Examen de Programación 3 y III (09/02/2013)

Instituto de Computación, Facultad de Ingeniería, UdelaR

1. Este examen dura 4 horas y contiene **2** carillas. El total de puntos es 100 y se requieren 60 para su aprobación.
2. En los enunciados llamamos C* a la extensión de C al que se agrega el operador de pasaje por referencia &, y las sentencias *new*, *delete*, el uso de *cout* y *cin y el tipo bool*.
3. NO se puede utilizar ningún tipo de material de consulta.
4. No se contestarán dudas durante la última media hora.

Se requiere:

- Numerar todas las hojas e incluir en cada una el nombre y la cédula de identidad.
- Utilizar las hojas de un solo lado y escribir con lápiz, iniciando cada ejercicio en hoja nueva.
- Poner en la primera hoja la cantidad de hojas entregadas, y entregar el **índice** indicando en qué hoja se respondió cada problema.

Parte Obligatoria

Esta parte es eliminatoria, para la aprobación del examen debe obtenerse un mínimo del **50% de esta parte (20 puntos)**. En caso de no llegar a dicho mínimo, **NO** se corregirán los problemas.

Ejercicio 1 (10 puntos)

a) Indique la veracidad de las siguientes afirmaciones justificando todos los pasos.
Sea $f : N \rightarrow R^*$ una función arbitraria. Se cumple entonces que:

i) $f(n) \in O(n) \rightarrow 2^{f(n)} \in O(2^n)$
ii) $f(n) \in \Omega(n) \rightarrow 2^{f(n)} \in \Omega(2^n)$

b) Demostrar la siguiente propiedad basándose en las definiciones de O y Ω

$$f(n) \in O(g), g(n) \in O(h) \rightarrow f(n) \in O(h)$$

Ejercicio 2 (10 puntos)

a) Enunciar el principio de optimalidad.
b) Mostrar la aplicación del principio de optimalidad al problema de encontrar el camino de menor costo entre dos vértices de un grafo con costos asociados a sus aristas.

Ejercicio 3 (10 puntos)

a) Implemente el algoritmo `Selection Sort` (C* o seudocódigo es suficiente).
b) Para una entrada de tamaño 3 [a, b, c] con b < a, dibuje las partes del árbol de decisión consideradas en el algoritmo en una ejecución genérica del algoritmo anterior.

Ejercicio 4 (10 puntos)

a) Dado un AVL vacío, y la siguiente lista de elementos: 48, 33, 18, 5, 27, 23, 78, 42, 30, 95, 39 haga un dibujo que muestre dicha estructura para cada inserción y cada rotación que aplique, indicando cual es la rotación que se aplica.

b) Dado un AVL vacío, indique una ordenación de la lista de elementos anterior de forma de minimizar la cantidad de rotaciones necesarias.

Problemas

Problema 1 (30 puntos)

Una casa de cambio tiene interés de desarrollar un programa para hacer conversiones entre monedas de distinto origen.

Dadas las monedas de origen diferente que opera la casa de cambio, ésta no tiene por qué tener una conversión directa para cada par de moneda de distinto origen, por lo tanto, puede ser necesario realizar varias conversiones para convertir de una moneda a otra.

Suponga que las monedas de distinto origen se identifican por un entero en el rango [1..M], y a su vez, se tiene disponible la función *ExisteConversionDirecta* que dadas dos monedas de distinto origen, retorna *verdadero* si existe una conversión directa entre ambas monedas.

```
bool ExisteConversionDirecta(int a, int b);
```

Asuma que el resultado de invocar *`ExisteConversionDirecta(a, b)`* es igual al resultado de invocar *`ExisteConversionDirecta(b, a)`*.

Se quiere determinar si es posible realizar la conversión entre dos monedas de distinto origen, y en caso de ser posible, determinar la cantidad mínima de conversiones a realizar.

Se pide:

a) **(20 puntos)** Formalizar el problema aplicando **Programación Dinámica**. Llame a la fórmula recursiva ***f***. Explicar qué representa el/los pasos base y cada paso recursivo de la solución, así como también, qué representa la función *f* indicando sus índices. Se debe indicar la invocación de la función para resolver el problema.

   Sugerencia: puede resultar útil basarse en alguna de las recurrencias vistas en el curso.

b) **(10 puntos)** Describa qué cambios tendría que hacer para no hallar solamente la cantidad mínima de conversiones a realizar entre dos monedas de distinto origen, sino también para reconstruir la secuencia de conversiones mínima. **No es necesario reformular la recurrencia**.

Problema 2 (30 puntos)

Se desea minimizar el costo de producción de un campo modelado como una matriz de NxM hectáreas donde cada una se identifica con un número k, $0 \leq k < N*M$. Todas las hectáreas deben ser cultivadas, para ello se cuenta con S tipos distintos de semillas y los vectores K y P de largo S donde:

- K[i] indica la **cantidad máxima** de hectáreas que se pueden sembrar con el cultivo i.
- P[i] indica el costo normal de producción del cultivo i.

Cada cultivo afecta a sus cultivos linderos de acuerdo a los coeficientes dados por la función *impacto*: NxN->R^+. Es decir por ejemplo, si se cultiva h en la hectárea C[i][j-1] y t en la C[i+1][j] , el costo normal del cultivo r sembrado en C[i][j] será afectado de la siguiente manera: *P[r]*impacto(h, r)*impacto(t, r)*; análogamente y simultáneamente lo harán sus otros cultivos linderos. Notar que cada cultivo podrá tener a lo sumo 4 cultivos linderos que afectarán su costo.

Se pide:

a) Formalizar el problema en términos de Backtracking. Considere definir funciones auxiliares e indicatriz para facilitar la formulación matemática.
b) Utilizando Backtracking y su formulación anterior, implemente en C* la función *optimizarProduccion(int** Campo, int** Costo)* que deja en *Campo* la asignación de los cultivos y en *Costo* el costo de cada hectárea (afectado por sus linderos) que minimizan el costo total de producción. Asuma que todas las constantes, vectores y variables *Campo* y *Costo* mencionadas son dadas y tienen visibilidad global. Se sugiere el uso e implementación de funciones auxiliares para la manipulación de la matriz y cálculo del precio de los cultivos.