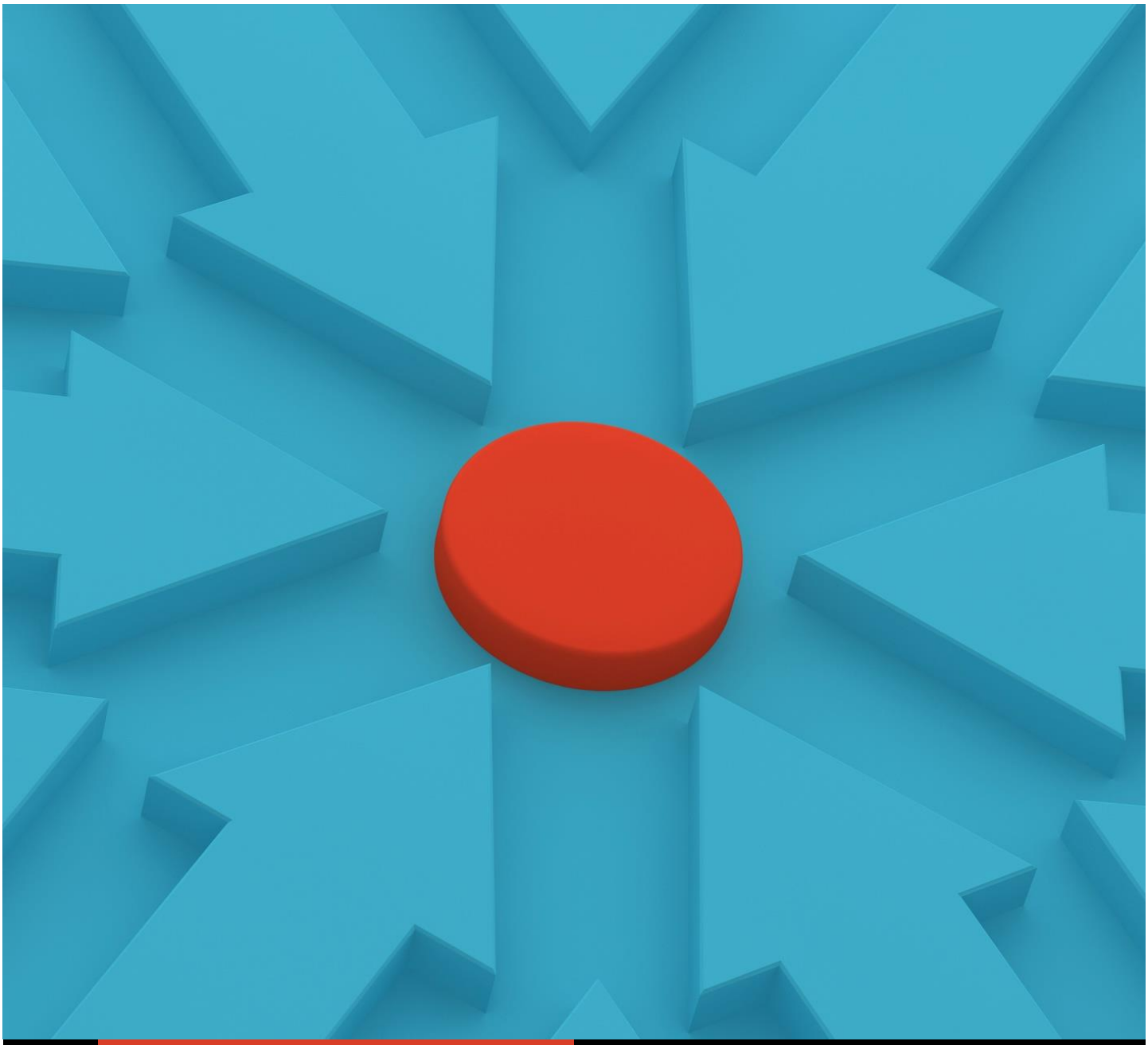
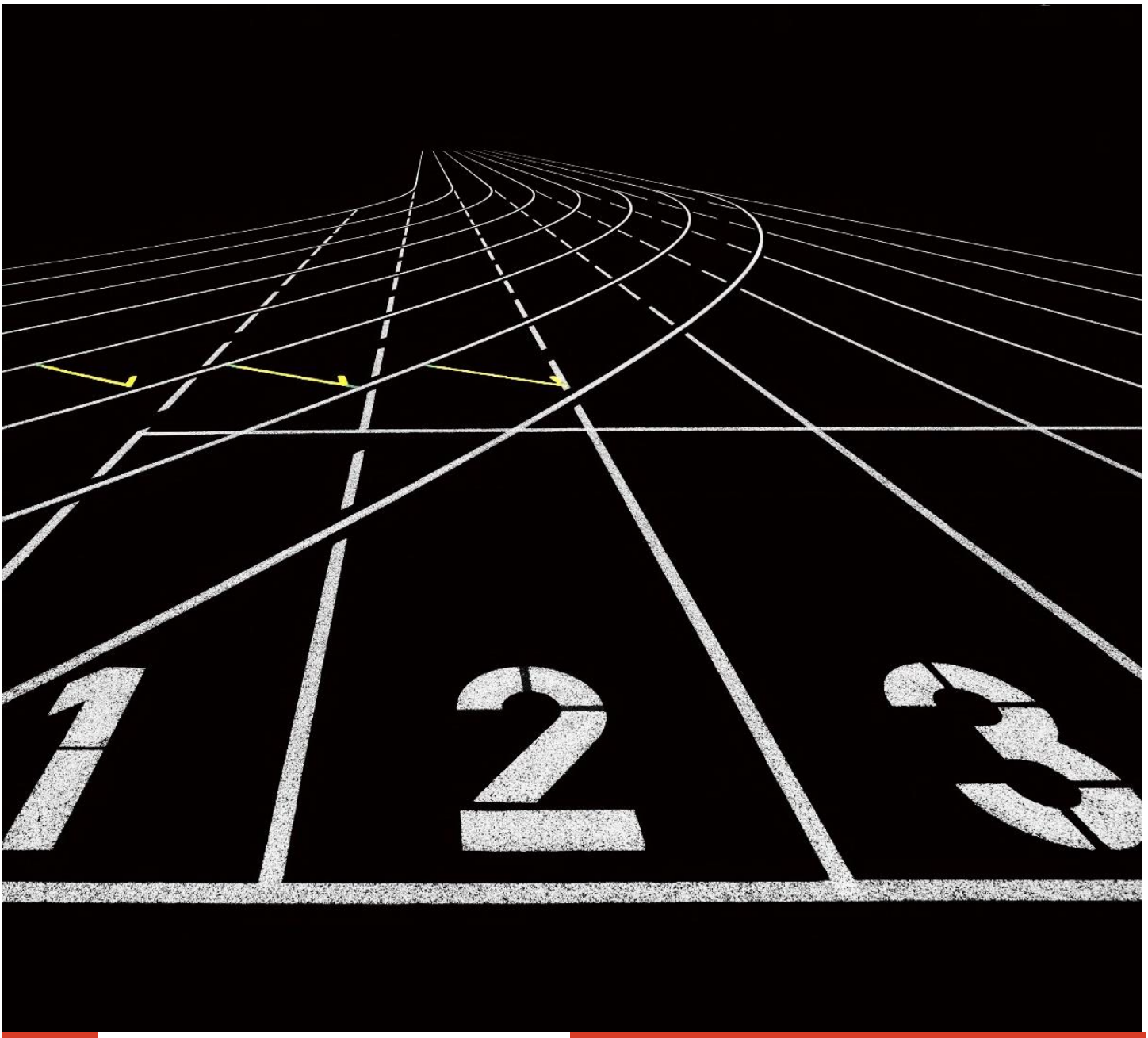


# MANUAL TÉCNICO

PROYECTO 1




# INTRODUCCION

La finalidad de todo manual técnico es la de proporcionar al lector las pautas de configuración y la lógica con la que se ha desarrollado una aplicación, la cual se sabe que es propia de cada programador; por lo que se considera necesario ser documentada.

# TEORIA DE GRAFOS

## Importaciones

Cabe resaltar que el import permite agregar a nuestro proyecto una o varias clases (paquete) según lo necesitemos. Para comprender y usar correctamente el import de Java, retomaremos los ejemplos dados en la sección de paquetes.

En el Proyecto se utilizaron varios tipos de importaciones como

**Tkinter:** interfaces graficas.

**Networkxx**: manipulación y visualización de los grafos.

**Matplotlib**: para poder trazar los caminos de los grafos.

```
import tkinter as tk
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
```

.

## Clases utilizadas

Las clases son fundamentales en la POO ya que permiten crear objetos que tienen atributos y métodos específicos, lo que hace que nuestro código sea más modular y fácil de mantener. Además, podemos crear múltiples objetos a partir de la misma clase, lo que nos permite reutilizar nuestro código y ahorrar tiempo.

**Clase Main:** nuestra pantalla y nuestro bucle principales

```
def main():
    root = tk.Tk()
    app = GraphEditor(root)
    root.mainloop()


if __name__ == "__main__":
    main()
```

**Clase GraphEditor:** aspecto gráfico y búsqueda de nuestro editor de grafos

```
class GraphEditor:
    def __init__(self, master):
        self.master = master
        self.master.title("Editor de Grafos")

        # Inicializar un grafo vacío
        self.graph = nx.Graph()

        # Crear lienzo para visualización del grafo original
```

## Etiquetas y botones

En Java, las etiquetas y los botones son componentes de la interfaz gráfica de usuario (GUI) que se utilizan comúnmente en la programación de aplicaciones.

### Inicializar un grafo vacío

```
# Inicializar un grafo vacío
self.graph = nx.Graph()
```

### Crear lienzo para visualización del grafo original

```
# Crear lienzo para visualización del grafo original
self.fig_original, self.ax_original = plt.subplots(figsize=(4, 4), facecolor='green')
self.ax_original.axis('off')  # Quitar los ejes del gráfico
self.canvas_original = FigureCanvasTkAgg(self.fig_original, master=self.master)
self.canvas_original.get_tk_widget().pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
```

## Crear lienzo para visualización del grafo resultado de la búsqueda en profundidad

```
# Crear lienzo para visualización del grafo resultado de la búsqueda en profundidad
self.fig_dfs, self.ax_dfs = plt.subplots(figsize=(4, 4), facecolor='green')
self.ax_dfs.axis('off')  # Quitar los ejes del gráfico
self.canvas_dfs = FigureCanvasTkAgg(self.fig_dfs, master=self.master)
self.canvas_dfs.get_tk_widget().pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)
```

## Botones

```
# Botones
self.add_node_button = tk.Button(master, text="Agregar Nodo", command=self.add_node, width=15)
self.add_node_button.pack(side=tk.BOTTOM)
self.add_edge_button = tk.Button(master, text="Agregar Arista", command=self.add_edge, width=15)
self.add_edge_button.pack(side=tk.BOTTOM)
self.clear_button = tk.Button(master, text="Limpiar", command=self.clear_graph, width=15)
self.clear_button.pack(side=tk.BOTTOM)
self.draw_graph_button = tk.Button(master, text="Dibujar Grafo", command=self.draw_graph, width=15)
self.draw_graph_button.pack(side=tk.BOTTOM)
self.dfs_button = tk.Button(master, text="Búsqueda por Profundidad (DFS)", command=self.dfs_traversal, width=20)
self.dfs_button.pack(side=tk.BOTTOM)
self.bfs_button = tk.Button(master, text="Búsqueda por Amplitud (BFS)", command=self.bfs_traversal, width=20)
self.bfs_button.pack(side=tk.BOTTOM)  # Botón para BFS
```

## Etiquetas y campos de entrada

```
# Etiquetas y campos de entrada
self.node_label = tk.Label(master, text="Nodo:")
self.node_label.pack(side=tk.BOTTOM)
self.node_entry = tk.Entry(master)
self.node_entry.pack(side=tk.BOTTOM)
```

## Campo de entrada para agregar aristas

```
# Campo de entrada para agregar aristas
self.edge_label = tk.Label(master, text="Arista:")
self.edge_label.pack(side=tk.BOTTOM)
self.edge_entry = tk.Entry(master)
self.edge_entry.pack(side=tk.BOTTOM)
```

## Campo de entrada para el nodo de inicio del DFS

```
# Campo de entrada para el nodo de inicio del DFS
self.start_node_label = tk.Label(master, text="Nodo de inicio para DFS:")
self.start_node_label.pack(side=tk.BOTTOM)
self.start_node_entry = tk.Entry(master)
self.start_node_entry.pack(side=tk.BOTTOM)
```

## Campo de entrada para el nodo de inicio del BFS

```
# Campo de entrada para el nodo de inicio del BFS
self.bfs_start_node_label = tk.Label(master, text="Nodo de inicio para BFS:")
self.bfs_start_node_label.pack(side=tk.BOTTOM)
self.bfs_start_node_entry = tk.Entry(master)
self.bfs_start_node_entry.pack(side=tk.BOTTOM)
```

# METODOS UTILIZADOS

Los métodos son las acciones que los objetos pueden realizar.

```
def add_node(self):
    # Método para agregar un nodo al grafo
    node = self.node_entry.get()
    if node:
        self.graph.add_node(node)
        self.node_entry.delete(0, tk.END)
```

```
def add_edge(self):
    # Método para agregar una arista al grafo
    edge = self.edge_entry.get()
    if edge:
        node1, node2 = edge.split()
        self.graph.add_edge(node1, node2)
        self.edge_entry.delete(0, tk.END)
```

```
def clear_graph(self):
    # Método para limpiar el grafo y restablecer los lienzos
    self.graph.clear()
    self.ax_original.clear()
    self.ax_original.set_facecolor('green')  # Restablecer el color de fondo verde
    self.ax_original.axis('off')  # Quitar los ejes del gráfico
    self.ax_dfs.clear()
    self.ax_dfs.set_facecolor('green')  # Restablecer el color de fondo verde
    self.ax_dfs.axis('off')  # Quitar los ejes del gráfico
    self.canvas_original.draw()
    self.canvas_dfs.draw()
```

```python
def draw_graph(self):
    # Método para dibujar el grafo en el lienzo original
    if self.graph.number_of_nodes() == 0:
        return

    self.ax_original.clear()
    self.ax_original.set_facecolor('green')  # Restablecer el color de fondo verde
    self.ax_original.axis('off')  # Quitar los ejes del gráfico
    nx.draw(self.graph, with_labels=True, ax=self.ax_original)
    self.canvas_original.draw()
```

```python
def dfs_traversal(self):
    # Método para realizar una búsqueda en profundidad en el grafo y dibujar el resultado
    if self.graph.number_of_nodes() == 0:
        return

    start_node = self.start_node_entry.get()  # Obtener el nodo de inicio del DFS desde la entrada de texto
    if not start_node:
        return

    visited_edges = set()
    for u, v in nx.dfs_edges(self.graph, source=start_node):
        visited_edges.add((u, v))

    # Limpiar el lienzo del grafo resultante de la búsqueda en profundidad
    self.ax_dfs.clear()
    self.ax_dfs.set_facecolor('green')  # Restablecer el color de fondo verde
    self.ax_dfs.axis('off')  # Quitar los ejes del gráfico

    # Dibujar solo las aristas visitadas en rojo y las no visitadas con un grosor menor
    for u, v in self.graph.edges():
        if (u, v) in visited_edges or (v, u) in visited_edges:
            nx.draw_networkx_edges(self.graph, pos=nx.circular_layout(self.graph), edgelist=[(u, v)], ax=self.ax_dfs, edge_color='r', width=2.0)
        else:
            nx.draw_networkx_edges(self.graph, pos=nx.circular_layout(self.graph), edgelist=[(u, v)], ax=self.ax_dfs, edge_color='k', width=0.5)

    # Dibujar los nodos
    nx.draw(self.graph, pos=nx.circular_layout(self.graph), with_labels=True, ax=self.ax_dfs, node_color='lightblue')

    # Actualizar lienzo del grafo resultante de la búsqueda en profundidad
    self.canvas_dfs.draw()
```

```python
def bfs_traversal(self):
    # Método para realizar una búsqueda en amplitud en el grafo y dibujar el resultado
    if self.graph.number_of_nodes() == 0:
        return

    start_node = self.bfs_start_node_entry.get()  # Obtener el nodo de inicio del BFS desde la entrada de texto
    if not start_node:
        return

    visited_edges = set()
    for u, v in nx.bfs_edges(self.graph, source=start_node):
        visited_edges.add((u, v))

    # Limpiar el lienzo del grafo resultante de la búsqueda en amplitud
    self.ax_dfs.clear()
    self.ax_dfs.set_facecolor('green')  # Restablecer el color de fondo verde
    self.ax_dfs.axis('off')  # Quitar los ejes del gráfico

    # Dibujar solo las aristas visitadas en rojo y las no visitadas con un grosor menor
    for u, v in self.graph.edges():
        if (u, v) in visited_edges or (v, u) in visited_edges:
            nx.draw_networkx_edges(self.graph, pos=nx.circular_layout(self.graph), edgelist=[(u, v)], ax=self.ax_dfs, edge_color='r', width=2.0)
        else:
            nx.draw_networkx_edges(self.graph, pos=nx.circular_layout(self.graph), edgelist=[(u, v)], ax=self.ax_dfs, edge_color='k', width=0.5)

    # Dibujar los nodos
    nx.draw(self.graph, pos=nx.circular_layout(self.graph), with_labels=True, ax=self.ax_dfs, node_color='lightblue')

    # Actualizar lienzo del grafo resultante de la búsqueda en amplitud
    self.canvas_dfs.draw()
```

# VIDEO EXPLICATIVO

https://youtu.be/_8S8DsK3NP0