Proyecto Final Análisis y Diseño de Algoritmos

Análisis de `coloreo`

La función `coloreo` es la más compleja de las dos, así que empezaremos por ella.

```
def coloreo(graph):
    n = len(graph)  # (1)
    colors = [-1] * n  # (n)
    degree = [(i, sum(graph[i])) for i in range(n)]  # (n^
2)
    degree.sort(key=lambda x: x[1], reverse=True)  # (n log
n)
    neighbor_colors = [set() for _ in range(n)]  # (n)

    current_color = 0  # (1)
    for node, _ in degree:  # (n)
        if colors[node] == -1:  # (n)
            colors[node] = current_color  # (n)
            for neighbor in range(n):  # (n * (n-1))
                if graph[node][neighbor] == 0 and colors[ne
ighbor] == -1 and current_color not in neighbor_colors[neig
hbor]:  # (n-1)^2
                    colors[neighbor] = current_color  # (n-
1)^2
                    for k in range(n):  # (n-1)^2 * (n)
                        if graph[neighbor][k] == 1:  # (n-
1)^3
                            neighbor_colors[k].add(current_
color)  # (n-1)^3
            current_color += 1  # O(n)
```

```
    return colors, current_color  # O(1)
```

Costo en el peor caso de `coloreo`

- Inicializar `colors`:$O(n)$
- Calcular `degree`:$O(n^2)$
- Ordenar `degree`: $O(nlogn)$
- Inicializar `neighbor_colors`: $O(n)$
- Doble bucle anidado para colorear el grafo: $O(n^3)$

Sumando estas complejidades:

$O(n) + O(n^2) + O(nlogn) + O(n) + O(n^3) = O(n^3)O(n) + O(n^2) + O(nlogn) + O(n) + O(n^3) = O(n^3)$

En el peor caso, `coloreo` tiene una complejidad asintótica de $O(n3)O(n3)$.

Costo en el mejor caso de `coloreo`

En el mejor caso, el grafo ya está coloreado óptimamente con el primer nodo:

- Inicializar `colors`: $O(n)$
- Calcular `degree`: $O(n^2)$
- Ordenar `degree`: $O(nlogn)$
- Inicializar `neighbor_colors`: $O(n)$
- Bucle anidado mínimo (p.ej., si todos los nodos se colorean al primer intento): $O(n^2)$

Sumando estas complejidades:

$O(n) + O(n^2) + O(nlogn) + O(n) + O(n^2) = O(n^2)O(n) + O(n^2) + O(nlogn) + O(n) + O(n^2) = O(n^2)$

En el mejor caso, `coloreo` tiene una complejidad asintótica de $O(n^2)$.

Costo en el caso promedio de `coloreo`

Para el caso promedio, asumimos que la estructura del grafo es aleatoria y cada nodo tiene un grado medio:

- Inicializar `colors`: $O(n)$
- Calcular `degree`:$O(n^2)$
- Ordenar `degree`: $O(nlogn)$
- Inicializar `neighbor_colors`: $O(n)$

En el caso promedio, `coloreo` también tiene una complejidad asintótica de $O(n^3)$.

Resumen

- Peor caso: $O(n^3)$

  $O(n^3)$
- Mejor caso: $O(n^2)$

  $O(n^2)$
- Caso promedio: $O(n^3)$

  $O(n^3)$

Estructuras de datos

1. Listas

a. Lista `colors`

- **Descripción**: `colors` es una lista que contiene los colores asignados a cada villa (o nodo) en el grafo.
- **Uso en el código**:
  - `num_selections = len(colors)` obtiene el número de selecciones (la cantidad de colores asignados).
  - `colors[selection]` se utiliza para comparar si una villa está asignada a un color específico.

b. Lista `villa_matrix`

- **Descripción**: `villa_matrix` es una lista de listas (matriz) que se genera y retorna en la función.

- **Uso en el código**:
  - `villa_matrix` es la matriz final que indica las selecciones de colores para cada villa.
  - Cada fila en `villa_matrix` representa una villa, y cada columna representa una selección de color.
  - La lista externa (`villa_matrix`) contiene una lista interna para cada villa.
  - La lista interna contiene valores `0` y `1` indicando si una villa tiene un color específico.

c. Lista interna dentro de `villa_matrix`

- **Descripción**: Cada elemento de la lista interna es un entero (`0` o `1`) que indica si la villa corresponde al color en la posición actual.
- **Uso en el código**:
  - La comprensión de listas doblemente anidada crea esta lista interna.
  - `int(villa == colors[selection])` produce `1` si la villa corresponde al color en la posición actual de `colors` y `0` en caso contrario.

2. Comprensiones de Listas

a. Comprensión de lista interna

- **Descripción**: Es una estructura que permite crear listas de manera concisa en una sola línea.
- **Uso en el código**:
  - `[int(villa == colors[selection]) for selection in range(num_selections)]` crea una lista donde cada elemento es `1` si la villa corresponde al color en la posición actual de `colors` y `0` en caso contrario.

b. Comprensión de lista externa

- **Descripción**: Es otra comprensión de lista que contiene la comprensión de lista interna.
- **Uso en el código**:
  - `[[int(villa == colors[selection]) for selection in range(num_selections)] for villa in range(num_villas)]` crea `villa_matrix`, que es una lista de listas,

donde cada sublista representa las selecciones de colores para una villa específica.

Pruebas e Interfaz de usuario

Pruebas realizadas

Se realizaron un total de 6 pruebas para validar el funcionamiento del código. Estas pruebas abarcaron una variedad de escenarios con diferentes tamaños y formas de datos. Los resultados de estas pruebas demostraron que el código funciona de manera eficiente y precisa, incluso cuando se trata de organizar hasta 100 equipos o más si es necesario.

Evidencias

El archivo `main.py` se encuentran las pruebas, basta con solo ejecutarlo y mostrará el resultado de los test.

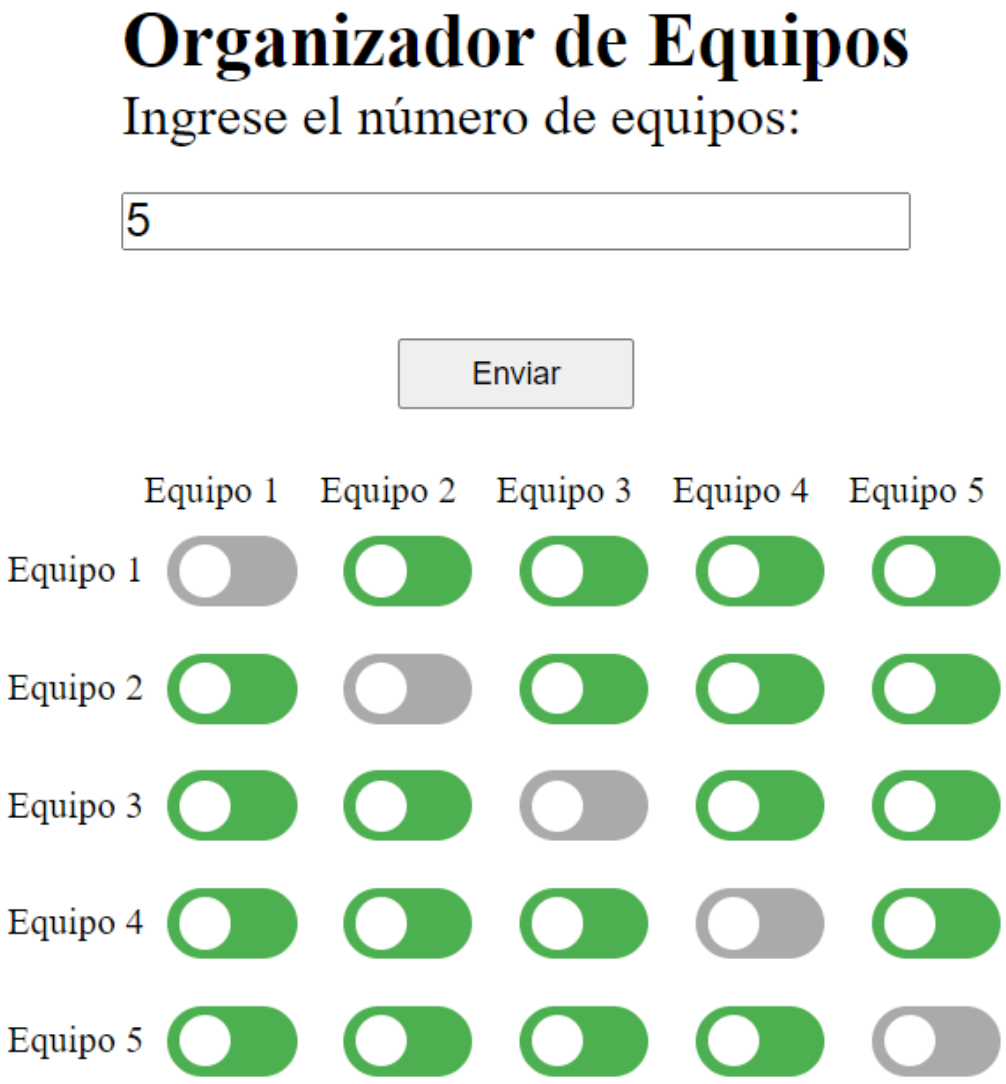
Se hizo uso de la librería `unittest` para realizar todas las pruebas, librería incorporada al lenguaje de Python.

```
......
----------------------------------------------------------------------
Ran 6 tests in 0.001s

OK
```

Interfaz de usuario

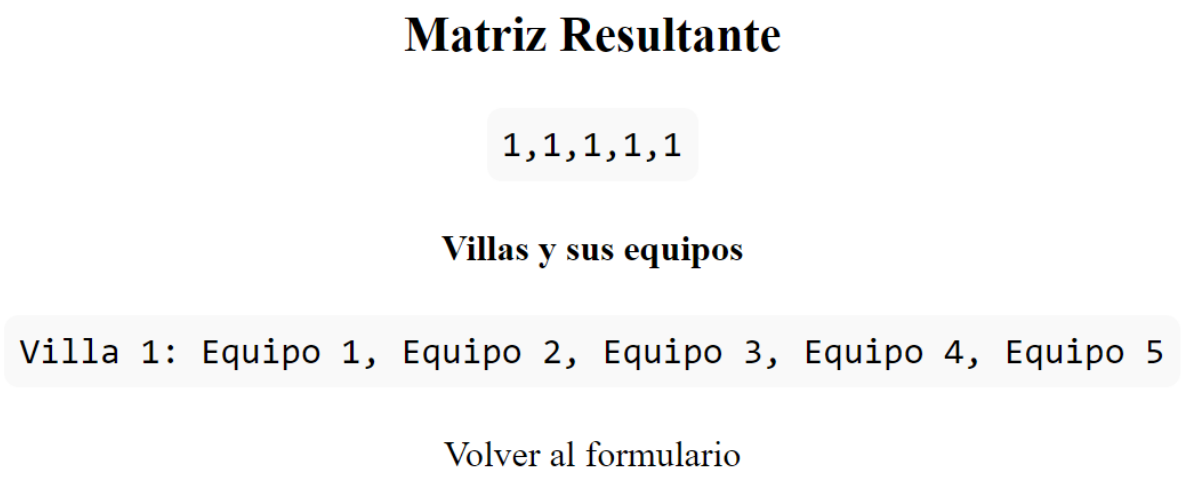
Se realizó una interfaz de usuario sencilla, que pide al usuario cuántos equipos necesita organizar. Posteriormente, se genera una matriz de tamaño n x n, donde n es el número de equipos, utilizando switches para recopilar datos sobre si los equipos se llevan bien entre sí, representados en filas y columnas. Si se selecciona un elemento, se selecciona automáticamente su correspondiente inverso, es decir, si se selecciona la fila 1, columna 3, también será seleccionada la fila 3, columna 1. (Imagen 1)



(Imagen 1)

La función principal de este programa es generar una distribución de datos basada en la información proporcionada. Para funcionar correctamente, el programa envía la información a un archivo de texto. Luego, las funciones dentro del programa analizan esta información y generan el resultado esperado.

El resultado se presenta en forma de matriz m x n, donde m representa el número de villas y n representa el número de equipos. Esta matriz proporciona una representación visual clara y organizada de cómo se distribuyen los equipos entre las villas según los criterios establecidos. (Imagen 2)



(Imagen 2)

Referencias

Los siguientes sitios web se utilizaron para obtener una mejor comprensión de los conceptos y técnicas necesarias para desarrollar este proyecto. Su uso ha ayudado a garantizar que el proyecto sea de la más alta calidad posible.

https://ellibrodepython.com/python-testing

https://docs.python.org/3/library/unittest.html

https://cienciadedatos.net/documentos/pygml01-introduccion-grafos-redes-python

https://www.techiedelight.com/es/greedy-coloring-graph/