Algoritmos y Estructuras de Datos II - 1° cuatrimestre 2011

Práctico 3: Técnicas avanzadas de programación

Docentes: Daniel Fridlender, Renato Cherini, Diego Lis, y Juan Cruz Rodriguez

1. El siguiente programa computa x^n para x : **int** y n : **nat** dados.

   fun *exp*(x : **int**, n : **nat**) **ret** e : **int**
   $e := 1$
   for $i := 1$ **to** n **do** $e := e * x$

   Utilizando el hecho de que $x^n = (x^{n/2})^2$ cuando n es par, y *dividiendo* el problema para números pares e impares:

   a) Escribí un programa que compute x^n de manera más eficiente.

   b) Calculá estrictamente el orden de su complejidad.

2. Dados a : **array**[1..n] **of real** y x : **real**, escribí un programa que decida si existen i, j con $1 \leq i, j \leq N$ y $i \neq j$ tal que $a[i] + a[j] = x$. Diseñá el programa de manera que su complejidad sea de $\mathcal{O}(nlog(n))$.

   Ayuda: el problema puede resolverse combinando métodos suficientemente eficientes de ordenación y búsqueda.

3. La *moda* de una secuencia de valores $x_1, x_2, \ldots, x_n$ es el valor x_i que más se repite en la secuencia (notá que para una misma secuencia puede haber más de una moda). Utilizando las ideas de *quicksort*:

   a) Escribí un programa que compute una moda de la secuencia de valores dada en un arreglo.

   b) Calculá el orden de complejidad en el "peor caso".

   Ayuda: notá que si partimos una secuencia de valores en dos subsecuencias *disjuntas*, la moda se puede calcular fácilmente a partir de las modas de las subsecuencias.

4. Dado a : **array**[1..n] **of int**, decimos que un valor x es *mayoritario* si ocurre $n/2$ veces en a. Escribí un programa que dado un arreglo a, decida si existe un valor mayoritario de a, y en el caso positivo, devuelva ese valor.

5. Dados a : **array**[1..N] **of int** y k : **nat** tal que $1 \leq k \leq N$, definimos el *k-ésimo menor valor de a* como el valor que ocuparía la posición k en el arreglo *sort*(a) (donde *sort* es cualquier procedimiento que ordena un arreglo de menor a mayor). Por ejemplo, el 3-ésimo menor valor de $[9, 10, 8, 8, 4, 4, 1, 3]$ es 4.

   Suponé que se cuenta con un programa **fun** *med*(a : **array of int**, izq : **nat**, der : **nat**) **ret** m : **int** que devuelve la *mediana* del arreglo a entre los indices izq y der, y cuya complejidad es de de $\mathcal{O}(n)$. La *mediana* de un arreglo a : **array**[1..n] puede pensarse como el $\lceil n/2 \rceil$-ésimo menor valor de a.

   Escribí un programa **fun** *select*(a : **array of int**, izq : **nat**, der : **nat**, k : **int**) **ret** n : **int** que devuelva el k-ésimo menor valor de a entre los índices izq y der, cuyo orden de complejidad sea *estrictamente* menor que $\mathcal{O}(nlog(n))$.

6. Es posible introducir una pequeña mejora en la eficiencia del algoritmo *quicksort* si prestamos atención a los elementos repetidos. La idea consiste en utilizar un valor de *pivot* para dividir el arreglo a ordenar en tres segmentos: el primero que contiene los valores *menores* que el pivot, el segundo que contiene los valores *iguales* al pivot, y finalmente aquel que contiene los valores mayores que el pivot.

   Modificá el programa *quicksort* visto en clase, reemplazando el procedimiento *pivot* por otro que devuelva un par de índices i, j de manera que $a[i..j-1]$ contenga un mismo valor pivot.

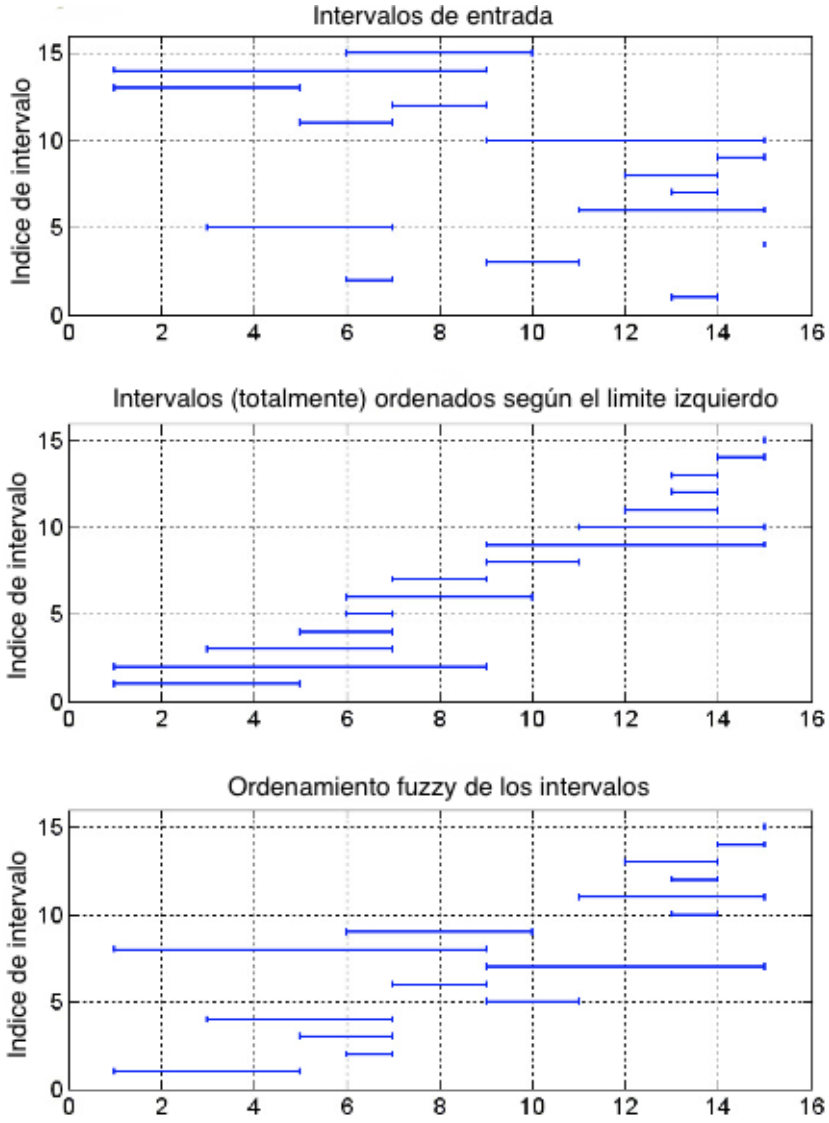

Figura 1: Ejemplo de intervalos, su orden total y su orden *fuzzy*

7. En muchas situaciones no es posible saber con precisión el valor de una cantidad (por ejemplo si medimos una magnitud física sujeta a *ruido*). En estas situaciones suele representarse una cantidad i como un intervalo $[a_i, b_i]$ con $a_i \leq b_i$, dentro del cual se supone que el valor de i se encuentra.

   Dada una secuencia de intervalos $i_1, i_2, \ldots, i_n$, un *ordenamiento fuzzy* de esta secuencia es una permutación de manera que $i_j \leq i_k$ si existen valores $c_j \in i_j$ y $c_k \in i_k$ tal que $c_j \leq c_k$ (¿qué sucede, respecto al orden, cuando los intervalos se solapan?).

   En la figura 7 se muestra un ejemplo de un ordenamiento según el extremo izquierdo y un ordenamiento *fuzzy* de la misma secuencia de intervalos. Notá que un ordenamiento (estándar) de los intervalos según extremo izquierdo es en particular un ordenamiento *fuzzy*, pero posiblemente su cómputo sea mas costoso.

   Diseñá un programa que ordene de manera *fuzzy* una secuencia de intervalos representados por dos arreglos a : **array**$[1..n]$ **of nat** y b : **array**$[1..n]$ **of nat** que almacenan la primer y segunda componente de los intervalos, respectivamente.

   La complejidad promedio del programa debe ser de $\mathcal{O}(nlog(n))$, pero el programa debe sacar ventaja de las posibles superposiciones de los intervalos para mejorar su eficiencia. En particular, en el caso en que todos los intervalos tengan algún punto en común, demostrá que el orden de complejidad es $\Theta(n)$.

8. Ejecutá paso a paso el algoritmo de Dijkstra que computa el *camino de costo mínimo* entre un nodo dado y los restantes nodos de un grafo, sobre el grafo con nodos $\{1, 2, \ldots, 8\}$ y pesos dados por la función w:

$$\begin{array}{llll} w((1,2)) = 7 & w((2,3)) = 4 & w((3,6)) = 4 & w((5,6)) = 6 \\ w((1,6)) = 3 & w((2,4)) = 2 & w((3,8)) = 6 & w((6,7)) = 5 \\ w((1,7)) = 5 & w((2,5)) = 1 & w((4,6)) = 8 & w((8,5)) = 2 \\ w((1,3)) = 3 & w((3,4)) = 5 & w((5,4)) = 3 & w((8,7)) = 3 \end{array}$$

   Considera a 1 como el nodo inicial. En cada paso explicitá el conjunto de nodos para los cuáles ya se ha computado el costo mínimo y el arreglo con tales costos.

9. Ejecutá paso a paso, graficando las soluciones parciales, los algoritmos de Prim y Kruskal que computan el *árbol generador mínimo* sobre el grafo del ejercicio anterior.

10. ¿Qué sucede si se ejecutan los algoritmos de Prim y Kruskal sobre un grafo no conexo?

11. Demostrá que el algoritmo voraz para el problema de la mochila *sin fragmentación* no siempre halla la solución óptima.

    Ayuda: Podés modificar el algoritmo visto en clase para que no permita fragmentación y después encontrar un ejemplo en cuya ejecución la solución computada no sea óptima.

12. En numerosas oportunidades se ha observado que decenas y hasta cientos de ballenas nadan juntas hacia la costa y quedan varadas en la playa sin poder moverse. Algunos sostienen que se debe a una pérdida de orientación posiblemente causada por la contaminación sonora de los océanos que interferiría con su capacidad de inter-comunicación. En estos casos los equipos de rescate realizan enormes esfuerzos para regresarlas al interior del mar para salvar sus vidas.

    Suponga que se encuentran n ballenas varadas en una playa. Se conocen los tiempos $s_1, s_2, \ldots, s_n$ que cada ballena tiene de sobrevida hasta ser asistida por el equipo de rescate. Además el tiempo necesario para salvar una ballena, trasladándola hasta el agua es constante t.

    Escribí un programa que compute el orden en que debe rescatarse las ballenas de manera de salvar al mayor número de ellas.

13. Sos el flamante dueño de un equipo de gps, y se lo ofreces a tus n amigos para que lo lleven con ellos cuando salgan de vacaciones el próximo verano. Lamentablemente cada uno de ellos irá a un lugar diferente y en algunos casos, los períodos de viaje se superponen. Por lo tanto es imposible prestarle el gps a todos, pero vos querés prestárselo al mayor número de amigos posible.

    Suponiendo que conoces los días de partida y regreso (p_i y r_i respectivamente) de cada uno de tus amigos, escribí un programa que determine a quién debes prestar el gps.

    ¿Cuál es el criterio para determinar, en un momento dado, a quien conviene prestarle el equipo?

14. Se desea realizar un viaje en un automóvil con autonomía a, desde la localidad l_0 hasta la localidad l_n pasando por las localidades $l_1, \ldots, l_{n-1}$ en ese orden. Se conoce cada distancia $d_i \leq a$ entre la localidad l_{i-1} y la localidad l_i (para $1 \leq i \leq n$), y se sabe que existe una estación de combustible en cada una de las localidades.

    Escribí un programa que compute el menor número de veces que es necesario cargar combustible para realizar el viaje, y las localidades donde se realizaría la carga.

    Supone que inicialmente el tanque de combustible se encuentra vacío.

15. Un submarino averiado descansa en el fondo del océano con un número n de sobrevivientes en su interior. Se conocen las cantidades $c_1, \ldots, c_n$ de oxígeno que cada uno de ellos consume por minuto. El rescate de sobrevivientes se puede realizar de a uno por vez, y cada operación de rescate lleva t minutos.

    a) Escribí un programa que determine el orden de salvamento de manera de rescatar el mayor número posible de sobrevivientes antes de que se agote el total C de oxígeno.

    b) Modificá la solución anterior suponiendo que por cada operación de rescate se puede llevar a la superficie a m sobrevivientes (con $m \leq n$).

16. Considerá una fabrica que cuenta con una planta de n operarios, y existen n tareas por realizar. Para cada operario cada tarea tiene un costo representada por una matrix $n \times n$.

    Escribí un programa que determine la forma óptima de asignar una tarea a cada operario, es decir, aquella que determine el menor costo total.

17. Dados n objetos de peso $p_1, \ldots, p_n$ y m cajas de capacidad $q_1, \ldots, q_m$ se desea almacenar los objetos en las cajas de modo de utilizar el menor número de cajas posible y sin exceder la capacidad de ninguna de ellas.

    Suponiendo que existen suficientes cajas de cada tipo, escribí un programa que calcule el menor número de cajas necesarias para almacenar los objetos.

18. Reconsiderá el problema del ejercicio 14. Sean $t_0, \ldots, t_{n-1}$ los tiempos de demora para cargar nafta en cada una de las localidades $l_0, \ldots, l_{n-1}$. El tiempo de demora no depende de la cantidad de nafta a cargar, sino que es la espera estimada para ser atendido en dicha localidad. Por consiguiente, cada vez que se carga nafta conviene llenar el tanque.

    Escribí un programa que calcule el tiempo mínimo que se puede desperdiciar esperando a ser atendido en las estaciones para realizar el viaje.

19. De repente te encontrás en un globo aerostático sobrevolando el océano y descubrís que la lona está levemente dañada y que el globo empieza a perder altura. Te preguntas cómo es posible que te encuentres en semejante situación, si hasta hace un minuto estabas en el aula haciendo ejercicios. Pero no tenés tiempo para contestarte esa pregunta, el globo cae peligrosamente. De pronto descubrís que tenés n objetos cuyos pesos $p_1, \ldots, p_n$ y valores sentimentales $v_1, \ldots, v_n$ conocés bien. Sabés que si te desprende de al menos P kilogramos el globo logrará recuperar altura y llegar a tierra firme. Afortunadamente, la suma de los pesos de los objetos supera holgadamente a P. Elegí cuáles objetos vas a arrojar para salvarte, de manera de desprenderte del menor valor (total) posible.

    Ayuda: Podes definir una función $m(i, j)$ como "el mínimo valor posible obtenible con los objetos 1, ..., n superando el peso j".

20. Suponé que existen n empresas cuyas acciones valen respectivamente $v_1, \ldots, v_n$. Además conocés los valores $w_1, \ldots, w_n$ que esas mismas acciones tendrán mañana (que no necesariamente representan un incremento).

    Escribí un programa que determine el máximo capital que se puede obtener mañana invirtiendo en acciones un monto D disponible hoy.

    Ayuda: Podes definir una función $m(i, j)$ como "el máximo valor alcanzable con dinero j comprando acciones de empresas $1, \ldots, i$".

21. **Problema de las 8 reinas con costos.** Dado un tablero de 8 por 8 en el que cada celda tiene asociado un costo, escribí un programa que encuentre el menor costo posible de colocar 8 reinas en dicho tablero sin que ningún par de reinas se puedan atacar mutuamente. En otras palabras, se pide computar el menor costo posible de elegir 8 celdas tal que cada fila, cada columna y cada diagonal contenga a lo sumo una de las 8 celdas elegidas. El costo de elegir 8 celdas es la suma de los costos de las celdas elegidas.

22. Tus amigos quedaron encantados con tu gps (ejercicio 13), tanto que de ahora en más ofrecen pagarte un alquiler por él. Además de la información sobre el día de partida y de regreso (p_i y r_i) de cada amigo, ahora se dispone del monto m_i que cada uno ofrece por el gps.

    Escribí un programa que determine el máximo monto de dinero que podés ganar alquilando el gps.

    Ayuda: Podés definir una función $m(i, j)$ como "el maximo valor obtenible hasta el día j alquilando el gps a amigos 1, ..., i"

23. Considerá un tablero con $n \times n$ cuadros y una función costo $c(i, j)$ que asocia un costo a cada cuadro (i, j). Se debe mover una pieza desde la fila inferior del tablero hacia la fila superior del mismo. En cada paso se puede mover la pieza hacia uno de los siguientes cuadros:

    - El cuadro inmediatamente superior.
    - El cuadro que se encuentra uno arriba y uno a la izquierda (es decir, en diagonal) si la pieza no estaba en la primera columna de la izquierda.

- El cuadro que se encuentra uno arriba y uno a la derecha (es decir, en la otra diagonal) si la pieza no estaba en la primera columna de la derecha.

a) Escríbí un programa que compute el mínimo costo de llevar una pieza desde la fila inferior hasta la fila superior. El costo total de dicho desplazamiento es la suma de los costos de los cuadros por los que pasa la pieza, incluyendo los cuadros inicial y final).

b) Modificá el programa anterior para que devuelva el camino de costo mínimo, es decir, la secuencia de cuadros que dan lugar al costo mínimo.

24. Optimizá la soluciones de los problemas 19, 23, 22 y 20 utilizando la técnica de *programación dinámica*.

25. Sea un tablero con 100 casillas numeradas desde la 1 a la 100. El tablero corresponde a un juego en el que cada jugador comienza en la celda 1 y, a su turno, arroja un dado que le permite avanzar entre 1 y 6 casilleros. Se debe llegar (o superar) la casilla 100 para terminar.

    Se desea calcular el mínimo número de turnos que hacen falta para terminar. La solución sería trivial (se puede recorrer el tablero en 17 turnos) si no fuera porque algunas celdas tienen premios o castigos:

    - Hay un conjunto de celdas **avanza**: cuando se cae en una celda de este conjunto, en el mismo turno se avanza 3 casilleros.
    - Hay un conjunto de celdas **denuevo**: cuando se cae en una celda de este conjunto, en el mismo turno se tira nuevamente el dado y se avanza según el dado indica.
    - Hay un conjunto de celdas **espera**: cuando se cae en una celda de este conjunto, se pierde el próximo turno.

    Estos 3 conjuntos son disjuntos (y no contienen ni al 1 ni al 100). Pero puede ocurrir que al avanzar 3 casilleros por haber caído en una celda de **avanza** se caiga en una nueva celda con premio o castigo, en cuyo caso también debe ejecutarse el nuevo premio o castigo. De la misma forma, al tirar nuevamente el dado por haber caído en una celda de **denuevo** se puede caer en una nueva celda que tiene premio o castigo y el mismo debe ejecutarse también. Esto, claramente, puede dar lugar a una secuencia larga de premios en un mismo turno.

    El problema puede resolverse utilizando la técnica de *backtracking*. La función $m(i)$ se define como "el número mínimo de turnos para terminar cuando uno ya se encuentra en el casillero i". Su formulación recursiva es la siguiente:

$$m(i) = \begin{cases} 0 & i \geq 100 \\ m(i+3) & i \in \textbf{avanza} \\ \text{mín}(m(i+1), m(i+2), \ldots, m(i+6)) & i \in \textbf{denuevo} \\ 2 + \text{mín}(m(i+1), m(i+2), \ldots, m(i+6)) & i \in \textbf{espera} \\ 1 + \text{mín}(m(i+1), m(i+2), \ldots, m(i+6)) & \textit{c.c.} \end{cases}$$

    Esta definición es exponencial (dado que una llamada a $m(i)$ puede generar hasta 6 llamadas recursivas). Utilizá la técnica de programación dinámica para encontrar una programa iterativo que solucione el problema.