RESOLUCIÓN DE PROBLEMAS ALGORÍTMICOS MEDIANTE LA PROGRAMACIÓN EN LA CLASE DE MATEMÁTICA

Teresa Pérez – Luis Langón – Santiago Vigo
tperezan@gmail.com – luislangon@adinet.com.uy – svigo@adinet.com.uy
CES, Uruguay – CES, Uruguay – CES, Uruguay

Tema: VI.2 Enseñanza experimental de la matemática.
Modalidad: MC
Nivel educativo: 3 Medio (11 a 17 años)
Palabras clave: algoritmia, informática, problemas, programación.

Resumen
Durante el mes de febrero de este año, varios profesores hemos realizado una pasantía a cargo de Sylvia da Rosa en el marco del programa Acortando Distancias, finalizando con la presentación de tres posters y la publicación de un librillo en el que se recogen diferentes ejemplos de secuencias didácticas para aplicar algoritmia y programación en la resolución de problemas en la clase de matemática e informática. Una de las extensiones naturales de la pasantía consiste en divulgar la experiencia realizada y promover la algoritmia como herramienta de resolución de cierto tipo de problemas matemáticos, así como su implementación en un lenguaje de programación, en este caso python.
Proponemos realizar un mini curso para compartir con los profesores de matemática, a través de una selección de actividades específicas sobre temas matemáticos variados, la caracterización y resolución de problemas algorítmicos y su implementación en lenguaje python. La elección del programa responde a su disponibilidad en las laptops que entrega el plan Ceibal.
Pasantes: Patricia Añon, Teresita Carrión, Luis Langón, Santiago Martorell, Daniela Pagés, Teresa Pérez, Santiago Vigo y Franco Vuan.
Sylvia da Rosa: Docente grado 4, INCO Fac. Ing. UDELAR. Doctora en Informática (UDELAR-PEDECIBA) con especialidad en Didáctica de la Informática.

Introducción:

En este mini-curso pretendemos explicar (a través de ejemplos concretos) el potencial de la inclusión de actividades que utilicen la programación en la clase de matemática.

Nos proponemos que al finalizar el curso, los participantes puedan proponer en sus clases actividades que impliquen el reconocimiento de un problema algorítmico, la escritura de una solución en lenguaje natural y su correspondiente implementación en el lenguaje python.

Comenzaremos con la implementación de programas que resuelvan problemas puntuales para finalizar con la construcción y utilización de módulos en base a la definición de funciones en relación a un cierto contenido matemático.

Primera parte:

- **Problema algorítmico, el programa y el intérprete.**

Nos proponemos a través de un ejemplo concreto mostrar cómo se ejecuta un programa escrito en lenguaje python y a qué llamamos problema algorítmico. Luego observar algunos aspectos iniciales de la sintaxis de python.

Para la ejecución e interpretación de un programa en el intérprete de python, seguiremos los siguientes pasos: crear una carpeta; descargar el archivo; abrir un terminal; encontrar la carpeta creada con "cambio de directorio" ("cd"); ejecutar python; ejecutar el programa con el comando "import *archivo*" y en caso de necesitar ejecutarlo nuevamente en la misma terminal con el comando "reload (*archivo*)".

El programa propuesto es solución al problema: "Escribir el conjunto de divisores de un número dado".

Analizando la ejecución del programa anterior concluimos de que se trata un problema algorítmico a los que Harel (1987) caracteriza como:

- Una especificación de una colección válida, posiblemente infinita de conjuntos de entrada,
- Una especificación de los elementos de salida deseados en función de los de entrada.

En definitiva debemos poder establecer en forma clara y sin ambigüedades qué información requiere el programa para iniciar y qué información devuelve. Uno de los principales motivos por los que fallan los estudiantes al escribir algoritmos proviene de una mala comprensión de la especificación del problema. (Da Rosa, 2013)

En este caso la especificación de entrada es un número natural y la de salida una lista, que corresponde al conjunto de divisores del número inicial.

El algoritmo es el mecanismo de resolución del problema, que consiste en una secuencia finita de pasos que partiendo de los datos iniciales lleva a la solución.

Un programa no es otra cosa que un texto, que debe estar escrito en un lenguaje reconocible para el intérprete (en este caso python). Según Dowek (2005), la elaboración de programas tiende un puente entre el lenguaje y la acción: *"Un programa de informática tiene como primera cualidad la de pertenecer a un lenguaje, es decir, de ser un texto. Pero ese texto tiene una segunda cualidad que es la de ser ejecutable, es decir es el agente de una acción."(p. 4).* Esto valoriza el papel del texto, ya que sin el texto no hay acción.

Pasaremos luego a ver el texto escrito en lenguaje python (código fuente), damos doble clic sobre el archivo acertijo.py para poder leerlo con el editor gedit. Para el programa que ejecutamos el texto es el
siguiente:

```
a = input("escriba un numero natural mayor que 0 o 0 para finalizar: ")
while a > 0:
     d = [x for x in range(1, a+1) if a%x==0]
     print "d =", d
     a = input("escriba un numero natural mayor que 0 o 0 para finalizar: ")
print "Gracias por usar mi programa!"
```

Al analizar el texto observamos dos tipos de sentencias (instrucciones): unas que permiten la interacción con el operador (persona que usa la máquina) y otras que son los pasos del algoritmo que lleva a la solución.
Se debe cerrar el editor y no el terminal para la próxima actividad.

- **El editor gedit**

El objetivo es escribir y ejecutar un programa, con un algoritmo muy simple para lograr una primera aproximación al uso del editor y utilizar algunas instrucciones más en lenguaje python. En este caso se utilizará el editor de texto gedit, que ya se usó en la actividad anterior.
1) Para abrir gedit (aplicaciones – accesorios – Editor de texto gedit)
2) Copie el siguiente texto, respete las tabulaciones (indentar) no use tildes, no se preocupe por los colores, éstos aparecerán cuando el archivo haya sido guardado con extensión .py

```
# entrada un numero
# salida  lista de multiplos hasta 1000
a = input ("ingrese numero: ")
m = 0
l =[]
while m < 1000:
     l = l + [m]
     m = a + m
print l
```

3) Grabe el archivo como multiplos.py seleccionando la misma carpeta que antes.
4) Como ya tenemos abierta una ventana de terminal donde se está ejecutando python en la carpeta en la que grabamos el archivo no es necesario repetir los pasos de la actividad anterior. Simplemente... import multiplos

Al ejecutar un programa, si surgen problemas, pueden deberse a errores en la escritura del programa, o sea errores en la sintaxis del texto, o a errores en el diseño del mismo, por lo que se hace necesario revisar la etapa de escritura del algoritmo en lenguaje natural o inclusive revisar la especificación del problema. Esto constituye un espacio privilegiado para desarrollar la autonomía y el rigor en los estudiantes. Como sostiene Dowek (2005) *"El rigor no aparece entonces como algo impuesto del exterior sino simplemente como una condición de buena comunicación entre el estudiante y la máquina." (p. 4)*

- **Especificar un problema, diseñar un algoritmo, implementarlo en el intérprete.**

El objetivo es seguir los pasos de resolución de un problema algorítmico y experimentar su implementación en el intérprete.

Para cada uno de los siguientes problemas los asistentes piensan un algoritmo, escriben el algoritmo en papel, dan las instrucciones en español, lo implementan en python y revisan el trabajo realizado.

Las actividades están pensadas para fortalecer el concepto de algoritmo, entrada - proceso – salida y experimentar con la aplicación de los siguientes comandos: el condicional, (if else), la iteración (while), además de los ya utilizados input y print.

Algunos problemas a resolver:

1. Dado un número decidir si es par o no.
2. Dados dos números indicar cuál es mayor.
3. Dado un número escribir los pares menores que él.
4. Dados tres números devolverlos en orden.
5. Dado un número a, devolver a!
6. Dado un número natural decidir si es primo.
7. Algoritmo para hallar el número verificador de la cédula de identidad. Veremos con un ejemplo cómo es este método[1].

Se necesita una cédula, por ejemplo 3.416.652-_ y el número 2987634, que es una constante dada (a la cual se le quita el primer 2 en caso de que la cédula tenga 6 dígitos).

Primer paso: Se multiplica cada dígito de la cédula por cada dígito del número dado y se anota la última cifra de cada resultado.

	3	4	1	6	6	5	2
x	2	9	8	7	6	3	4
Resultados:	6	6	8	2	6	5	8

Segundo paso: Se suman todos los números que se anotaron en el paso anterior y se anota la última cifra de esa suma.

6 + 6 + 8 + 2 + 6 + 5 + 8 = 41 última cifra: 1

Tercer paso:

Si es cero, entonces el dígito verificador es 0.

Si no es cero debe calcularse la diferencia con 10.

10 – 1 = 9 el dígito verificador es 9.

Segunda parte:

Las funciones, una necesidad para ordenar el proceso de programación.

Vamos a ver ahora como aplicar los conceptos vistos aplicados a otra rama de la matemática, pasamos entonces a la Geometría Analítica, este es un tema con mucho contenido algorítmico.

Supongamos que queremos resolver el problema de calcular el perímetro de un cuadrilátero a partir de las coordenadas de sus vértices.

Al pensar la solución, surge la necesidad de resolver un problema más sencillo: calcular la distancia entre dos puntos conocidas sus coordenadas.

Crearemos entonces una función que dados dos puntos devuelva la distancia entre ellos, para luego utilizarla en el programa que calcule el perímetro del cuadrilátero.

Función distancia:

Especificación de la entrada y salida

Entrada: coordenadas de los puntos (dos pares ordenados)

Salida: la distancia entre los puntos (un número)

Algoritmo: calculo la raíz cuadrada de la suma de los cuadrados de las diferencias coordenada a coordenada.

1) Abrir el editor gedit. Copiar el siguiente texto. Guardarlo en un archivo llamado *geo.py*

(Observación: Las oraciones precedidas de # son comentarios que no afectan el texto del programa)

```
# formulas de geometria analitica
import math

# distancia entre dos puntos
def distancia ((a,b),(c,d)):
    return math.sqrt((a-c)**2 + (b-d)**2)
```

Este archivo el lo que llamaremos un módulo de funciones, en él escribiremos las funciones que se utilizan en geometría analítica, comenzamos con ésta, luego agregaremos más. Este módulo no es un programa sino una colección de funciones que serán utilizadas por otros programas.

2) Abrir otra ventana en el editor gedit y copiar el siguiente texto:

```
# programa para probar la funcion anterior
import geo
(a,b) = input ("escriba las coordenadas del punto A ")
(c,d) = input ("escriba las coordenadas del punto B ")
print "d (A,B) = ",geo.distancia ((a,b),(c,d))
```

Este texto es un programa para ver el funcionamiento de la función distancia.

import geo (De esta manera se informa que utilizaremos las funciones del módulo *geo.py*)

geo.distancia ((a,b),(c,d)) (llama a la función distancia que se encuentra dentro del módulo *geo.py*)

Observemos que el módulo *geo.py* tiene la sentencia *import math*, para poder utilizar las funciones del módulo *math.py*, que es un módulo que tiene funciones matemáticas ya definidas, en este caso se utiliza raíz cuadrada *sqrt*.

3) Abrir una ventana de terminal, ir a la carpeta donde guardamos los archivos, ejecutar python y escribir *import geometria* para probar nuestra función distancia.

La función distancia no solo puede usarse directamente sino que también, como habíamos pensado originalmente, para definir otra función como perímetro de un cuadrilátero:

```
def perimetrocuadrilatero(p1,p2,p3,p4):
        return distancia(p1,p2)+distancia(p2,p3)+distancia(p3,p4)+distancia(p4,p1)
```

Otros problemas:

- Dados una recta y un punto determinar si el punto pertenece a la recta
- Dados dos puntos y una recta en la que no le pertenecen, determinar si están en igual o distinto semiplano con borde en la recta.
- Dados tres puntos, determinar si están alineados o no.
- Dados tres puntos no alineados calcular el perímetro del triángulo que determinan.
- Dados n puntos en determinado orden, hallar el perímetro del polígono que determinan.

Conclusiones:

Brindar a los estudiantes herramientas que les permitan resolver problemas algorítmicos a través de la programación, creemos que genera un ámbito privilegiado para desarrollar las competencias de resolución de problemas, comunicación y representaciones que proponen los estándares del NTCM. (NCTM, 2003).

Los comandos disponibles en los lenguajes de programación están relacionados directamente a conceptos matemáticos y lógicos, esto implica utilizar, en muchos casos, procedimientos diferentes a los que utilizaría al resolver el problema con otros recursos, favoreciendo analizar los diferentes caminos disponibles y qué contenidos matemáticos entran en juego en cada caso.

Creemos, como sostiene Artigue (2004), que el uso de la programación no debe convertirse en un objeto de enseñanza en sí mismo, sino en una herramienta que permita mejores aprendizajes matemáticos, favoreciendo además la motivación de los estudiantes.

Finalmente, creemos que el pensamiento computacional, favorece fuertemente la posibilidad de convertir un problema complejo en uno más sencillo y abarcable, desarrollar diferentes niveles de abstracción, cuestionarse acerca de qué problemas los resuelve en forma más sencilla una máquina y cuáles no, entre otras cuestiones, por lo que su promoción no debe imitarse a aquellos con una inclinación hacia las ciencias sino con todos los estudiantes. (Wing, 2006)

Glosario de comandos utilizados:

Los comandos o sentencias son órdenes a la computadora para que ésta las realice.

- a = input (“ingrese un numero: “)

Muestra la leyenda *ingrese un numero* y espera que se escriba con el teclado hasta que se pulse enter, luego guarda la información ingresada en una variable a la que llamaremos *a*. Es importante tener en cuenta que no se pueden usar tildes ni ñ.

- print (“ese numero es par”)

Escribe en la pantalla el mensaje *ese numero es par*

- m = m + a

Suma el contenido de las variables a y m y guarda el resultado en la variable m, el signo de igual significa asignación, no debe confundirse con una identidad.

- Condicional:

if a > 0:	Si el contenido de la variable a es mayor

print (“ese numero es positivo”) else: print (a, “no es positivo”)	que cero entonces Escribe: *ese numero es positivo* sino Escribe: el valor de la variable *no es positivo.*

- Iteración

while i < 100: i = i + 1 print (“pude contar hasta 100”)	mientras la variable i sea menor que 100 incrementar en 1 la variable i luego escribe: *pude contar hasta 100*

- Definir una función:

def cuadrado(a): return a*a	define una función que se llama cuadrado a la cual entra un parámetro (variable) que le llama *a* devuelve el resultado de hacer *a* por *a*.

Referencias bibliográficas

Artigue, M (2004), Problemas y desafíos en educación matemática: ¿Qué nos ofrece hoy la didáctica de la matemática para afrontarlos? *Educación Matemática*, Diciembre, año/vol. 16, número 003. Santillana. Distrito Federal, México pp. 5-28.

Da Rosa, S (2013) coord. Matemática y programación. Recuperado de: http://www.ineed.edu.uy/sites/default/files/matyprogversionfinal.pdf

Dowek, G. (2005) Quelle informatique enseigner au lycee? Recuperado de: https://who.rocq.inria.fr/Gilles.Dowek/lycee.html

Fernández Gallardo, P. y Gil, O. 2007. Una introducción a los códigos detectores de errores y correctores de errores. Recuperado de: http://www.fing.edu.uy/~omargil/educmate/codyal31-71.pdf

Harel, D. (1987) *Algorithmics: The Spirit of Computing*, Addison-Wesley, Reading, MA, (425 pp.).

NCTM. (2003) *Principios y estándares para la Educación Matemática*. Traducción de Manuel Fernandez SAEM Thales Sevilla ISBN 84-933040-3-4

Wing, J. (2006) Computational Thinking, *Communications of the ACM*, Vol. 49, Nr. 3.

Anexo:

Una solución en español de algunos problemas:

- Dados una recta y un punto determinar si el punto pertenece a la recta

Entrada: coeficientes de la ecuación de la recta (en forma general), coordenadas del punto

Salida: Verdadero o Falso

Algoritmo:

1. Evalúo la ecuación de la recta con las coordenadas del punto y la comparo con 0 devuelvo el resultado de esa comparación

- Dados dos puntos y una recta a la que no pertenecen determinar si están en igual o distinto semiplano con borde en la recta.

Entrada: coordenadas de los 2 puntos; coeficientes de una ecuación general de la recta.

Salida: un mensaje de si están o no en el mismo semiplano.

Algoritmo:

Sustituyo en la ecuación de la recta las coordenadas de cada punto.

Multiplico los resultados.

Evalúo si este número es mayor o menor a 0 (no puede ser 0 porque los puntos no pertenecen a la recta).

Si es menor a 0 los puntos están en semiplanos opuestos. Si es positivo están en el mismo semiplano.

- Dados tres puntos, determinar si están alineados o no.

Entrada: las coordenadas de los 3 puntos

Salida: Están o no alineados

Algoritmo:

Se calcula el cociente incremental de AB y de AC (El cociente incremental es $\frac{y_2-y_1}{x_2-x_1}$)

Si los números obtenidos son iguales: "están alineados"; si son distintos: "no están alineados".

- Dados tres puntos no alineados determinar el perímetro del triángulo que determinan.

Entrada: Una lista con las coordenadas de los vértices del triángulo (tres pares ordenados).

Salida El perímetro del triángulo (un número)

Algoritmo:

Calcular la distancia entre el primer y el segundo punto

Sumarle la distancia entre el segundo y el tercer punto

Sumarle la distancia entre el tercer y el primer punto

- Dados n puntos en determinado orden, calcular el perímetro del polígono que determinan.

Entrada: Una lista con las coordenadas de los puntos considerados en forma concíclica (una lista de pares ordenados)

Salida: El perímetro del polígono (un número)

Algoritmo:

Al no saber la cantidad de puntos una extensión directa del algoritmo anterior no es posible. Sin embargo la idea que se empleará será parecida. Aplicaremos un algoritmo recursivo. Para simplificar el razonamiento aplicaremos un algoritmo para sumar las distancias de todos los segmentos desde el determinado por el primer punto con el segundo, hasta el del penúltimo con el último, a este número le llamaremos perímetro *poligonal* y a ese resultado le sumaremos la distancia desde el primer al último punto.

La función poligonal toma una lista con las coordenadas de los puntos, y devuelve su perímetro.

El perímetro de un punto es 0 y el perímetro de una lista mayor de puntos es la distancia entre el primero y el segundo de la lista + el perímetro de la lista a la que se le saca el primer punto.

Esquemáticamente

perpol= 0 si la lista tiene un solo elemento

perpol=dist(A,B)+ perpol(lista menos A) donde A y B son el primer y el segundo punto de la lista.

Entonces perímetro de un polígono:

distancia(A,X)+perpol(A) donde A y X son el primer y el último punto de la lista.

Para calcular el perímetro de un polígono lo hacemos con dos funciones, una perpoligonal que calcula la suma de las medidas de los segmentos desde el primero hasta el último y perímetro que a ese resultado le suma la medida del segmento determinado por el primer y el último punto.

Algoritmos en python

```
def pertenece(r,p):
        return (r[0]*p[0]+r[1]*p[1]+r[2])==0

def mismosemiplano(r,p1,p2):
        if (r[0]*p1[0]+r[1]*p1[1]+r[2])*(r[0]*p2[0]+r[1]*p2[1]+r[2])>0:
                resultado="Estan en el mismo semiplano"
        else:
                resultado="Estan en semiplanos distintos"
        return resultado

def alineados(p1,p2,p3):
        return (p1[1]-p2[1])/(p1[0]-p2[0])==(p3[1]-p2[1])/(p3[0]-p2[0])
```

```
# la poligonal es una linea formada por segmentos de recta consecutivos no alineados.
#calculamos recursivamente su perimetro que no es otra cosa que la suma de las medidas de sus lados

def perpoligonal(A):
        if len(A)==1
                return 0
        else
                return distancia(A[0],A[1])+perpoligonal(A[1:])

# Calculamos el perimetro como la suma del segmento del primer punto hasta el ultimo mas el perimetro
# de la poligonal desde el primero hasta el ultimo
def perimetro(A):
        return distancia(A[0],A[-1])+perpoligonal(A)
```