

Manual Técnico

Requisitos del sistema:

Python: Un entorno de desarrollo de Python para la realización de dicha aplicación.

IDE Visual Studio Code: Para la realización del código se necesita un ide para poder desarrollar el código.

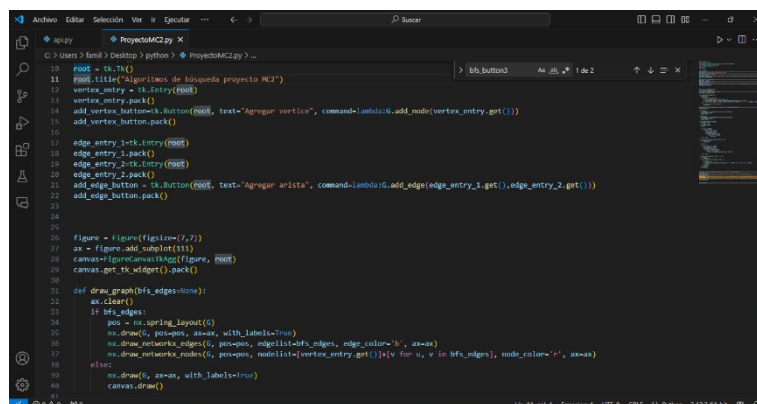
Estructura de código:

En base a los diferentes códigos proporcionados, se realizó la estructura de esta aplicación, en el cual se implementaron dos diferentes código dentro de la aplicación desarrollando a través de múltiples librerías lo que son los diferentes algoritmos de búsqueda a lo ancho y en profundidad.

Iniciamos creando una gráfica vacía para poder realizar los diferentes grafos, y mediante las funciones lambda agregada al botón se iban agregando ya sea un nodo o una arista a la gráfica, además por medio de la función draw_graph se crean los diferentes grafos según lo solicitado.

Se separo en dos métodos los diferentes algoritmos donde se puede ver que en un alista se almacenan los datos y de esta forma se hace la simulación de si se visito o no, y si no estaba se van agregando a dicha lista, y asi se iban almacenando y asi consecutivamente. Con el de profundidad se hizo algo similar con la diferencia que regresaba a ver esos datos.

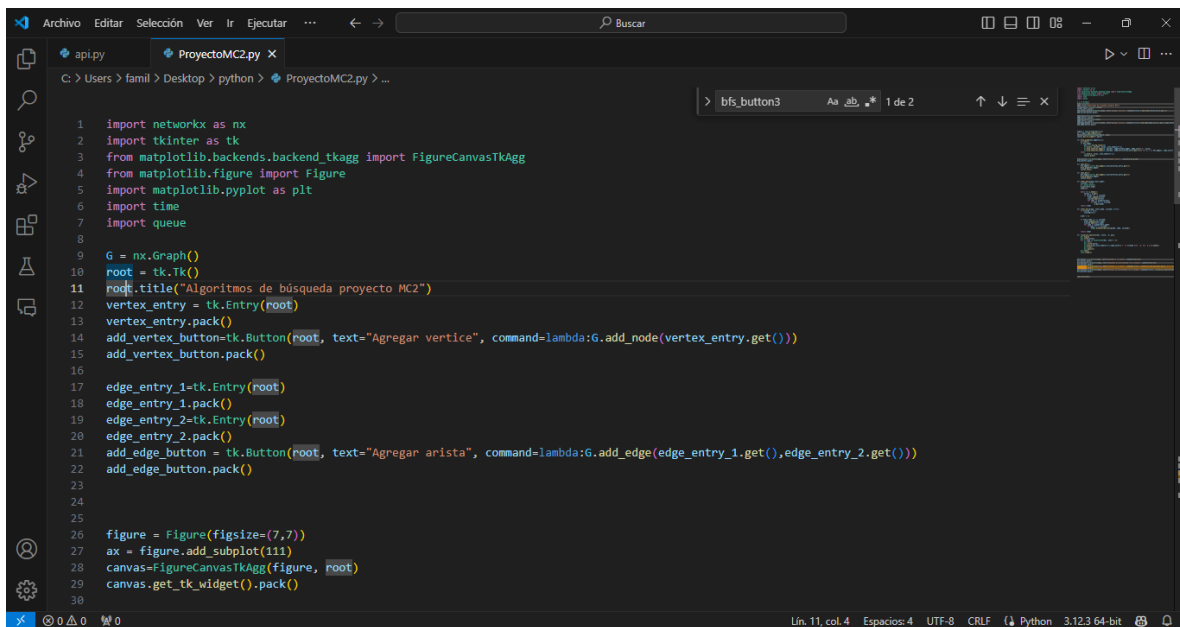
Por último, para la representación a una grafica se le da los parámetros, en este caso dichas listas, para poder graficarla mediante las funciones.



```
10 root = tk.Tk()
11 root.title("Algoritmos de búsqueda proyecto MC2")
12 vertex_entry = tk.Entry(root)
13 vertex_entry.pack()
14 add_vertex_button = tk.Button(root, text="Agregar vertice", command=lambda: add_node(vertex_entry.get()))
15 add_vertex_button.pack()
16
17 edge_entry_1 = tk.Entry(root)
18 edge_entry_1.pack()
19 edge_entry_2 = tk.Entry(root)
20 edge_entry_2.pack()
21 add_edge_button = tk.Button(root, text="Agregar arista", command=lambda: add_edge(edge_entry_1.get(), edge_entry_2.get()))
22 add_edge_button.pack()
23
24
25
26 figure = Figure(figsize=(7,7))
27 ax = figure.add_subplot(111)
28 canvas = FigureCanvasTkAgg(figure, root)
29 canvas.get_tk_widget().pack()
30
31 def draw_graph(bfs_edges=None):
32     ax.clear()
33     if bfs_edges:
34         pos = nx.spring_layout(G)
35         nx.draw(G, pos=pos, ax=ax, with_labels=True)
36         nx.draw_networkx_edges(G, pos=pos, edgelist=bfs_edges, edge_color='b', ax=ax)
37         nx.draw_networkx_nodes(G, pos=pos, nodelist=[vertex_entry.get(i) for i in bfs_edges], node_color='r', ax=ax)
38     else:
39         nx.draw(G, ax=ax, with_labels=True)
40     canvas.draw()
```

El código se realizó con diferentes lógicas que implementa a través de librerías que grafican coordenadas y procesos matemáticos, así como tiempo para la simulación de los diferentes resultados.

Se utilizó mediante funciones lambda, se implementaron en los diferentes botones y la librería de networks se realizó la graficación de los diferentes grafos que se vayan utilizando, la librería de tinker para la realización en conjunto con networks de implementación gráfica de los diferentes grafos. La librería math para hacer los diferentes cálculos matemáticos que se realizan durante el programa. Incluso time para la simulación de los algoritmos.



```
1 import networkx as nx
2 import tkinter as tk
3 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
4 from matplotlib.figure import Figure
5 import matplotlib.pyplot as plt
6 import time
7 import queue
8
9 G = nx.Graph()
10 root = tk.Tk()
11 root.title("Algoritmos de búsqueda proyecto MC2")
12 vertex_entry = tk.Entry(root)
13 vertex_entry.pack()
14 add_vertex_button = tk.Button(root, text="Agregar vertice", command=lambda:G.add_node(vertex_entry.get()))
15 add_vertex_button.pack()
16
17 edge_entry_1=tk.Entry(root)
18 edge_entry_1.pack()
19 edge_entry_2=tk.Entry(root)
20 edge_entry_2.pack()
21 add_edge_button = tk.Button(root, text="Agregar arista", command=lambda:G.add_edge(edge_entry_1.get(),edge_entry_2.get()))
22 add_edge_button.pack()
23
24
25
26 figure = Figure(figsize=(7,7))
27 ax = figure.add_subplot(111)
28 canvas=FigureCanvasTkAgg(figure, root)
29 canvas.get_tk_widget().pack()
30
```