Pontificia Universidad JAVERIANA Cali

Conceptos y Modelos de programación | **Taller Modelo Declarativo Febrero 20 de 2007**

> "El estudiante de la Pontificia Universidad Javeriana, como agente de su propia formación, es corresponsable de la Identidad Institucional, uno de cuyos cimientos es tener como hábito un comportamiento Ético en todos los ámbitos de la vida. En este sentido me comprometo a realizar con total integridad esta evaluación, solamente empleando los recursos autorizados para su desarrollo".
> *Consejo académico, Acta Nro 79, abril 19 de 2004*

Nombre: _____

Pregunta	1	2	3	4	5	Total
Puntos	25	10	25	25	15	100
Cal.						

Nota: A continuación se presenta el taller correspondiente al modelo declarativo visto en clase. Esta actividad debe ser entregada el día 7 de Marzo antes de las 6:00 pm en un sobre (físico) que contenga un disquete y un documento (si es necesario). Pasada esta fecha no se recibirá el trabajo. El taller puede ser presentado en grupos de máximo 2 personas. Todos las implementaciones deben ser realizadas en Mozart/Oz.

Recursión en el modelo declarativo

1 (25 Puntos)
Defina una función que tome dos listas con longitud N de la forma $[X_1, X_2, \ldots, X_N]$ y $[Y_1, Y_2, \ldots Y_N]$, y retorne su convolución simbólica $[X_1 \# Y_N, X_2 \# Y_{N-1}, \ldots, X_N \# Y_1]$. En ningún momento la función propuesta deberá recorrer la lista más de una vez. Escriba la ecuación de recurrencia correspondiente a su propuesta y justifíquela. Recuerde que $\{List.nth\ L\ N\}$ tiene complejidad $O(n)$.

2 (10 Puntos)
Una matriz puede ser representada en mozart mediante una lista de listas. Dada una matriz cuadrada de dimensiones $n \times n$, escriba una función que calcule su determinante. Determine la ecuación de recurrencia de esta función, y calcule tanto la complejidad espacial como la temporal. Solamente se pueden utilizar las técnicas vistas en el curso.

Concurrencia en el modelo declarativo

3 (25 Puntos)
Existen diversos algoritmos para ordenar secuencias (listas), por ejemplo: *quick-sort, merge-sort, shell-sort*. Además de estos, existen algoritmos que aprovechan la existencia de concurrencia en los lenguajes de programación para realizar esta tarea de una manera tal vez más eficiente. A continuación se muestra un algoritmo de ordenamiento llamado *odd even transposition sort*.

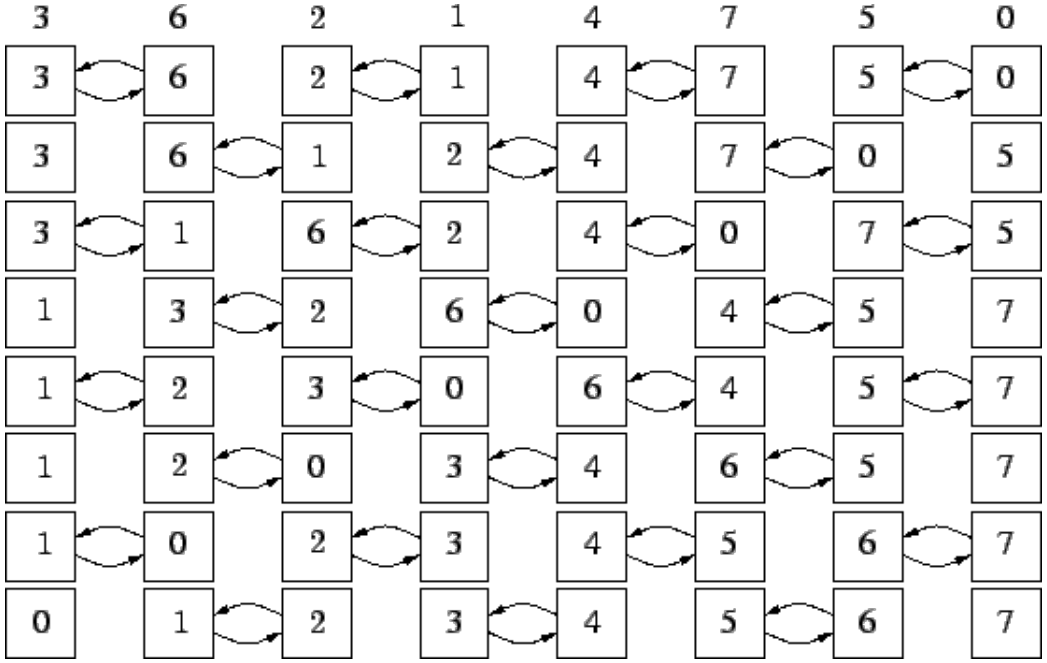

FIGURA 1: Odd even transposition sort para ordenar [3 6 2 1 4 7 5 0].

1. Sea L lista de números a ordenar de longitud N.
2. Colocar un hilo por cada elemento en una posición *par* p de L. Este hilo debe comparar el elemento en la posición p con el de la posición $p-1$ y cambiarlos si el elemento en la posición p es menor que el de la posición $p-1$.
3. Colocar un hilo por cada elemento en una posición *impar* p de L. Este hilo debe comparar el elemento en la posición p con el de la posición $p-1$ y cambiarlos si el elemento en la posición p es menor que el de la posición $p-1$.
4. Repetir los pasos 2 y 3 hasta que la lista quede completamente ordenada. Note que esta verificación debe ser realizada con una complejidad constante.

Para realizar este ejercicio suponga que la lista inicia en la posición 1. La Figura 1 muestra un ejemplo donde se ordena la lista de números [3 6 2 1 4 7 5 0].

4 (25 Puntos)
Dada una tupla de números que se encuentra ordenada, escriba una función que concurrente retorne la posición de un elemento e en la tupla. La función recibe como parámetros la tupla (T), el elemento (e) y el número máximo de hilos p que se pueden utilizar por cada iteración de la función. El algoritmo consiste en una serie de iteraciones para reducir el tamaño del arreglo donde se encuentra el elemento e. En cada iteración, la tupla se divide en $p+1$ partes iguales. Este proceso se repite hasta que el tamaño de la tupla analizada sea igual a p. A continuación se realiza una comparación directa y se retorna la posición de e. En la Figura 2 se muestra un ejemplo del funcionamiento del algoritmo para $p=2$ y $e=23$.

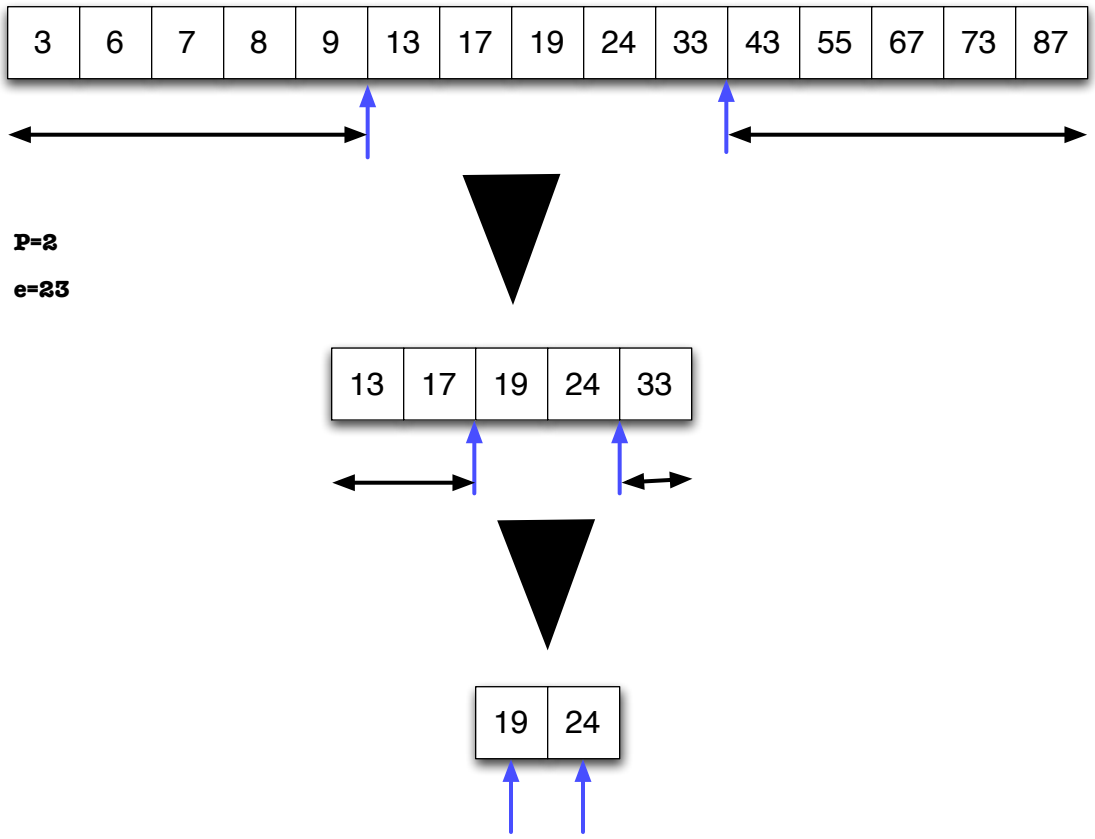

FIGURA 2: Búsqueda concurrente

Programación perezosa

5 (15 Puntos)

Con un árbol binario se pueden representar expresiones matemáticas como se muestra a continuación:

Los operadores de este tipo de expresiones no se restringen solamente a: +, -, *, /. Usted deberá tener en cuenta otras operaciones matemáticas como: $sin(x)$, $cos(x)$, x^n, $ln(x)$, etc., que pueden estar definidas por el usuario. Tenga en cuenta que un nuevo operador matemático no debe dar lugar a cambios en su programa pero sÌ en sus parámetros.

Implemente una interfaz gráfica que permita representar este tipo de estructura de datos, tomando como entrada una formula matemática en notación postfija. Su implementación debe permitir al usuario seleccionar con el *mouse* un nodo del árbol y debe mostrar el valor de la expresión hasta ese momento. Por ejemplo, en el nodo C del árbol anterior, imprimiría 2, pero en el nodo A 18.

Además de esto, las operaciones sobre los datos solamente se deben realizar por demanda, es decir, solo deben ser evaluados aquellos nodos que sean necesarios para la visualización de un nodo determinado.

Un ejemplo de entrada para su programa puede ser la siguiente lista: $[4\ 5\ '+'\ 1\ 2\ '+'\ '*']$.