



Tecnológico de Monterrey

Grupo:

TC3002B.201

Materia:

Desarrollo de aplicaciones avanzadas de ciencias computacionales

Evidencia Final Compiladores: Desarrollo de herramienta de soporte al proceso de análisis de imágenes

Integrantes:

Salomon Dabbah Beracha - A01028445

Marco Antonio Torres - A01025334

Martín Palomares García - A01066569

Fecha de entrega:

03/05/24

Reglas de la gramática implementada.....	3
1. Precedencia de operadores.....	5
2. Llamadas a funciones.....	6
3. Asignación de variables.....	7
4. Implementación de flujos de imágenes y 5. Aplicación de filtros de Open CV.....	8
6. Cada una de las nuevas características implementadas.....	11
a. Aceptar archivos y ejecutar el contenido.....	11
b. Todas las tareas entregadas (independientemente de la calificación si existiese alguna).....	12
c. Aceptar cualquier función de numpy para manejo de matrices como np.where, np.mean, np.std. Al menos 9 de ellas.....	12
d. Implementación de visualización de histogramas con opencv.....	17
e. Implementación de un algoritmo complejo como herramienta en el lenguaje: CannyEdgeDetection.....	18
f. Implementación de pruebas automatizadas. Uso de un marco de TESTING para probar todas las características de tu gramática.....	18

Reglas de la gramática implementada

```
<assignment> ::= VARIABLE SETTO <expression>
                | VARIABLE SETTO <flow>
                | <expression>
<flow> ::= VARIABLE CONNECT <flow_functions>
<flow_functions> ::= <flow_function_call> CONNECT <flow_functions>
                | <flow_function_call>
<flow_function_call> ::= VARIABLE LPAREN <params> RPAREN
<params_opt> ::= <params> | ε
<expression> ::= <expression> PLUS <term>
                | <expression> MINUS <term>
                | <term>
                | <string>
<string> ::= STRING
<term> ::= <term> TIMES <exponent>
        | <term> DIVIDE <exponent>
        | <exponent>
<exponent> ::= <factor> EXP <factor>
        | <factor>
        | LPAREN <expression> RPAREN
<factor> ::= NUMBER
        | VARIABLE
        | <function_call>
<function_call> ::= VARIABLE LPAREN <params_opt> RPAREN
<params> ::= <expression> | <expression> COMMA <params>
```

Terminales

COMMA	: 24
CONNECT	: 3 4
DIVIDE	: 14
EXP	: 16
LPAREN	: 6 18 22 23
MINUS	: 9
NUMBER	: 19
PLUS	: 8
RPAREN	: 6 18 22 23
SETTO	: 1 2
STRING	: 12
TIMES	: 13
VARIABLE	: 1 2 3 6 20 22 23
error	:

Descripción de las funciones implementadas como herramientas y accesorios a la gramática

Función	Parámetro/s	Descripción
load_image	imagen	Carga una imagen al compilador y la lee.
save_image	nombre_del_archivo, imagen	Guarda la imagen en el dispositivo con el nombre seleccionado.
show_image	imagen	Despliega la imagen en el dispositivo.
search_cv2	nombre_de_la_funcion	Se usa para ver si una función existe en la librería de cv2, en caso de que exista, la función la regresa, de lo contrario arroja un error diciendo que no existe.
gen_matrix	filas, columnas, argumentos	Crea una matriz de x filas, por y columnas, y se le introducen los argumentos, es importante tener en cuenta que se tienen que introducir la cantidad de argumentos resultante de la multiplicación de x por y, por ejemplo, si la matriz sería de 3 x 3 tendría que tener 9 argumentos exactamente.
gen_vector	argumentos	Crea un vector que contiene los argumentos que se le proporcionen.
my_mean	argumentos	Obtiene la media de los argumentos proporcionados.
my_sum	argumentos	Obtiene la sumatoria de los argumentos proporcionados.
my_median	argumentos	Obtiene la mediana de los argumentos proporcionados.
my_std	argumentos	Obtiene la desviación estándar de los argumentos proporcionados.
my_max	argumentos	Obtiene el elemento más grande de los argumentos proporcionados.
my_min	argumentos	Obtiene el elemento más pequeño de los argumentos proporcionados.
my_sin	argumentos	Obtiene los senos de los argumentos proporcionados.
my_cos	argumentos	Obtiene los cosenos de los argumentos proporcionados.
my_tan	argumentos	Obtiene las tangentes de los argumentos proporcionados.
histogram	imagen	Convierte la imagen a escala de grises y calcula su histograma, que muestra la distribución de intensidades de píxeles de 0 (negro) a 255 (blanco).
canny_edge	imagen	Convierte la imagen a escala de grises y detecta los bordes utilizando el método de Canny. Luego, muestra la imagen original y la imagen de los bordes.
gray_scale	imagen	Carga la imagen y la convierte en blanco y negro.

Demostraciones

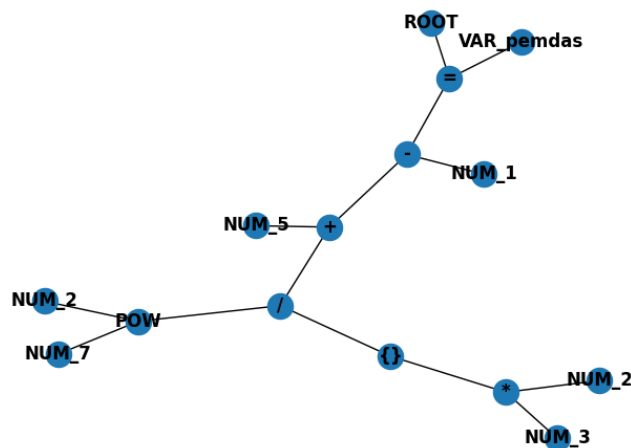
1. Precedencia de operadores

Corremos una ecuación en donde se prueben las operaciones aritméticas que tenemos en nuestro compilador:

$$5 + (3 * 2)/7^2 - 1 = 202/49 = 4.1224489796$$

```
input>> pmdas = 5 + (3 * 2) / 7 ^ 2 - 1
From Node 14 []
From Node 1 []
From Node 2 []
From Node 3 []
From Node 4 [3, 2]
From Node 5 [6]
From Node 6 []
From Node 7 []
From Node 8 [7, 2]
From Node 9 [6, 49]
From Node 10 [5, 0.12244897959183673]
From Node 11 []
From Node 12 [5.122448979591836, 1]
From Node 13 ['pmdas', 4.122448979591836]
From Node 0 [4.122448979591836]
Result: 4.122448979591836
```

Como se puede observar el resultado es correcto y por lo tanto también es correcta la precedencia de operadores.



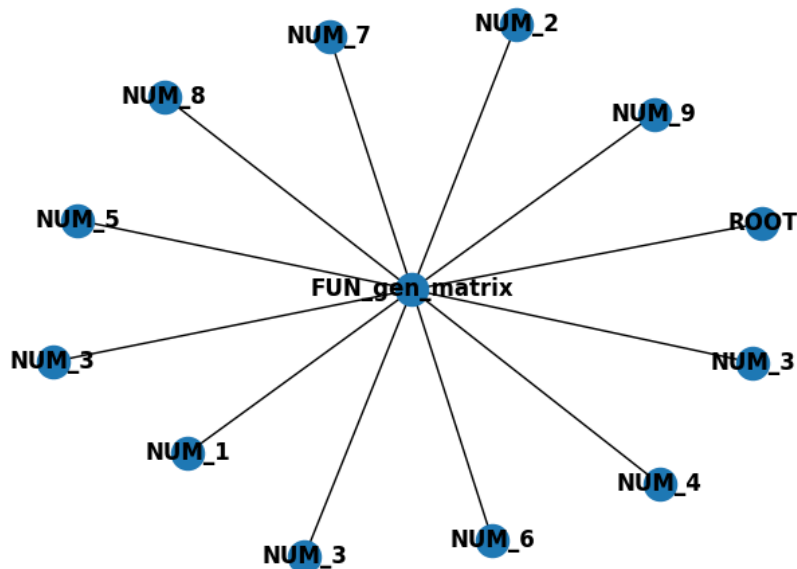
2. Llamadas a funciones

En este caso corremos una función que genera una matriz, esta recibe de parámetros las filas, columnas y elementos que estarán dentro de la matriz:

```
input>> gen_matrix(3,3,1,2,3,4,5,6,7,8,9)
[{'type': 'NUMBER', 'label': 'NUM_3', 'value': 3, 'counter': 1}, {'type': 'NUMBER', 'label': 'NUM_3', 'value': 3, 'counter': 2}, {'type': 'NUMBER', 'label': 'NUM_1', 'value': 1, 'counter': 3}, {'type': 'NUMBER', 'label': 'NUM_2', 'value': 2, 'counter': 4}, {'type': 'NUMBER', 'label': 'NUM_3', 'value': 3, 'counter': 5}, {'type': 'NUMBER', 'label': 'NUM_4', 'value': 4, 'counter': 6}, {'type': 'NUMBER', 'label': 'NUM_5', 'value': 5, 'counter': 7}, {'type': 'NUMBER', 'label': 'NUM_6', 'value': 6, 'counter': 8}, {'type': 'NUMBER', 'label': 'NUM_7', 'value': 7, 'counter': 9}, {'type': 'NUMBER', 'label': 'NUM_8', 'value': 8, 'counter': 10}, {'type': 'NUMBER', 'label': 'NUM_9', 'value': 9, 'counter': 11}]

From Node 1 []
From Node 2 []
From Node 3 []
From Node 4 []
From Node 5 []
From Node 6 []
From Node 7 []
From Node 8 []
From Node 9 []
From Node 10 []
From Node 11 []
From Node 12 [3, 3, 1, 2, 3, 4, 5, 6, 7, 8, 9]
**** HERE Function call: gen_matrix ****
From Node 0 [array([[1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9]])]
```

El resultado es el esperado y contamos con una matriz de 3x3 con los argumentos del 1 al 9 ordenados de forma correcta.

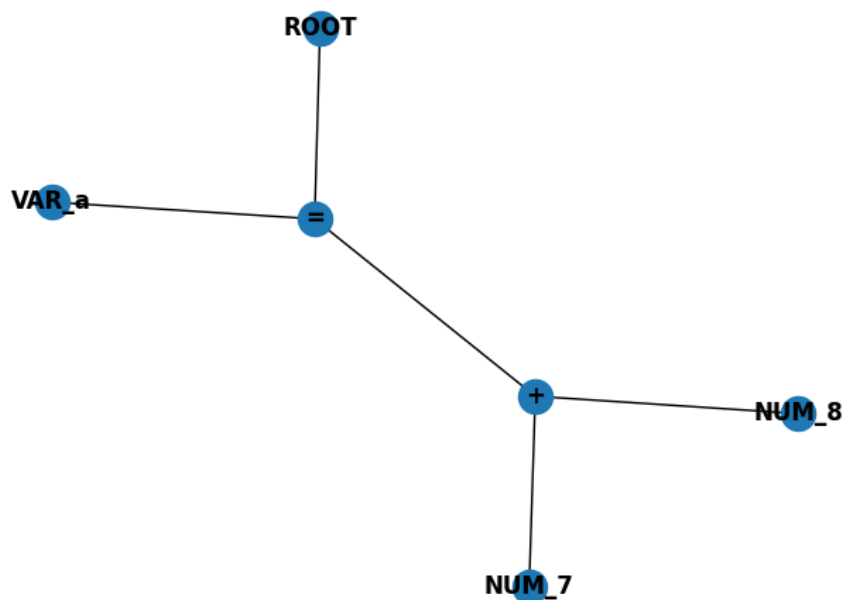


3. Asignación de variables

Asignamos la suma de $7 + 8$ a la variable "a":

```
input>> a = 7 + 8
From Node 5 []
From Node 1 []
From Node 2 []
From Node 3 [7, 8]
From Node 4 ['a', 15]
From Node 0 [15]
Result: 15
```

Podemos observar que la suma se hace correctamente y se asigna a la variable con éxito.



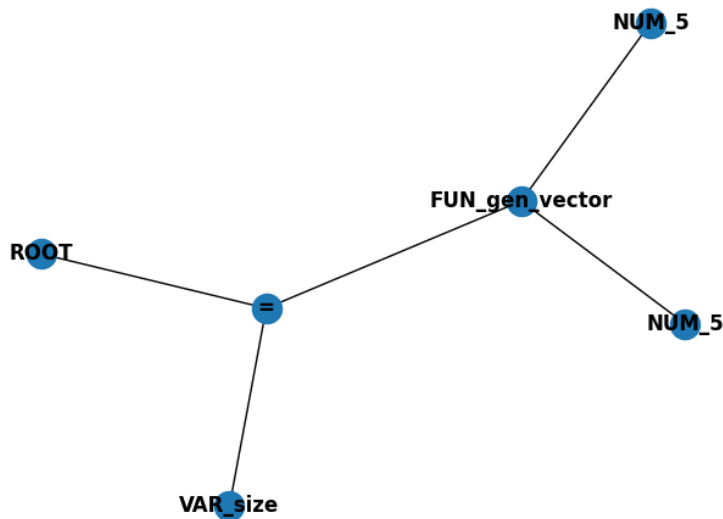
4. Implementación de flujos de imágenes y 5. Aplicación de filtros de Open CV

Usaremos esta imagen para implementar flujos y aplicaciones de filtros de Open CV, en este caso utilizaremos blur():



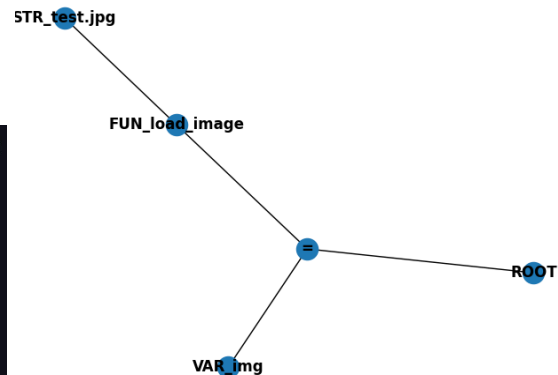
Generamos un vector que será un array de dos elementos llamado size (entre más grande sea este número mayor será el blur, en este caso usaremos [5,5]):

```
input>> size = gen_vector(5,5)
[{'type': 'NUMBER', 'label': 'NUM_5', 'value': 5, 'counter': 1}, {'type': 'NUMBER', 'label': 'NUM_5', 'value': 5, 'counter': 2}]
From Node 5 []
From Node 1 []
From Node 2 []
From Node 3 [5, 5]
**** HERE Function call: gen_vector ****
From Node 4 ['size', array([5, 5])]
From Node 0 [array([5, 5])]
```



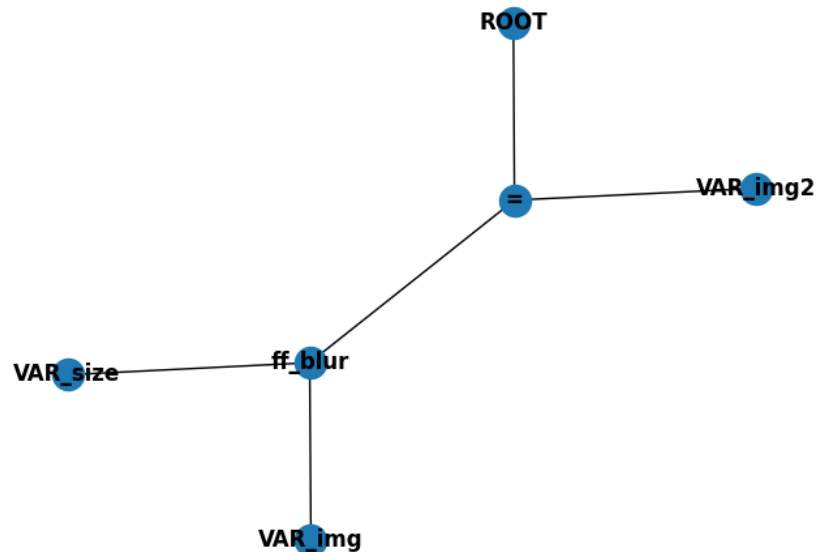
Usamos la función de load_image() para guardar la imagen en una variable llamada "img"

```
input>> img = load_image("test.jpg")
[{'type': 'STRING', 'label': 'STR_test.jpg', 'value': 'test.jpg', 'counter': 1}]
From Node 4 []
From Node 1 []
From Node 2 ['test.jpg']
**** HERE Function call: load_image ****
From Node 3 ['img', array([[252, 243, 233],
[252, 243, 233],
...
[ 1, 2, 0],
[ 1, 2, 0],
[ 1, 2, 0]],
[[253, 244, 234],
[253, 244, 234],
[253, 244, 234],
...
[ 1, 2, 0],
[ 1, 2, 0],
[ 1, 2, 0]],
[[254, 244, 234],
[254, 244, 234],
[254, 244, 234],
...
[ 1, 2, 0],
[ 1, 2, 0],
[ 1, 2, 0]]],
...]
```



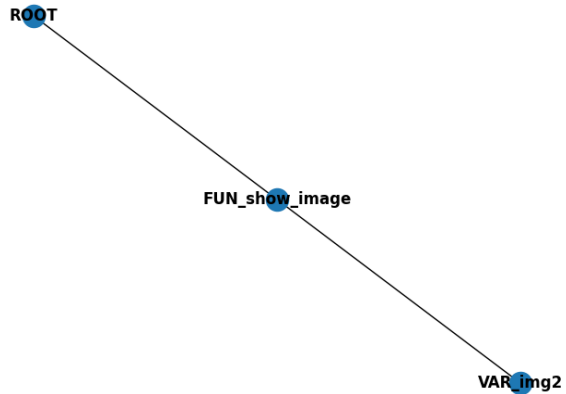
Implementamos el flujo de la imagen y aplicamos el filtro de Open CV de blur con el tamaño antes especificado.

```
input>> img2 = img -> blur(size)
From Node 5 []
From Node 3 []
From Node 1 []
From Node 2 [array([[252, 243, 233],
[252, 243, 233],
...
[ 1, 2, 0],
[ 1, 2, 0],
[ 1, 2, 0]],
[[253, 244, 234],
[253, 244, 234],
[253, 244, 234],
...
[ 1, 2, 0],
[ 1, 2, 0],
[ 1, 2, 0]],
[[254, 244, 234],
[254, 244, 234],
[254, 244, 234],
...
[ 1, 2, 0],
[ 1, 2, 0],
[ 1, 2, 0]]],
...]
```



Mostramos la imagen final en el dispositivo con el blur aplicado:

```
input>> show_image(img2)
[{'type': 'VARIABLE', 'label': 'VAR_img2', 'value': 'img2', 'counter': 1}]
From Node 1 []
From Node 2 [array([[253, 244, 234],
                    [253, 244, 234],
                    [253, 244, 234],
                    ...,
                    [ 1, 2, 0],
                    [ 1, 2, 0],
                    [ 1, 2, 0]],
                    [[253, 244, 234],
                    [253, 244, 234],
                    [253, 244, 234],
                    ...,
                    [ 1, 2, 0],
                    [ 1, 2, 0],
                    [ 1, 2, 0]],
                    [[254, 244, 234],
                    [254, 244, 234],
                    [254, 244, 234],
                    ...,
                    [ 1, 2, 0],
                    [ 1, 2, 0],
                    [ 1, 2, 0]]],
            ...]
```



Como se puede observar sí se aplicó un filtro en la imagen y se ve más borrosa que al principio:



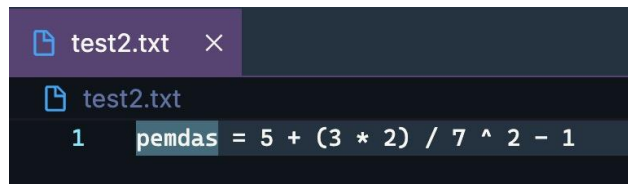
Nota: en caso de que se quisiera aumentar el blur se podría hacer de dos formas, incrementar las cifras de size o aplicar varios flujos de blur de la siguiente forma:

```
img2 = img -> blur(size) -> blur(size) -> blur(size)
```

6. Cada una de las nuevas características implementadas

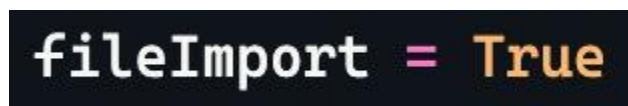
a. Aceptar archivos y ejecutar el contenido

Contamos con el archivo de “test2.txt” que en este caso está compuesto de una sola línea



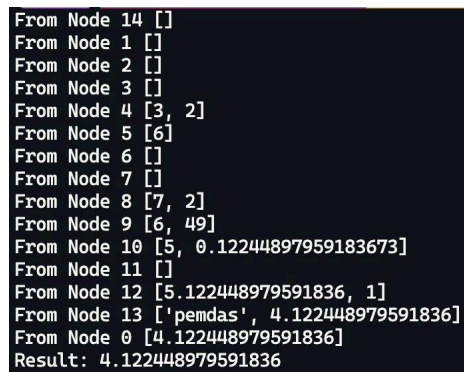
```
test2.txt ×
test2.txt
1 pemdas = 5 + (3 * 2) / 7 ^ 2 - 1
```

En el archivo de “translator.py” fijamos la variable de fileImport a True para que importe el archivo

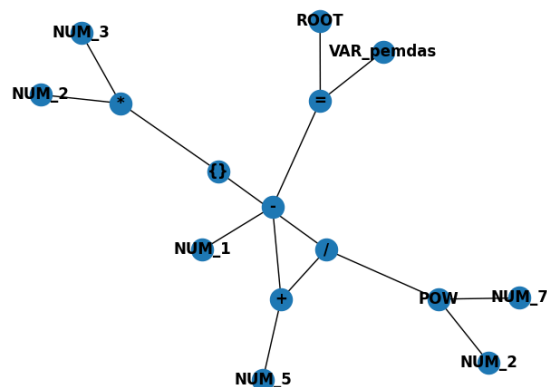


```
fileImport = True
```

Corremos el código y observamos que el archivo se ejecuta correctamente:



```
From Node 14 []
From Node 1 []
From Node 2 []
From Node 3 []
From Node 4 [3, 2]
From Node 5 [6]
From Node 6 []
From Node 7 []
From Node 8 [7, 2]
From Node 9 [6, 49]
From Node 10 [5, 0.12244897959183673]
From Node 11 []
From Node 12 [5.122448979591836, 1]
From Node 13 ['pemdas', 4.122448979591836]
From Node 0 [4.122448979591836]
Result: 4.122448979591836
```



- b. Todas las tareas entregadas (independientemente de la calificación si existiese alguna)



Conjuntos - notacion

4 de abr 10 pts



Creacion de lenguajes

5 de abr 10 pts



Lenguajes específicos

8 de abr 10 pts



Expresiones regulares

8 de abr 10 pts



Automatas

11 de abr 10 pts



Mi parser multiplica y divide

12 de abr 10 pts



Gramáticas libres de contexto

15 de abr 10 pts



Top Down Analyzer

18 de abr 10 pts



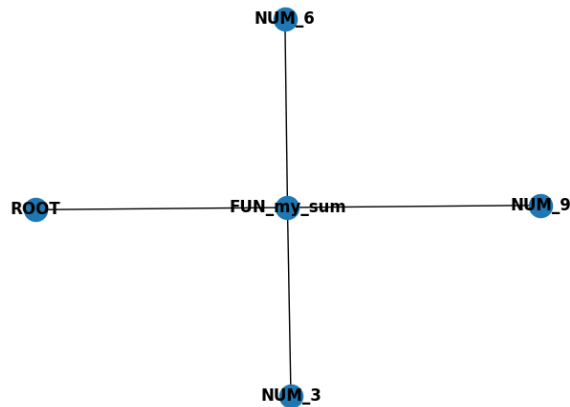
Bottom Up Analyzer

23 de abr 10 pts

- c. Aceptar cualquier función de numpy para manejo de matrices como np.where, np.mean, np.std. Al menos 9 de ellas.

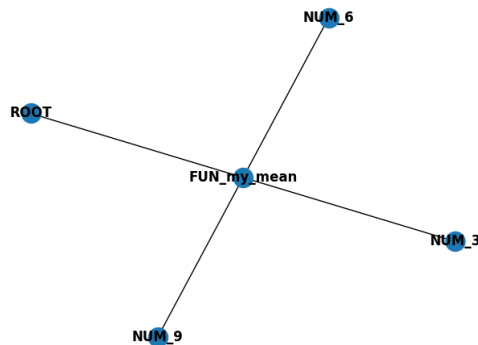
Corremos la función de sumatoria con 3 elementos de prueba 3,6 y 9 y obtenemos el resultado correcto que es 18:

```
input>> my_sum(3,6,9)
[{'type': 'NUMBER', 'label': 'NUM_3', 'value': 3, 'counter': 1}, {'type': 'NUMBER', 'label': 'NUM_6', 'value': 6, 'counter': 2}, {'type': 'NUMBER', 'label': 'NUM_9', 'value': 9, 'counter': 3}]
From Node 1 []
From Node 2 []
From Node 3 []
From Node 4 [3, 6, 9]
**** HERE Function call: my_sum ****
From Node 0 [18]
```



Corremos la función de media con 3 elementos de prueba 3,6 y 9 y obtenemos el resultado correcto que es 6:

```
input>> my_mean(3,6,9)
[{'type': 'NUMBER', 'label': 'NUM_3', 'value': 3, 'counter': 1}, {'type': 'NUMBER', 'label': 'NUM_6', 'value': 6, 'counter': 2}, {'type': 'NUMBER', 'label': 'NUM_9', 'value': 9, 'counter': 3}]
From Node 1 []
From Node 2 []
From Node 3 []
From Node 4 [3, 6, 9]
**** HERE Function call: my_mean ****
From Node 0 [6.0]
```

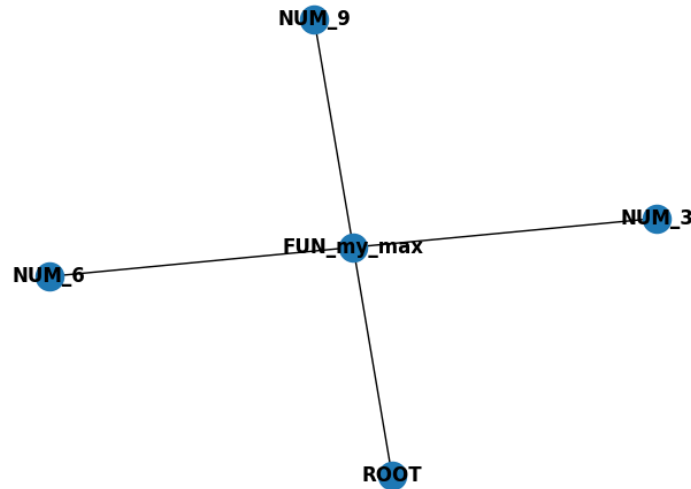


Corremos la función de max con 3 elementos de prueba 3,6 y 9 y obtenemos el resultado correcto que es 9:

```

input>> my_max(3,6,9)
[{'type': 'NUMBER', 'label': 'NUM_3', 'value': 3, 'counter': 1}, {'type': 'NUMBER', 'label': 'NUM_6', 'value': 6, 'counter': 2}, {'t
ype': 'NUMBER', 'label': 'NUM_9', 'value': 9, 'counter': 3}]
From Node 1 []
From Node 2 []
From Node 3 []
From Node 4 [3, 6, 9]
**** HERE Function call: my_max ****
From Node 0 [9]

```

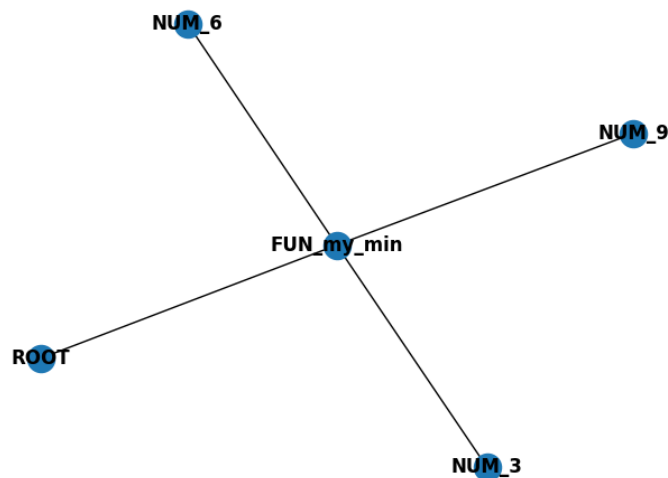


Corremos la función de min con 3 elementos de prueba 3,6 y 9 y obtenemos el resultado correcto que es 3:

```

input>> my_min(3,6,9)
[{'type': 'NUMBER', 'label': 'NUM_3', 'value': 3, 'counter': 1}, {'type': 'NUMBER', 'label': 'NUM_6', 'value': 6, 'counter': 2}, {'t
ype': 'NUMBER', 'label': 'NUM_9', 'value': 9, 'counter': 3}]
From Node 1 []
From Node 2 []
From Node 3 []
From Node 4 [3, 6, 9]
**** HERE Function call: my_min ****
From Node 0 [3]

```

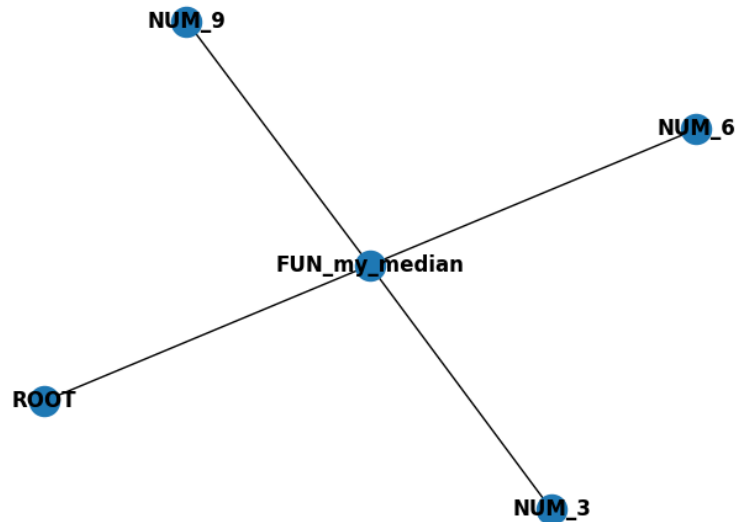


Corremos la función de mediana con 3 elementos de prueba 3,6 y 9 y obtenemos el resultado correcto que es 6:

```

input>> my_median(3,6,9)
[{'type': 'NUMBER', 'label': 'NUM_3', 'value': 3, 'counter': 1}, {'type': 'NUMBER', 'label': 'NUM_6', 'value': 6, 'counter': 2}, {'t
ype': 'NUMBER', 'label': 'NUM_9', 'value': 9, 'counter': 3}]
From Node 1 []
From Node 2 []
From Node 3 []
From Node 4 [3, 6, 9]
**** HERE Function call: my_median ****
From Node 0 [6.0]

```

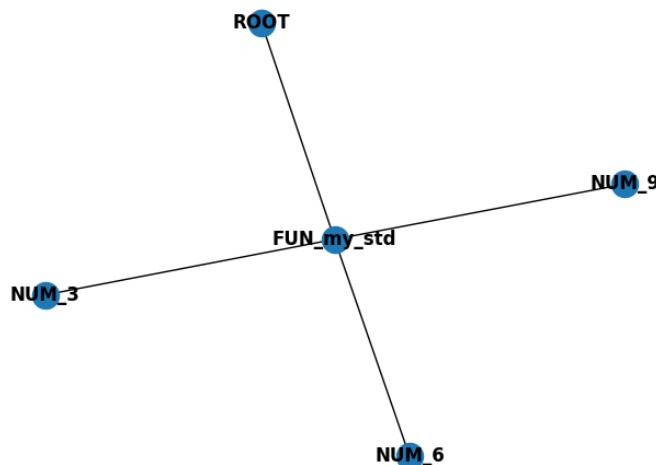


Corremos la función de desviación estándar poblacional con 3 elementos de prueba 3,6 y 9 y obtenemos el resultado correcto que es 2.44949:

```

input>> my_std(3,6,9)
[{'type': 'NUMBER', 'label': 'NUM_3', 'value': 3, 'counter': 1}, {'type': 'NUMBER', 'label': 'NUM_6', 'value': 6, 'counter': 2}, {'t
ype': 'NUMBER', 'label': 'NUM_9', 'value': 9, 'counter': 3}]
From Node 1 []
From Node 2 []
From Node 3 []
From Node 4 [3, 6, 9]
**** HERE Function call: my_std ****
From Node 0 [2.449489742783178]

```

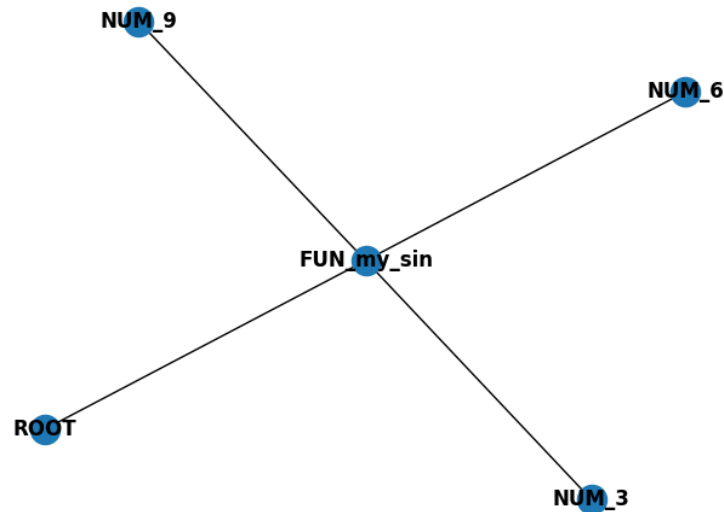


Corremos la función de seno con 3 elementos de prueba 3,6 y 9 y obtenemos el resultado correcto que es 0.1411, -0.2794 y 0.4121:

```

input>> my_sin(3,6,9)
[{'type': 'NUMBER', 'label': 'NUM_3', 'value': 3, 'counter': 1}, {'type': 'NUMBER', 'label': 'NUM_6', 'value': 6, 'counter': 2}, {'t
ype': 'NUMBER', 'label': 'NUM_9', 'value': 9, 'counter': 3}]
From Node 1 []
From Node 2 []
From Node 3 []
From Node 4 [3, 6, 9]
**** HERE Function call: my_sin ****
From Node 0 [array([ 0.14112001, -0.2794155 ,  0.41211849])]

```

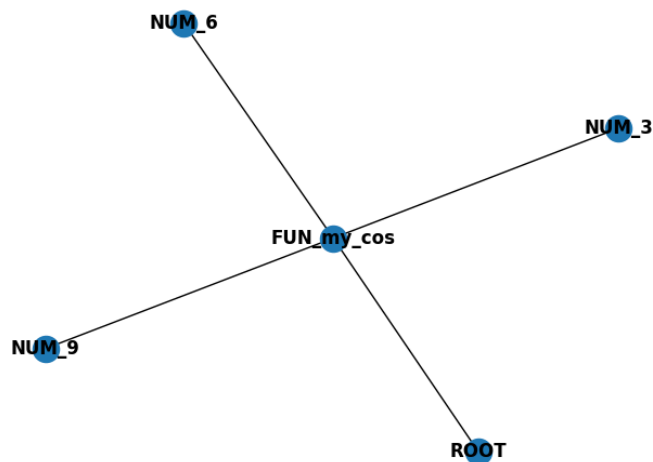


Corremos la función de coseno con 3 elementos de prueba 3,6 y 9 y obtenemos el resultado correcto que es -0.9899, -0.9601 y 0.9111:

```

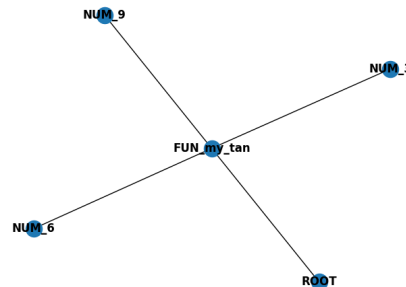
input>> my_cos(3,6,9)
[{'type': 'NUMBER', 'label': 'NUM_3', 'value': 3, 'counter': 1}, {'type': 'NUMBER', 'label': 'NUM_6', 'value': 6, 'counter': 2}, {'t
ype': 'NUMBER', 'label': 'NUM_9', 'value': 9, 'counter': 3}]
From Node 1 []
From Node 2 []
From Node 3 []
From Node 4 [3, 6, 9]
**** HERE Function call: my_cos ****
From Node 0 [array([-0.9899925 ,  0.96017029, -0.91113026])]

```



Corremos la función de tangente con 3 elementos de prueba 3,6 y 9 y obtenemos el resultado correcto que es -0.1425, -0.2910 y -0.4523:


```
input>> my_tan(3,6,9)
[{'type': 'NUMBER', 'label': 'NUM_3', 'value': 3, 'counter': 1}, {'type': 'NUMBER', 'label': 'NUM_6', 'value': 6, 'counter': 2}, {'t
ype': 'NUMBER', 'label': 'NUM_9', 'value': 9, 'counter': 3}]
From Node 1 []
From Node 2 []
From Node 3 []
From Node 4 [3, 6, 9]
**** HERE Function call: my_tan ****
From Node 0 [array([-0.14254654, -0.29100619, -0.45231566])]
```



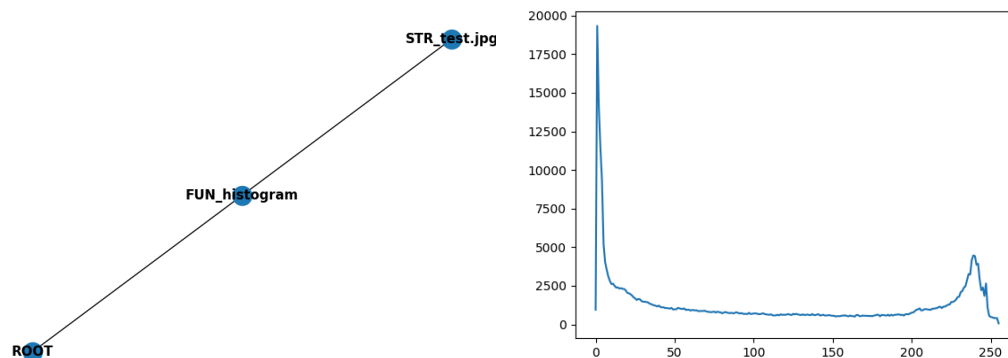
d. Implementación de visualización de histogramas con opencv

Usamos esta imagen para calcular su histograma, que muestra la distribución de intensidades de píxeles de 0 (negro) a 255 (blanco):



```
input>> histogram("test.jpg")
[{'type': 'STRING', 'label': 'STR_test.jpg', 'value': 'test.jpg', 'counter': 1}]
From Node 1 []
From Node 2 ['test.jpg']
**** HERE Function call: histogram ****
From Node 0 [None]
```

Podemos observar el histograma resultante de la distribución de las intensidades de píxeles:



- e. Implementación de un algoritmo complejo como herramienta en el lenguaje:
CannyEdgeDetection

Corremos la función de `canny_edge()` para convertir la imagen a escala de grises y detectar los bordes utilizando el método de CannyEdgeDetection

```
input>> canny_edge("test.jpg")
[{'type': 'STRING', 'label': 'STR_test.jpg', 'value': 'test.jpg', 'counter': 1}]
From Node 1 []
From Node 2 ['test.jpg']
**** HERE Function call: canny_edge ****
From Node 0 [None]
```

Original Image



Edge Image



- f. Implementación de pruebas automatizadas. Uso de un marco de TESTING para probar todas las características de tu gramática

← Run pytest

✓ tests #8 Re-run all jobs ...

Summary

Jobs

- test

Run details

- Usage
- Workflow file

test

succeeded 4 days ago in 18s

Search logs

- > ✓ Set up job 0s
- > ✓ Run actions/checkout@v2 1s
- > ✓ Set up Python 0s
- > ✓ Install dependencies 13s
- > ✓ Run tests 2s
- > ✓ Post Set up Python 0s
- > ✓ Post Run actions/checkout@v2 0s
- > ✓ Complete job 0s

<https://github.com/MartinPaGarcia/Desarrollo-de-aplicaciones-avanzadas/actions/runs/8841733482/job/24279282690>