Examen de Programación 3 y III (10/02/2011)

Instituto de Computación, Facultad de Ingeniería, UDELAR

1. Este examen dura 4 horas y contiene 3 carillas. El total de puntos es 100. Para su aprobación necesita 60 puntos.
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* y el uso de *cout* y *cin*.
3. NO se puede utilizar ningún tipo de material de consulta.
4. No se contestaran dudas durante la última media hora.

Se requiere:

- **Numerar e incluir el nombre completo y cédula de identidad en la esquina superior derecha de cada hoja.**
- Utilizar las hojas de un sólo lado y escribir con lápiz, iniciando cada ejercicio en hoja nueva.
- Completar la carátula con la cantidad de hojas entregadas, y un **índice** indicando en qué hoja se respondió cada ejercicio y problema.

Parte Teórica Obligatoria

Esta parte es eliminatoria, vale **40** puntos y usted debe obtener como mínimo el **50% de esta parte (20 puntos)**. En caso de no llegar a dicho mínimo, **NO** se corregirán los problemas. Usted podrá encontrar planteos prácticos. Los mismos deben ser resueltos justificando detalladamente la correspondencia con la base teórica que utilice en su respuesta.

Ejercicio 1 (10 puntos)

Dado la siguiente descripción de un algoritmo que tiene como entrada un vector de datos de tamaño N (asuma que N es potencia de 3), calcule la complejidad asintótica (Θ):

El algoritmo divide los datos de entrada en tres vectores de igual tamaño, realiza una llamada recursiva sobre **cada uno** de estos vectores, y por último realiza un proceso sobre los datos de entrada con coste cuadrático ($a.N^2$).

Sugerencia: Podría ser de utilidad saber que $\sum_{i=0}^{\infty} \frac{1}{k^i}$ converge para valores k>1 y considerar $N=log_3(n)$ como cambio de variable cuando desee reducir.

Ejercicio 2 (10 puntos)

a) Describa en forma concisa el método Greedy y sus características principales.

b) Un albañil desea invertir sus *M* pesos en comprar la mayor cantidad de metros posibles de cuerdas.
Existen *N* tipos de cuerdas cuyos largos son acotados. El albañil puede unir pedazos de cuerdas, y la uniones entre ellas son “perfectas” (no se gasta cuerda extra).

Basandose en Greedy, implemente la estrategia *albanil()* que resuelve el problema.

```
void albanil(
      float M, // Dinero del albañil
      int N,  // Cantidad de tipos de cuerdas
      float* metros_disponibles,//metros_disponibles[i]:Mts diponibles de la cuerda i
      float* costo_por_metro,// costo_por_metro[i]:Costo por metro de la cuerda i
      float* sol // sol[i]: Cantidad de metros a comprar de la cuerda i.
);
```

Puede asumir que se tiene implementado una **función de ordenación: *sort(Tipo[])***, siempre y cuando defina el criterio de ordenación.

Ejercicio 3 (10 puntos)

Un niño quiere construir una torre de piezas de madera de **exactamente** N cm de altura. Para ello cuenta con M tipos distintos de piezas, cada uno con una altura dada *h[i]* (1<=i<=M) donde *h[i] < h[j]* si *i < j.* Se quiere minimizar la cantidad de fichas que se utilizan en la construcción.

Se pide:

a) Demuestre que el principio de optimalidad es aplicable a este problema.

b) Sea *f(i,j)* la cantidad mínima de piezas necesarias para construir una torre de altura *j*, utilizando sólo piezas de tipo *k, 1<= k<= i*; **plantee la formula recursiva para calcular *f*** e indique como debe invocarse la función.

Ejercicio 4 (10 puntos)

a) Implemente el algoritmo de ordenación *QuickSort(int* T, int ini, int fin)* utilizando la función:

```
// retorna el índice del pivote seleccionado de la entrada T entre [ini..fin]
int elegirPivote (int* T, int ini, int fin)
```

Cualquier otra función auxiliar que utilice deberá ser implementada.

b) Responda de manera **concisa** justificando **brevemente**:
- Clasifique la estrategia utilizada por el algoritmo.
- Indique su complejidad en caso promedio y peor caso
- Indique de que depende su estabilidad.

Problemas

Problema 1 (30 puntos)

El *diámetro* de un grafo es la mayor distancia entre todo par de vértices. Donde, para un par de vértices, la *distancia* se define como la cantidad de aristas del camino más corto entre ellos.

Sea un grafo no-dirigido, *G*, conteniendo *n* vértices que se identifican con enteros desde cero a $n - 1$:
a) Describa brevemente como se podría determinar el diámetro de *G* mediante BFS.
b) Implemente un algoritmo en C* que lo resuelva según lo descrito en a).

Notas:
i) Asuma el grafo como dado, cualquier otra estructura que utilice deberá definirla y cargarla según corresponda.
ii) Use la representación de lista de adyacencia para representar el grafo y asuma la existencia del TAD lista (*deque*) de enteros con operaciones para instrumentar colas de estos. Implemente cualquier otra función auxiliar que utilice.

Problema 2 (30 puntos)

El Sudoku es un juego muy conocido que consta de completar un grilla de n^2xn^2 casilleros con dígitos de *1 al* n^2 de manera que no haya repetidos en cada columna ni fila. A demás cada submatriz de *n x n* (ver ejemplo) deberá contener todos los dígitos de *1 a* n^2. La complejidad del juego es dada por la cantidad de valores predeterminados, es decir, casilleros con valores fijos que restringen las posibilidades de completar la grilla.

La siguiente figuras muestran un ejemplo de un escenario con *n=3*, valores predeterminados y las posibles opciones para completar el casillero Sudoku[1][1] (Fig. A); y la grilla una vez completada (Fig. B).

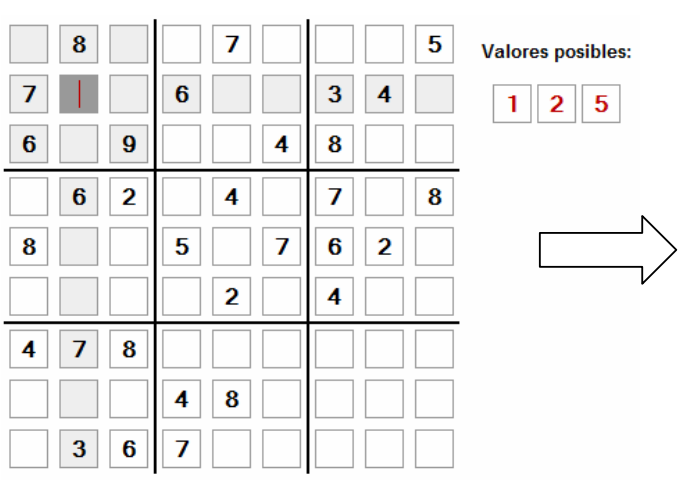

Fig.A

Fig. B

Se pide:

a) Considerando como operación la asignación de elementos a la grilla, plantee el pseudocódigo de la función ***SudokuSinR(inr**matriz, int n)*** que resuelve el problema para una grilla sin valores predefinidos en $\Theta(N^2)$. La siguiente es una grilla completada con un método que cumple lo anterior.

```
+-----------------------+
| 1 2 3 | 4 5 6 | 7 8 9 |
| 4 5 6 | 7 8 9 | 1 2 3 |
| 7 8 9 | 1 2 3 | 4 5 6 |
|-------+-------+-------|
| 2 3 4 | 5 6 7 | 8 9 1 |
| 5 6 7 | 8 9 1 | 2 3 4 |
| 8 9 1 | 2 3 4 | 5 6 7 |
|-------+-------+-------|
| 3 4 5 | 6 7 8 | 9 1 2 |
| 6 7 8 | 9 1 2 | 3 4 5 |
| 9 1 2 | 3 4 5 | 6 7 8 |
+-----------------------+
```

b) **Formalizar** una solución al problema del Sudoku de tamaño $N= n^2$ en términos de **BackTracking**.

c) **Implementar** el algoritmo ***Sudoku(int** matriz, int n)*** en C* que resuelva el problema según lo especificado en b) (para esta parte puede asumir que los casilleros no predefinidos contienen el valor 0 inicialmente).

Nota: Cuenta con las siguientes funciones:

- *int* toVector(int idSubMatriz),* que retorna los valores de la submatriz identificada por un valor en *[0..n^2-1]* (por filas), en un vector de tamaño n^2. Considerando el ejemplo en **a)** : *[2,3,4,5,6,7,8,9,1] = ToVector(3).*

- *int subMatrix(int fila, int col),* que dada una posición de la matriz retorna el id de la submatriz a la que pertenece.

Se sugiere utilizar métodos auxiliares para controlar las reglas del juego.