UNIVERSIDAD SIMÓN BOLIVAR **18 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 consite simplemente retirar el objeto superior de la pila S_2 si dicha pila no esta vacia. Si la pila S_2 esta vacia, entonces primero se retira de forma sucesiva el objeto superior de S_1 y se inserta en S_2 hasta que S_1 este vacio. Esto invierte el orden de los obsjetos de S_1 dentro de S_2.

```
DESENCOLAR(Q)
1  si Q.S_2 esta vacia entonces
2          mientras Q.S_1 no este vacia 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 $O(1)$ puesto 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 creditos a cada operacion ENCOLAR y 0 creditos a cada operación DESENCOLAR. Cuando se encola un objeto se cancela 1 credito al inserta 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 credito por la operación POP en S_1 y 1 credito por la operación PUSH en S_2 por cada objeto durante el proceso que invierte el orden de los objetos; y finalmente 1 credito por la operación POP en S_2 del objeto a desencolar. Con este reazonamiento se observa que la cantidad de creditos 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 subárboles 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)

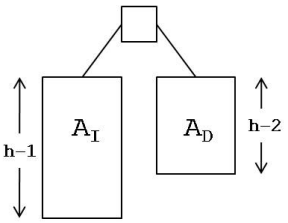

Figura 1: Árbol AVL

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 mas de 1 en todo árbol AVL. De esta manera es bastante claro cual es la formula de recurrencia para el numero de nodos en funcion 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 numero de nodos en el árbol de altura h. Observese que al sumar 1 en ambos lados a la ecuación 1 se obtiene la formula de recurrencia de los números de Fibonacci con el indice deplazado 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 formula de De Moivre se obtiene que,

$$n_h + 1 \approx \frac{\varphi^{h+3}}{\sqrt{5}} \qquad h \gg 1 \tag{3}$$

$$h \approx 1{,}44 \log n_h \tag{4}$$

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 solucion óptima.

Solución:

Una estrategia greedy evidente es insertar los objetos con mayor beneficio primero. Pero esta estrategia no funciona bien cuando los objetos mas "atractivos" tienen un peso mucho mayor en comparación con su beneficio. Otra posible estrategia greedy seria insertar los objetos mas livianos primero. Esta estrategia falla de forma similar a la anterior cuando los objetos livianos tienen un beneficio muy bajo comparados con el beneficio de objetos de mayor peso. Una estrategia greddy mas sofisticada, que busca evitar los problemas de las dos ya mencionadas, seria insertar los objetos con mayor beneficio por unidad de peso primero. Sin embargo, esta estrategia tampoco resulta siempre en la solución óptima. Por lo tanto, no se puede garantizar que alguna de las estrategias genere la solucion optima. De hecho, es incluso posible que ninguna de dichas estrategias produsca la solución óptima en algunos ejemplares de este problema. La siguiente tabla muestra un ejemplo de este caso con una mochila de capacidad 100.

w_i	v_i	v_i/w_i	Máx v_i	Mín w_i	Máx v_i/w_i	óptimo
100	400	4	1	0	0	0
50	350	7	0	0	1	1
45	180	4	0	1	0	1
20	100	5	0	1	1	0
Peso total			100	65	70	95
Beneficio total			400	280	450	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 es 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 lineas 6 y 11. Modificar el algoritmo carece de sentido debido a que no sigue siendo necesario. Es posible llenar la mochila unicamente con el tipo de objeto 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.

```
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) $G = (V, E)$, con E no vacío, la matroide de cociclos $M = (E, F)$ de G tiene como conjunto base a E y un independiente es un conjunto de aristas que al eliminarlas simultáneamente de G no aumenta el número de componentes conexas.

a) Muestre que $M = (E, F)$ es una matroide.

b) Un subconjunto de E que no esté en 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.

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 lagoritmo divide y venceras. Dicho algoritmo se muestra a continuacion con el nombre de PLANIFICAR-TORNEO. Los parametros del procedimento son una matriz T de $(n-1) \times n$ donde se almacena el oponente del par (dia, jugador); el dia inicial de emparejamiento (d_0) de los jugadores j_0 hasta j_f; y los indices (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 la figura 2 y el cuadro 2.

PLANIFICAR-TORNEO($T[1 \ldots n-1][1 \ldots n], d_0, j_0, j_f$)

▷ El algoritmo supone que n es potencia de 2.

▷ $1 \leq d_0 < (n-1) \quad \wedge \quad 1 \leq j_0 < n \quad \wedge \quad 1 < j_f \leq n$

1 $m \leftarrow \frac{j_f - j_0 + 1}{2}$ ▷ Divide el conjunto de jugadores en dos parte iguales.

2 **para** $d \leftarrow 0$ **hasta** $m-1$ **hacer**
3 **para** $i \leftarrow 0$ **hasta** $m-1$ **hacer**
4 $j_1 = j_0 + i$
5 $j_2 = j_0 + m + (d+i) \bmod m$
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($T, d_0 + m, j_0 + m, j_f$)

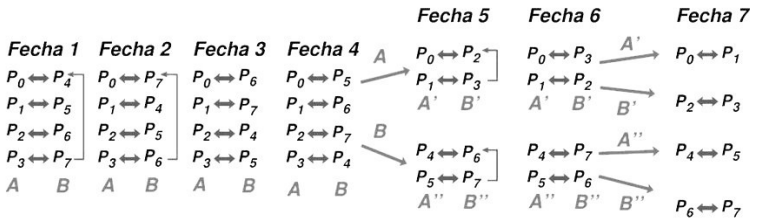

Figura 2: Solución divide y venceras para $n = 8$

dia \ 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 solucion al caso en que n es par, consiste en emparejar, el primer día, a los primero $n/2$ 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 numero de jugadores es impar, entonces solo hace falta insertar un jugador mas, que marca el día libre de los demas jugadores. A continuación se muestra el pseudocodigo del algoritmo. Observese que algoritmo coloca -1 para indicar los dias libre en el caso de n impar, y que en las lineas 12 y 13 se calcula directamente las operaciones de permutación e inversión, mencionadas anteriormente, de los ultimos $n-1$ jugadores.

PLANIFICAR-TORNEO($T[1 \ldots n][1 \ldots n]$)

```
 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)

dia \ jugador	1	2	3	4	5
1	-	5	4	3	2
2	2	1	-	5	4
3	3	4	1	2	-
4	4	-	5	1	3
5	5	3	2	-	1

Cuadro 3: Una solución del problema de planificación para $n = 5$