# Problemas de programacion de computadores

Sebastian Rodriguez Ardila
email: sebrodriguezar@unal.edu.co

Diciembre 2020

## 1 La Granja

1. Litros de leche que se producen en una granja.

Objetos Conocidos: largo y ancho del corral, (n y m $\in$ N), metros cuadrado que necesita una vaca para producir leche (m $\in$ N) y los litros de leche que produce una vaca (x $\in$ N).

Objetos Desconocidos: cantidad de litros de leche que se producen en la granja, (resul $\in$ N)

Función:

$$resul : N \times N \to N$$

$$(n, x) \to resul = n * x$$

Ubicación del código del programa: Linea 3 - 10, del archivo: Ejercicios Programacion.py .

2. Cantidad de huevos de gallina al mes en una granja.

Constantes: Dias del mes (diasMes = 30), huevos que pone cada 3 días, una gallina de la primera mitad de las gallinas (1) y huevos que pone cada 5 días, una gallina de la segunda mitad de las gallinas (1).

Objetos Conocidos: cantidad de aves (A $\in$ N) y cantidad de gallinas ($\frac{1}{3}A \in$ N).

Objetos Desconocidos: cantidad huevos de gallina al mes en la granja (h $\in$ N).

Función:

$$h : N \times N \to N$$

$$(A, diasMes) \rightarrow h = \frac{\frac{1}{3}A}{2} \times \frac{diasMes}{3} + \frac{\frac{1}{3}A}{2} \times \frac{diasMes}{5}$$

Ubicación del código del programa: Linea 13 - 19, del archivo: Ejercicios Programacion.py .

3. Cantidad en kilos de escorpiones que se pueden vender sin que decrezca la población a menos de 2/3

Constantes: peso en gramos de cada tamaño de escorpión (grande = 50g, mediano = 30g y pequeño = 20g) y cantidad de población que se puede vender (1/3), ∈ N.

Objetos Conocidos: número de escorpiones pequeños, (p ∈ N),número de escorpiones medianos, (m ∈ N) y número de escorpiones grandes, (g ∈ N).

Objetos Desconocidos: Cantidad en kilos de escorpiones que se pueden vender (totalK/1000 ∈ N), donde totalk esta en unidades gramos.

Función:

$$totalK : N \times N \times N \rightarrow N$$

Nota: La venta iniciara por vender todos los grandes, luego los medianos y por ultimo los pequeños. No se parara la venta de escorpiones, mientras no se alla vendido exactamente un 1/3 de la población total, tercioP = $\frac{m+p+g}{3}$, es decir hasta que tercioP = 0.

$$(p, m, g) = \begin{cases} tercioP - 1,\ g - 1,\ totalk + 50 & \text{si } g > 0 \\ tercioP - 1,\ m - 1,\ totalk + 30 & \text{si } g = 0\ y\ m > 0 \\ tercioP - 1,\ p - 1,\ totalk + 20 & \text{si } g = 0\ y\ m = 0\ y\ p > 0 \end{cases} \quad | \ tercioP > 0 \tag{1}$$

Ubicación del código del programa: Linea 24 - 42, del archivo: Ejercicios Programacion.py .

4. Material para cerramiento del corral más económico(con tablas, varilla o alambre).

Constantes: Hileras de tablas de madera para un lado del corral (4), hileras de varilla para un lado del corral (8), hileras de alambre para un lado del corral (5) y ecuaciones para calcular el total de valor por cada material.

Objetos Conocidos: largo y ancho del corral, (n y m ∈ N), precio por metro de tabla, (p ∈ N), precio por metro de varilla, (q ∈ N) y precio por metro de alambre, (s ∈ N).

Objetos Desconocidos: material para lograr el cerramiento mas economico ,

(resul ∈ ASCII).

Ecuaciones:

$$totalM(n,m,p) = n*p*4*2 + m*p*4*2$$
$$totalV(n,m,q) = n*q*8*2 + m*q*8*2$$
$$totalA(n,m,s) = n*s*5*2 + m*s*5*2$$

Función:

$$cerramientoEco : N \times N \times N \times N \times N \rightarrow ASCII^*$$
$$(n,m,p,q,s) \rightarrow resul = \begin{cases} madera & \text{si } totalM < totalV \ y \ totalM < totalA \\ varilla & \text{si } totalV < totalM \ y \ totalV < totalA \\ alambre & \text{si } totalA < totalM \ y \ totalA < totalV \end{cases} \quad (2)$$

Ubicación del código del programa: Linea 51 - 70, del archivo: Ejercicios Programacion.py .

# 2 Numericos

5. Función potencia de un entero elevado a un entero.

Objetos Conocidos: base de la función potencia (base ∈ Z) y exponente de la función potencia (exp ∈ Z).

Objetos Desconocidos: resultado de la función potencia (resul ∈ Z).

Función:

$$potencia : Z \times Z... \rightarrow Z$$
$$(base, exp) \rightarrow resul = base^{exp}$$

Ubicación del código del programa: Linea 76 - 83 , del archivo: Ejercicios Programacion.py .

6. Función que determina si un número es divisible por otro.

Objetos Conocidos: dividendo (divid ∈ Z) y el divisor (divis ∈ Z) con divis mayor que 0.

Objetos Desconocidos: cadena de texto que indica si un número es divisible por otro (resul ∈ B).

Función:

$$divisible : Z \times Z \rightarrow B$$

$$(divid, divis) \rightarrow resul = \begin{cases} True & \text{si } divid \ \% \ divis = 0 \\ False & \text{si } divid \ \% \ divis \neq 0 \end{cases} \quad (3)$$

siendo % la operación mod (módulo).

Ubicación del código del programa: Linea 86 - 95 , del archivo: Ejercicios Programacion.py .

7. Determinar si un número es primo.

Objetos Conocidos: número al que se evaluara si es primo o no (num ∈ N).

Objetos Desconocidos: cadena de texto que indica si el número es primo o no (resul ∈ B).

Función:

$$primo : N \rightarrow B$$

$$(num) \rightarrow resul = \begin{cases} False & \text{si } num = 1 \ o \ num \ \% \ \forall_{n=2}^{num-1} n = 0 \\ True & \text{si } num = 2 \ o \ \ \% \ forall_{n=2}^{num-1} n \neq 0 \end{cases} \quad (4)$$

siendo % la operación módulo.

Ubicación del código del programa: Linea 98 - 115, del archivo: Ejercicios Programacion.py .

8. Dados dos números naturales, determinar si son primos relativos.

Objetos Conocidos: dos números (num1 y num2 ∈ N)

Objetos Desconocidos: cadena de texto que indica si los números son primos relativos (resul ∈ B).

Función:

$$primoRelativo : N \times N \rightarrow B$$

$$\forall_{i=0}^{num-1} n_i <= numero1 \ o \ \forall_{i=0}^{num-1} n_i <= numero2$$

$$(num1, num2) \rightarrow resul =$$

$$\begin{cases} False & \text{si } num1\% \ \forall_{n=0}^{num-1} n = 0 \ o \ num2\% \ \forall_{n=0}^{num-1} n = 0 \\ True, & \text{si } en \ otro \ caso \end{cases} \tag{5}$$

siendo % la operación módulo.

Ubicación del código del programa: Linea 120 - 135, del archivo: Ejercicios Programacion.py .

9. Determinar si un número es múltiplo de la suma de otros dos números.

Objetos Conocidos: primer número a sumar (num1 $\in$ R), segundo número a sumar (num2 $\in$ R) y un número al que se evaluara si es múltiplo de la suma de los dos numeros anteriores (numMul $\in$ R).

Objetos Desconocidos: cadena de texto que indica si el número (numMul) es múltiplo de la suma de los dos numeros (num1 y num2) (resul $\in$ B).

Función:

$$multiplo : R \times R \times R \to B$$

$$(num1num2, numMul) \to resul =$$

$$\begin{cases} True & \text{si } numMul \ \% \ (num1 + num2) = 0 \\ False & \text{si } numMul \ \% \ (num1 + num2) \neq 0 \end{cases} \tag{6}$$

siendo % la operación módulo. Si la operación da como resultado cero (0), el número si es multiplo de la suma de los números.

Ubicación del código del programa: Linea 138 - 149, del archivo: Ejercicios Programacion.py .

10. Dados los coeficientes de un polinomio de grado dos, evaluar el polinomio en un valor dado.

Objetos Conocidos: los tres coeficientes de un polinomio de segundo grado (a2, a1, a0 $\in$ R, con a2 $\neq$ 0) y el valor dado para evaluar el polinomio (k $\in$ R).

Objetos Desconocidos: Valor del resultado (resul $\in$ R).

Función:

$$polGrado2 : R \times R \times R \times R \to R$$

$$(a2, a1, a0, k) \to resul = a2 * k^2 + a1 * k + a0$$

Ubicación del código del programa: Linea 152 - 161, del archivo: Ejercicios Programacion.py .

11. Dados los coeficientes de un polinomio de grado dos, calcular coeficiente lineal de la derivada.

Objetos Conocidos: los tres coeficientes de un polinomio de segundo grado (a2, a1, a0 $\in$ R, con con a2 $\neq$ 0).

Objetos Desconocidos: Valor del coeficiente lineal de la derivada (coeflin $\in$ R).

Función:

$$coeflin : R \times R \times R \times R \to R$$

$$(a2, a1, a0)' = coeflin = (a2 * x^2 + a1 * x + a0)' = (2 * a2) * x + a1$$

Donde 2*a2 es el coeficiente lineal de la derivada.

Ubicación del código del programa: Linea 164 - 172, del archivo: Ejercicios Programacion.py .

12. Dados los coeficientes de un polinomio de grado dos y un número real, evaluar la derivada del polinomio en ese número.

Objetos Conocidos: los tres coeficientes de un polinomio de segundo grado (a2, a1, a0 $\in$ R, con a2 $\neq$ 0) y el valor dado para evaluar la derivada del polinomio (k $\in$ R).

Objetos Desconocidos: Valor del resultado (resul $\in$ R).

Función:

$$deriPolGrado2 : R \times R \times R \times R \to R$$

$$(a2, a1, a0, k)' \to resul = (2 * a2) * k + a1$$

Ubicación del código del programa: Linea 175 - 184, del archivo: Ejercicios Programacion.py .

13. Dado un natural, determinar si es un número de Fibonacci o no.

Objetos Conocidos: número ingresado (num $\in$ N) y la función para crear la serie de Fibonacci hasta num (fibonacci)

Objetos Desconocidos: cadena de texto que indica si el número pertenece a la serie de Fibonnaci (result ∈ B)

Función serie de Fibonnaci:

$$fibonacci : N \to N$$
$$\forall_{n=0}^{num-1} fibonacci(n) = \begin{cases} fibonacci(n-1) + fibonacci(n-2), & \text{si } n \geq 2 \\ n, & \text{si } n < 2 \end{cases} \tag{7}$$

Función:

$$numFibonacci : R \to B$$
$$(num) \to resul = \begin{cases} True, & \text{si } \forall_{i=0}^{num-1} \ni fibonnaci_{i+1} = num \\ False, & \text{si } en\ otro\ caso \end{cases} \tag{8}$$

.

Ubicación del código del programa: Linea 187 - 211, del archivo: Ejercicios Programacion.py .

# 3 Geométricos

14. Dadas la pendiente y el punto de corte de dos rectas, determinar si son paralelas, perpendiculares o ninguna de las anteriores.

Objetos Conocidos: pendiente recta uno (m1 ∈ R), pendiente recta dos (m2 ∈ R), coordenada $y$ del punto corte de la recta uno respecto al el eje $y$, (pcUno ∈ R) y coordenada $y$ del punto corte de la recta dos con respecto al el eje $y$, (pcDos ∈ R).

Objetos Desconocidos: cadena de texto que indica si las rectas son paralelas, perpendiculares o ninguna de las anteriores (result ∈ ASCII).

Función:

$$relacionRectas : R \times R \times R \times R \to ASCII^*$$
$$(m1, m2, pcUno, pcDos) \to resul = \begin{cases} son\ paralelas, & \text{si } m1 = m2\ y\ pcUno \neq pcDos \\ son\ perpendiculares, & \text{si } m1 * m2 = -1 \\ ninguna\ de\ las\ dos, & en\ otro\ caso \end{cases} \tag{9}$$

Ubicación del código del programa: Linea 218 - 234, del archivo: Ejercicios Programacion.py.

15. Dadas la pendiente y el punto de corte de dos rectas, determinar los puntos de intersección al origen (ordenada y abscisa).

Objetos Conocidos: pendiente recta uno (m1 $\in$ R), pendiente recta dos (m2 $\in$ R), coordenada $y$ del punto corte de la recta uno respecto al el eje $y$, (pcUno $\in$ R) y coordenada $y$ del punto corte de la recta dos con respecto al el eje $y$, (pcDos $\in$ R).

Objetos Desconocidos: coordenada $y$ de la ordenada de una recta, (yOrdenada $\in$ R) y coordenada $x$ de la abscisa de una recta, (xAbscisa $\in$ R).

La siguiente función es valida para determinar los puntos de intersección al origen, tanto de la recta uno. Función:

$$interseccionOrigen : R \times R \to R$$

$$(m, pCorte) = (0,\ pCorte)\ y\ (\frac{-pCorte}{m},\ 0)$$

donde yOrdenada = pCorte.
Ubicación del código del programa: Linea 237 - 246, del archivo: Ejercicios Programacion.py.

16. Dado el radio de un círculo, calcular el área del triángulo que circunscribe el circulo (triángulo afuera).
Nota: Se conocen pocos valores para dar una respuesta numérica, en este caso se pedirá el valor del perímetro del triángulo que circunscribe el circulo.

Objetos Conocidos: radio del círculo, (radio $\in$ R), valor del perímetro del triángulo que circunscribe el circulo, (perim $\in$ R) y función del semi-perímetro del triangulo que circunscribe la circunferencia, (s $\in$ R)

Objetos Desconocidos: el área del triángulo que circunscribe la circunferencia, (result $\in$ R)

Función del semi-perímetro del triangulo que circunscribe la circunferencia:

$$s(a, b, c) = \frac{1}{2} * (a + b + c) = \frac{1}{2} * (perim)$$

Función:

$$areaTri : R \times R \to R$$

$$(radio, perim) \to result = \frac{1}{2} * (perim) * radio$$

Donde a,b y c son respectivamente los lados de un triangulo.
Ubicación del código del programa: Linea 249 - 256, del archivo: Ejercicios Programación.py.

17. Dado el radio de un círculo, calcular el éarea y perímetro del cuadrado, pentágono y hexágono adentro (inscrito en un círculo) y afuera (inscribiendo al círculo).

Objetos Conocidos: radio del círculo, (radio ∈ R), nombre de la figura a calcular su área y perímetro inscritas y circunscritas en la circunferencia (figura ∈ ASCII), y ecuaciones del área y perímetro de las figuras inscritas y circunscritas en la circunferencia.

Objetos Desconocidos: área y perímetro de las figuras inscritas y circunscritas en la circunferencia. (areaYPerimInscritos y areaYPerimInscribiendo ∈ R)

Figuras Inscritas Función:

$$areaYPerimInscritos : R \times ASCII^* \to R$$

$$(radio, figura) = \begin{cases} a = \frac{2*radio}{\sqrt{2}} \quad p = 4 * \frac{2*radio}{\sqrt{2}}, & \text{si } cuadrado \\ a = \frac{5*radio^2*\sqrt{10+2*\sqrt{5}}}{8} \quad p = \frac{5*radio*\sqrt{10+2*\sqrt{5}}}{2}, & \text{si } pentagono \\ a = 6 * radio \quad perimetro = \frac{3*radio^2*\sqrt{3}}{2}, & \text{si } hexagono \end{cases} \tag{10}$$

Figuras Inscribiendo Función:

$$areaYPerimInscribiendo : R \times ASCII^* \to R$$

$$(radio, figura) = \begin{cases} a = (2 * radio)^2 \quad p = (2 * radio) * 4, & \text{si } cuadrado \\ a = 1.72048 * (\frac{2*radio}{\sqrt{1+\frac{2}{\sqrt{5}}}})^2 \quad p = 5 * \frac{2*radio}{\sqrt{1+\frac{2}{\sqrt{5}}}}, & \text{si } pentagono \\ a = \frac{6*2*radio*\sqrt{3}}{3} \quad perimetro = 2 * radio^2 * \sqrt{3}, & \text{si } hexagono \end{cases} \tag{11}$$

Ubicación del código del programa: Linea 260 - 296, del archivo: Ejercicios Programación.py.

18. Cantidad de telaraña que requiere la araña para su telaraña de $\pi * r^2$.

Objetos Conocidos: radio de la telaraña de $\pi * r^2$, (r ∈ N) y ecuaciones para el

perimetro de un hexagono inscrito en una circunferencia.

Objetos Desconocidos: cantidad de telaraña que necesita la araña para su telaraña de $\pi * r^2$, (cantidadTelarana $\in$ N).

Ecuación perímetro hexágono inscrito en circunferencia:

$$perimetro : 6L = 6r$$

donde L es un lado del hexágono, que en este caso al estar inscrito en una circunferencia de radio r, L = r.

Función:

$$cantidadTelarana : N \to N$$

$$(r) = cantidadTelarana = \begin{cases} 0, & \text{si } r = 0 \\ 6, & \text{si } r = 1 \\ (6 * r) + cantidadTelarana(r-1), & \text{si } en\ otro\ caso \end{cases} \quad (12)$$

Ubicación del código del programa: Linea 299 - 311 , del archivo: Ejercicios Programación.py.

# 4 Varios

19. Si en la UN están podando éarboles y cada rama tiene P hojas, y a cada arbol le quitaron K ramas, cuántos arboles se deben podar para obtener T hojas?.

Objetos Conocidos: número de hojas a podar (t $\in$ N), número de hojas por rama (P $\in$ N) y número de ramas podadas en un árbol (K $\in$ N)

Objetos Desconocidos: número de arboles a podar (resul $\in$ N)

Función:

$$podarArboles : N \times N \times N \to N$$

$$(t, p, k) \to resul = \frac{\frac{t}{p}}{k}$$

Ubicación del código del programa: Linea 316 - 324, del archivo: Ejercicios

Programación.py.

20. Si un amigo, no tan amigo, me presta K pesos a i pesos de interés diario, ¿cuánto le pagaré en una semana si el interés es simple?, ¿y cuánto si el interés es compuesto?.

Objetos Conocidos: cantidad de dinero prestada (k $\in$ N), cantidad a pagar de dinero prestado (1 semana = 7 dias = n $\in$ N), interés diario (i $\in$ N) y las formulas para calcular el interés simple e interés compuesto.

Objetos Desconocidos: dinero a pagar con interés simple (pagoIntSim $\in$ R) y el dinero a pagar con interés compuesto (pagoIntComp $\in$ ASCII).

Función:

$$pagoIntSim : N \times N \times N \to R$$

$$pagoIntComp : N \times N \times N \to R$$

$$(k, i) = pagoIntSim = k(1 + (i/100) * (n/360))$$

$$(k, i) = pagoIntComp = k(1 + (i/100))^n$$

Nota: n se divide en 360 para convertir el periodo de tiempo de años a días, solamente en el interés simple, ya que en el interés compuesto, n es un número de veces al año e i se divide entre 100 para convertir el interés diario, de porcentaje a decimal.
Ubicación del código del programa: Linea 327 - 340, del archivo: Ejercicios Programación.py.

21. Número de formas distintas de ubicar fichas rojas(1x1) y azules(1x2) de lego sobre base de 1xn y luego incluyendo también fichas amarillas(1x3).

Objetos Conocidos: tamaño n de la base de 1xn, (n $\in$ N).

Objetos Desconocidos: número de formas distintas de ubicar sobre una base de 1xn las fichas de lego azules y rojas, posteriormente fichas azules, rojas y amarillas, (numUbicaciones $\in$ N).

Función:

$$numUbicaciones : N \to N$$

$$(n) = numUbicaciones = \begin{cases} 0, & \text{si } n = 0 \\ 1, & \text{si } n = 1 \\ numUbicaciones(n-1) + numUbicaciones(n-2), & \text{si } en\ otro\ caso \end{cases} \tag{13}$$

Para el caso en que se incluyan fichas amarillas: Función:

$$numUbicaciones : N \to N$$

$$(n) = \begin{cases} 0, & \text{si } n = 0 \\ 1, & \text{si } n = 1 \\ 2, & \text{si } n = 2 \\ numUbicaciones(n-1) + numUbicaciones(n-2) + numUbicaciones(n-3), & \text{si } en\ otro\ caso \end{cases} \tag{14}$$

Ubicación del código del programa: Linea 343 - 371, del archivo: Ejercicios Programación.py.

# 5 Arreglos

22. Implementar la criba de Eratostenes para calcular los números primos en el rango 2 a n, donde n es un número natural dado por el usuario.

Objetos Conocidos: número dado por el usuario, (n ∈ N).

Objetos Desconocidos: arreglo de booleanos, donde el elemento false hace parte de la criba y true no hace parte. (criba ∈ B)

Función:

$$criba : N \to B^n$$

$$listaCriba_i = listaCriba_0 = listaCriba[1] = False, \forall_{i=2}^{n} listaCriba_i = True$$

$$(n) = \begin{cases} \forall_{x=2}^{n+1} \forall_{y=2}^{n+1} listaCriba_y = False, & \text{si } x^2 \ < \ n \ , y \neq x \ y \ \ y\%x = 0 \\ listaCriba, & \text{si } en\ otro\ caso \end{cases} \tag{15}$$

Siendo % la operación módulo.

Ubicación del código del programa: Linea 376 - 391, del archivo: Ejercicios Programación.py.

23. Desarrollar un algoritmo que calcule la suma de los elementos de un arreglo de números reales/naturales.

Objetos Conocidos: arreglo de números reales/naturales (arreglo ∈ N o R).

Objetos Desconocidos: suma total de los elementos del arreglo (sumTotal ∈ N o R)

Función:

$$sumTotal : R^n \to R$$

Si son naturales

$$sumTotal : N^n \to N$$

$$(arreglo) \to sumTotal = \sum_{i=0}^{n-1} arreglo_i = arreglo_0 + arreglo_1 + ... + arreglo_{n-1}$$

donde n es el largo del arreglo

Ubicación del código del programa: Linea 395 - 405, del archivo: Ejercicios Programación.py.

24. Desarrollar un algoritmo que calcule el promedio de un arreglo de números reales/naturales.

Objetos Conocidos: arreglo de números reales/naturales (arreglo ∈ N o R).

Objetos Desconocidos: promedio de los elementos del arreglo (promedio ∈ N o R)

Función:

$$promedio : R^n \to R$$

Si son naturales

$$promedio : N \to N$$

$$(arreglo) = \frac{\sum_{i=0}^{n-1} arreglo_i}{i} = \frac{arreglo_0 + ... + arreglo_{n-1}}{i}$$

donde n es el largo del arreglo.

Ubicación del código del programa: Linea 409 - 428, del archivo: Ejercicios Programación.py.

25. Desarrollar un algoritmo que calcule el producto punto de dos arreglos de números reales/naturales de igual tamaño.

Objetos Conocidos: primer arreglo de números reales/naturales (v ∈ N o R) y el segundo arreglo de números (w ∈ ASCII R).

Objetos Desconocidos: resultado del producto punto entre el arreglo v y el arreglo w (result ∈ N o R).

Función:

$$result : R^n \times R^n \to R$$

Si son naturales

$$result : N^n \times N^n \to N$$

$$(v, w) = result = \sum_{i=0}^{n-1} v_i * w_i = v_0 * w_0 + v_1 * w_1 + ... + v_{n-1} * w_{n-1}$$

donde n es el largo tanto de arreglo v como del arreglo w.

Ubicación del código del programa: Linea 433 - 443, del archivo: Ejercicios Programación.py.

26. Desarrollar un algoritmo que calcule el mínimo de un arreglo de números reales/naturales.

Objetos Conocidos: arreglo de números (arreglo ∈ N o R).

Objetos Desconocidos: número mínimo del arreglo de números reales (minimo ∈ N o R).

Función:

$$minimo : R^n \to R$$

Si son naturales

$$minimo : N^n \to N$$

$$minimo = arreglo_0$$

Valor inicial de minimo.

$$(arreglo) \to minimo = \begin{cases} arreglo_i, & \text{si } \forall_{i=0}^{n-1} arreglo_i < minimo \\ minimo, & \text{si } en\ otro\ caso \end{cases} \tag{16}$$

donde n es el largo del arreglo.

Ubicación del código del programa: Linea 446 - 457, del archivo: Ejercicios Programación.py.

27. Desarrollar un algoritmo que calcule el máximo de un arreglo de números reales/naturales.

Objetos Conocidos: arreglo de números (arreglo ∈ N o R).

Objetos Desconocidos: número máximo del arreglo de números reales/naturales (maximo ∈ N o R).

Función:

$$maximo : R^n \to R$$

Si son naturales

$$maximo : N^n \to N$$

$$maximo = arreglo_0$$

Valor inicial de maximo.

$$(arreglo) \to maximo = \begin{cases} arreglo_i, & \text{si } \forall_{i=0}^{n-1} arreglo_i > maximo \\ maximo, & \text{si } en\ otro\ caso \end{cases} \tag{17}$$

donde n es el largo del arreglo.

Ubicación del código del programa: Linea 460 - 483, del archivo: Ejercicios Programación.py.

28. Desarrollar un algoritmo que calcule el producto directo de dos arreglos de números reales/naturales de igual tamaño.

Objetos Conocidos: primer arreglo de números reales/naturales (v ∈ N o R) y el segundo arreglo de números reales/naturales (w ∈ N o R).

Objetos Desconocidos: arreglo con el producto directo de los vectores v y w (productoD ∈ N o R)

Función:

$$productoDirecto : R^n \to R^n$$

Si son naturales

$$productoDirecto : N^n \to N^n$$

$$(arreglo) = \forall_{i=0}^{n-1} productoD_i = v_i * w_i.$$

donde n es el largo tanto de arreglo v como del arreglo w.

Ubicación del código del programa: Linea 473 - 483, del archivo: Ejercicios Programación.py.

29. Desarrollar un algoritmo que determine la mediana de un arreglo de números reales/naturales. La mediana es el número que queda en la mitad del arreglo después de ser ordenado.

Objetos Conocidos: arreglo de números (arreglo ∈ N o R)

Objetos Desconocidos: mediana del arreglo de números reales/naturales (mediana ∈ N o R)

Nota: Antes de iniciar la mediana, se debe ordenar los elementos del arreglo de manera ascendente.

Función:

$$mediana : R^n \to R$$

Si son naturales

$$mediana : N^n \to N$$

$$(arreglo) \to mediana = \begin{cases} \frac{arreglo_{(n/2)-1}+arreglo_{(n/2)}}{2}, & \text{si } n\%2 = 0 \\ arreglo_{(n-1)/2}, & \text{si } en\ otro\ caso \end{cases} \quad (18)$$

donde n es el largo del arreglo y (n/2)-1, (i/2),(i-1)/2 ∈ N y % la operación módulo.

.

Ubicación del código del programa: Linea 487 - 510, del archivo: Ejercicios Programación.py.

30. Hacer un algoritmo que deje al final de un arreglo de números reales/naturales, todos los ceros que aparezcan en dicho arreglo.

Objetos Conocidos: arreglo de números (arreglo ∈ N o R), un contador que cuenta la cantidad de ceros en un arreglo, (cont ∈ N), una función que realiza un corrimiento hacia la izquierda de los elementos y una función que reemplaza elementos de un array por una cantidad especifica de ceros al final del array.

Objetos Desconocidos: arreglo de números reales con ceros en el final (result ∈ N o R)

Función que realiza un corrimiento hacia la izquierda:

$$corrimientoIzq : N \times N^n \to N^n$$

$$(pos, arreglo) = \forall_{j=pos}^{n-1} arreglo_j = arreglo_{j+1}$$

Función que reemplaza una cantidad especifica de elementos de un array por ceros:

$$reemplazarPorCeros : N \times N^n \to N^n$$
$$(num, arreglo) = \forall_{k=n-num}^{n-1} arreglo_k = 0$$

Función:

$$result : R^n \to R^n$$

Si son naturales

$$result : N^n \to N^n$$
$$(arreglo) = \sum_{i=0}^{n-1} = \begin{cases} 1, corrimientoIzq(i, arreglo) & \text{si } arreglo_i = 0 \\ 0, & \text{si } en\ otro\ caso \\ reemplazarPorCeros(cont, arreglo), & \text{si } i = n-1 \end{cases} \tag{19}$$

donde n es el largo total del arreglo. Ubicación del código del programa: Linea 513 - 532, del archivo: Ejercicios Programación.py.

31. Hacer un algoritmo que calcule los números en decimal que representa dicho arreglo de unos y ceros. reales/naturales.

Objetos Conocidos: arreglo de unos y ceros (arreglo $\in$ N).

Objetos Desconocidos: número decimal que representa dicho arreglo de unos y ceros (decimal $\in$ N).

Función:

$$binarioADecimal : N^n \to N$$

$$(arreglo) = \forall_{i=0}^{n-1} = \begin{cases} 2^i, & \text{si } arreglo_i = 1 \\ 0, & \text{si } en\ otro\ caso \end{cases} \tag{20}$$

donde n es el largo del arreglo.

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

32. Hacer un algoritmo que dado un número entero no negativo, cree un arreglo de unos y ceros que representa el número en binario al revés

Objetos Conocidos: numero ingresado (arreglo $\in Z \geq 0$).

Objetos Desconocidos: arreglo de unos y ceros que representa el número en

binario al revés (binario ∈ N).

Función:

$$decimalABinario : Z \to N^n$$

$$(arreglo) \to \forall\, decimal/2 = \begin{cases} binario_i = decimal\%2, decimal/2 & \text{si } decimal/2 \neq 0 \\ binario_i = decimal & \text{si } en\ otro\ caso \end{cases} \tag{21}$$

Siendo % la operación módulo y decimal/ 2 ∈ Z.

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

33. Hacer un algoritmo que calcule el Máximo Común Divisor (MCD) para un arreglo de enteros positivos.

Objetos Conocidos: arreglo de números (arreglo ∈ $Z \geq 0$).

Objetos Desconocidos: Máximo Común Divisor (MCD) (mcdValor ∈ N).

Función Máximo Común Divisor de dos números :

$$mcd : Z \times Z \to N$$

$$(x, y) \to mcd = \begin{cases} x & \text{si } y = 0 \\ mcd(y, x\%y) & \text{si } en\ otro\ caso \end{cases} \tag{22}$$

Función :

$$calcularmcd : Z^n \to N$$

$$(arreglo) \to mcdValor = \begin{cases} mcd(array_i, array_{i+1}) \forall_{i=0}^{n-1} & \text{si } i = 0 \\ mcd(mcdValor, array_i) \forall_{i=0}^{n-1} & \text{si } i \geq 2 \end{cases} \tag{23}$$

Siendo % la operación módulo y donde n es el largo del arreglo.

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

34. Hacer un algoritmo que calcule el Mínimo Común Múltiplo (MCM) para un arreglo de entero positivos.

Objetos Conocidos: arreglo de números (arreglo $\in Z \geq 0$).

Objetos Desconocidos: Mínimo Común Múltiplo (MCM) (mcmValor $\in$ N).

Función Máximo Común Divisor de dos números :

$$mcd : Z \times Z \to N$$

$$(x, y) \to mcd = \begin{cases} x & \text{si } y = 0 \\ mcd(y, x\%y) & \text{si } en\ otro\ caso \end{cases} \tag{24}$$

Función :

$$calcularmcm : Z^n \to N$$

$$(arreglo) \to mcmValor =$$

$$= \begin{cases} mcd(array_i, array_{i+1}), multiplicar = array_i * array_{i+1}, total = \frac{multiplicar}{mcmValor}\ \forall_{i=0}^{n-1} & \text{si } i = 0 \\ mcd(mcmValor, array_i), multiplicar = total * array_i, total = \frac{multiplicar}{mcmValor}\ \forall_{i=0}^{n-1} & \text{si } i \geq 2 \end{cases} \tag{25}$$

donde n es el largo del arreglo.

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

# 6 Conjuntos como arreglos

35. Calcula en un arreglo la unión de los conjuntos y la imprime.

Objetos Conocidos: dos arreglos de números (conjuntoUno,conjuntoDos $\in$ R) y una función para retirar las repeticiones de un arreglo (en este caso un conjunto).

Objetos Desconocidos: arreglo que contiene la unión de los dos arreglos de números (union $\in$ R).

Función que quita repeticiones en un conjunto :

$$quitarRep : R^n \to R^*$$

$$(conjunto) \to \forall_{i=0}^{n-1} conjuntoSinRep_i = conjunto_i\ , \text{si } conjunto_i \neq conjunto_{i+1}$$

Función :

$$union : R^n \times R^n \to R^*$$

$$(conjuntoUno, conjuntoDos) \to union =$$
$$quitarRep(conjuntoUno + conjuntoDos) = conjuntoUno \cup conjuntoDos$$

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

36. Calcula en un arreglo la intersección de los conjuntos y la imprime.

Objetos Conocidos: dos arreglos de números (conjuntoUno,conjuntoDos ∈ R).

Objetos Desconocidos: arreglo que contiene la intersección de los dos arreglos de números (conjuntoInter ∈ R).

Función :

$$conjuntoInter : R^n \times R^n \to R^*$$

$$(conjuntoUno, conjuntoDos) \to \forall_{i=0}^{n} \forall_{j=0}^{m} conjuntoInter_i$$
$$= \begin{cases} conjuntoUno_i, \\ \text{si } conjuntoUno_i = conjuntoDos_j \end{cases} \tag{26}$$
$$conjuntoInter = conjuntoUno \cap conjuntoDos$$

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

37. Calcula en un arreglo la diferencia del primero con el segundo y la imprime.

Objetos Conocidos: dos arreglos de números (conjuntoUno,conjuntoDos ∈ R) y una bandera (flag "∈ B).

Objetos Desconocidos: arreglo que contiene la diferencia de los dos arreglos de números (conjuntoDif ∈ R).

Función :

$$conjuntoDif : R^n \times R^n \to R^*$$
$$flag_0 = False$$

Valor inicial de flag.

$$(conjuntoUno, conjuntoDos) \rightarrow \forall_{i=0}^{n} \forall_{j=0}^{m} conjuntoDif_i$$

$$= \left\{ \begin{array}{ll} conjuntoUno_i, & \text{si } flag = True \ \forall conjuntoUno_i = conjuntoDos_j \end{array} \right. \tag{27}$$

$$conjuntoDif = conjuntoUno \setminus conjuntoDos$$

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

38. Calcula en un arreglo la diferencia simetrica de los conjuntos y la imprime.

Objetos Conocidos: dos arreglos de números (conjuntoUno,conjuntoDos ∈ R).

Objetos Desconocidos: arreglo que contiene la diferencia simetrica de los dos arreglos de números (conjuntoDifSim ∈ R).

Nota: Se hara uso de la funcion diferencia del ejercicio 37
Función :

$$conjuntoDifSim : R^n \times R^n \rightarrow R^*$$

$$(conjuntoUno, conjuntoDos) \rightarrow conjuntoDifSim =$$

$$= diferencia(conjuntoUno, conjuntoDos) + diferencia(conjuntoDos, conjuntoUno)$$

$$conjuntoDifSim = conjuntoUno \triangle conjuntoDos$$

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

39. Lee un entero y determina si el elemento pertenece o no a cada uno de los conjuntos y lo imprime.

Objetos Conocidos: número (numeroEnt ∈ Z) y dos arreglos de números (conjuntoUno,conjuntoDos ∈ R).

Objetos Desconocidos: cadena de texto que indica si el número entero se encuentra o no, contenido en los conjuntos (result ∈ B).

Función :

$$pertenece : Z \times R^n \times R^n \rightarrow B$$

$$(conjuntoUno, conjuntoDos, numeroEnt) \rightarrow result$$

$$\begin{cases} True & \text{si } \forall_{i=0}^{n-1} \exists conjuntoUno_i = numeroEnt \text{ y } \forall_{j=0}^{m-1} \exists conjuntoDos_j = numeroEnt \\ False & \text{si } en\ otro\ caso \end{cases} \tag{28}$$

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

40. Determina sí el primer conjunto esta contenido en el segundo y lo imprime.

Objetos Conocidos: dos arreglos de números (conjuntoUno,conjuntoDos ∈ R).

Objetos Desconocidos: cadena de texto que indica si el primer conjunto esta contenido en el segundo (result ∈ B).

Función :

$$R^n \times R^n \rightarrow B$$

$$(conjuntoUno, conjuntoDos) \rightarrow \sum_{k=0}^{n} =$$

$$\begin{cases} 1 & \text{si } \forall_{i=0}^{n} \forall_{j=0}^{m} \exists conjuntoUno_i = conjuntoDos_j \\ 0 & \text{si } en\ otro\ caso \\ True & \text{si } \sum = n \\ False & \text{si } \sum < n \end{cases} \tag{29}$$

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

42. Desarrollar el programa anterior usando la representación modificada con las operaciones entre conjuntos optimizadas (ordenar un conjunto).

Objetos Conocidos: un arreglo de números (conjunto ∈ R) y una variable aux (aux ∈ R).

Objetos Desconocidos: arreglo que contiene a otro arreglo ordenado de forma descendente (conjuntoOrd ∈ R).

Función :

$$R^n \to R^n$$

$$(conjunto) = \forall_{i=0}^{n-2} \forall_{j=i+1}^{m-1} =$$

$$\begin{cases} aux = conjunto_i, conjunto_i = conjunto_j, conjunto_j = aux & \text{si } conjunto_j < conjunto_i \\ conjunto_i & \text{si } en\ otro\ caso \end{cases} \tag{30}$$

Nota: Utilizando la función ordenar conjunto, se optimizan las funciones 35-40 en sus operaciones y en la impresión de resultados.
Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

# 7 Polinomios como arreglos

43. Lee un real e imprime la evaluación de los dos polinomios en dicho dato.

Objetos Conocidos: número leido (num ∈ R) y dos polinomios (polinomioUno,polinomioDos ∈ R).

Objetos Desconocidos: resultados de evaluar los polinomios en el número leido (resultUno,resultDos ∈ R).

Función para evaluar polinomioUno:

$$R \times R^n \to R$$

$$(num, polinomioUno) = \sum_{i=0}^{n} polinomioUno_i * num^i = resultUno$$

Función para evaluar polinomioDos:

$$R \times R^n \to R$$

$$(num, polinomioDos) = \sum_{j=0}^{m} polinomioDos_j * num^j = resultDos$$

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

44. Calcula el polinomio suma y lo imprime.

Objetos Conocidos: dos polinomios (polinomioUno,polinomioDos ∈ R).

Objetos Desconocidos: resultados de sumar los polinomios (polSumar ∈ R).

Función:

$$R^n \times R^m \to R^*$$

$(polinomioUno, polinomioDos) \to \forall_{i=0}^{p} polSumar_i = polinomioUno_i + polinomioDos_i$

Siendo p el largo del polinomio mas grande.
Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

45. Calcula el polinomio resta y lo imprime.

Objetos Conocidos: dos polinomios (polinomioUno,polinomioDos ∈ R).

Objetos Desconocidos: resultados de restar los polinomios (polRestar ∈ R).

Función:

$$R^n \times R^m \to R^*$$

$(polinomioUno, polinomioDos) \to \forall_{i=0}^{p} polRestar_i = polinomioUno_i + polinomioDos_i$

Siendo p el largo del polinomio mas grande.
Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

46. Calcula el polinomio multiplicación y lo imprime.

Objetos Conocidos: dos polinomios (polinomioUno,polinomioDos ∈ R).

Objetos Desconocidos: resultados de multiplicar los polinomios (polMultiplicar ∈ R).

Función:

$$R^n \times R^m \to R^*$$

$$(polinomioUno, polinomioDos) \to \forall_{k=0}^{i+j} polMultiplicar_k$$

$$= \sum_{i=0}^{n} \sum_{j=0}^{m} polinomioUno_i * polinomioDos_j$$

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

# 8 Matrices

50. Desarrollar un algoritmo que permita sumar dos matrices de números reales (enteros).

Objetos Conocidos: dos matrices matrizUno,matrizDos ($\in$ R) y función para crear una matriz de ceros (crearMat).

Objetos Desconocidos: matriz resultados de sumar dos matrices (matSuma $\in$ R).

Función crear matriz de ceros:

$$R \times R \to R^{*}*$$

$$(x, y) \to \forall_{i=0}^{x-1} \forall_{i=0}^{y-1} matSuma_{xy} = 0$$

Función:

$$R^n \times R^n \to R^n$$

$$(matrizUno, matrizDos) \to \forall_{i=0}^{n-1} \forall_{i=0}^{n-1} matSuma_{ij} = matrizUno_{ij} + matrizDos_{ij}$$

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

51. Desarrollar un algoritmo que permita multiplicar dos matrices de números reales (enteros).

Objetos Conocidos: dos matrices matrizUno,matrizDos ($\in$ R) y función para crear una matriz de ceros (crearMat).

Objetos Desconocidos: matriz resultados de multiplicar dos matrices (matMul $\in$ R).

Función crear matriz de ceros:

$$R \times R \to R^{*}*$$

$$(x, y) \to \forall_{i=0}^{x-1} \forall_{i=0}^{y-1} matMul_{xy} = 0$$

Función:

$$R^{n} \times R^{n} \to R^{n}$$

$$(matrizUno, matrizDos) \to \forall_{i=0}^{n-1} \forall_{i=0}^{n-1} matMul_{ij} = matrizUno_{ij} * matrizDos_{ij}$$

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

52. Desarrollar un programa que sume los elementos de una columna dada de una matriz.

Objetos Conocidos: matriz (matriz $\in$ R), columna de la matriz a la cual se sumaran todos sus elementos (col $\in$ N) y función para crear una matriz de ceros (crearMat).

Objetos Desconocidos: resultado de sumar todos los elementos de una columna en una matriz (sumCol $\in$ R).

Función crear matriz de ceros:

$$R \times R \to R^{*}*$$

$$(x, y) \to \forall_{i=0}^{x-1} \forall_{i=0}^{y-1} matMul_{xy} = 0$$

Función:

$$R^{n} \times R \to R$$

$$(matriz, col) \to \sum_{i=0}^{n-1} sumCol = matriz_{i,col}$$

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

53. Desarrollar un programa que sume los elementos de una fila dada de una matriz.

Objetos Conocidos: matriz (matriz $\in$ R), fila de una matriz a la cual se sumaran todos sus elementos (fila $\in$ N) y función para crear una matriz de ceros (crearMat).

Objetos Desconocidos: resultado de sumar todos los elementos de una fila en una matriz (sumFila $\in$ R).

Función crear matriz de ceros:

$$R \times R \to R^{*}*$$

$$(x, y) \to \forall_{i=0}^{x-1} \forall_{i=0}^{y-1} matMul_{xy} = 0$$

Función:

$$R^{n} \times R \to R$$

$$(matriz, fila) \to \sum_{i=0}^{n-1} matriz_{fila,i} = sumFila$$

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

54. Desarrollar un algoritmo que determine si una matriz es mágica.

Objetos Conocidos: matriz (matriz $\in$ R) y función para sumar elementos de una fila, función para sumar elementos de una columna, función para sumar elementos de la diagonal principal de una matriz y función para sumar elementos de la diagonal secundaria de una matriz (sumarEnFila, sumarEnColumna, sumarDiagonal y sumarDiagonal2).

Objetos Desconocidos: booleano que determina si la matriz es mágica o no (result $\in$ B).

Función sumarEnFila: Revisar ejercicio 53. Función sumarEnColumna: Revisar ejercicio 52. Función sumarDiagonal:

$$R^{n} \to R$$

$$(matriz) \rightarrow \sum_{i=0}^{n-1} matriz_{ii} = sumDiag$$

Función sumarDiagonal2:

$$R^n \rightarrow R$$

$$(matriz) \rightarrow sumDiag2 = \begin{cases} \sum_{i=0}^{n-1} matriz_{ij} & \text{si } i = n-j-1 \ y \ j = n-i-1 \\ 0 & \text{si } en \ otro \ caso \end{cases} \tag{31}$$

Función:

$$R^n \rightarrow B$$

$$(matriz) \rightarrow result = \sum_{i=0}^{n-1} sumFila = matriz_{fila,i}$$

$$\begin{cases} True & \text{si } sumarEnFila(matriz, i) = sumarEnFila(matriz, i+1) \ y \\ & \text{si } sumarEnColumna(matriz, i) = sumarEnColumna(matriz, i+1) \ y \\ & \text{si } sumarDiagonal(matriz) = sumarDiagonal2(matriz) \\ false & \text{si } en \ otro \ caso \end{cases} \tag{32}$$

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

55. Desarrollar un algoritmo que dado un entero, reemplace en una matriz todos los números mayores a ese entero por un uno y a los menores/iguales por un cero.

Objetos Conocidos: matriz (matriz ∈ R) y número (∈ Z).

Objetos Desconocidos: matriz reemplazada por unos y ceros (matriz ∈ N).

Función:

$$R^{nxn} \times Z \rightarrow N^{nxn}$$

$$(matriz, num) \rightarrow \forall_{i=0}^{n-1} \forall_{j=0}^{n-1}$$

$$(matriz, num) \rightarrow \forall_{i=0}^{n-1} \forall_{j=0}^{n-1} matriz_{ij} = \begin{cases} 1 & \text{si } matriz_{ij} > num \\ 0 & \text{si } en \ otro \ caso \end{cases} \tag{33}$$

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

56. Desarrollar un programa que calcule el determinante de una matriz cuadrada.

Objetos Conocidos: matriz cuadrada (matriz ∈ R), una función para borrar una fila de una matriz, una función para borrar una columna de una matriz, una función para calcular el determinante de una matriz 2x2 y una función para calcular el determinante de una matriz 3x3 (borrarFila, borrarColumna, detDosXDos, detTresXTres).

Objetos Desconocidos: determinante de una matriz cuadrada (det ∈ R).

Función borrarFila:

$$R^{nn} \times N \to R^{n-1,n}$$

$$(matriz, fila) \to \forall_{i=0}^{n-1} NuevaMatriz_i = \begin{cases} matriz_i & \text{si } i \neq fila \\ NuevaMatriz & \text{si } en\ otro\ caso \end{cases} \tag{34}$$

Función borrarColumna:

$$R^{nn} \times N \to R^{n,n-1}$$

$$(matriz, col) \to \forall_{i=0}^{n-1} \forall_{j=0}^{n-1} NuevaMatriz_{ij} = \begin{cases} matriz_{ij} & \text{si } j \neq col \\ NuevaMatriz & \text{si } en\ otro\ caso \end{cases} \tag{35}$$

Función detDosXDos:

$$R^{nn} \to R$$

$$(matriz) \to det = matriz_{00} * matriz_{11} - matriz_{10} * matriz_{01}$$

Función detTresXTres:

$$R^{nn} \to R$$

$$(matriz) \to \sum_{j=0}^{n-1} (matriz_{0j} * ((-1)^j) * A_{0j}) = det$$

Siendo $A_{0j}$ = detDosXDos(borrarCol(borrarFila(matriz,0),j))
Función:

$$R^n \to R$$

$$(matriz) \to det =$$

$$\begin{cases} detDosXDos(matriz) & \text{si } n = 2 \\ detTresXTres(matriz) & \text{si } n = 3 \\ \sum_{j=0}^{n-1}(matriz_{0j} * ((-1)^j) * A_{0j}) & \text{si } n = 4 \end{cases} \tag{36}$$

Siendo $A_{0j}$ = detTresXTres(borrarCol(borrarFila(matriz,0),j))

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

57. Desarrollar un programa que dadas una matriz cuadrada A y un arreglo de números reales del mismo tamaño B, calcule una solución x para el sistema de ecuaciones lineales Ax = B.

Objetos Conocidos: matriz (matriz ∈ R), un vector (vector∈ R) y una función para calcular una matriz inversa (matInversa).

Objetos Desconocidos: vector solución x (vecSol ∈ $R$).

Función matInversa:

$$R^{nxn} \to R^{nxn}$$

Para matrices 2x2

$$A^{-1} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}^{-1} = \frac{1}{|A|} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}.$$

Para matrices 3x3

$$A^{-1} = \frac{(Adj(A))^T}{|A|}, Adj(A)_{ij} = (-1)^{i+j}|A|$$

Siendo el exponente T la matriz adjunta.

Función:

$$R^{nxn} \times R^n \to R^n$$

$$(matriz, vector) \to \forall_{i=0}^{n-1} vecSol_i = \sum_{j=0}^{n-1} matriz_{ij} * vector_j$$

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

58. Desarrollar un programa que calcule la inversa de una matriz cuadrada.

Objetos Conocidos: matriz (matriz ∈ R).

Objetos Desconocidos: matriz inversa (matInv ∈ $R$).

Función matInversa:

$$R^{nxn} \rightarrow R^{nxn}$$

Para matrices 2x2

$$A^{-1} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}^{-1} = \frac{1}{|A|} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}.$$

Para matrices 3x3

$$A^{-1} = \frac{(Adj(A))^T}{|A|}, Adj(A)_{ij} = (-1)^{i+j}|A|$$

Siendo el exponente T la matriz adjunta.

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

59. Desarrollar un programa que tome un arreglo de tamaño $n^2$ y llene es espiral hacia adentro una matriz cuadrada de tamaño n.

Objetos Conocidos: arreglo de tamaño $n^2$ (arreglo ∈ R).

Objetos Desconocidos: matriz en espiral (matrizEsp ∈ R).

Nota: Función solo para arreglo tamaño 9. Función:

$$R^n \rightarrow R^{nn}$$

$$(arreglo) \rightarrow \forall_{i=0}^{n-1} \sum_{k=0}^{n-1} k =$$

$$\begin{cases} \forall_{j=0}^{n-1} matrizEsp_{ij} = arreglo[k], 1 & \text{si } k = 0 \ y \ k <= 2 \\ \forall_{j=1}^{n-1} matrizEsp_{j,n-1} = arreglo[k], 1 & \text{si } k => 2 \ y \ k <= 4 \\ \forall_{j=n-2}^{0} matrizEsp_{n-1,j} = arreglo[k], 1 & \text{si } k => 4 \ y \ k <= 6 \\ \forall_{j=n-2}^{1} matrizEsp_{j0} = arreglo[k], 1 & \text{si } k => 6 \ y \ k <= 7 \\ \forall_{j=1}^{n-2} matrizEsp_{n-2,j} = arreglo[k], 1 & \text{si } k = 8 \\ matrizEsp & \text{si } en\ otro\ caso \end{cases} \tag{37}$$

Ubicación del código del programa: Linea , del archivo: Ejercicios Programación.py.

# 9 Cadenas

71. Desarrollar un algoritmo que reciba como entrada un carácter y de como salida el numero de ocurrencias de dicho carácter en una cadena de caracteres.

Objetos Conocidos: carácter de entrada (caracter ∈ ASCII).

Objetos Desconocidos: numero de ocurrencias del caracter de entrada, en una cadena, (ocurrencia ∈ N).

Nota: Como una cadena de caracteres (string), se comporta como un arreglo de caracteres, se puede acceder a una posición especifica de la cadena para determinar el caracter en esa posición.
Función:

$$ocurrencia : ASCII^n \to N$$

$$(caracter) = \sum_{i=0}^{n-1} = \begin{cases} 1 & \text{si } cadena_i = caracter \\ 0 & \text{si } en\ otro\ caso \end{cases} \tag{38}$$

donde n es el largo de la cadena.
Ubicación del código del programa: Linea 535 - 551 , del archivo: ejercicios programacion entrega 2.py .

72. Desarrollar un algoritmo que reciba como entrada dos cadenas y determine si la primera es subcadena de la segunda.

Objetos Conocidos: primera cadena de texto ingresada por el usuario, (cadenaUno ∈ ASCII) y segunda cadena de texto ingresada por el usuario, (cadenaDos ∈ ASCII).

Objetos Desconocidos: cadena de texto que indica al usuario si la primera cadena texto ingresada es subcadena de la segunda cadena de texto ingresada, (resul ∈ B).

Ver nota ejercicio 71.
Función:

$$result : ASCII^n \times ASCII^m \to B$$

$$(cadenaUno, cadenaDos) = \forall_{i=0}^{m-1} cadenaDos_i \begin{cases} True, & \text{si } cadenaDos_i^{i+n} = cadenaUno_0^n \\ False, & \text{si } en\ otro\ caso \end{cases} \tag{39}$$

donde n es el largo de la cadena uno y m el largo de la cadena dos.
Ubicación del código del programa: Linea 553 - 566, del archivo: ejercicios programacion entrega 2.py .

73. Desarrollar un algoritmo que reciba dos cadenas de caracteres y determine si la primera esta incluida en la segunda, todos los caracteres (con repeticiones) de la cadena uno están presentes en la cadena dos, sin importar el orden.

Objetos Conocidos: primera cadena de texto ingresada por el usuario, (cadenaUno ∈ ASCII), segunda cadena de texto ingresada por el usuario, (cadenaDos ∈ ASCII), y una bandera para evitar que un caracter contado de la primera cadena se ha contado dos veces, (flag ∈ B).

Objetos Desconocidos: cadena de texto que determine si la primera esta incluida en la segunda, (resul ∈ B).

Ver nota ejercicio 71.
Función:

$$result : ASCII^n \times ASCII^m \to B$$

$$(cadenaUno, cadenaDos) = \sum_{i=0}^{n-1}$$

$$= \begin{cases} 1, flag = true & \text{si } \exists cadenaUno_i = cadenaDos_j \ y\ flag = false \\ True & \text{si } \sum = n \\ False & \text{si } en\ otro\ caso \end{cases} \tag{40}$$

donde n es el largo de la cadena uno, y m es el largo de la cadena dos.
Ubicación del código del programa: Linea 568 - 589, del archivo: ejercicios programacion entrega 2.py .

74. Desarrollar un algoritmo que invierta una cadena de caracteres.

Objetos Conocidos: cadena de entrada, (cadena ∈ ASCII).

Objetos Desconocidos: cadena de entrada invertida, (cadenaInvertida ∈ ASCII)

Ver nota ejercicio 71.
Función:

$$invertir : ASCII^n \to ASCII^n$$

$$(cadena) = cadenaInvertida = cadena_{i=n-1}^{0}$$

donde n es el largo de la cadena de entrada.
Ubicación del código del programa: Linea 591 - 603, del archivo: ejercicios programacion entrega 2.py .

75. Desarrollar un algoritmo que determine si una cadena de caracteres es palíndrome. Una cadena se dice palíndrome si al invertirla es igual a ella misma.

Objetos Conocidos: cadena de entrada, (cadena ∈ ASCII).

Objetos Desconocidos: cadena de texto que determine si una cadena de caracteres es palíndrome, (resul ∈ b).

Ver nota ejercicio 71.
Nota 2: Para este ejercicio se reutilizara la función invertir(), del ejercicio 74, para invertir una cadena de texto. Función:

$$palindrome : ASCII^n \rightarrow b$$

$$(cadena) = \begin{cases} True, & \text{si } cadena_{i=0}^{n-1} = invertir(cadena_{i=0}^{n-1}) \\ False, & \text{si } en\ otro\ caso \end{cases} \tag{41}$$

Ubicación del código del programa: Linea 605 - 623, del archivo: ejercicios programacion entrega 2.py .

76. Desarrollar un algoritmo que determina si una cadena de caracteres es frase palíndrome. Una cadena se dice frase palíndrome si la cadena al eliminarle los espacios es palíndrome.

Objetos Conocidos: cadena de entrada, (cadena ∈ ASCII) y una función para borrar espacios en blanco (borrarEspacios).

Objetos Desconocidos: cadena de texto que determine si una cadena de caracteres es frase palíndrome, (resul ∈ B).

Ver nota ejercicio 71.
Nota 2: Para este ejercicio se reutilizara la función invertir(), del ejercicio 74, para invertir una cadena de texto.
Función para borrar espacios en blanco:

$$borrarEspacios : ASCII^n \rightarrow ASCII^m$$

$$(cadenaEnt) = \forall_{i=0}^{n-1} cadenaEnt_i \begin{cases} aux + cadenaEnt_i, \\ \text{si } cadenaEnt_i \neq \text{" "} \end{cases} \tag{42}$$

donde n es el largo de la cadena de entrada . Función:

$$frasePalindrome : ASCII^n \rightarrow B$$

$$(cadena) = \begin{cases} True, & \text{si } borrarEspacios(cadena_{i=0}^{n-1}) = invertir(borrarEspacios(cadena_{i=0}^{n-1})) \\ False, & \text{si } en\ otro\ caso \end{cases} \tag{43}$$

Ubicación del código del programa: Linea 625 - 656, del archivo: ejercicios programacion entrega 2.py .

77. Desarrollar un algoritmo que realice el corrimiento circular a izquierda de una cadena de caracteres. El corrimiento circular a izquierda es pasar el primer carácter de una cadena como éultimo carácter de la misma.

Objetos Conocidos: cadena de entrada, (cadena ∈ ASCII).

Objetos Desconocidos: cadena de entrada con corrimiento circular a izquierda, (cadenaCorriIzq ∈ ASCII).

Ver nota ejercicio 71.
Función:

$$corriCircularIzq : ASCII^n \rightarrow ASCII^n$$

$$(cadena) = cadenaCorriIzq \begin{cases} cadenaCorriIzq_{i=0}^{n-2} = cadena_{i=1}^{n-1}, & \text{si } i \neq 0 \\ cadenaCorriIzq_{n-1} = cadena_0, & \text{si } en\ otro\ caso \end{cases} \tag{44}$$

donde n es el largo de la cadena de entrada.
Ubicación del código del programa: Linea 658 - 674, del archivo: ejercicios programacion entrega 2.py .

78. Desarrollar un algoritmo que realice el corrimiento circular a derecha de una cadena de caracteres. El corrimiento circular a derecha de una cadena es poner el éultimo carácter de la cadena como primer carácter de la misma.

Objetos Conocidos: cadena de entrada, (cadena ∈ ASCII).

Objetos Desconocidos: cadena de entrada con corrimiento circular a la derecha, (cadenaCorriDer ∈ ASCII).

Ver nota ejercicio 71.
Función:

$$corriCircularDer : ASCII^n \rightarrow ASCII^n$$

$$(cadena) = corriCircularDer = \begin{cases} corriCircularDer_{i=1}^{n-1} = cadena_{i=0}^{n-2}, & \text{si } i \neq n-1 \\ corriCircularDer_0 = cadena_{n-1}, & \text{si } en\ otro\ caso \end{cases} \tag{45}$$

donde n es el largo de la cadena de entrada.
Ubicación del código del programa: Linea 676 - 692, del archivo: ejercicios programacion entrega 2.py .

79. Desarrollar un algoritmo que codifique una cadena de caracteres mediante una cadena de correspondencias de caracteres dada. La cadena de correspondencias tiene como el primer carácter el carácter equivalente para el carácter 'a', el segundo carácter para la 'b'y así sucesivamente hasta la 'z'. No se tiene traducción para las mayúsculas ni para la 'ñ'.

Objetos Conocidos: cadena de entrada a codificada, (cadena ∈ ASCII), cadena de entrada de correspondencias, (cadenaCor ∈ ASCII), arreglo con las letras del abecedario, solo minusculas, (abecedario ∈ ASCII) y función para convertir una letra, (letra ∈ ASCII), en un número, (j ∈ N).

Objetos Desconocidos: cadena de entrada ya codificada (cadenaCod).

Ver nota ejercicio 71.
El array, abecedario, es un array de tipo $ASCII^{26}$. Función para convertir una letra a número:

$$letraANumero : ASCII^{j} \to N$$

$$(letra) = \forall_{j=0}^{25} \begin{cases} j, & \text{si } \exists letra = abecedario_j \\ -1, & \text{si } en\ otro\ caso \end{cases} \tag{46}$$

Función:

$$codifique : ASCII^{n} \times ASCII^{m} \to ASCII^{n}$$

$$(cadena) = \forall_{i=0}^{n-1} cadenaCod_i \begin{cases} \text{" "}, & \text{si } cadena_i = \text{" "} \\ cadena_i, & \text{si } x = -1 \\ cadenaCor(x), & \text{si } en\ otro\ caso \end{cases} \tag{47}$$

donde x = letraANumero($cadena_i$) y n es el largo de la cadena de entrada.
Ubicación del código del programa: Linea 694 - 722, del archivo: ejercicios programacion entrega 2.py .

80. Desarrollar un algoritmo que decodifique una cadena de caracteres mediante una cadena de correspondencias de caracteres dada. La cadena de correspondencias tiene como el primer carácter el carácter equivalente para el carácter 'a', el segundo carácter para la 'b'y así sucesivamente hasta la 'z'. No se tiene traducción para las mayúsculas ni para la 'ñ'.

Objetos Conocidos: cadena de entrada a decodificar, (cadena ∈ ASCII), cadena de entrada de correspondencias, (cadenaCor ∈ ASCII), arreglo con las letras del abecedario, solo minusculas, (abecedario ∈ $ASCII^{*}$), función para determinar la posición de una letra de cadenaCor y función para convertir una número, (num ∈ N), en una letra, (letra ∈ ASCII).

Objetos Desconocidos: cadena de entrada ya decodificada, (cadenaDec ∈ ASCII).

Ver nota ejercicio 71.

función para determinar la posición de una letra :

$$posicionLetra : ASCII^k \times ASCII^n \to N$$

$$(letra, cadenaCor) = \forall_{k=0}^{25} = \begin{cases} k, & \text{si } \exists letra = cadenaCor_k \\ -1, & \text{si } en\ otro\ caso \end{cases} \tag{48}$$

Función para convertir un número a letra:

$$numeroALetra : N \to ASCII$$

El array, abecedario,es un array de tipo $ASCII^{26}$.

$$(num) = \forall_{j=0}^{25} = \begin{cases} abecedario_j, & \text{si } num = j \\ -1, & \text{si } en\ otro\ caso \end{cases} \tag{49}$$

Función:

$$decodifique : ASCII^n \times ASCII^m \to ASCII^n$$

$$(cadena) = \forall_{i=0}^{n-1} cadenaDec_i = \begin{cases} \text{" "}, & \text{si } cadena_i = \text{" "} \\ cadena_i, & \text{si } x = -1 \\ numeroALetra(x), & \text{si } en\ otro\ caso \end{cases} \tag{50}$$

donde x = posicionLetra($cadena_i$,cadenaCor) y n es el largo de la cadena de entrada.
Ubicación del código del programa: Linea 724 - 758, del archivo: ejercicios programacion entrega 2.py .