

**“Año del Bicentenario, de la consolidación de nuestra Independencia,  
y de la conmemoración de las heroicas batallas de Junín y Ayacucho”**



**CARRERA: INGENIERÍA DE SOFTWARE  
PROYECTO FINAL  
VIRTUALIZACIÓN DE SISTEMAS OPERATIVOS Y LA NUBE**

**ESTUDIANTES:**

**Carpio Guevara Rorigo Sebastian  
Flores Leon Miguel Angel**

**DOCENTE:**

**Luque Mamani Edson Fracisco**

**Arequipa, Perú  
30 de noviembre de 2024**

# Índice

<b>I</b>	<b>Introducción</b>	<b>1</b>
<b>II</b>	<b>Uso de polars</b>	<b>2</b>
<b>III</b>	<b>Creación del grafo</b>	<b>2</b>
<b>IV</b>	<b>Explicación del Código</b>	<b>2</b>
	IV-A Código . . . . .	2
<b>V</b>	<b>Funcionamiento detallado del código</b>	<b>3</b>

## I. INTRODUCCIÓN

En este informe se presenta el desarrollo del proyecto final, el cual tiene el objetivo de analizar y visualizar la estructura de un grafo para una red social, esto con el proposito de descubrir patrones de interez y obtener informacion sobre la conectividad social, las estrcuturas de la comunidad y propiedades de la red.

Esta red social es conformada por dos bases de datos, una formada por diez millones de usuarios y la segunda esta formada por las ubicaciones de estos usuraiois, Este conjunto de datos es un subconjunto de todos los usuarios de la red social, junto con sus conexiones y ubicaciones el grafo que crea el programa consiera a los usuarios como nodos y sus conexiones como aristas, por lo que forman un componente conectado del grafo total.

Para el análisis de los datos optamos por utilizar la libreria "polars."esto debido a la velocidad con la que trabaja y la eficiencia en el uso de los recursos, factores en los que supera a otras librerias como "pandas".

La construcción del grafo se logra cargandolo primero y luego almacenandolo en formato csv, luego el programa correra pruebas en el grafo y en los datos.

El de este programa es hacer un grafo realista y funcional de la red social con el objetivo de analizar a las distintas comunidades que esta posee y comprender como se relacionan, ademas de eso se evalua el uso de algoritmos para la carga y lectura e datos con el proposito de encontrar la opcion mas eficiente".

## II. USO DE POLARS

La librería polars está escrita en Rust, lo que lo hace más eficiente a la hora de procesar grandes volúmenes de datos a comparación de las librerías basadas en NumPy, como caso de ejemplo si se ejecuta un groupby + sum() pandas tardaría un promedio de 31 segundos, mientras que polars tardaría un promedio de 4.5 segundos lo que representa una diferencia de tiempo de un 85 por ciento con respecto a pandas, esto debido a que polars ejecuta las operaciones en paralelo de manera automática, dividiéndolas entre los núcleos del procesador.

Por lo tanto el uso utiliza esta librería es la mejor opción para la carga y el tratamiento de los grandes volúmenes de datos requeridos.

## III. CREACIÓN DEL GRAFO

El grafo se carga con los datos manejados con polars y la encargada de crear el grafo es la librería Networkx, el programa carga las localizaciones en forma de latitud y longitud, luego se cargan los usuarios y se crean las aristas para finalizar con la medición de tiempo, esto con el objetivo de saber cuánto tiempo le toma al programa crear el grafo, el uso de try es para hacer búsqueda de excepciones dentro de la base de datos.

## IV. EXPLICACIÓN DEL CÓDIGO

### IV-A. Código

```

1 import igraph as ig
2 import time
3 import logging
4 import polars as pl
5 import pickle
6 import numpy as np
7
8 logging.basicConfig(level=logging.INFO,
9                     format='%(asctime)s - %(levelname)s
10                      - %(message)s')
11
12 def crear_grafo_igraph(ubicaciones_path,
13                       usuarios_path, block_size=4_000_000)
14     :
15     start_time = time.time()
16     logging.info("Cargando ubicaciones...")
17
18     ubicaciones = np.loadtxt(
19         ubicaciones_path, delimiter=',')
20     num_nodos = ubicaciones.shape[0]
21     ubicaciones = [tuple(row) for row in
22                     ubicaciones]
23     logging.info(f"Se cargaron {num_nodos}
24                 ubicaciones.")
25
26     logging.info("Cargando usuarios y
27                 creando aristas (modo robusto y
28                 rápido)...")
29     edges = []
30     with open(usuarios_path, 'r', encoding=
31              'utf-8', errors='ignore') as
32         file:
33         block = []
34         for line_number, line in enumerate(
35             file, start=1):

```

```

24         src = line_number - 1
25         # Salta líneas vacías o con
26         # solo espacios
27         if not line.strip():
28             continue
29         # Solo procesa si el nodo
30         # fuente es válido
31         if not (0 <= src < num_nodos):
32             continue
33         conexiones = set()
34         for x in line.split(','):
35             x = x.strip()
36             if not x.isdigit():
37                 continue
38             dst = int(x) - 1
39             # Solo agrega si el destino
40             # es válido y no es
41             # un self-loop
42             if 0 <= dst < num_nodos and
43                 dst != src:
44                 conexiones.add(dst)
45             block.extend((src, dst) for dst
46                         in conexiones)
47             if (line_number) % block_size
48                 == 0:
49                 edges.extend(block)
50                 block = []
51         if block:
52             edges.extend(block)
53     logging.info(f"Se procesaron {len(edges)}
54                 aristas.")
55
56     g = ig.Graph(directed=True)
57     g.add_vertices(num_nodos)
58     g.add_edges(edges)
59     g.vs["location"] = ubicaciones
60
61     return g
62
63 def generar_tabla_grafos_ordenados(grafo,
64                                     num_nodos=50):
65     """
66     Genera una tabla con información de los
67     primeros y últimos nodos del
68     grafo.
69
70     """
71     nodos = list(range(grafo.vcount()))
72     primeros_nodos = nodos[:num_nodos]
73     ultimos_nodos = nodos[-num_nodos:]
74     nodos_seleccionados = primeros_nodos +
75                             ultimos_nodos
76
77     data = []
78     for idx, nodo in enumerate(
79         nodos_seleccionados, start=1):
80         loc = grafo.vs[nodo]["location"]
81         if loc is not None:
82             lat, lon = loc
83         else:
84             lat, lon = None, None
85         conexiones = grafo.successors(nodo)
86         num_conexiones = len(conexiones)
87         print(f"Nodo {nodo+1}: Conexiones
88               {len(conexiones)}")
89         # Mostrar base 1
90         data.append({
91             'Index': idx,
92             'Nodo': nodo + 1, # Mostrar
93             # base 1
94             'Latitud': lat,
95             'Longitud': lon,
96             'Conexiones': num_conexiones
97         })
98     tabla = pl.DataFrame(data)
99     return tabla

```

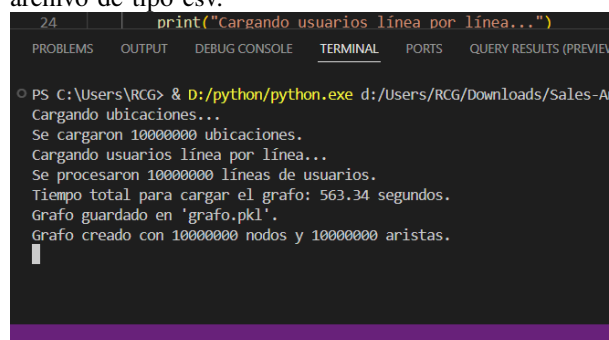
```

85 def realizar_eda(grafo):
86     try:
87         num_nodos = grafo.vcount()
88         num_aristas = grafo.ecount()
89         grados = grafo.degree()
90         max_grado = max(grados)
91         min_grado = min(grados)
92         promedio_grado = sum(grados) /
93             ↪ num_nodos
94
95         logging.info(f"Número de nodos: {
96             ↪ num_nodos}")
97         logging.info(f"Número de aristas: {
98             ↪ num_aristas}")
99         logging.info(f"Grado máximo: {
100             ↪ max_grado}")
101         logging.info(f"Grado mínimo: {
102             ↪ min_grado}")
103         logging.info(f"Grado promedio: {
104             ↪ promedio_grado:.2f}")
105     except Exception as e:
106         logging.error(f"Error durante el
107             ↪ EDA: {e}")
108
109 # Define las rutas de los archivos de
110     ↪ entrada y salida
111 ubicaciones_archivo = '10_million_location.
112     ↪ txt'
113 usuarios_archivo = '10_million_user.txt'
114
115 # Crear el grafo
116 grafo = crear_grafo_igraph(
117     ↪ ubicaciones_archivo,
118     ↪ usuarios_archivo)
119
120 # Cargar el grafo desde el archivo guardado
121 if grafo:
122     realizar_eda(grafo)
123     tabla_grafos =
124         ↪ generar_tabla_grafos_ordenados(
125         ↪ grafo, num_nodos=50)
126     print(tabla_grafos)
127     tabla_grafos.write_csv('tabla_grafos.
128         ↪ csv')
129     print("Tabla guardada en 'tabla_grafos.
130         ↪ csv'.")

```

Listing 1. Código para lectura de datos y carga del grafo

Este código es la carga de datos y la generación del grafo dentro del programa como archivo de tipo csv.



```

24 print("Cargando usuarios línea por línea...")
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS (PREVIEW)
PS C:\Users\RCG> & D:/python/python.exe d:/Users/RCG/Downloads/Sales-A-
Cargando ubicaciones...
Se cargaron 10000000 ubicaciones.
Cargando usuarios línea por línea...
Se procesaron 10000000 líneas de usuarios.
Tiempo total para cargar el grafo: 563.34 segundos.
Grafo guardado en 'grafo.pkl'.
Grafo creado con 10000000 nodos y 10000000 aristas.

```

Imagen 1, prueba del funcionamiento del grafo

## V. FUNCIONAMIENTO DETALLADO DEL CODIGO

Primero se importan las librerías, se usa `igraph` para trabajar con los grafos, `loggin` para registrar la información de la ejecución, `polars` para el tratamiento de grandes volúmenes de datos y

`time` que es el módulo para medir el tiempo, luego se inicia el contador, después se lee el archivo de ubicaciones y se renombran las columnas a `lat` y `log`, el programa recorre el archivo de ubicaciones cada fila se convierte en un nodo con el atributo `"location"` para cargar el grafo, se abre el archivo de usuarios y se lee línea por línea ajustando el valor, se añade una arista desde el nodo `"line number +1"` a cada nodo `j`, luego se muestra el tiempo total de ejecución y se almacena el grafo como un archivo csv.

Luego se abre el archivo csv y se carga.