ICOM2016 – 1er Parcial

6 de octubre de 2016

Notas:

1. Al finalizar, enviar por e-mail los archivos fuente de cada ejercicio con nombre APELLIDO_NOMBRE_Ejer_N.c a icom.cabib@gmail.com
2. Uso de prácticos: **se pueden utilizar los trabajos prácticos propios realizados.**
3. Uso de Internet**: solo para la consulta de referencias de funciones de C**

Problema 1: Hanoi++

Utilizando **strings** para representar las torres, con los caracteres `'0',...,'9'` para representar los discos de distintos tamaños, reimplemente el algoritmo de las torres de Hanoi, moviendo efectivamente los discos (caracteres) entre las distintas torres (strings). Recuerde que sólo se puede mover el disco que está en el tope de una torre al tope de otra torre.
Inicialmente los discos estarian apilados en la torre **A**, con el `'9'` en la base y el `'0'` en el tope, y las torres **B** y **C** estarían vacías (pero con espacio suficiente para almacenar todos los discos):

```
char A[13]="A:9876543210";
char B[13]="B:"; // rellena con 0 hasta el final del arreglo
char C[13]="C:";
```

El prototipo de la función sería ahora:

```
void hanoi(int n, char from[], char aux[], char to[]);
```

Para verificar el progreso del algoritmo, imprima el estado de las torres cada cierto número de movimientos (ayuda: cada $2^n - 1$ movimientos, los **i** discos más pequeños deberían estar todos apilados en alguna torre y los restantes apilados en otra torre y la torre restante vacía).
Identifique los subproblemas e implemente funciones para resolverlos, que deberá luego utilizar en la función hanoi

Problema 2: Números de Uber.

Se dice que un número es el enésimo **número Uber** $T(n)$ si es el menor número que se puede descomponer como *n* sumas distintas de dos cubos positivos.

Se desea conocer (no usando la WEB) los valores de $T(1)$, $T(2)$ y $T(3)$.

Una forma de hacerlo (costosa, pero viable para este orden de números) es armar una matriz V de 500x500 en donde:

$$V[i][j] = i^3 + j^3 \qquad para\ i > 0\ y\ j > 0$$

Si después se pasan los valores de la matriz a un vector en donde cada elemento del vector mantenga la terna { i, j, V[i][j] } y se ordena en forma creciente de acuerdo a V[i][j], la primer secuencia igual (respecto a V[i][j]) de largo 1 definirá $T(1)$, la primer secuencia igual de largo 2 definirá $T(2)$ y la primer secuencia igual de largo 3 definirá $T(3)$.

Para el vector se sugiere usar elementos del tipo:

```
typedef struct {
    int i;
    int j;
    int v;
} Terna_t;
```

Se solicita calcular $T(1)$, $T(2)$ y $T(3)$, así como también las sumas que los definen.

Nota: Se podría omitir el armado de la matriz y generar directamente el vector. Asimismo, en el vector, podría omitirse V[i][j] y calcularlo cada vez que haga falta.

Problema 3: Estimación de quiebre.

Se desea ajustar con dos rectas una serie de datos como los que se muestran en las figuras. El ajuste de las dos rectas en simultáneo debe minimizar la función $\mathbf{Chi^2(a_1,b_1,a_2,b_2)=Chi_1^2(a_1,b_1)+Chi_2^2(a_2,b_2)}$, donde $\mathbf{a_i}$, $\mathbf{b_i}$, $\mathbf{Chi_i}$ son la ordenada, la pendiente y la función Chi cuadrado del ajuste de la recta i . La intersección de las dos rectas es un estimador del punto de quiebre de los datos.

Implementar una función **`estimaQuiebre`** que retorne el índice del dato en el que se produce el quiebre.

En el archivo 'regresion_lineal.h' se encuentra declarada la estructura **`FitParams`** y la función **`fitLineal`** (implementada en 'regresión_lineal.c'):

```
typedef struct {
    double a, b, siga, sigb, chi2,sigdat;
} FitParams;

FitParams fitLineal(int ndata, double x[], double y[]);
```

La función **`fitLineal`** ajusta una recta y=a+b*x a los **`ndata`** puntos formados por los pares **`x[i]`**, **`y[i]`** y retorna la estructura de parámetros que caracterizan el ajuste. En particular nos interesa el **Chi^2** para encontrar el mínimo de **$Chi^2(a_1,b_1,a_2,b_2)$**.

Implementar un programa que lea un archivo con una columna de 256 datos. Estos datos son los valores de la ordenada **`y[i]`** y su índice será el de la abscisa **`x[i]=i`**. Luego aplique la función **`estimaQuiebre`** para encontrar el índice del dato en el que se produce el quiebre. Probar su programa con los archivos de datos que se muestran en la figura.