Building a Graph Visualization Interface with Tkinter: A Step-by-Step Guide

Contents

1 Introduction

In this guide, we will build a Python application using Tkinter to create a graphical interface for constructing graphs and visualizing graph traversal algorithms like Depth-First Search (DFS), Recursive DFS, and Breadth-First Search (BFS). We will start by setting up the general interface, progressively adding functionality and complexity, and explain each step in detail.

2 Objectives

Our main objectives are:

- Create a resizable GUI interface with a canvas on the left and controls on the right.
- Implement functionality to draw nodes and edges on the canvas.
- Allow users to move nodes and update connected edges accordingly.
- Integrate graph traversal algorithms and visualize them on the canvas.

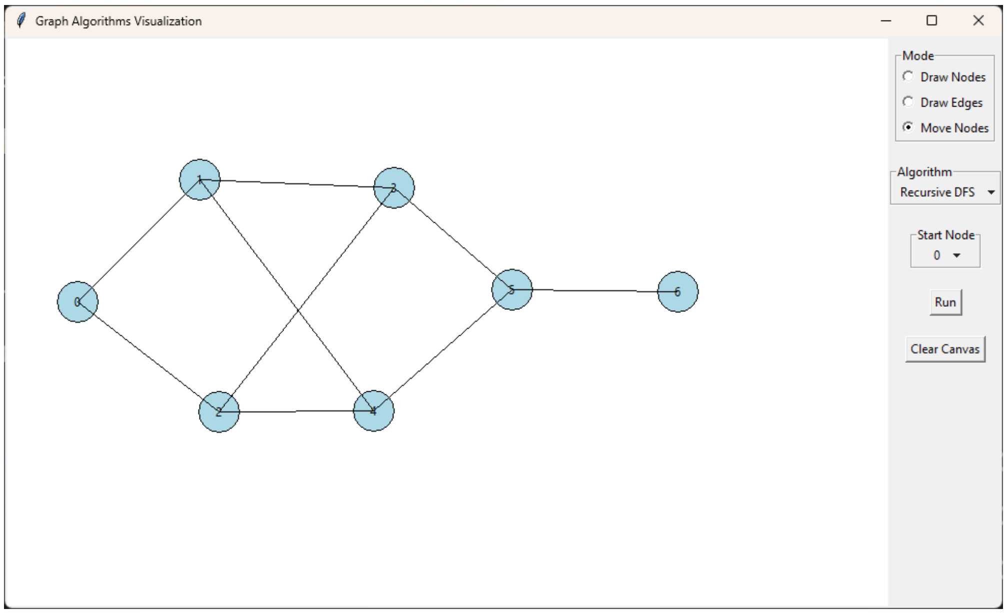

Figure 1: Final interface

3 Setting Up the General Interface

We begin by creating the general structure of the interface, which includes a canvas for drawing and a control panel for user interactions.

3.1 Importing Necessary Modules

First, import the necessary modules:

```python
import tkinter as tk
from tkinter import ttk
```

3.2 Initializing the Main Application Window

We create the main application window and set its title.

```python
root = tk.Tk()
root.title("Graph␣Visualization␣Tool")
root.mainloop()
```

3.3 Creating the Main Frames

To organize the layout, we use frames. We'll have a main frame that contains two sub-frames: one for the canvas and one for the controls.

More information about frames: Tkinter frame.

```python
# Create main frame
main_frame = tk.Frame(root)
main_frame.pack(fill=tk.BOTH, expand=True)
```

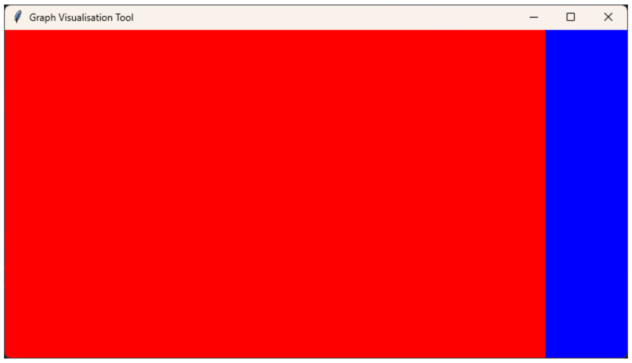

Figure 2: General Structure

3.4 Adding the Canvas and Control Panel

We add a canvas on the left and a control panel on the right.

```
# Create canvas frame
canvas_frame = tk.Frame(main_frame)
canvas_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

# Create control frame
control_frame = tk.Frame(main_frame)
control_frame.pack(side=tk.RIGHT, fill=tk.Y)
```

In order to visualize the structure we can add some dimensions and colors, in Fig. [2], we can see the GUI generated so far.

```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
root.title("Graph␣Visualisation␣Tool")

# Create main frame
main_frame = tk.Frame(root)
main_frame.pack(fill=tk.BOTH, expand=True)

# Create canvas frame
canvas_frame = tk.Frame(main_frame, height=100, width=200, bg='red')
canvas_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

# Create control frame
control_frame = tk.Frame(main_frame, height=100, width=100, bg='blue')
control_frame.pack(side=tk.RIGHT, fill=tk.Y)
root.mainloop()
```

3.5 Creating the Canvas

We create a canvas widget within the canvas frame.

```
canvas = tk.Canvas(canvas_frame, bg="white")
canvas.pack(fill=tk.BOTH, expand=True)
```

3.6 Adding Controls to the Control Panel

3.6.1 Mode Radio List

We can now add buttons, labels, and other widgets to the control panel.

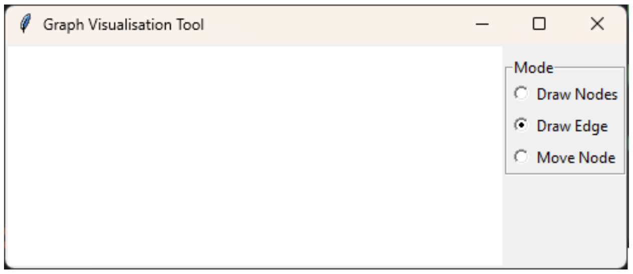

Figure 3: Mode Selector

```
# Example control: Mode selection
mode_frame = tk.LabelFrame(control_frame, text="Mode")
mode_frame.pack(pady=10)

node_var = tk.StringVar()
node_var.set("Draw Edge")

draw_node_radio = tk.Radiobutton(mode_frame, text="Draw Nodes", variable=node_var, value="
    Draw Nodes")
draw_edge_radio = tk.Radiobutton(mode_frame, text="Draw Edge", variable=node_var, value="
    Draw Edge")
move_node_radio = tk.Radiobutton(mode_frame, text="Move Node", variable=node_var, value="
    Move Node")

draw_node_radio.pack(anchor=tk.W)
draw_edge_radio.pack(anchor=tk.W)
move_node_radio.pack(anchor=tk.W)
```

We can shorten the code by directly packing the Radio Butons on their creation.

```
mode_frame = tk.Label(control_frame, text="Mode")
mode_frame.pack(pady=10)

node_var = tk.StringVar()
node_var.set("Draw Edge")

tk.Radiobutton(mode_frame, text="Draw Nodes", variable=node_var, value="Draw Nodes").pack(
    anchor=tk.W)
tk.Radiobutton(mode_frame, text="Draw Edge", variable=node_var, value="Draw Edge").pack(
    anchor=tk.W)
tk.Radiobutton(mode_frame, text="Move Node", variable=node_var, value="Move Node").pack(
    anchor=tk.W)
```

3.6.2 Algorithm Selection Drop List & Start Node Drop List

Algorithm selection

```
algorithms = ["Recursive DFS", "DFS", "BFS"]
selected_alg = tk.StringVar()
alg_menu = ttk.OptionMenu(alg_frame, selected_alg, "BFS", *algorithms)
alg_menu.pack()

start_node_frame = tk.LabelFrame(control_frame, text="Start Node")
start_node_frame.pack(pady=10)
```

Node Selection

```
start_node_frame = tk.LabelFrame(control_frame, text="Start Node")
start_node_frame.pack(pady=10)

start_node = tk.IntVar()
```

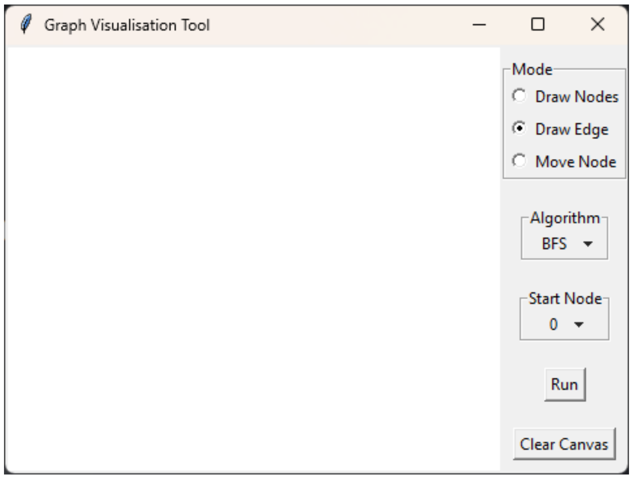

Figure 4: All Controls

```

start_node_menu = ttk.OptionMenu(start_node_frame, start_node)
start_node_menu.pack()
```

3.6.3 Run & Clear Canvas Buttons

It must be mentioned, that the button also support biding to methods via the parameter command. We will see in the future chapters.

```
# Run button
run_button = tk.Button(control_frame, text="Run")
run_button.pack(pady=10)

# Clear Canvas button
clear_button = tk.Button(control_frame, text="Clear Canvas")
clear_button.pack(pady=10)
```

4 Implementing Canvas Interactions

Now that we have the basic interface, we can add functionality to interact with the canvas, such as drawing nodes and edges.

4.1 Rendering a Circle (Node) on the Canvas

We define a function to draw a node at the position where the user clicks.

```
def add_node(event):
    x,y = event.x, event.y
    r = 20
    # id     =          create_oval(x0,    y0,    x1,    y1,    option, ...)
    node_id = canvas.create_oval(x - r, y - r, x + r, y + r, fill="lightblue", outline="
        lightblue")
    canvas.create_text(x, y, text=str(node_id))
```

4.2 Binding Mouse Events to the Canvas

We bind the left mouse button click to the `add_node` function.

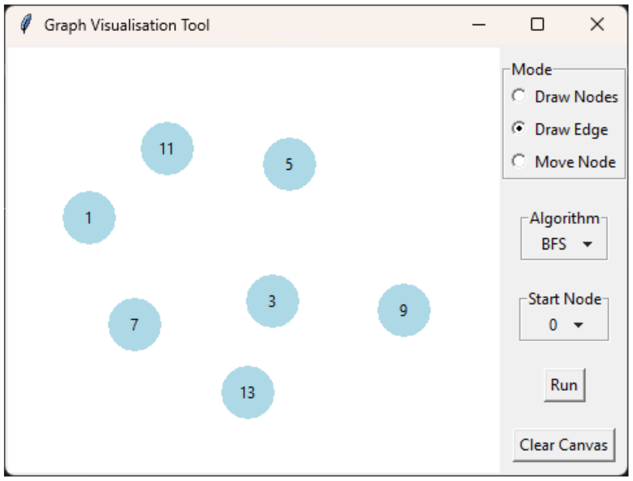

Figure 5: Draw Nodes in Canvas

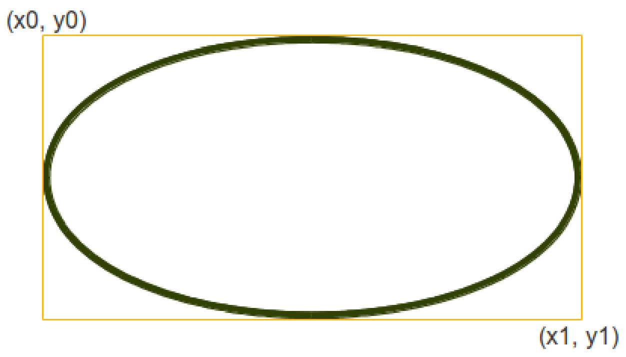

Figure 6: Coordinates on the circle.Source: Python-Course Eu Website. Canvas Widgets in Tkinter.

```
canvas.bind("<Button-1>", add_node)
```

4.3 Implementing Node Movement

We can allow nodes to be moved by clicking and dragging.

```
# Variables to store the selected node
selected_node = None

def select_node(event):
    global selected_node
    # Find the node under the cursor
    items = canvas.find_overlapping(event.x, event.y, event.x, event.y)
    for item in items:
        if canvas.type(item) == "oval":
            selected_node = item
            break

def move_node(event):
    if selected_node:
        x, y = event.x, event.y
        r = 20  # Node radius
        canvas.coords(selected_node, x - r, y - r, x + r, y + r)
        # Update the position of the text label if any

```

```python
def release_node(event):
    global selected_node
    selected_node = None

# Bind the events
canvas.bind("<Button-1>", select_node)
canvas.bind("<B1-Motion>", move_node)
canvas.bind("<ButtonRelease-1>", release_node)
```

5 Enhancing the Interface