LENGUAJES DE PROGRAMACIÓN

Solución al Trabajo Práctico - Junio de 2019

EJERCICIO 1

Un sistema dinámico de tiempo discreto M-dimensional es un mapa iterativo $f : \mathbb{R}^M \to \mathbb{R}^M$ de la forma

$$X_{k+1} = f(X_k) \tag{1.1}$$

donde $k = 0, 1, \ldots$ es el tiempo discreto y $X \in \mathbb{R}^M$ es el estado. Comenzando con X_0, que es el estado inicial, e iterando (1.1) se obtiene una serie de estados conocida como órbita. Un ejemplo es el mapa de Hénon[1], un sistema dinámico no lineal de tiempo discreto bidimensional representado por las ecuaciones de estado siguientes:

$$x_{k+1} = -\alpha \cdot x_k^2 + y_k + 1 \tag{1.2}$$
$$y_{k+1} = \beta \cdot x_k \tag{1.3}$$

donde (x, y) es el estado bidimensional del sistema. Asignaremos a los parámetros los valores siguientes:

$$\alpha = 1.4 \qquad\qquad \beta = 0.3 \tag{1.4}$$

Escriba un programa en C++ que, dado un estado inicial (x_0, y_0), calcule el estado del sistema (x_k, y_k) en los instantes de tiempo discreto $k = 1, \ldots, N$ y escriba dichos valores en la consola, en tres columnas separadas por tabuladores. En la pri-

[1] Hénon, M. (1976). A two-dimensional mapping with a strange attractor. Commun. math. Phys. 50: 69 – 77.

mera columna deben escribirse los valores consecutivos de k, con $k = 0, 1, \ldots, N$. En la segunda y tercera columnas, deben escribirse los correspondientes valores x_k e y_k, en formato científico con 6 dígitos decimales.

Ejecute el programa para $N = 20$ y el estado inicial $x_0 = 0$, $y_0 = 0$. Muestre en la memoria una captura de pantalla del resultado obtenido.

Solución al Ejercicio 1

```
#include <iostream>
#include <iomanip>

const double alfa = 1.4;
const double beta = 0.3;
const int N = 20;

int main() {
    double x_k, y_k;
    std::cout << "Introduzca los valores iniciales:"
              << std::endl;
    std::cin >> x_k >> y_k;
    std::cout << "k\tx_k\t\ty_k" << std::endl;
    std::cout << 0 << "\t"
             << std::scientific << std::setprecision(6)
             << x_k << "\t" << y_k << std::endl;
    for (int k=1; k<=N; k++) {
        double x_kp1 = -alfa*x_k*x_k + y_k + 1;
        y_k = beta*x_k;
        x_k = x_kp1;
        std::cout << k << "\t"
                  << std::scientific << std::setprecision(6)
                  << x_k << "\t" << y_k << std::endl;
    }
    return 0;
}
```

EJERCICIO 2

En una bolsa hay 100 bolas, numeradas de 0 a 99. Consideremos el experimento siguiente. Extraemos al azar una bola de la bolsa, anotamos su número y volvemos a introducirla en la bolsa. Suponemos que la extracción de cada una de las bolas es equiprobable. Realizamos repetidamente estas operaciones hasta que extraemos la bola número 0. Llamaremos n al número de bolas diferentes numeradas de 1 a 99 que hemos extraído de la bolsa (una o más veces) antes de extraer la bola número 0.

Vamos un ejemplo. Supongamos que hemos anotado los números siguientes:

12, 23, 65, 1, 23, 2, 12, 10, 11, 1, 12, 0

Las bolas extraídas (una o más veces) antes de la número cero han sido las número 12, 23, 65, 1, 2, 10 y 11. Por tanto, en este experimento hemos obtenido $n = 7$. Obsérvese que aunque una bola se extraiga varias veces en el experimento, sólo se contabiliza una vez en dicho experimento.

Escriba un programa en C++ que realice N réplicas independientes (es decir, empleando diferentes secuencias de números pseudo aleatorios) del experimento, calculando en cada caso el valor de n. A continuación, el programa debe escribir en la consola el valor medio de los N valores de n obtenidos.

Como generador de números aleatorios, puede emplear la función `rand()`, que está definida en `stdlib.h`.

Ejecute el programa para $N = 10000$. Muestre en la memoria una captura de pantalla donde pueda verse el resultado obtenido de ejecutar el programa.

Solución al Ejercicio 2

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <vector>

const int N = 10000;       // Numero de replicas del experimento
const int nBolas = 100;  // Numero de bolas que hay en la bolsa

int main() {
    srand(time(NULL));
    double valorMedio_n = 0;
    for (int i=0; i<N; i++) {
        int u;          // Numero de bola extraida de la bolsa
        int n = 0;    // Numero de bolas diferentes extraidas
        std::vector<bool> bolaExtraida(nBolas, false);
        do {
            u = rand() % nBolas;      // u en el rango 0 a (nBolas-1);
            if (!bolaExtraida[u]) {
                n++;
                bolaExtraida[u] = true;
            }
        } while (!bolaExtraida[0]);
        valorMedio_n += n-1;
    }
    std::cout << "Media = " << valorMedio_n/N << std::endl;
    return 0;
}
```

EJERCICIO 3

Diremos que un vector es *ralo* cuando una gran proporción de sus componentes vale cero. Por ejemplo, el vector $\mathbf{x}$ tiene 100 componentes, de los cuales sólo 5 son diferentes de cero. Éstos son los componentes de índice 4, 11, 37, 44 y 88 (la numeración del índice comienza en cero).

$$\begin{array}{rcccccccccc} \mathbf{x} = \{ & 0, & 0, & 0, & 0, & 0.1, & 0, & 0, & 0, & 0, & 0, \\ & 0, & 7.7, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0 \\ & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\ & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 1.3, & 0, & 0 \\ & 0, & 0, & 0, & 0, & 6.2, & 0, & 0, & 0, & 0, & 0, \\ & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0 \\ & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\ & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\ & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & -1.2, & 0, \\ & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0 \quad \} \end{array} \tag{1.5}$$

Una forma de almacenar $\mathbf{x}$ sería guardando el valor de cada uno de sus 100 componentes. Sin embargo, dado que es un vector ralo, resulta más económico (en términos del espacio ocupado en la memoria del ordenador) almacenar sólo aquellos componentes diferentes de cero y asumir que todos los demás componentes valen cero.

Representando mediante una pareja (valor, índice) cada componente diferente de cero, el vector $\mathbf{x}$ podría almacenarse de la forma siguiente:

$$\{ \quad (0.1, 4), \quad (7.7, 11), \quad (1.3, 37), \quad (6.2, 44), \quad (-1.2, 88) \quad \} \tag{1.6}$$

Obsérvese que las parejas están ordenadas en orden creciente de su índice. Esta representación del vector se denomina *forma empaquetada*, mientras que la representación mostrada en (1.5) se denomina *forma expandida*. Denominaremos *agrupamiento* a la operación consistente en obtener la forma empaquetada a partir de la forma expandida y denominaremos *expansión* a la operación inversa.

Escriba un programa en C++ que:

1. Declare dos vectores $\mathbf{x_1}$ y $\mathbf{x_2}$, mediante la especificación de sus formas empaquetadas. Asigne los valores que usted desee a los componentes de los

vectores. Emplee las estructuras de datos que usted considere más adecuadas para implementar la forma empaquetada de los vectores.

2. Calcule el producto escalar de ambos vectores operando sus formas empaquetadas, sin expandir ninguno de ellos. El programa debe mostrar el resultado en la consola.

3. Calcule el vector $\mathbf{z}$, definido como

$$\mathbf{z} = \alpha_1 \cdot \mathbf{x_1} + \alpha_2 \cdot \mathbf{x_2} \tag{1.7}$$

donde α_1 y α_2 son constantes reales. El calculo debe hacerse operando las formas empaquetadas de $\mathbf{x_1}$ y $\mathbf{x_2}$, sin expandir ninguno de los dos vectores. El programa debe mostrar en la consola, en forma empaquetada, el vector $\mathbf{z}$ obtenido.

Explique en la memoria las pruebas que usted ha hecho para verificar que su programa funciona correctamente. Asimismo, muestre en la memoria el resultado de la ejecución de su programa para las siguientes cuatro selecciones de α_1 y α_2:

$$\begin{array}{ll} \alpha_1 = 0 & \alpha_2 = 0 \\ \alpha_1 = 1 & \alpha_2 = 0 \\ \alpha_1 = 0 & \alpha_2 = 1 \\ \alpha_1 = 6.2 & \alpha_2 = -1.3 \end{array} \tag{1.8}$$

Recuerde que sólo forman parte de la forma empaquetada aquellos componentes cuyo valor es diferente de cero.

Recuerde también que las parejas (valor, índice) deben estar ordenadas en orden creciente de su índice.

Solución al Ejercicio 3

```
#include <iostream>
#include <vector>


struct Componente {
    double val;
    int ind;
};


int main() {
// Definición de x1
    Componente x1[] = {{0.1,1 },{0.1,4 },{7.7,11 },{1.3,37 },
                       {12,40},  {6.2,44 },{-1.2,86 },{-1.2,88 },
                       {1.2,188 },{1.6,799 },{-3.2,1388 }};
// Definición de x2
    Componente x2[] = {{-0.1,0 },{0.5,10 },{3.9,25 },
                       {8.3,36 },{5.3,38 },{1.3,39 },{6.2,45 },
                       {-1.2,88 },{1.3,137 },{6.2,188 },{-2.2,534 }};
// Número de componentes de los arrays
    int N1 = sizeof(x1) / sizeof(*x1);
    int N2 = sizeof(x2) / sizeof(*x2);
// Producto escalar
    double prod = 0;
    int k = 0;
    for (int i1 = 0; k < N2 && i1 < N1; i1++) {
        for (int i2 = k; i2 < N2; i2++) {
            if (x1[i1].ind < x2[i2].ind) {
                k = i2;
                break;
            } else if (x1[i1].ind == x2[i2].ind) {
                prod += x1[i1].val * x2[i2].val;
                k = i2 + 1;
                break;
            }
        }
    }
    std::cout << "Producto escalar = " << prod << std::endl;
```

```
// Suma de múltiplos
    const double alfa1 = 6.2;
    const double alfa2 = -1.3;
    std::vector<Componente> z;
    k = 0;
    int i1 = 0, i2 = 0;
    while ( i1 < N1 || i2 < N2 ){
        if ( i1 == N1 ){
            if ( alfa2 != 0 ){
                Componente c = { alfa2*x2[i2].val, x2[i2].ind };
                z.push_back(c);
            }
            i2++;
        } else if ( i2 == N2 ){
            if ( alfa1 != 0 ){
                Componente c = { alfa1*x1[i1].val, x1[i1].ind };
                z.push_back(c);
            }
            i1++;
        } else if (x1[i1].ind > x2[i2].ind){
            if ( alfa2 != 0 ){
                Componente c = { alfa2*x2[i2].val, x2[i2].ind };
                z.push_back(c);
            }
            i2++;
        } else if (x1[i1].ind == x2[i2].ind){
            double val = alfa1*x1[i1].val+alfa2*x2[i2].val;
            if ( val != 0 ){
                Componente c = { val, x1[i1].ind };
                z.push_back(c);
            }
            i1++;
            i2++;
        } else {
            if ( alfa1 != 0 ){
                Componente c = { alfa1*x1[i1].val, x1[i1].ind };
                z.push_back(c);
            }
            i1++;
        }
    }

    std::cout << "Suma de multiplos" << std::endl;
    for (unsigned int i = 0; i < z.size(); i++)
        std::cout << z[i].val << ", " << z[i].ind << std::endl;
    return 0;
}
```

EJERCICIO 4

Supongamos que un empleado trabaja en una oficina revisando expedientes. Al comienzo de la jornada laboral, el empleado está ocioso y no hay ningún expediente pendiente de revisión en la oficina. A lo largo de la jornada laboral van llegando, de uno en uno, expedientes a la oficina. Cada expediente está identificado de manera unívoca mediante un número entero N_{ident}.

Cada vez que llega un expediente a la oficina, puede darse una de estas dos situaciones:

- Si el empleado está ocioso, entonces comienza inmediatamente a revisar el expediente. El empleado pasa de estar ocioso a estar ocupado.
- Si en ese instante el empleado está ocupado, el empleado no interrumpe su trabajo. El expediente se pone en la cola de espera de revisión.

Cuando el empleado termina de revisar un expediente, el expediente abandona la oficina. Puede entonces darse una de estas dos situaciones:

- Si no hay expedientes en la cola de espera de revisión, el empleado pasa de estar ocupado a estar ocioso.
- Si, por el contrario, hay expedientes en la cola de espera de revisión, el empleado comienza inmediatamente a revisar uno cualquiera de ellos.

En un fichero de texto llamado *datosIn.txt* está almacenada la información relativa a la llegada y revisión de los expedientes. Cada fila del fichero hace referencia a un expediente y está compuesta por tres números enteros, separados por espacios en blanco, cuyo significado es el siguiente.

- En la primera columna está escrito el identificador del expediente, N_{ident}.
- En la segunda columna se indica el instante de tiempo en que se recibe en la oficina ese expediente. El tiempo se mide en minutos, considerando que la jornada laboral del empleado comienza en el minuto cero. Obsérvese que los expedientes se reciben de uno en uno y que en el fichero *datosIn.txt* aparecen ordenados por orden de llegada.
- En la tercera columna se indica el tiempo, expresado en minutos, que necesitará el empleado para revisar ese expediente.

Escriba un programa en C++ que calcule el porcentaje del tiempo de la jornada laboral durante el cual el empleado permanece ocioso. La jornada laboral comienza en el instante cero y finaliza en el instante $T = 480$ minutos. El programa debe escribir dicho porcentaje en la consola.

Explique en la memoria las pruebas que ha hecho para verificar que su programa funciona correctamente.

Además, incluya en la memoria capturas de pantalla del resultado que ha obtenido al ejecutar su programa tres veces, usando los siguientes tres ficheros de entrada.

Primer fichero de entrada:

```
1     10    10
2     100   10
3     200   28
```

Segundo fichero de entrada:

```
1     0     120
2     100   100
3     200   20
```

Tercer fichero de entrada:

```
1     1    8
11    10   120
2     20   100
34    30   20
4    100   40
5    160   20
26   380   200
7    440   90
```

Solución al Ejercicio 4

```
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <algorithm>

const char nombreFichI[] = "datosIn.txt";

struct Expediente {
    int ident;        // Identificador
    int t_llegada;    // Instante en que llega
    int t_revision;// Tiempo necesario para revisarlo
};

const int Tfin = 480;    // Instante final de la simulación

int main() {
    // Lectura del fichero
    std::ifstream inFich(nombreFichI,std::ios::in);
    if (!inFich) {
        std::cout << "ERROR al abrir el fichero "
            << nombreFichI << std::endl;
        return 1;
    }
    std::vector<Expediente> expedientes;
    while (!inFich.eof()) {
        Expediente e;
        inFich >> e.ident >> e.t_llegada >> e.t_revision;
        if (!inFich.eof()) {
            expedientes.push_back(e);
        }
    }
    inFich.close();
    if ( expedientes.size() == 0 ) {
        std::cout << "Fichero vacio" << std::endl;
        return 0;
    }
```

```
    // Cálculo del porcentaje de tiempo ocioso
    int Tocio = 0;                  // Tiempo total ocioso
    int TrabajoPendiente = 0; // Tiempo de trabajo pendiente
    int tLast = 0;                  // Instante del último evento
    for (unsigned int i=0; tLast<Tfin && i<expedientes.size(); i++) {
        if (TrabajoPendiente == 0) {
            Tocio += std::min(Tfin,expedientes[i].t_llegada) - tLast;
            TrabajoPendiente = expedientes[i].t_revision;
        } else if (TrabajoPendiente >= expedientes[i].t_llegada - tLast) {
            TrabajoPendiente += expedientes[i].t_revision -
                                (expedientes[i].t_llegada - tLast);
        } else {
            Tocio += std::max(0,
                      std::min(Tfin,expedientes[i].t_llegada)
                      - tLast - TrabajoPendiente);
            TrabajoPendiente = expedientes[i].t_revision;
        }
        tLast = expedientes[i].t_llegada;
    }
    if (Tfin > tLast + TrabajoPendiente)
        Tocio += Tfin - tLast - TrabajoPendiente;
    // Salida por consola
    std::cout << "Porcentaje de tiempo ocioso = "
          << (double)Tocio*100/Tfin << std::endl;
    return 0;
}
```