Reporte de Laboratorio Equipo 1
ArrayBaseList

César Aldahir Flores Gámez*, Lorena Marisol Romero Hernández*,
José Andrik Martínez Rodríguez*,
Francisco Gael Sustaita Reyna*
*Ingeniería en Tecnologías de la Información
Universidad Politécnica de Victoria

***Resumen*—Este trabajo presenta una aplicación de escritorio desarrollada en Python con el objetivo de proporcionar una solución eficiente y fácil de usar para agregar elementos a una lista. La aplicación utiliza la estructura de datos Array-Based List en una interfaz gráfica de usuario creada con PyQT5. En el trabajo se describe el diseño y la implementación de la aplicación, así como la integración de la estructura de datos con la interfaz gráfica. También se discuten las optimizaciones de rendimiento y la escalabilidad de la aplicación. Los resultados muestran la eficacia y la utilidad de la estructura de datos Array-Based List en una aplicación práctica. Este trabajo puede ser utilizado como punto de partida para desarrollar aplicaciones más complejas basadas en la estructura de datos Array-Based List.**

I. Introducción

En este proyecto hemos desarrollado una aplicación de escritorio en Python que utiliza la estructura de datos Array-Based List en una interfaz gráfica de usuario creada con PyQT5, ya que PyQt5 es una biblioteca de enlace para la herramienta de diseño gráfico Qt, que es ampliamente utilizada en el desarrollo de aplicaciones GUI [1]. El objetivo de la aplicación es proporcionar una solución eficiente y fácil de usar para agregar elementos a una lista, utilizando la estructura de datos Array-Based List.

En este trabajo, presentaremos el diseño y la implementación de la aplicación, explicando cómo se integra la estructura de datos Array-Based List con la interfaz gráfica de usuario. Describiremos cómo hemos utilizado PyQT5 para crear la interfaz de usuario y cómo hemos implementado la funcionalidad necesaria para agregar elementos a la lista.
Al final de este proyecto, esperamos haber demostrado la eficacia y la utilidad de la estructura de datos Array-Based List en una aplicación práctica y haber proporcionado una solución escalable y eficiente para agregar elementos a una lista en una aplicación de escritorio desarrollada con PyQT5 y Python.

II. Desarrollo Experimental

Para el desarrollo de este proyecto fue necesario investigar el funcionamiento, y para que pueden ser utilizadas cada una de las siguientes estructuras de datos:

- **ArrayStack:** es una pila (stack) implementada utilizando un arreglo (array). Permite añadir y remover elementos solamente por un extremo (el tope de la pila), por lo que sigue el principio LIFO (Last-In-First-Out). Se puede utilizar para realizar operaciones que sigan este principio, como por ejemplo, en la implementación de sistemas de historial en navegadores web o editores de texto [2].

- **FastArrayStack:** es similar al ArrayStack, pero se diferencia en que tiene una mayor eficiencia en cuanto a tiempos de ejecución. Es útil en situaciones donde se requiere una alta velocidad de procesamiento, como en el procesamiento de datos en tiempo real[3].

- **ArrayQueue:** es una cola (queue) implementada utilizando un arreglo. Permite añadir elementos por un extremo (el final de la cola) y removerlos por el otro extremo (el frente de la cola), siguiendo el principio FIFO (First-In-First-Out). Es útil en situaciones donde se requiere procesar datos en orden de llegada, como en la planificación de procesos en sistemas operativos [4].

- **ArrayDeque:** es una doble cola implementada utilizando un arreglo. Permite añadir y remover elementos por ambos extremos de la cola, lo que la hace más flexible que la ArrayQueue. Se puede utilizar en situaciones donde se requiere una estructura de datos versátil para manejar una lista de elementos [5].

- **DualArrayDeque:** es una estructura de datos que combina las funcionalidades de un ArrayDeque y una lista enlazada doble. Permite agregar y eliminar elementos tanto por el inicio como por el final de la lista, y además, tiene una mayor eficiencia en la inserción y eliminación de elementos en cualquier posición. Es útil en situaciones donde se requiere una estructura de datos que permita operaciones de inserción y eliminación en cualquier posición de la lista [6].

- **RootishArrayStack:** es una pila implementada utilizando una estructura de datos llamada rootish array", que es una matriz de arreglos cuyo tamaño aumenta progresivamente [7]. Esta estructura permite realizar operaciones de apilar y desapilar elementos en

tiempo constante, lo que la hace ideal para situaciones donde se requiere una alta eficiencia en la ejecución de estas operaciones [8].

El código desarrollado contiene varias clases para implementar diferentes estructuras de datos en Python utilizando la biblioteca PyQt5. Cada clase está diseñada para representar gráficamente la estructura de datos correspondiente en una ventana de la interfaz de usuario. De igual forma, se implementaron las funciones de QPainter para realizar las animaciones correspondientes. QPainter es una clase de la biblioteca gráfica de PyQt5 que permite dibujar formas, imágenes y texto en widgets de la interfaz de usuario. Proporciona una API de bajo nivel para la manipulación de píxeles y una API de alto nivel para la creación de gráficos vectoriales [9]. Otra librerira que se tiene que importar antes de correr el programa es la liberia

La clase ArrayStack implementa una pila utilizando una lista Python para almacenar los elementos. Cada vez que se agrega o se quita un elemento de la pila, la interfaz gráfica se actualiza para reflejar los cambios. Dicha clase contiene los siguientes métodos:

- En el método **__init__**: se crea una lista vacía self.stack que se utilizará para almacenar los elementos de la pila.
- El método **paintEvent**: se utiliza para dibujar la pila en la ventana. Primero se crea un objeto QPainter que se utiliza para dibujar formas y texto. Luego se establecen el color de la pluma y el patrón de relleno. Se define el ancho y la altura de cada rectángulo que representará un elemento de la pila. Se inicializa x y y para la posición inicial del primer rectángulo. Se utiliza un bucle for para iterar sobre los elementos de la lista self.stack. En cada iteración, se dibuja un rectángulo en la posición actual y se escribe el valor del elemento en el centro del rectángulo. La posición x se actualiza para la siguiente iteración.
- El método **push**: se utiliza para agregar un elemento a la pila. El elemento se agrega a la lista self.stack y se llama al método update para redibujar la ventana.
- El método **pop**: se utiliza para eliminar el elemento superior de la pila. Si la lista self.stack no está vacía, el último elemento se elimina utilizando el método pop y se llama al método update para redibujar la ventana.

La clase FastArrayStack también implementa una pila, pero utiliza una lista Python preasignada para mejorar el rendimiento en la inserción de elementos. En este caso, se utiliza un puntero top para llevar un seguimiento del elemento superior en la pila. La clase mencionada, contiene los siguientes métodos:

- El método **paintEvent** dibuja los elementos de la pila en la ventana utilizando un objeto QPainter. Cada elemento se representa como un rectángulo con un ancho de 50 y una altura de 30. El valor del elemento se muestra en el centro del rectángulo.
- El método **push** agrega un elemento a la pila. Primero, incrementa la variable top en uno. Si top es menor que la longitud de la lista stack, se modifica el elemento en la posición top con el valor del nuevo elemento. Si top es mayor o igual a la longitud de la lista, entonces el nuevo elemento se agrega al final de la lista. En ambos casos, se llama al método update para redibujar la pila en la ventana.
- El método **pop** elimina el elemento en el tope de la pila. Si la lista stack no está vacía, se llama al método pop para eliminar el último elemento de la lista. Luego, se decrementa la variable top en uno y se llama al método update para redibujar la pila en la ventana.

La clase ArrayDeque implementa una deque utilizando una lista Python y los punteros front y rear. En Python, los punteros se refieren a las referencias a objetos [10]. La lista se redimensiona dinámicamente según sea necesario y se actualiza la interfaz gráfica cuando se agregan o eliminan elementos. Esta clase tiene las siguientes propiedades:

- **array**: Un arreglo que almacena los elementos de la cola.
- **front**: Un índice que indica el primer elemento de la cola.
- **rear**: Un índice que indica el último elemento de la cola.

La clase tiene los siguientes métodos:

- **__init__()**: Constructor de la clase. Inicializa el arreglo array con 2 elementos vacíos, y establece los índices front y rear en -1.
- **paintEvent(event)**: Método que se encarga de dibujar la cola en la pantalla. Itera sobre el arreglo y dibuja cada elemento en un rectángulo con texto en el centro.
- **push_front(item)**: Agrega un elemento al frente de la cola. Si el frente está en la posición 0, duplica el tamaño del arreglo para poder agregar el elemento. Si no hay elementos en la cola, establece front y rear en 0. Luego, decrementa el índice front y agrega el elemento en la nueva posición. Llama al método update() para redibujar la cola.
- **push_back(item)**: Agrega un elemento al final de la cola. Si la cola está llena, duplica el tamaño del arreglo para poder agregar el elemento. Incrementa el índice rear y agrega el elemento en la nueva posición. Si no hay elementos en la cola, establece front en 0. Llama al método update() para redibujar la cola.
- **pop_front()**: Elimina el elemento del frente de la cola y lo devuelve. Si no hay elementos en la cola, devuelve None. Si el frente y el rear apuntan al mismo elemento, establece front y rear en -1. Si no, incrementa el índice front. Llama al método update() para redibujar la cola.
- **pop_back()**: Elimina el elemento del final de la cola y lo devuelve. Si no hay elementos en la cola, devuelve None. Decrementa el índice rear. Si rear es menor que front, establece front y rear en -1. Llama al método update() para redibujar la cola.

La clase ArrayQueue implementa una cola utilizando una lista

Python y los punteros front y rear. Al igual que ArrayDeque, la lista se redimensiona dinámicamente según sea necesario y la interfaz gráfica se actualiza al agregar o eliminar elementos. Sin embargo, en este caso se utiliza un enfoque diferente para recorrer la cola, iterando sobre los elementos de front a rear. La clase utiliza los siguientes métodos:

- El método **__init__** inicializa la cola con una matriz de tamaño 2 y los punteros front y rear se inicializan en -1 para indicar que la cola está vacía.
- El método **paintEvent** es un método gráfico que dibuja los elementos de la cola en la ventana. Cada elemento se dibuja como un rectángulo y se etiqueta con el valor del elemento.
- El método **enqueue** agrega un elemento a la cola. Si la cola está llena, se duplica el tamaño de la matriz para permitir que la cola tenga más elementos. El elemento se agrega al final de la cola, actualizando el puntero rear si es necesario, y se actualiza la ventana.
- El método **dequeue** retira y devuelve el primer elemento de la cola. Si la cola está vacía, no hace nada y simplemente retorna None. Si hay elementos en la cola, se retira el primer elemento actualizando el puntero front. Si después de retirar el elemento la cola está vacía, se restablecen los punteros front y rear a -1. La ventana se actualiza después de retirar el elemento.

La clase DualArrayDeque implementa una estructura de datos deque, que es una secuencia lineal de elementos que permite agregar y eliminar elementos en ambos extremos de la secuencia. Los métodos utilizados en esta clase son:

- El constructor **__init__** de la clase: inicializa dos instancias de la clase deque() de Python, una para representar el frente de la deque (self.front) y otra para representar el final de la deque (self.rear).
- El método **paintEvent()**: dibuja los elementos de la deque. La deque se dibuja como una serie de rectángulos con el contenido de la deque centrado en cada rectángulo. La deque se divide en dos partes con una línea vertical en el medio para indicar la posición de la mitad de la deque.
- El método **add_front()**: añade un elemento al frente de la deque, utilizando el método appendleft() de la clase deque(). Luego, llama al método update() para redibujar los elementos de la deque.
- El método **remove_front()**: elimina y devuelve el elemento del frente de la deque, utilizando el método popleft() de la clase deque(). Si la deque está vacía, no hace nada y devuelve None. Luego, llama al método update() para redibujar los elementos de la deque.
- El método **add_rear()**: añade un elemento al final de la deque, utilizando el método append() de la clase deque(). Luego, llama al método update() para redibujar los elementos de la deque.
- El método **remove_rear()**: elimina y devuelve el elemento del final de la deque, utilizando el método pop() de la clase deque(). Si la deque está vacía, no hace nada y devuelve None. Luego, llama al método update() para redibujar los elementos de la deque.

La clase RootishArrayStack define una estructura de datos de pila basada en una lista de bloques. La implementación de la pila utiliza una técnica de partición de raíz cuadrada, donde la pila se divide en bloques de tamaño variable y cada bloque es una lista que puede contener una cantidad variable de elementos. Dicha clase utiliza los siguientes elementos:

- El constructor de la clase inicializa la ventana de la interfaz gráfica, crea una pila vacía y una lista de bloques vacía que se utilizarán para almacenar los elementos de la pila. El método paintEvent es el encargado de dibujar la pila en la ventana.
- El método **push** se utiliza para agregar un elemento a la pila. Si la lista de bloques está vacía o el último bloque está lleno, se crea un nuevo bloque para almacenar el elemento. Si el último bloque no está lleno, el elemento se agrega al último bloque existente. El elemento también se agrega a la pila y se llama al método update para actualizar la interfaz gráfica.
- El método **pop** se utiliza para quitar un elemento de la pila. Si la pila no está vacía, el último elemento de la pila se quita y se elimina del último bloque. Si el último bloque está vacío después de quitar el elemento, se elimina el último bloque. El método update se llama para actualizar la interfaz gráfica.
- El método **paintEvent** dibuja la pila en la ventana utilizando la clase QPainter de PyQt. Primero se establecen los colores de lápiz y pincel. Luego, se dibujan los rectángulos y se escribe el contenido de cada bloque dentro de ellos. El método se llama automáticamente cada vez que se actualiza la interfaz gráfica, lo que sucede después de llamar al método update.

Por último, La clase MainWindow es la ventana principal de la aplicación. En la función __init__, se configura la geometría y el título de la ventana y se llama al método initUi, que crea los botones y los conecta con los métodos que abren las ventanas correspondientes a cada estructura de datos.

Cada una de las funciones open_window_X es responsable de crear una ventana para una estructura de datos específica (X), en donde se muestra la estructura y se pueden realizar operaciones como agregar y eliminar elementos. Cada ventana se configura con su título y geometría, se crea la estructura de datos y se conectan los botones de la ventana con las funciones correspondientes que realizan las operaciones en la estructura de datos, para así, mostrar la ventana.

III. Resultados

En cuanto a los resultados obtenidos con nuestro proyecto de array, podemos mencionar que este fue exitoso en su totalidad, ya que el programa es capaz de realizar los

recorridos de manera correcta en cada uno de los caso. A continuacion se explicaran el funcionamiento del programa con cada recorrido.

Menú Principal:
Cuando se ejecuta el programa, en un principio se muestra un menú, donde el usuario puede seleccionar que tipo de recorrido desee realizar, como se aprecia en la figura 1.

Figura 1: Menú de opciones, menú que se ejecuta al iniciar el programa

En esta ventana el usuario puede seleccionar el tipo de recorrido que quiera realizar, bastando en solo dar clic en las opciones.

ArrayStack:
En esta opción el usuario puede escoger en el menú 2, ya sea un push o pop.

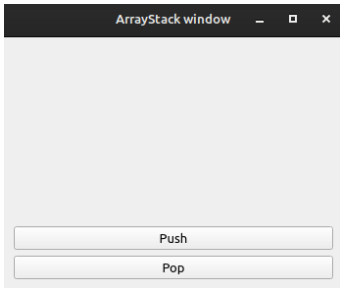

Figura 2: Menú de ArrayStack, menú que se ejecuta al seleccionar la opción ArrayStack

Si se selecciona un pop antes realizar un push simplemente el programa no hará nada, por lo que es necesario que el usuario primeramente genere un push. Cuando el usuario seleccione ”push”se mostrara una ventana en donde el usuario ingresara el elemento que desee agregar a la estructura, como se muestra en la figura 3.

Una vez se haya dado clic en push, el elemento que se haya asignado se mostrara de manera gráfica en la ventana, como se muestra en la figura 4

Si el usuario selecciona push, se retirara el elemento que se encuentre más a la derecha de la pantalla, ya que este fue el último elemento que fue asignado, como se muestra en la figura 5.

FastArrayStack:
Si el usuario selecciona esta opción, básicamente tendrán las

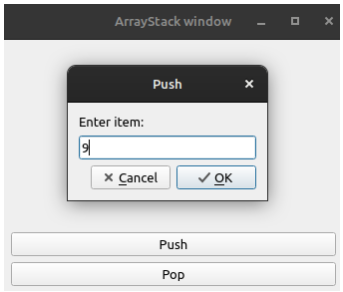

Figura 3: Menú de ArrayStack, menú que se ejecuta al seleccionar la opción ArrayStack

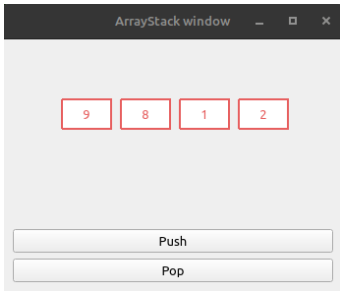

Figura 4: Pantalla de ArrayStack, se muestran elementos asignados por el botón push

mismas acciones que en la opción de ArrayStack, donde la única diferencia es que esta estructura es más rápida que la mencionada anteriormente.

Deque:
Si el usuario selecciona esta opción, se abrirá la siguiente ventana 6, donde se muestran cuatro acciones, push front, push back, pop front y pop back

Si el usuario selecciona la opción de push front, o push back, se mostrara la siguiente ventana para que pueda asignar el elemento que desee, como se muestran en las figuras 7 y 8

Una vez asignado al elemento, se mostrará de las siguientes maneras, dependiendo si se escogió la opción push front o push back 9.

El usuario, cuenta con 2 opciones para realizar un pop, ya sea el elemento Front, o el elemento back, que dependiendo de los casos se desacoplara de la estructura como se muestran en las figuras 10 y 11

ArrayQueue:
Si el usuario selecciona esta opción, se abrirá la siguiente ventana 12, donde se muestran dos acciones, equeue y dequeue, que seria encolar y desencolar.

De la misma manera, si el usuario selecciona primero la opción dequeue, este simplemente no hará nada hasta que tenga un elemento que desencolar.

Cuando el usuario seleccione la opción .[en]queue”, se abrirá la siguiente ventana 13, donde de igual manera el usuario debe de colocar el elemento que desea encolar.

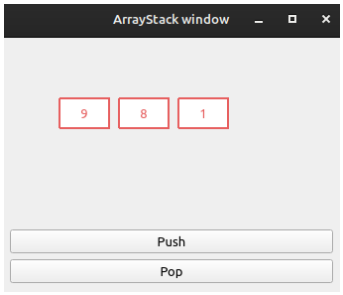

Figura 5: Pantalla de ArrayStack, se muestran que el último elemento fue eliminado

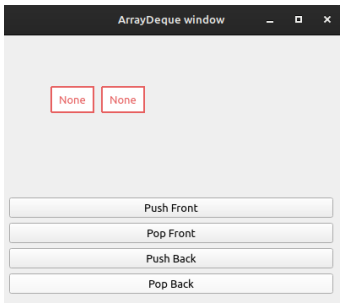

Figura 6: Menú ArrayDeque

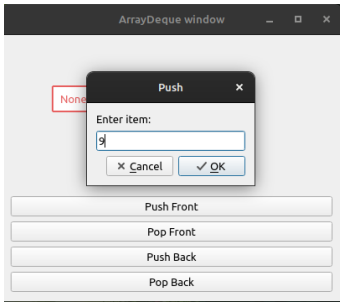

Figura 7: Push Front

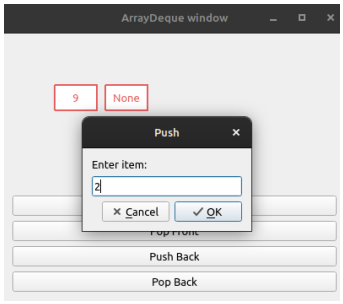

Figura 8: Push Back

Una vez asignado, se mostrará el elemento agregado de la siguiente manera, como se aprecia en la figura 14, dando a entender que se encoló de manera correcta.

Cuando el usuario seleccione la opción dequeue, este va a eliminar el primer elemento que se asignó, como se muestra en la figura 15, a diferencia del arrayStack que se eliminó el último elemento asignado.

DualArrayQueue:

En esta estructura, se le brinda al usuario más opciones, a diferencia de las estructuras anteriores, debido a que este tipo de estructura usa dos estructuras queue para funcionar, por lo que se ofrecen opciones para encolar o desencolar, en la estructura queue de enfrente, o la de atrás, como se muestra en la figura 16

Si el usuario selecciona la opción de add to front, o add to rear, se mostrara la siguiente ventana para que pueda asignar el elemento que desee, como se muestran en las figuras 17 y 18

Una vez asignado al elemento, se mostrará de las siguientes maneras, dependiendo si se escogió la opción AddToFront o AddToRear 19.

El usuario, cuenta con 2 opciones para desencolar, y de la misma manera, puede ser la primera estructurá cola de enfrente 20 o la segunda 21, pero de la misma manera se desencolará el primer elemento que se haya asignado a su respectiva cola.

Rootish Array Stack:

Por último, tenemos la estructura RootishArrayStack, donde si el usuario selecciona dicha opción, aparecerá el menú que se muestra en la figura 22, donde se muestra las opciones de push y pop.

Si el usuario selecciona la opción push, él debe de colocar el elemento que desee añadir, como se muestra en la figura 23, donde posteriormente debe de dar clic en ok para añadirlo.

Una vez colocado se añadirá el elemento de la misma manera que se agregaron en las otras estructuras. En la figura 24, se pueden apreciar varios elementos añadidos de una manera distinta, ya que son varias listas que se pueden acoplar, donde se menciona más detalladamente este procedimiento en el desarrollo anterior.

Cuando el usuario seleccione la opción pop, se desacoplara el último elemento que fue añadido con el botón push, como se aprecia en la figura 25, donde el último elemento añadido fue un ”3”

Para finalizar con esta sección, podremos mencionar que las pruebas que se realizaron con este programa fueron exitosas en un 100 % por lo que podríamos mencionar que la solución propuesta cumplió con lo solicitado

IV. Conclusión

El resultado de acuerdo a nuestro proyecto de realizar una aplicación que pueda realizar distintas implementaciones de varias estructuras de datos “Array” fue un éxito, ya que dichas implementaciones las realiza de manera correcta y exitosa por medio de una interfaz grafica desarrollada en pyqt5, además

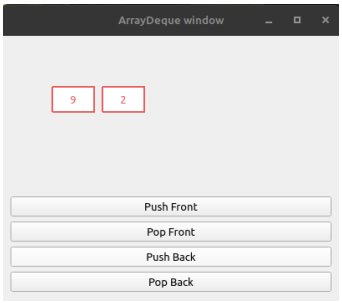

Figura 9: Mostrar elemento push front

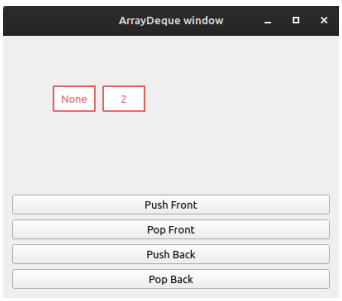

Figura 10: Pop Front

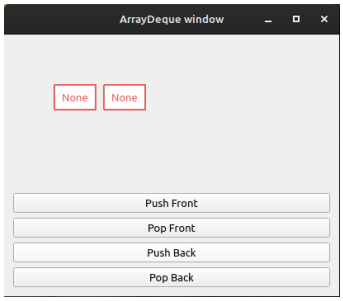

Figura 11: Pop Back

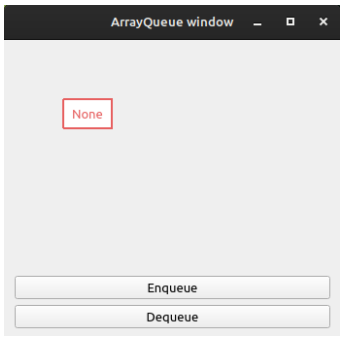

Figura 12: Menú Queue

de haber incorporado un menú muy entendible y amigable para el usuario, por lo que no se presentan problemas a la hora del manejo de la aplicación. Para lograr las inserciones, analizamos la herramienta online llamada, visualgo, que es una herramienta que hace distintos recorridos e inserciones de distintas estructuras de datos, y los representa de una manera visual y entendible, donde fue un proceso difícil llegar a mostrarlo en la interfaz pyqt5, por lo que una vez logrado realizar las primeras inserciones de "ArrayStack", no fue de mucha complicación realizar los otros, a excepción del último, el "RootishArrayStack", debido a que había pequeños fallos al momento de insertar o desacoplar elementos. De acuerdo a la ejecución del mismo, las pruebas que se realizaron en un inicio fueron erróneas, ya que como se mencionó, hubo problemas al momento de mostrarlo en la interfaz desarrollada en pyqt5. Después de cierto tiempo, logramos mostrar dichas inserciones de manera correctas, por lo que las últimas pruebas que se realizaron fueron exitosas, logrando así un resultado exitoso con el proyecto desarrollado.

Referencias

[1] Mark Summerfield. *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming*. Addison-Wesley Professional, 2019.

[2] Robert Sedgewick y Kevin Wayne. *Algorithms*. Addison-Wesley, 2011.

[3] Robert Sedgewick y Kevin Wayne. *Algorithms*. Addison-Wesley, 2011.

[4] Thomas H Cormen et al. *Introduction to Algorithms*. MIT press, 2009.

[5] Thomas H Cormen et al. *Introduction to Algorithms*. MIT press, 2009.

[6] Andrej Brodnik et al. "Dual arrays: a doubly-indexed sequence representation". En: *Algorithmica* 23.3 (1999), págs. 198-215.

[7] Andrej Brodnik et al. "Rootish array–a simple data structure for non-uniform memory access". En: *Algorithmica* 24.3-4 (1999), págs. 214-238.

[8] Harald Prokop. "A purely functional representation of catenable sorted lists". En: *Journal of Functional Programming* 12.3 (2002), págs. 231-247.

[9] David Amos. *Python: How to Work with Pointers*. 2021. URL: https://realpython.com/pointers-in-python/ (visitado 13-03-2022).

[10] Python Software Foundation. *Python's memory model*. 2021. URL: https://docs.python.org/3/c-api/memory.html.

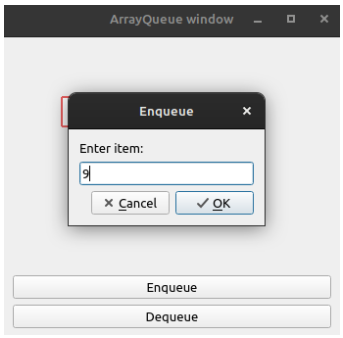

Figura 13: Ventana para encolar elemento

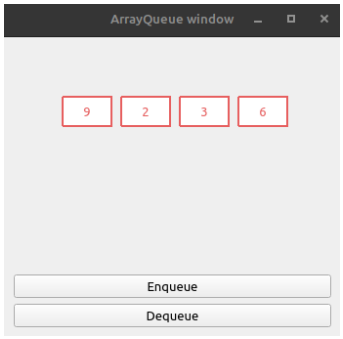

Figura 14: Ventana ya con elemento asignado en la cola

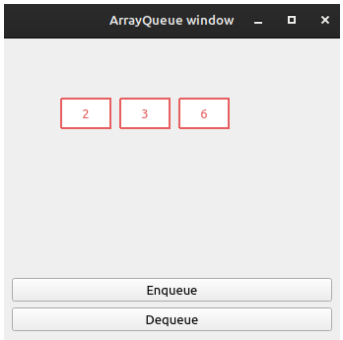

Figura 15: Ventana ya con elemento desencolado de la cola

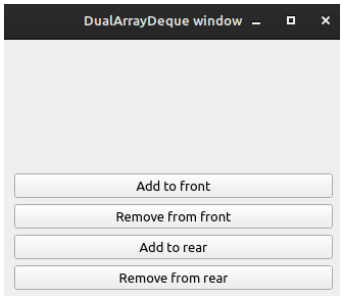

Figura 16: Menú DualArrayQueue

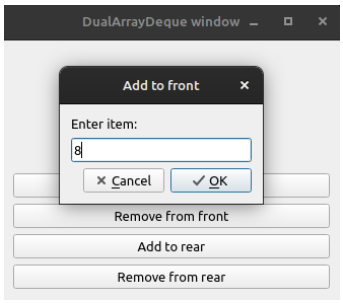

Figura 17: Asignar elemento AddToFront

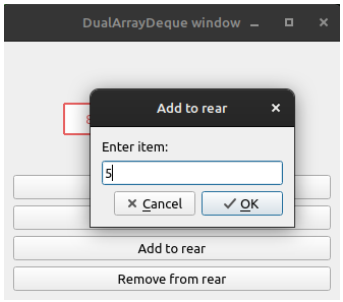

Figura 18: Asignar elemento AddToRear

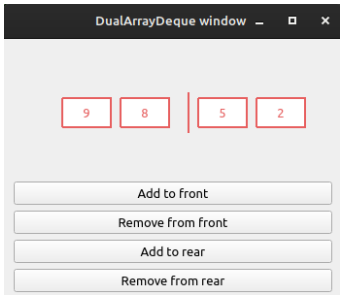

Figura 19: Mostrar elemento AddToFront

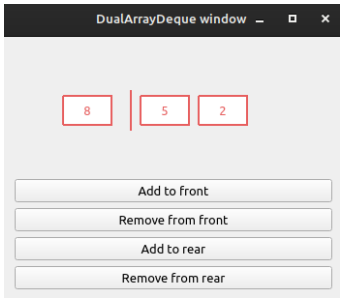

Figura 20: Desencolar elemento Front

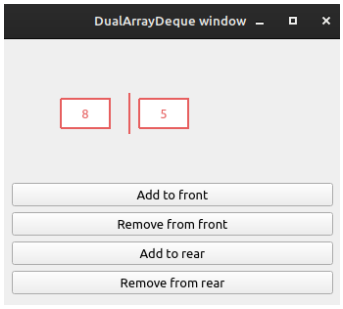

Figura 21: Desencolar elemento Rear

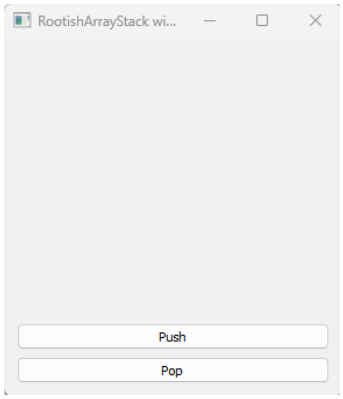

Figura 22: Menú Rootish Array Stack

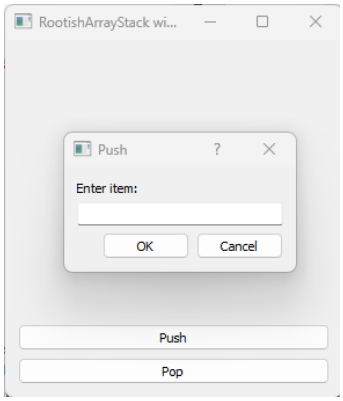

Figura 23: Push Rootish Array Stack

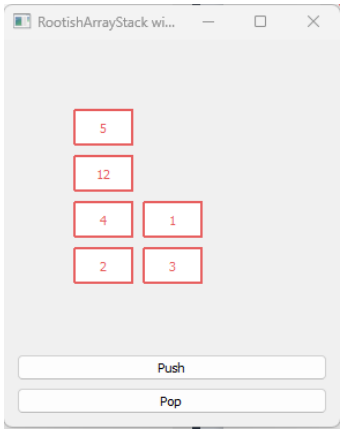

Figura 24: Elementos puestos Rootish Array Stack

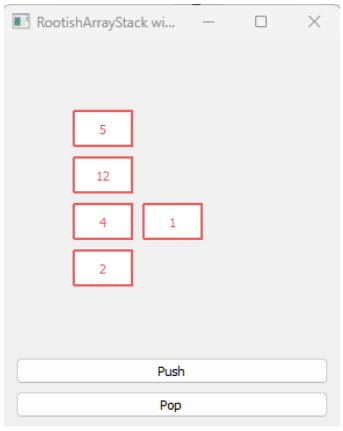

Figura 25: Pop Rootish Array Stack