LENGUAJES DE PROGRAMACIÓN

Solución al Trabajo Práctico - Junio de 2021

EJERCICIO 1

La *suma de Riemann* es un método para obtener una aproximación de la integral definida I de la función $f(x)$ con respecto a x, en el intervalo finito $[a, b]$, esto es:

$$I = \int_a^b f\,(x) \cdot dx$$

Partiendo del hecho de que la integral definida es igual al área bajo la curva de la función $f(x)$, desde $x = a$ a $x = b$, el método consiste en dividir el área en un conjunto de N rectángulos y sumar sus áreas.

Una de las formas de definir los rectángulos es de manera que la altura del rectángulo sea igual al valor medio de la función en sus extremos. Es decir, si se divide el intervalo $[a, b]$ en N subintervalos $[x_0, x_1]$, $[x_1, x_2]$, ..., $[x_{N-1}, x_N]$, con $x_0 = a$, $x_N = b$, y $x_i < x_{i+1}$ para $i = 0, \ldots, N-1$, la aproximación a la integral I se calcula de la forma siguiente:

$$I \simeq \sum_{i=0}^{N-1} \frac{f\,(x_{i+1}) + f\,(x_i)}{2} \cdot (x_{i+1} - x_i)$$

Escriba un programa en C++ que calcule la integral definida de la función

$$f(x) = x^m$$

en el intervalo $[a, b]$, donde m es un número entero mayor que cero, y a y b son números reales, satisfaciéndose $a < b$.

El programa debe realizar las acciones siguientes:

1. Declarar una constante global de tipo **int** llamada `N` y asignarle el valor 10000. Esta variable almacena el número N de rectángulos del método.

2. Escribir un mensaje en la consola solicitando al usuario que introduzca los valores de m, a y b. Leer los valores de consola, almacenándolos en una variable **int** llamada `m`, y en variables **double** llamadas `a` y `b`.

3. Comprobar que se satisface $a < b$ y también que se satisface $m > 0$. En caso contrario, el programa debe indicarlo mediante un mensaje escrito en la consola y terminar.

4. Calcular mediante el método descrito anteriormente la integral definida de $f(x) = x^m$ en el intervalo $[a, b]$, dividiendo para ello dicho intervalo en N subintervalos de igual longitud. Esto es:

$$x_{i+1} - x_i = \frac{b-a}{N} \qquad \text{para} \quad i = 0, \ldots, N-1$$

5. Escribir en la consola, en formato científico con 10 dígitos de precisión, el valor estimado de la integral definida, el valor de dicha integral calculado a partir de la expresión analítica de la integral,

$$I = \int_a^b x^m \cdot dx = \left[\frac{x^{m+1}}{m+1}\right]_a^b = \frac{b^{m+1} - a^{m+1}}{m+1}$$

   y también la diferencia entre ambos valores.

6. Terminar.

Solución al Ejercicio 1

```cpp
#include <iostream>
#include <iomanip>
#include <cmath>

const int N = 10000;

int main() {
  // Entrada de datos por consola
  std::cout << "Introduzca m: ";
  int m;
  std::cin >> m;
  std::cout << "Introduzca los extremos del intervalo (a, b): ";
  double a, b;
  std::cin >> a >> b;
  // Comprobación de datos
  if (!(a < b) || !(m > 0)) {
    std::cout << "Datos no validos" << std::endl;
    return 0;
  }
  // Estimación de la integral
  double Inc_x = (b - a) / N;
  double I = 0.5 * (std::pow(a, m) + std::pow(b, m));
  for (int i = 1; i < N; i++)
    I += std::pow(a + i * Inc_x, m);
  I *= Inc_x;
  // Salida por consola
  double Iexacta = (std::pow(b, m + 1) - std::pow(a, m + 1)) / (m + 1);
  std::cout << std::scientific << std::setprecision(10)
      << "Valor aprox.:\t" << I
      << "\nValor exacto:\t" << Iexacta
      << "\nError:\t" << Iexacta - I
      << std::endl;
  return 0;
}
```

EJERCICIO 2

Una matriz real $\mathbf{A}$, con N filas y N columnas, es una matriz con diagonal estrictamente dominante por filas cuando satisface:

$$|a_{i,i}| > \sum_{\substack{j=1 \\ j \neq i}}^{N} |a_{i,j}| \qquad \text{para todo} \quad i = 1, \ldots, N$$

donde $|a_{i,j}|$ representa el valor absoluto del elemento de la matriz que está situado en la fila i, columna j.

Escriba un programa en C++ que realice las acciones siguientes.

1. Declarar una constante global **int** llamada N y asignarle el valor 4.
2. Declarar un array bidimensional global de **double** llamado $\mathbf{A}$, con tamaño $N \times N$.
3. Solicitar al usuario que introduzca por consola los elementos de $\mathbf{A}$.
4. Calcular si la matriz introducida por el usuario es una matriz con diagonal estrictamente dominante por filas. Mostrar en la consola un mensaje indicándolo. En caso afirmativo, terminar.
5. Buscar si, permutando columnas de la matriz $\mathbf{A}$, es posible obtener una matriz con diagonal estrictamente dominante por filas. Si existe, mostrar en la consola dicha matriz. Si no existe, mostrar un mensaje en la consola indicándolo.
6. Terminar.

A continuación se muestran tres ejemplos: (1) $\mathbf{A}_1$ es una matriz con diagonal estrictamente dominante por filas; (2) $\mathbf{A}_2$ no lo es, pero mediante permutaciones de sus columnas es posible obtener una matriz con diagonal estrictamente dominante por filas: la matriz $\mathbf{A}_1$; (3) $\mathbf{A}_3$ ni es una matriz con diagonal estrictamente dominante por filas, ni puede transformarse en una mediante permutaciones de sus columnas.

$$\mathbf{A}_1 = \begin{pmatrix} 6 & -1 & 2 & 0 \\ 1 & 4 & 1 & -1 \\ 2 & -1 & 18 & 5 \\ 1 & -1 & 0 & -4 \end{pmatrix}, \quad \mathbf{A}_2 = \begin{pmatrix} 0 & -1 & 6 & 2 \\ -1 & 4 & 1 & 1 \\ 5 & -1 & 2 & 18 \\ -4 & -1 & 1 & 0 \end{pmatrix}, \quad \mathbf{A}_3 = \begin{pmatrix} 0 & -1 & 6 & 2 \\ -1 & 4 & 1 & 12 \\ 5 & -1 & 2 & 18 \\ -4 & -1 & 1 & 0 \end{pmatrix}$$

Solución al Ejercicio 2

```cpp
#include <iostream>
#include <cmath>

const int N = 4;
double A[N][N];

int main() {
  // Entrada de la matriz por consola
  std::cout << "Introduzca A (por filas):\n";
  for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
      std::cin >> A[i][j];
  // Análisis de A
  bool diagDominante = true;
  for (int i = 0; diagDominante && i < N; i++) {
    double suma = 0;
    for (int j = 0; j < N; j++)
      if (i != j)
        suma += std::abs(A[i][j]);
    diagDominante = (std::abs(A[i][i]) > suma);
  }
  if (diagDominante) {
    std::cout << "Tiene diagonal estrictamente dominante\n";
    return 0;
  }
  std::cout << "No tiene diagonal estrictamente dominante\n";
  // Búsqueda de permutaciones de columnas
  int pos_max[N] = { 0 };
  for (int i = 0; i < N; i++) {
    double sumaFila = std::abs(A[i][0]);
    for (int j = 1; j < N; j++) {
      sumaFila += std::abs(A[i][j]);
      if (std::abs(A[i][j]) > std::abs(A[i][pos_max[i]]))
        pos_max[i] = j;
    }
    if (2 * std::abs(A[i][pos_max[i]]) <= sumaFila) {
      std::cout << "No puede transformarse" << std::endl;
      return 0;
    }
    for (int k = 0; k < i; k++)
      if (pos_max[k] == pos_max[i]) {
        std::cout << "No puede transformarse" << std::endl;
        return 0;
      }
  }
  // Escritura en consola de la matriz permutada
  std::cout << "Matriz con columnas permutadas" << std::endl;
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++)
      std::cout << A[i][pos_max[j]] << "\t";
    std::cout << std::endl;
  }
  return 0;
}
```

EJERCICIO 3

Se desea calcular el valor, en sucesivos instantes de tiempo, del nivel de agua dentro de un depósito cilíndrico que tiene un orificio circular en su base, a través del cual sale el agua.

En el instante inicial, el nivel de agua dentro del depósito es h_0 y se abre el orificio que hay en la base del depósito. Entonces, el agua comienza a salir a través del orificio situado en la base del depósito, disminuyendo el nivel de agua en el depósito a medida que transcurre el tiempo.

El nivel de agua h en el depósito viene descrito por la ecuación diferencial ordinaria siguiente:

$$\frac{dh}{dt} = -\frac{\alpha \cdot A}{B} \cdot \sqrt{2 \cdot g \cdot h}$$

donde t es el tiempo, A es la sección del orificio, B es la sección de la base (circular) del depósito y g es la aceleración gravitatoria. El factor de contracción $\alpha = 0.6$ describe el hecho de que la sección del flujo saliente de agua es menor que la sección del orificio.

Para integrar numéricamente la ecuación diferencial ordinaria anterior, emplearemos el *método de Euler mejorado*. Para explicar el método, representaremos la ecuación de la forma genérica siguiente:

$$\frac{dh}{dt} = f(h)$$

Llamaremos h_i al valor de h en el instante t_i, y h_{i+1} al valor de h en el instante $t_{i+1} = t_i + \Delta t$. El tamaño del paso en el tiempo del método de integración es Δt. El método calcula h_{i+1} a partir de h_i en dos pasos, de la forma descrita a continuación.

1. *Paso predictor*. Se calcula una primera aproximación, $h_{i+1}^{(p)}$, empleando para ello el método de Euler:

$$h_{i+1}^{(p)} = h_i + f(h_i) \cdot \Delta t$$

2. *Paso corrector*. Se calcula h_{i+1} de la forma:

$$h_{i+1} = h_i + \frac{f(h_i) + f(h_{i+1}^{(p)})}{2} \cdot \Delta t$$

Escriba un programa en C++ que calcule y escriba en un fichero de texto llamado *nivelAgua.txt* el valor del nivel de agua h (expresado en cm) en los instantes 0, 10 minutos, 20 minutos, $\ldots$, hasta que el nivel de agua sea menor que h_{fin}.

El fichero debe tener dos columnas: en la primera debe aparecer el instante de tiempo (en minutos) y en la segunda el nivel de agua (en cm) en dicho instante. Los números deben escribirse en formato fijo, con un dígito decimal.

El diámetro de la base circular del depósito mide 200 cm. El diámetro del orificio mide 1 cm. La aceleración gravitatoria vale aproximadamente $g = 980$ cm/s^2. En el instante inicial, $t_0 = 0$ s, el nivel de agua es $h_0 = 250$ cm.

El tamaño del paso del método de integración es $\Delta t = 1$ s. La condición de finalización es que el nivel de agua sea menor que $h_{fin} = 1$ mm.

Además de escribir la información anteriormente indicada en el fichero de texto *nivelAgua.txt*, el programa debe escribir en la consola el instante de tiempo, expresado en segundos, en que se satisface la condición de finalización.

Solución al Ejercicio 3

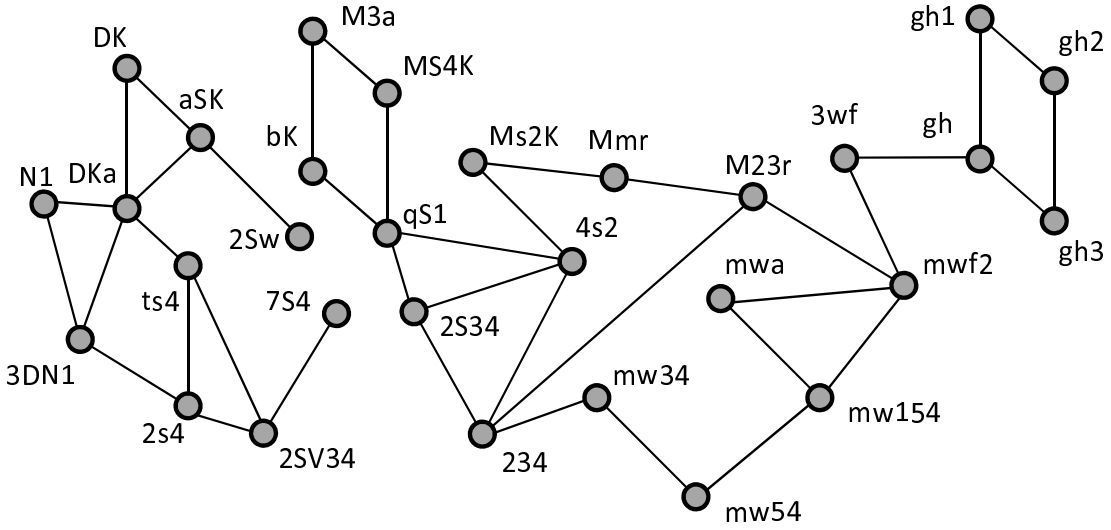
```cpp
#include <iostream>
#include <iomanip>
#include <cmath>
#include <stdexcept>
#include <sstream>
#include <fstream>

const char fich[] = "nivelAgua.txt";
const double B = std::acos(-1) * 100 * 100;  // cm^2
const double A = std::acos(-1) * 0.5 * 0.5;  // cm^2
const double alfa = 0.6;
const double g = 980;                        // cm/s^2

const double t_ini = 0;      // s
const double h_ini = 250;    // cm
const double h_fin = 0.1;    // cm

const double Inc_t = 1;      // s
const double Tcom = 600;    // s

double f(double h) throw (std::invalid_argument) {
  if (h < 0)
    throw std::invalid_argument("Error: h menor que cero");
  return -alfa * A / B * std::sqrt(2 * g * h);
}
```

```
26
27 int main() {
28   // Inicialización
29   double h = h_ini;
30   double t = t_ini;
31   double t_com = Tcom;
32   std::stringstream ss;
33   ss << std::fixed << std::setprecision(1)
34     << t / 60 << "\t" << h << "\n";
35   try {
36     // Algoritmo de integración
37     while (h >= h_fin) {
38       double h_p = h + f(h) * Inc_t;
39       h += 0.5 * (f(h) + f(h_p)) * Inc_t;
40       t += Inc_t;
41       if (t >= t_com || h < h_fin) {
42         ss << t / 60 << "\t" << h << "\n";
43         t_com += Tcom;
44       }
45     }
46     std::cout << "t final = " << t << " s" << std::endl;
47   } catch (std::invalid_argument &exc) {
48     std::cout << exc.what() << std::endl;
49   }
50   // Escritura en fichero
51   std::ofstream outFich(fich, std::ios::out | std::ios::trunc);
52   if (!outFich) {
53     std::cerr << "Error al abrir fichero para escritura";
54     return 0;
55   }
56   outFich << ss.str() << std::endl;
57   outFich.close();
58   return 0;
59 }
```

EJERCICIO 4

Un *grafo* consiste en puntos, llamados *vértices*, y líneas conectándolos, llamadas *aristas*. El problema de encontrar el camino más corto a través de una red de ordenadores puede modelarse mediante grafos, en los cuales los vértices representan los ordenadores y las aristas las conexiones entre ellos.

En la figura situada bajo estas líneas se muestra un ejemplo. Cada ordenador tiene asignado un identificador, compuesto por una secuencia de números y/o letras, que lo designa de manera unívoca.



Seleccionados dos vértices, puede no existir ningún camino a través de las aristas que los conecte, puede existir un único camino, o más de uno. La longitud de un camino es igual al número de aristas por las que pasa el camino.

En nuestro problema en concreto, el grafo se almacena en un fichero de texto llamado *redes.txt*, que contiene dos columnas de identificadores. Cada línea del fichero describe una arista, que une los dos vértices indicados mediante los dos identificadores.

Por ejemplo, el fichero *redes.txt* listado a continuación describe el sistema mostrado en la figura anterior.

Fichero redes.txt

```
N1     DKa
N1     3DN1
3DN1   DKa
3DN1   2s4
2s4    ts4
2s4    2SV34
2SV34  ts4
2SV34  7S4
ts4    DKa
DKa    DK
DKa    aSK
DK     aSK
aSK    2Sw
M3a    bK
M3a    MS4K
bK     qS1
MS4K   qS1
qS1    4s2
qS1    2S34
2S34   4s2
2S34   234
234    4s2
234    M23r
4s2    Ms2K
234    mw34
mw34   mw54
mw54   mw154
mw154  mwa
mw154  mwf2
mwa    mwf2
mwf2   M23r
mwf2   3wf
M23r   Mmr
Mmr    Ms2K
3wf    gh
gh     gh1
gh     gh3
gh1    gh2
gh2    gh3
```

Escriba un programa en C++ que realice las acciones descritas a continuación.

1. Leer el fichero *redes.txt*. Si se produce error, indicarlo mediante un mensaje en consola y terminar.
2. Escribir un mensaje en la consola solicitando al usuario que introduzca el identificador del ordenador origen y el identificador del ordenador destino. Leer de consola ambos identificadores.
3. Si alguno de los identificadores introducidos por el usuario no aparece en el fichero *redes.txt*, indicarlo mediante un mensaje escrito en consola y terminar.
4. Determinar si existe al menos un camino entre el ordenador origen y destino y, en caso afirmativo, calcular la longitud del camino más corto. Escribir el resultado en la consola.
5. Terminar.

Seguidamente se muestran tres casos de prueba de la ejecución del programa para el fichero *redes.txt* anteriormente listado.

Caso de prueba 1

```
Introduzca ID origen: N1
Introduzca ID destino: M23r
No existe camino
```

Caso de prueba 2

```
Introduzca ID origen: 7S4
Introduzca ID destino: 2Sw
Longitud camino = 5
```

Caso de prueba 3

```
Introduzca ID origen: M3a
Introduzca ID destino: gh2
Longitud camino = 10
```

Solución al Ejercicio 4

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <map>

const std::string fich = "redes.txt";

struct Nodo {
  std::vector<std::string> id_adyacentes;
  int etiqueta;
};

int main() {
  // Lectura del fichero al mapa directorio
  std::ifstream inFich(fich.c_str(), std::ios::in);
  if (!inFich) {
    std::cerr << "Error al abrir el fichero" << std::endl;
    return 0;
  }

  std::map<std::string, Nodo> directorio;
  std::string ident1, ident2;
  while (inFich >> ident1 >> ident2) {
    std::map<std::string, Nodo>::iterator it;
    it = directorio.find(ident1);
    if (it != directorio.end()) {
      it->second.id_adyacentes.push_back(ident2);
    } else {
      std::vector<std::string> ady(1, ident2);
      Nodo nn = { ady, -1 };
      directorio[ident1] = nn;
    }
    it = directorio.find(ident2);
    if (it != directorio.end()) {
      it->second.id_adyacentes.push_back(ident1);
    } else {
      std::vector<std::string> ady(1, ident1);
      Nodo nn = { ady, -1 };
      directorio[ident2] = nn;
    }
  }
  inFich.close();

  // Entrada por consola identificadores origen y destino
  std::string ID_origen, ID_destino;
  std::cout << "Introduzca ID origen: ";
  std::cin >> ID_origen;
  std::cout << "Introduzca ID destino: ";
  std::cin >> ID_destino;

```

```
52   // Comprobación de los ID
53   if (directorio.find(ID_origen) == directorio.end()) {
54     std::cerr << ID_origen << " no encontrado";
55     return 0;
56   }
57   if (directorio.find(ID_destino) == directorio.end()) {
58     std::cerr << ID_destino << " no encontrado";
59     return 0;
60   }
61
62   // Búsqueda del camino
63   int lonCamino = 0;
64   std::vector<std::string> etiquetar(1, ID_origen);
65   while (etiquetar.size() > 0) {
66     std::vector<std::string> etiquetarN;
67     for (unsigned int i = 0; i < etiquetar.size(); i++) {
68       if (etiquetar[i] == ID_destino) {
69         std::cout << "Longitud camino = " << lonCamino << "\n";
70         return 0;
71       }
72       std::map<std::string, Nodo>::iterator it = directorio.find(
73           etiquetar[i]);
74       if (it->second.etiqueta == -1) {
75         it->second.etiqueta = lonCamino;
76         etiquetarN.insert(etiquetarN.end(),
77             it->second.id_adyacentes.begin(),
78             it->second.id_adyacentes.end());
79       }
80     }
81     lonCamino++;
82     etiquetar = etiquetarN;
83   }
84   std::cout << "No existe camino" << std::endl;
85   return 0;
86 }
```