 0 hasta Δ hacer
 6          r ← T[i + j]
 7          T[i + j] ← T[k + 1 + j]
 8          T[k + 1 + j] ← r

 9  i_p ← i + Δ
10  si k < i_p entonces
11          k_p ← k + Δ
12  sino
13          k_p ← k

14  si k_p < n entonces
15          INTERCAMBIO(T, k_p, i_p)
```

No es difícil observar que el peor caso se da cuando $k = 1$. En esta situación, la simplificación siempre es un arreglo con sólo un elemento menos, por lo cual la ejecución completa del algoritmo debe intercambiar $n - 1$ veces el primer y el segundo elemento del arreglo. Esto indica que INTERCAMBIO es $O(n)$.