**Curso básico de introducción a la algoritmia.**

El objetivo del curso es dotar a los alumnos de las herramientas, procedimientos y conocimientos teórico-prácticos necesarios para enfrentarse a los enunciados y problemas presentados en las asignaturas de programación, garantizando así un nivel base que ayude a los asistentes al curso a mejorar su rendimiento académico en dichas asignaturas.

El temario del curso viene establecido por las necesidades de los alumnos, tomando como punto de referencia el temario de las asignaturas “Metodología de la Programación I” y “Estructuras de Datos I” de la titulación de “Ingeniero Técnico en Informática de Sistemas”. Dado que estas asignaturas se impartían en pseudolenguaje y ADA 95 y el temario de las asignaturas equivalentes es tratado de cara al lenguaje JAVA, el profesor del curso ha utilizado este último lenguaje como principal.

En primer lugar, se imparte un conocimiento teórico a los asistentes acerca de cuestiones tales como tipos de datos primitivos (enteros, reales, tipo carácter y booleanos) además de sentencias de control de flujo (estructuras repetitivas, condicionales y recursividad). También, se impartieron conocimientos acerca de pruebas de programas y metodologías de programación ágil (pruebas para los programas frente a programas para las pruebas).

Una vez asimilados los conceptos teóricos, los asistentes comienzan el aprendizaje práctico de esos mismos conocimientos y la asimilación mediante distintos problemas propuestos. De la misma manera, se les inculcan también conocimientos sobre traza de programas y cómo dividir un enunciado en problemas más simples para llegar a una solución. Por ejemplo, se planteó a los alumnos un problema en el cual tendrían que desarrollar un programa que verificase si un número (entero y positivo) dado es palíndromo o no, es decir, si el número leído de izquierda a derecha, es igual al número leído de derecha a izquierda. En primer lugar se aconsejó a los alumnos hacer una prueba para un caso base, el cual fue visto por los mismos como que cualquier número menor de 10 (es decir, de un único dígito) tenía que ser palíndromo.

Código del primer Test 1-1

@Test

public void test1(){

n=2;

assertEquals(n,2);

}

Además se ideó una prueba en la cual se probaban todos los números de un dígito.

Código del Test 1-2

@Test

public void test2(){

for(int i=0;i<10;i++){

assertEquals((espal(i)),true);

}

}

Una vez realizado el código que verificaba la correctitud de estas dos pruebas, se llegó al caso general, y los alumnos razonaron que la mejor opción podía ser apoyarse en una función auxiliar que diese la vuelta al número dado, ya que un número palíndromo tiene que ser igual, por definición, a ese mismo número dado la vuelta.

Código de la función auxiliar 2-1

public int dadovuelta(int n){

int reves=0;

while(n>0){

reves=reves\*10+n%10;

n=n/10;

}

return reves;

}

Con lo cual, con esta metodología, el problema propuesto resulta más fácil de resolver.

Código de resolución del problema propuesto 3-1

public boolean espal(int n){

if(n<10)return true;

else{

if(dadovuelta(n)==n)return true;

else return (n==dadovuelta(n));

}

}

A continuación, se abordó el problema referente a los arrays, ya que los alumnos pidieron que se diera una mayor importancia a este tema porque se vieron sobrepasados por los ejercicios propuestos por sus asignaturas de programación en este tema en concreto, además de por la proximidad de un parcial del mismo tema.

Tras una explicación teórica del funcionamiento de los arrays, se les planteó a los alumnos problemas típicos de vectores y matrices, entre ellos, uno de los ejercicios de los parciales que tuvieron los alumnos durante el desarrollo del curso.

Ejercicios de vectores:

Como introducción, se les plantea a los alumnos el cálculo del producto escalar; a continuación se les incrementó un poco el nivel al pedirles que ordenen un vector de enteros y a continuación se les propuso comprobar si un vector es palíndromo o no. Seguidamente, se propuso diversos ejercicios con matrices, además de resolver uno de los ejercicios de los parciales que trajeron los alumnos en el cual se trabajaba con una matriz de caracteres representativos de los números romanos. Tras conseguir realizar el ejercicio, se les planteó algo más complicado: buscar una matriz dentro de otra.

Los alumnos mejoraron progresivamente en el manejo de estas estructuras de datos, con lo que el profesor procedió a incrementar el nivel introduciendo el concepto de recursividad. Tras una explicación teórica del término y de cómo funciona la recursividad, se procedió a realizar algunos ejercicios sencillos como la realización de una suma o una multiplicación de forma recursiva.

Llegados a este punto, los alumnos reconocieron que no entendían cómo funcionaba el paso de parámetros. Se les explicó de forma teórica cómo funciona el paso de parámetros y así mismo se probó si se podía pasar una fila de una matriz como si se tratase de un vector indicando únicamente la fila (primera dimensión) que se quería pasar a otra función, descubriendo que sí se puede hacer, ya que la otra forma conocida, era copiar en un vector auxiliar la fila con la que se desease trabajar.

Llegados a este punto, se procedió a explicarle a los alumnos como trabajar con tipos de datos “propios”, es decir, crear sus propios tipos de datos basándose en los tipos base anteriormente explicados, con lo que la mejor elección (basándose en los temarios de las asignaturas anteriormente citadas) era seguir con los arboles binarios de búsqueda, para lo cual, hubo que introducir teóricamente el concepto de árbol, nodo, nodos “padre” e “hijo”, punteros y objetos.

Tras la explicación teórica, se procedió a la asimilación práctica de dichos conceptos, y la elección del árbol binario de búsqueda se refuerza debido a que es el tipo abstracto de datos que mejor explota el funcionamiento de la recursividad, con lo cual, al mismo tiempo se refuerza la asimilación del funcionamiento de dicha herramienta.

Finalmente se le expuso a los alumnos el uso de metodologías ágiles al mismo tiempo que se les planteó las dos alternativas existentes respecto a la prueba de programas.

Los alumnos pudieron comprobar el funcionamiento de la realización de las pruebas ANTES de comenzar a desarrollar el código frente a la alternativa más común de realizar en primer lugar el código y una vez terminado, comenzar a realizar baterías de pruebas.

Se adjunta al final de este informe los códigos fuente de los ejercicios propuestos.

**Asistencia de los alumnos.**

Se observa, que la mayoría de los alumnos asistentes al curso se interesaron por el mismo debido a la proximidad de exámenes parciales de arrays, con lo cual, tras la primera clase (que tuvo asistencia masiva) la población de asistentes se redujo drásticamente a la mitad. Tras este primer descenso, el número de asistentes se mantuvo hasta las dos últimas clases (después de la ampliación de la duración del curso en otras dos semanas), con lo cual, se puede concluir que la asistencia al curso fue muy satisfactoria, teniendo en cuenta que tras el primer día, los asistentes demostraron verdadero interés por aprender y no solo por aprobar sus exámenes. Así mismo, los alumnos que asistieron con asiduidad al curso pudieron plantear y resolver dudas que traían de sus clases de programación y que, por vergüenza o por no tener un trato muy cercano con sus profesores, prefirieron compartir dichas dudas en el curso.

**Carencias a tener en cuenta por parte de los alumnos.**

Los asistentes al curso se sorprendieron de manera alarmante cuando se les habló de tests, prueba de programas y la realización de las mismas, ya que expusieron en el curso que nunca se les había enseñado el uso de dicha herramienta y a la gran mayoría les pareció de mucha utilidad. Del mismo modo, los asistentes no tenían ni la más remota idea de cómo utilizar la herramienta de depuración del entorno de desarrollo eclipse, ya que argumentaban que el entorno dado en la asignatura (AULAGA) carece de herramienta de depuración o no les ha sido explicada. Por tanto, se les enseñó el uso, manejo y funcionamiento de estas dos herramientas y, además, para el caso en el cual no pudieran utilizar ninguna de ellas, se les enseñó a hacer trazas de los programas a mano, en un folio, lo cual, también aseguraron no haberlo aprendido en sus asignaturas de programación, aunque algunos puntualizaron que ciertos profesores les sugirieron el uso de esta técnica durante la realización de sus ejercicios de examen.

**Agradecimientos de los alumnos.**

Los alumnos, al finalizar el curso, pidieron que se prorrogase la duración del mismo, ya que les estaba resultando muy útil e interesante, además de tener a alguien más cercano a ellos a quien poder preguntarle sus dudas. También han pedido que quede constancia de su agradecimiento tanto a la delegación de estudiantes, por la organización de este curso, como a la dirección del centro por su apoyo al mismo.

**Conclusiones.**

Teniendo en cuenta lo anteriormente expuesto, considero (personalmente) que la realización de este curso es un complemento muy apropiado a las asignaturas de programación de primer y segundo curso, ya que puede servir de refuerzo a los alumnos y puede ayudar a los asistentes al mismo a afrontar, con mayor seguridad en sí mismos, los ejercicios y prácticas propuestos por los profesores. Además, el curso, puede suponer un refuerzo positivo, ya que los alumnos dedican más horas al aprendizaje teórico-práctico del temario de las asignaturas sin suponer un esfuerzo adicional para los profesores.

**Anexo:**

**Códigos desarrollados en el curso: Iniciación**

import static org.junit.Assert.\*;

import org.junit.Test;

public class palindromo {

int n;

public int dadovuelta(int n){

int reves=0;

while(n>0){

reves=reves\*10+n%10;

n=n/10;

}

return reves;

}

public boolean espal(int n){

if(n<10)return true;

else{

if(dadovuelta(n)==n)return true;

else return (n==dadovuelta(n));

}

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// TODO Auto-generated method stub

}

@Test

public void test1(){

n=2;

assertEquals(n,2);

}

@Test

public void test2(){

for(int i=0;i<10;i++){

assertEquals((espal(i)),true);

}

}

@Test

public void test3(){

n=123;

assertEquals(dadovuelta(n),321);

}

@Test

public void test4(){

n=234;

assertEquals(espal(n),false);

n=123454321;

assertEquals(espal(n),true);

}

}

**Arrays: vectores y matrices.**

import static org.junit.Assert.\*;

import org.junit.Test;

import java.util.Random;

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

/\*\*

\*

\* @author Kike

\*/

public class mainvectores {

public int productoescalar(int[] vector, int[] vector2){

int res=0;

if(vector.length==vector2.length){

for(int i=0;i<vector.length;i++){

res+=(vector[i]\*vector2[i]);

}

}

return res;

}

public static boolean ordenavector(int[] vector){

int cambio=0;

for(int i=0;i<vector.length-1;i++){

for(int j=i+1;j<vector.length;j++){

if(vector[i]>vector[j]){

cambio=vector[i];

vector[i]=vector[j];

vector[j]=cambio;

}

}

}

return true;

}

public boolean palindromo(int[] vector){

for(int i=0;i<vector.length;i++){

if(vector[i]==vector[vector.length-i-1]){

if((i==vector.length-i-1)||(i+1==vector.length-i-1)){

return true;

}

}

else break;

}

return false;

}

public boolean palinrecur(int[] vec, int indice){

if(indice==vec.length-1-indice){

return true;

}

if(indice+1==vec.length-1-indice){

if(vec[indice]==vec[vec.length-1-indice]){

return true;

}

}

if(vec[indice]==vec[vec.length-1-indice]){

return palinrecur(vec,++indice);

}

else return false;

}

public boolean filasiguales(int[][] matriz){

boolean res=false;

int k=0;

boolean[] vector= new boolean [matriz[0].length];

for(k=0;k<vector.length;k++)vector[k]=false;

for(int i=0;i<matriz[0].length-1;i++){

for(int j=1;j<matriz[i].length-i;j++){

for(k=0;k<matriz[i].length;k++){

if (matriz[i][k]==matriz[i+j][k]){

vector[k]=true;

}

else{

break;

}

}

for(k=0;k<matriz[0].length;k++){

if (vector[k]==false){

res=false;

break;

}

}

if((k==vector.length)&&(vector[k-1]==true)){

return true;

}

}

}

return res;

}

public boolean contenido(int[] vector, int[] vector2){

boolean aux=false;

int i=0,j=0;

for (i=0;i<vector2.length;i++){

if(vector[0]==vector2[i]){

for(j=0;j<vector.length;j++){

if(vector[j]==vector2[i+j]){

aux=true;

}

else{

aux=false;

break;

}

}

if((aux)&&(j==vector.length)){

return aux;

}

}

}

return aux;

}

public boolean repetidos(int[] vector){

for(int i=0;i<vector.length-1;i++){

for (int j=i+1;j<vector.length;j++){

if(vector[i]==vector[j]){

return true;

}

}

}

return false;

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

int[] vector = new int [10];

boolean aux2;

for(int i=0;i<vector.length;i++){

Random aux = new Random();

vector[i]=(aux.nextInt()%10);

}

aux2=ordenavector(vector);

}

@Test

public void test1(){

int[] vector = new int [300];

int[] vector2= {4,3,2,1};

int[] vector3={1,2,3,4};

for(int i=0;i<300;i++){

Random aux = new Random();

vector[i]=aux.nextInt();

}

assertEquals(ordenavector(vector),true);

ordenavector(vector2);

//assertEquals(vector2,vector3);

}

@Test

public void test2(){

int[] vector = new int[10];

int[] vector2 = {1,2,3,4,3,2,1};

for(int i=0;i<vector.length;i++){

vector[i] = i;

}

//assertEquals(palindromo(vector),false);

//assertEquals(palindromo(vector2),true);

}

@Test

public void test3(){

int[] vector = {1,2,3,4};

int[] vector2 = {3,2,5,1};

assertEquals(productoescalar(vector,vector2),26);

}

@Test

public void test4(){

int[][] matriz = new int[10][10];

int[][] matriz2 = new int[10][10];

for(int i=0;i<matriz.length;i++){

for(int j=0;j<matriz.length;j++){

matriz[i][j]=j;

matriz2[i][j]=i;

}

}

assertEquals (filasiguales(matriz),true);

assertEquals (filasiguales(matriz2),false);

}

@Test

public void test5(){

int[] vector = {1,3,5,7};

int[] vector2= {1,3,5,2,4,5,1,3,5,7,2,3,4,5,6,1,3,5,6,1,3,5,7,1,1,1,3,5,7};

assertEquals(contenido(vector,vector2),true);

}

@Test

public void test6(){

int[] vector = {1,2,3,4};

int[] vector2 = {1,2,3,1};

assertEquals(repetidos(vector),false);

assertEquals(repetidos(vector2),true);

}

@Test

public void testpalindromo(){

int[] vectorunitario = {0};

int[] vector2 = {2,3,3,3,2};

assertEquals(palindromo(vectorunitario),true);

assertEquals(palindromo(vector2),true);

}

@Test

public void testrecursivo(){

int[] vectorunitario = {0};

int[] vector2 = {2,3,564,3,2};

assertEquals(palinrecur(vectorunitario,0),true);

assertEquals(palinrecur(vector2,0),true);

}

@Test

public void testrepetidos(){

int[] v={9};

//assertEquals(repetidos(v),false);

}

}

**Matrices.**

import org.junit.Test;

import static org.junit.Assert.\*;

public class matrices{

public int pasoparametros(){

char[][] matriz = new char[26][26];

char[] vector=new char[26];

int aux=0;

for (char i= 'A';i<='Z';i++){

vector[aux]=i;

aux++;

}

for (int i=0;i<matriz.length;i++){

for (int j=0;j<matriz[i].length;j++){

matriz[i][j]=vector[i];

}

}

aux=aux(matriz[12]);

return aux;

}

public boolean contenida(int[][] mat1, int[][] mat2){

boolean aux=false;

int k=0,l=0;

for(int i=0;i<mat2.length;i++){

for(int j=0;j<mat2[i].length;j++){

if((mat1[0][0]==mat2[i][j])&&(mat2.length-i>=mat1.length)&&(mat2[i].length-j>=mat1[i].length)){

for(k=0;k<mat1.length;k++){

for(l=0;l<mat1[i].length;l++){

if(mat1[k][l]==mat2[i+k][j+l]){

aux=true;

}

else{

aux=false;

break;

}

}

if(l!=mat1[i].length){

break;

}

}

if((aux)&&(k==mat1.length)&&(l==mat1[i].length)){

return aux;

}

}

}

}

return aux;

}

public int numRomano(char[][] mat){

int[] sumas = new int[mat.length];

int indice=0;

for(int i=0;i<mat.length;i++){

sumas[i]=aux(mat[i]);

}

for(int j=1;j<sumas.length;j++){

if(sumas[j]>=sumas[indice]){

indice=j;

}

}

return indice;

}

public int aux(char[] vaux){

int suma=0;

for(int i=0;i<vaux.length;i++){

switch(vaux[i]){

case 'I': suma+=1;

break;

case 'V': suma+=5;

break;

case 'X': suma+=10;

break;

case 'L': suma+=50;

break;

case 'C': suma+=100;

break;

case 'D': suma+=500;

break;

case 'M': suma+=1000;

break;

default: suma+=0;

}

}

return suma;

}

public int sumarecursiva(int a,int b){

if(b==0)return a;

return (sumarecursiva(a+1,b-1));

}

public int multrecur(int a, int b){

if(b==1){

return a;

}

else{

return a+(multrecur(a,b-1));

}

}

public static void main(String[] args){

}

@Test

public void pruebacontenido(){

int[][] m1={{1,2,3},{1,2,3},{1,2,3}};

int[][] m2={{1,2,3},{1,2,3},{1,2,3}};

int[][] m3=new int[9][9];

for(int i=0;i<m3.length;i++){

for(int j=0;j<m3[0].length;j++){

m3[i][j]=j;

}

}

assertEquals(contenida(m1,m2),true);

assertEquals(contenida(m1,m3),true);

}

@Test

public void pruebanumRomano(){

char[][] mat = {{'M','V','D'},{'L','M','X'},{'X','D','V'}};

assertEquals(numRomano(mat),0);

}

@Test

public void pruebasuma(){

int a=6;

int b=3;

assertEquals(sumarecursiva(a,b),9);

}

@Test

public void pruebamult(){

assertEquals(multrecur(15,15),225);

}

@Test

public void pruebapaso(){

assertEquals(pasoparametros(),26\*1000);

}

}

**Árbol binario de búsqueda.**

import static org.junit.Assert.\*;

import org.junit.Test;

import java.util.\*;

public class arbolbin{

class nodobin{

int clave;

nodobin[] enlaces;

public nodobin(int info){

clave = info;

enlaces=new nodobin[2];

enlaces[0]=null;

enlaces[1]=null;

}

}

protected int numElem;

private nodobin raiz;

public arbolbin(){

raiz=null;

numElem=0;

}

public int tamano(){

return numElem;

}

public boolean esta(nodobin raiz,int valor){

if(raiz==null){

return false;

}

else{

if(raiz.clave==valor){

return true;

}

else{

if(valor<raiz.clave){

return esta(raiz.enlaces[0],valor);

}

else return esta(raiz.enlaces[1],valor);

}

}

}

public void insertar(int n){

nodobin aux;

boolean flag=true;

if(raiz!=null){

if(!esta(n)){

aux=raiz;

while(flag){

if(n<aux.clave){

if(aux.enlaces[0]!=null){

aux=aux.enlaces[0];

}

else{

aux.enlaces[0]=new nodobin(n);

numElem++;

flag=false;

}

}

else{

if(aux.enlaces[1]!=null){

aux=aux.enlaces[1];

}

else{

aux.enlaces[1]=new nodobin(n);

numElem++;

flag=false;

}

}

}

}

}

else{

raiz=new nodobin(n);

numElem++;

}

}

public boolean esVacio(){

return tamano()==0;

}

public boolean esta(int valor){

return esta(raiz,valor);

}

public void componentes(ArrayList<Integer> vector){

componentes(raiz, vector);

}

public void componentes(nodobin nodo, ArrayList<Integer> vector){

if(nodo!=null){

vector.add(nodo.clave);

componentes(nodo.enlaces[0], vector);

componentes(nodo.enlaces[1], vector);

}

}

public void copia(arbolbin arbol){

if (raiz==null)return;

ArrayList<Integer> vector=new ArrayList<Integer>();

componentes(vector);

for(int i=0;i<vector.size();i++){

arbol.insertar(vector.get(i));

}

}

public int maximo(){

int aux=0;

if(raiz!=null){

nodobin auxi=raiz;

while(auxi!=null){

aux=auxi.clave;

auxi=auxi.enlaces[1];

}

}

return aux;

}

public static void main(String[] args){

arbolbin prueba=new arbolbin();

arbolbin copiado=new arbolbin();

int maximo;

prueba.insertar(57);

prueba.insertar(70);

prueba.insertar(25);

prueba.insertar(96);

prueba.insertar(42);

prueba.insertar(23);

prueba.insertar(32);

prueba.insertar(86);

prueba.insertar(10);

prueba.insertar(100);

ArrayList<Integer> vector =new ArrayList<Integer>();

ArrayList<Integer> vector2 =new ArrayList<Integer>();

prueba.componentes(vector);

prueba.copia(copiado);

copiado.componentes(vector2);

maximo=prueba.maximo();

}

@Test

public void testesta(){

arbolbin arbolprueba = new arbolbin();

for(int i=0;i<15;i++){

arbolprueba.insertar(i);

}

arbolprueba.insertar(18);

arbolprueba.insertar(21);

arbolprueba.insertar(10);

arbolprueba.insertar(20);

assertEquals(arbolprueba.esta(20),true);

assertEquals(arbolprueba.tamano(),18);

}

@Test

public void testmax(){

arbolbin prueba=new arbolbin();

prueba.insertar(57);

prueba.insertar(70);

prueba.insertar(25);

prueba.insertar(96);

prueba.insertar(42);

prueba.insertar(23);

prueba.insertar(32);

prueba.insertar(86);

prueba.insertar(10);

prueba.insertar(100);

assertEquals(prueba.maximo(),100);

}

}