Gestor de Cursos y Rutas Académicas

Danny Alejandro Bañol Osorio, Julian Andres Bonilla Mortigo, Jose Luis Rativa Medina.

No. de equipo de trabajo: {*G*}

I. INTRODUCCIÓN

Según el direccionamiento de la asignatura Estructura de Datos, se plantea desde un proyecto grupal dar solución a un problema en el que se establezcan y se expongan funcionalidades en torno a las estructuras de datos con las que nos vamos familiarizando a lo largo del desarrollo de la asignatura; para ello decidimos desde nuestra perspectiva de estudiantes, plantear una solución a una de las problemáticas actuales sobre la gestión de asignaturas o planes académicos a lo largo de la carrera. El presente documento es, entonces, el desarrollo de la exploración de rutas a posibles soluciones a dicho planteamiento.

II. DESCRIPCIÓN DEL PROBLEMA A RESOLVER

Los estudiantes deben y necesitan planificar correctamente su plan de estudios, según sus proyectos personales y profesionales, sin las preocupaciones usuales de saber cuáles prerrequisitos tiene una asignatura. Para esto, el estudiante debe tener en cuenta si está cursando asignaturas por fuera de la línea de avance de un programa, si ha cambiado el estado de una asignatura respecto a un plan académico o si la plataforma del *Sistema de Información Académico* (SIA) está funcionando correctamente para acceder a esta información.

Esta necesidad inicial llevó a comprender que existe un problema de fondo asociado; que, sólo el 28% de los estudiantes se gradúan a tiempo de la Universidad [1] y que una de las razones por las cuales los estudiantes no pueden finalizar su plan académico a tiempo, radica en que no se tiene claridad en las asignaturas que deben seguir para terminar dicho programa [2].

La solución que se plantea al problema implicaría por ejemplo el desarrollo de una *Progressive Web App* (PWA) (más adelante se definirá qué es una PWA) , como herramienta que permita a los estudiantes seleccionar los cursos que integrarán la ruta a seguir a lo largo de su plan de estudios. De la misma manera, facilitará la comprensión de la carga académica que comprende cada curso, y minimizará el impacto en la toma de decisiones [3] ya que el estudiante tendrá un plan a seguir en su plan de estudios y no se verá en la situación de tomar decisiones a última hora. Crear una ruta académica tiene que ser parte de una buena experiencia en la institución, no otro problema.

III. USUARIOS DEL PRODUCTO DE SOFTWARE

En esta sección se hace mención de las características y clasificación de los usuarios (perfiles/roles) que utilizarán el producto.

- Estudiantes: Los principales usuarios del producto serán los estudiantes que quieran planificar su ruta académica de acuerdo a sus elecciones.

IV. REQUERIMIENTOS FUNCIONALES DEL SOFTWARE

A continuación se presentan las actividades que el producto debe realizar y ejecutar.

Entre los posibles requerimientos funcionales del sistema, se incluyen:

- Los usuarios deberán escoger, entre el catálogo de asignaturas, las que deseen cursar con el fin de generar sugerencias.
- Consultar las asignaturas disponibles.
- Los usuarios son los que pueden ingresar datos al sistema. Dichos datos pueden ser nombre de asignaturas, sus prerrequisitos, alguna ruta donde se guarden un archivo previamente generado por el programa.
- Los reportes se presentan inicialmente en consola al usuario. En próximos avances implementaremos blazor para crear un ambiente más interactivo.
- El sistema procesa la información de acuerdo a las elecciones del usuario.
- Menú:
  El usuario puede elegir entre algunas opciones que puede realizar el programa. En está pantalla el sistema muestra las opciones y procesa la opción escogida.
- Organizar asignaturas:

  En está pantalla el sistema pide el número de asignaturas a organizar por prerrequisitos, por lo tanto pide para cada asignatura el nombre y seguidamente los prerrequisitos. Luego de esto, el sistema analiza y calcula un semestre apropiado para cursar cada materia según los requisitos que está tenga.
- Selecciona asignaturas para generar rutas:

  En está pantalla el sistema pregunta el nombre de la asignatura, busca en las asignaturas del sistema y sugiere una carrera a cursar.
- Buscar asignatura:
  En está pantalla el sistema pregunta por el nombre de la asignatura, la busca en las asignaturas del sistema y le muestra información relevante al usuario acerca de estas.

- Cargar un archivo txt para ver mis asignaturas:

En está pantalla el sistema requerirá una ruta donde se guarde un archivo txt previamente generado por el programa en la opción 1, lo leerá y lo mostrará al usuario para su visualización.

- Consultar asignaturas disponibles

En está pantalla el sistema simplemente muestra todas las asignaturas disponibles.

A continuación se presentan las funcionalidades iniciales para el programa.

- ***Organizar ruta académica.***

- *Descripción:* El usuario ingresa el nombre de las asignaturas que va a cursar junto con los prerrequisitos que estas piden para ser cursadas y de acuerdo a esto el programa sugiere el semestre en el que debería ver la asignatura.

- *Acciones iniciadoras y comportamiento esperado*:

  1. Usuario selecciona opción Organizar ruta académica, seguido a esto el sistema le pregunta al usuario el número de asignaturas a organizar.
  2. El usuario ingresa el nombre de n asignaturas que desee cursar junto con el nombre de sus prerrequisitos.
  3. De acuerdo a los prerrequisitos de cada asignatura el sistema le asigna un semestre para cursarla.
  4. Una vez las asignaturas tengan un semestre asignado, el programa le muestra al usuario las asignaturas y el semestre que se le fue asignado para cursarla.
  5. El sistema genera un archivo de texto plano donde queda guardada la información para el usuario.

- ***Cargar archivo para ver asignaturas.***

- *Descripción:* El usuario podrá cargar un archivo txt para que pueda visualizar las asignaturas que va a cursar.

- *Acciones iniciadoras y comportamiento esperado*:

  1. Usuario selecciona opción *Cargar un archivo txt para ver mis asignaturas.*
  2. El sistema pregunta al usuario por la dirección del directorio donde se encuentra el archivo de texto plano. El usuario debe poner la dirección de la carpeta donde está el archivo.
  3. El sistema pregunta el nombre del archivo y el usuario debe poner el nombre del archivo sin la extensión .txt.
  4. Si la ruta es incorrecta, el sistema le informará al usuario y le pedirá que intente de nuevo.
  5. Si la ruta es correcta, el sistema lee el archivo y organiza las asignaturas por semestre y permite su visualización.

- ***Sugerir ruta académica.***

- *Descripción:* De acuerdo a la o las asignaturas que el usuario desee cursar, el software le sugerirá un programa académico que contiene está asignatura, así cómo la ruta a seguir para culminar el programa académico.

- *Acciones iniciadoras y comportamiento esperado*:

  6. El usuario ingresa el nombre de una asignatura que desee cursar.
  7. El sistema busca la asignatura.
  8. El sistema busca los programas asociados.
  9. De acuerdo al o los programas académicos a los que pertenece la asignatura, el sistema sugerirá una o varias rutas a seguir.

- ***Obtener información de asignaturas.***

- *Descripción:* El usuario ingresa el nombre de una asignatura con la intención de obtener información adicional, cómo el número de créditos, componente, entre otros.

  - Descripción breve de la funcionalidad.

- *Acciones iniciadoras y comportamiento esperado*:
  1. Usuario selecciona opción *obtener información de asignaturas*.
  2. El sistema pregunta al usuario por el nombre de la asignatura a buscar.
  3. El sistema busca la asignatura y muestra la información en pantalla al usuario.
  4. Si no se encuentra la asignatura, el sistema comunicará al usuario que la asignatura no fue encontrada..

.

V. DESCRIPCIÓN DE LA INTERFAZ DE USUARIO PRELIMINAR

Para la página inicial de la aplicación se pensó que tenga una barra de navegación con los botones de *inicio*, que llevará al usuario a la página inicial, *Planifica tu estudio,* que llevará al usuario al apartado donde elige asignaturas y planifica la ruta académica, *Tutorial,* que llevará al usuario a un video demostrativo de cómo usar el producto y por último un botón llamado *¿Qué somos?* que llevará al usuario a una sección donde se explica que es el producto y para qué sirve. También se mostrará el nombre del producto (que puede cambiar, no está decidido del todo) y algunas funciones cómo se muestra en la imagen [6]. En la página planifica estudio se dispondrán

de links para dirigirse a las demás funcionalidades. En la imagen 8 se puede ver el boceto para la página de organización de tus asignaturas, en la que habrá un campo para ingresar el nombre de la asignatura, junto con sus créditos, y prerrequisitos. Sumado a esto un botón *Ingresar* para proporcionarle los datos al sistema y un botón *Organizar* para cumplir con la funcionalidad *Organiza tus asignaturas*

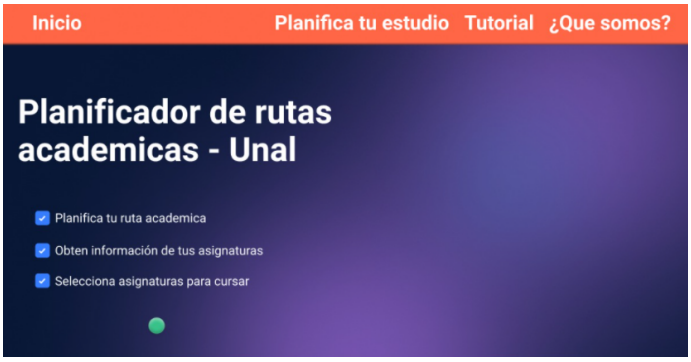

Imagen 1.Boceto página inicial.

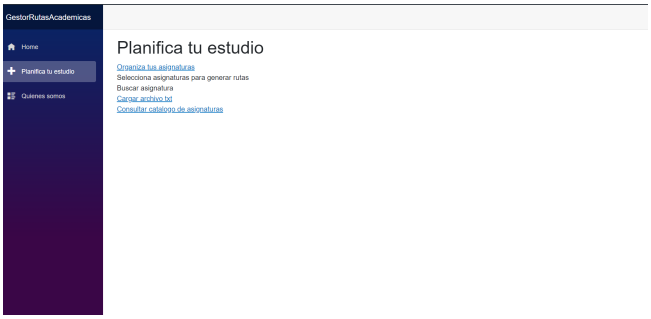

Imagen 2.Boceto página planifica tu estudio

Imagen 3.Boceto página Organiza tus asignaturas.

VI. ENTORNOS DE DESARROLLO Y DE OPERACIÓN

El entorno de desarrollo usado es Blazor, el cual es un framework desarrollado por Microsoft que permite desarrollar PWA. Una PWA es cómo un híbrido entre una aplicación web y una aplicación de escritorio, es decir, una PWA es una página web cómo otras pero utiliza service workers que permiten que sea ejecutada como una aplicación normal más que como aplicación web. Una vez que el usuario corre la aplicación en su navegador por primera vez mientras está Online el navegador descarga los recursos necesarios para poder operar Offline [4].

Blazor funciona principalmente bajo el lenguaje de programación C#, también desarrollado por Microsoft. Del mismo modo, utiliza Css para darle estilo a las páginas web y Html para darle la estructura a las páginas. Blazor es una alternativa para desarrollar aplicaciones web ya que usa C# en lugar de JavaScript(Js) . A pesar de lo anterior, es posible llamar librerías de Js que se consideren útiles para el desarrollo del proyecto[5].

Debido a que lo que se quiere desarrollar es una PWA, ésta operará en el navegador del usuario y por tanto el sistema operativo en el que se ejecute puede ser cualquiera, siempre y cuando cuente con un navegador y del mismo modo, puede ser ejecutado por cualquier configuración de Hardware.

VII. PROTOTIPO DE SOFTWARE INICIAL

En el siguiente enlace se encuentra el repositorio donde se encuentra el ejecutable del programa y su código fuente, además de varios documentos adicionales.
https://github.com/Jrativa/Gestor-de-Cursos-y-Rutas-Acad-micas

En la carpeta *dist* se encuentra la carpeta *bin* dentro está se debe dirigir a *Debug* y posteriormente a *net6.0* dentro de está carpeta está el ejecutable llamado *GestorRutasPrototipo.exe.*
En la carpeta *Scr* se encuentra el código fuente del programa.

VIII. IMPLEMENTACIÓN Y APLICACIÓN DE LAS ESTRUCTURAS DE DATOS

En este prototipo se implementaron las siguientes estructuras de datos.

- **Lista:**
  Implementación: Para el uso de listas, se hizo uso de listas enlazadas o LinkedList, en la cual, se realizó una clase *Nodo*, la cual tiene cómo atributos un atributo de tipo *nodo* que apunta al siguiente nodo y el *dato* que se guarda en el nodo. La clase *Nodo* hace contiene los métodos *Print,* para imprimir el dato del nodo y si hay nodos subsecuentes imprimir sus datos de la misma manera, *AddNode***,** para añadir un nodo con un dato que se quiera insertar, *AddSorted,* para añadir un dato en un nodo de manera que los valores de la lista queden ordenados.

  De igual manera se creó una clase *MyLinkedList* que tiene cómo atributo dos datos de tipo *Nodo,* uno guarda una referencia al primer nodo o cabeza de la lista y el otro guarda una referencia al último nodo o final de la lista. La lista implementa los métodos *Empty* (Vaciar la lista), *AddToEnd* (Añadir un dato al final de la lista), *AddToBeginnig* (Añadir un dato al inicio de la lista), *GetIndexOf* (Obtener el índice de

un dato en la lista), *GetLength*(Obtener la longitud de la lista), *GetValue*(Obtener el dato guardado en la lista según el índice), *GetNodeOf*(Obtener el nodo de la lista según el índice) , *Print*(Mostrar los datos guardados en la lista), *Remove*(Remover un dato de la lista) *Update* (Actualizar un dato de la lista).

La Linkedlist se utilizó para guardar las asignaturas que se ofertan, así cómo para guardar listas de prerrequisitos y programas asociados a las asignaturas con el fin de sugerir rutas académicas.

- **Cola:** Para implementar la cola, se creó una clase *MyQueue* que hereda de la clase *MyLinkedList* e implementa los métodos *enqueue()* (emplea el método *AddToEnd* de la lista enlazada) , *dequeue()* (emplea el método *Remove(0)* de la lista enlazada) , *front()* (emplea el método *GetValue(0)* de la lista enlazada) , *back()* (retorna el dato del nodo final).

  La cola fue vital para que en la opción de *Organizar asignaturas,* el usuario pudiera ingresar los nombres de las asignaturas y de está manera darle un turno a cada asignatura para calcular el semestre en el que se deberían cursar.

- **Pilas:** Para implementar la pila, se creó una clase *MyStack* que hereda de la clase *MyLinkedList* e implementa los métodos *push()* (emplea el método *AddToBeginning* de la lista enlazada) , *pop()* (emplea el método *Remove(0)* de la lista enlazada) , *peek()* (emplea el método *GetValue(0)* de la lista enlazada).

  La pila fue útil al momento de organizar las asignaturas de acuerdo al semestre que se les calculó previamente gracias a la cola.

IX. PRUEBAS DEL PROTOTIPO Y ANÁLISIS COMPARATIVO

Se escogieron tres funcionalidades para realizar pruebas para varios tamaños de datos de prueba (n), los cuales fueron:

- 10 mil datos,
- 100 mil datos,
- 1 millón de datos,
- 10 millones de datos, y
- 50 millones de datos.

- **Sugerir rutas académicas:** Se utilizó, en primer medida, LinkedList para está funcionalidad y para distintos tamaños de datos se realizaron 10 mediciones del tiempo de ejecución, se obtuvo el promedio y se graficó de la siguiente manera:

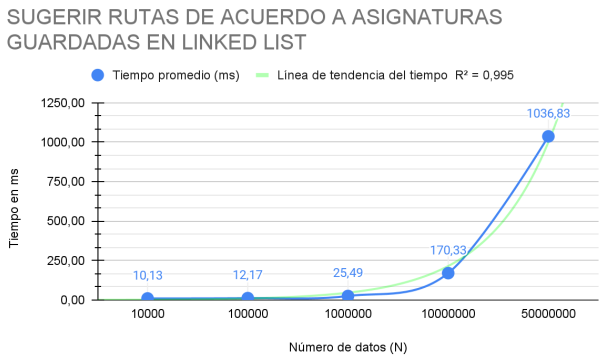

Gráfico 1. Tiempos de ejecución para Sugerir rutas académicas utilizando linked list.

Tabla 1. Tiempos de ejecución para Sugerir rutas académicas utilizando linked list.

N	Tiempo promedio (ms)
10000	10,13
100000	12,17
1000000	25,49
10000000	170,33
50000000	1036,83

En lugar de usar 100 millones de datos para la última prueba, usamos 50 millones de datos porque al momento de generar 100 millones de datos aleatorios el computador bajaba su rendimiento y el programa no corría óptimamente.

El algoritmo usado para la funcionalidad fue el siguiente:

```
public Asignatura buscarAsignatura(
    String Nombre,
    MyLinkedList<Asignatura> asignaturas)
{
    int n=asignaturas.GetLength();
    Node<Asignatura> aux= asignaturas.GetNodeOf(0);
    for (int i = 0; i <n;i++){
        if(aux.data.nombreAsignatura==Nombre)
            {
            return aux.data;
            }
        else{aux=aux.nextNode;}
    }
    return null;

}
1 reference
public void SugerirRutas(Asignatura asignatura){
    try{
        asignatura.MostrarProgramaAsociado();
    }
    catch (Exception e)
    {
        Console.WriteLine("No hay rutas", e.Message);
    }}
```

Imagen 4. Algoritmo utilizado para sugerir rutas.

El algoritmo usado emplea el método *buscarAsignatura,* el cual se ejecuta n veces en el peor caso ya que tiene que recorrer la lista de datos completa, por lo tanto, tiene una complejidad O(n) y el método *SugerirRutas,* se ejecuta una vez, pero invoca al método *MostrarProgramaAsociado* de la clase *Asignatura,* el cual se ejecuta tres veces, ya que se encarga de mostrar 3 programas asociados, por tanto la funcionalidad tiene una complejidad O(n).

En segunda medida, se realizó la misma funcionalidad pero guardando las asignaturas en listas basadas en Array y se obtuvieron los siguientes resultados:

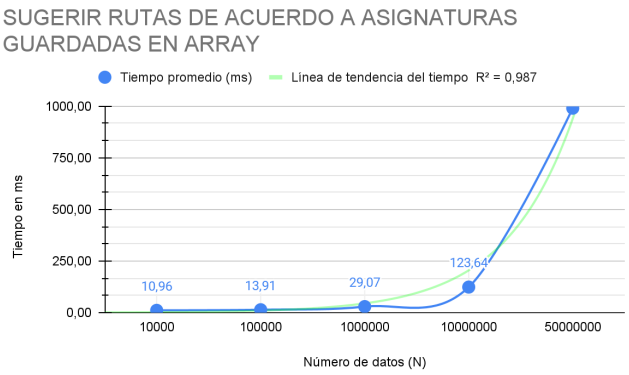

Gráfico 2. Tiempos de ejecución para Sugerir rutas académicas utilizando Array.

Tabla 2. Tiempos de ejecución para Sugerir rutas académicas utilizando Array.

N	Tiempo promedio (ms)
10000	10,96
100000	13,91
1000000	29,07
10000000	123,64
50000000	990,26

El algoritmo utilizado fue el mismo, con la diferencia de que *buscarAsignaturas* recibe un array en lugar de una lista enlazada. Es decir que el algoritmo tiene una complejidad de O(n).

En los datos proporcionados por las gráficas y tablas se aprecia que no existe demasiada diferencia en términos de tiempo entre la lista enlazada y el array. Por ejemplo, para n igual o menor a 1000000 de datos, la lista enlazada tuvo un rendimiento un poco mejor que el array pero para n igual a 10000000 y n igual a 50000000 el array tuvo un rendimiento un poco mejor que la lista enlazada. Debido a esto decidimos utilizar la lista enlazada para el prototipo de software debido a que la lista enlazada es dinámica mientras que el array no y no existen demasiadas diferencias en términos de tiempo de ejecución.

Esta prueba fue realizada en un procesador :Intel(R) Core(TM) i7-9750 H CPU @ 2.60GHz 2.60 GHz.

- **Cargar archivo para ver asignaturas:** Se utilizó, en primera medida, una pila basada en linked list para está funcionalidad y para distintos tamaños de datos se realizaron 10 mediciones del tiempo de ejecución, se obtuvo el promedio y se graficó de la siguiente manera.

Gráfico 3. Tiempos de ejecución para cargar, ordenar y ver asignaturas utilizando pilas basadas en linked list.

Tabla 3. Tiempos de ejecución para ordenar y ver asignaturas utilizando pilas basadas en linked list.

N	Tiempo promedio (ms)
1000	20,01
10000	48,52
100000	203,73
1000000	1831,30
10000000	64878,20

En lugar de usar 100 millones de datos para la última prueba, usamos 1000 datos porque al momento de generar 100 millones de datos aleatorios el computador bajaba su rendimiento y el programa no corría óptimamente.

El algoritmo usado para la funcionalidad fue el siguiente:

```
public static void LeerArchivo(string rutaArchivo)
{
    string linea;
    MyLinkedList<string> renglones = new MyLinkedList<string>();
    try
    {
        StreamReader lector = new StreamReader(rutaArchivo);
        linea = lector.ReadLine();
        Console.WriteLine(Int32.Parse(linea.Split(" ")[1]) + 3);
        MyStack<string> renglonesApilados1 = new MyStack<string>();
        MyStack<string> renglonesApilados2 = new MyStack<string>();
        MyStack<string> renglonesApilados3 = new MyStack<string>();
        MyStack<string> renglonesApilados4 = new MyStack<string>();
        MyStack<string> renglonesApilados5 = new MyStack<string>();
        MyStack<string> renglonesApilados6 = new MyStack<string>();
        MyStack<string> renglonesApilados7 = new MyStack<string>();
        MyStack<string> renglonesApilados8 = new MyStack<string>();
        MyStack<string> renglonesApilados9 = new MyStack<string>();
        MyStack<string> renglonesApilados10 = new MyStack<string>();
        while (linea != null)
        {
            renglones.AddToEnd(linea);
            switch ((linea.Split(" ")[1]))
            {
                case "1": renglonesApilados1.Push(linea); break;
                case "2": renglonesApilados2.Push(linea); break;
                case "3": renglonesApilados3.Push(linea); break;
                case "4": renglonesApilados4.Push(linea); break;
                case "5": renglonesApilados5.Push(linea); break;
                case "6": renglonesApilados6.Push(linea); break;
                case "7": renglonesApilados7.Push(linea); break;
                case "8": renglonesApilados8.Push(linea); break;
                case "9": renglonesApilados9.Push(linea); break;
                case "10": renglonesApilados10.Push(linea); break;

            }
            linea = lector.ReadLine();
        }
        lector.Close();
        mostrarRenglones(renglonesApilados1);
        mostrarRenglones(renglonesApilados2);
        mostrarRenglones(renglonesApilados3);
        mostrarRenglones(renglonesApilados4);
        mostrarRenglones(renglonesApilados5);
        mostrarRenglones(renglonesApilados6);
        mostrarRenglones(renglonesApilados7);
        mostrarRenglones(renglonesApilados8);
        mostrarRenglones(renglonesApilados9);
        mostrarRenglones(renglonesApilados10);
    }
```

Imagen 5. Algoritmo utilizado para leer y organizar archivos.

El algoritmo usado emplea el método *LeerArchivo,* el cual se ejecuta n veces, pues tiene que leer cada línea del archivo, por lo tanto, tiene una complejidad O(n) y el método *MostrarRenglones,* se ejecuta dependiendo del tamaño de cada pila que entre todas las pilas forman n veces por lo tanto sería de O(n). En total el orden sería 2O(n), omitimos la constante y quedamos con que la funcionalidad tiene un orden de O(n). Si bien en la gráfica N se aprecia cómo si fuera un crecimiento cuadrático, esto es porque está gráfica está en otra escala para poder observar con mayor claridad los tiempos para cada valor de n.

En segunda medida, se realizó la misma funcionalidad pero con pilas basadas en Array y se obtuvieron los siguientes resultados:

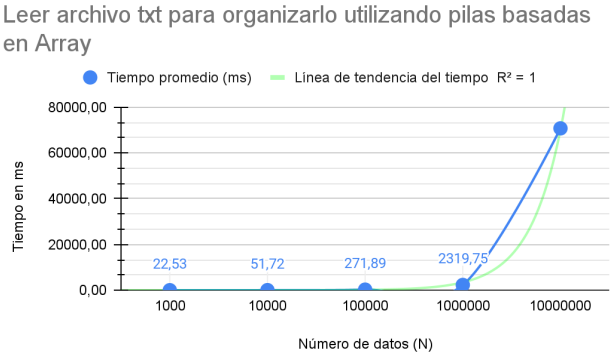

Gráfico 4. Tiempos de ejecución para cargar archivos, ordenar y ver asignaturas utilizando pilas basadas en Array.

Tabla 4. Tiempos de ejecución para ordenar y ver asignaturas utilizando pilas basadas en array.

N	Tiempo promedio (ms)
1000	22,53
10000	51,72
100000	271,89
1000000	2319,75
10000000	70783,23

En esté caso, el algoritmo usado fue el mismo, es decir tiene complejidad O(n). Si bien en la gráfica N se aprecia cómo si fuera un crecimiento cuadrático, esto es porque está gráfica está en otra escala para poder observar con mayor claridad los tiempos para cada valor de n. Se puede apreciar en las tablas 3 y 4 que la pila basada en lista enlazada tuvo un mejor desempeño cumpliendo la tarea para cada valor de n. Por lo tanto decidimos emplear la pila basada en lista enlazada para está funcionalidad. Esta prueba fue realizada en un procesador :Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.60 GHz.

- **Organizar asignaturas :**
  Está prueba se realizó con la implementación de colas basadas en Linked List, al igual que las pruebas anteriores, se realizaron 10 pruebas con 5 tamaños de datos diferentes N, se obtuvo el promedio y se gráfico de la siguiente manera:

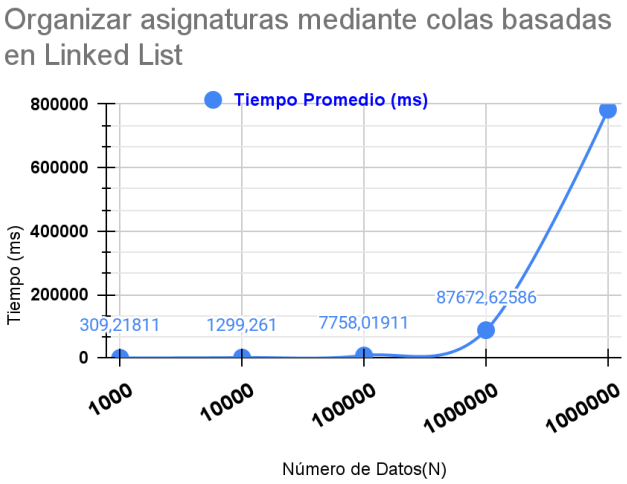

Gráfico 5. Tiempos de ejecución para organizar asignaturas utilizando colas basadas en Linked List.

TABLA 5. TIEMPOS DE EJECUCIÓN PARA ORGANIZAR COLAS BASADAS EN LINKED LIST.

N	Tiempo Promedio (ms)
1000	309,21811
10000	1299,261
100000	7758,01911
1000000	87672,62586
10000000	782795,3026

En lugar de usar 100 millones de datos para la última prueba, usamos 1000 datos porque al momento de generar 100 millones de datos aleatorios el computador bajaba su rendimiento y el programa no corría óptimamente.

El algoritmo que se utilizó fue el siguiente:

```
public static void pedirAsignaturas()
{
    MyQueue<Asignatura> asignaturas = new MyQueue<Asignatura>();
    Console.WriteLine("¿Cuantas asignaturas quieres planificar ?");
    string numero = Console.ReadLine();
    int n = 1000000;
    for (int i = 0; i < n; i++)
    {
        Console.WriteLine("Nombre de la asignatura:");
        string nombreAsignatura = ""+i;
        Console.WriteLine("Nombre de prerrequisitos(separados por comas):");

        string prerrequisito = " ";
        MyLinkedList<string> prerrequisitos = new MyLinkedList<string>();
        Asignatura aux = new Asignatura(); aux.nombreAsignatura = nombreAsignatura;
        foreach (string materia in prerrequisito.Split(separator: ","))
        {
            prerrequisitos.AddToEnd(data:materia.Trim());
        }
        aux.NombrePrerrequisito = prerrequisitos;
        asignaturas.enqueue(aux);
    }
    Organizar(asignaturas);
}
```

Imagen 6. Algoritmo utilizado para organizar asignaturas

```
1 reference
public static void Organizar(MyQueue<Asignatura> Asignaturas)
{
    MyQueue<Asignatura> Asignaturas2 = new MyQueue<Asignatura>();
    Console.WriteLine("Introduzca la ruta del directorio donde quiere que se guarde su archivo: ");
    string rutaArchivo = Console.ReadLine();
    Console.WriteLine("Introduzca el nombre para su archivo:");
    string nombreArchivo = Console.ReadLine();
    try
    {
        using (StreamWriter outputFile = new StreamWriter(rutaArchivo + (char)(92) + nombreArchivo + ".txt"))
        {
            Asignatura aux1 = new Asignatura();
            for (int i = 0; i < Asignaturas.GetLength(); i++)
            {
                aux1 = Asignaturas.dequeue(); i--;
                Asignaturas2.enqueue(aux1);
                aux1.SetSemestre(AsignarSemestre(aux1, Asignaturas, Asignaturas2));
                outputFile.WriteLine("Semestre " + aux1.GetSemestre() + " --- " + aux1.nombreAsignatura);
            }
        }
    }
    catch { Console.WriteLine("La ruta ingresada no es valida"); }
}
```

El algoritmo usado emplea el método *pedirAsignaturas,* el cual se ejecuta n veces,donde n es la cantidad de asignaturas que se van agregando a la consola, teniendo una complejidad O(n). Esté método invoca al método *Organizar* al final, el cual va leyendo cada asignatura guardada en la cola, es decir se ejecuta n veces, con complejidad O(n) y les va asignando un semestre con el método *Asignar semestre* el cual recibe la asignatura y busca sus prerrequisitos en las asignaturas ingresadas con el método *Buscar asignatura* el cual cómo ya vimos anteriormente tiene complejidad de O(n), por lo tanto el método Organizar tiene complejidad de O(n^2). Entonces la totalidad de la funcionalidad tiene una complejidad de O(n^2) ya que tomamos la complejidad de mayor orden.

En segunda medida, se realizó la misma funcionalidad pero con colas basadas en Array y se obtuvieron los siguientes resultados:

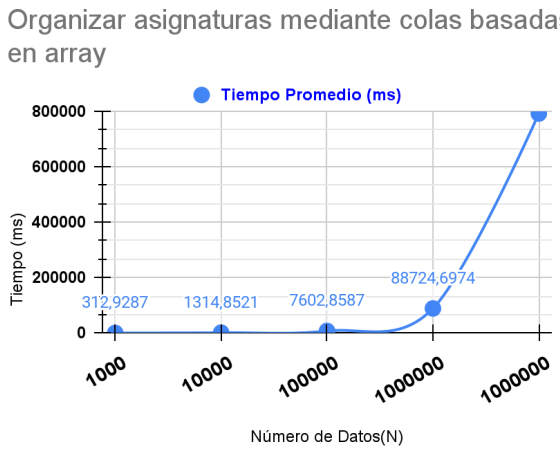

GRÁFICO 6. TIEMPOS DE EJECUCIÓN PARA ORGANIZAR ASIGNATURAS UTILIZANDO COLAS BASADAS EN ARRAY..

TABLA 6. TIEMPOS DE EJECUCIÓN PARA ORGANIZAR COLAS BASADAS EN ARRAY.

N	Tiempo Promedio (ms)
1000	312,9287
10000	1314,8521
100000	7602,8587
1000000	88724,6974
10000000	792188,8462

En esté caso, el algoritmo usado fue el mismo, es decir tiene complejidad O(n^2). Las dos implementaciones de la cola tuvieron resultados similares en cuanto a tiempo de ejecución por lo que decidimos usar la cola basada en lista enlazada, ya que fue un poco superior.

Esta prueba fue realizada en un Procesador Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, 2808 Mhz, 4 procesadores principales, 8 procesadores lógicos.

X. INFORMACIÓN DE ACCESO AL VIDEO DEMOSTRATIVO DEL PROTOTIPO DE SOFTWARE

En el siguiente enlace encontrará una demostración del prototipo de software desarrollado: https://www.loom.com/share/e6b818fd14fe4799982a84cff78f0e48

XI. ROLES Y ACTIVIDADES

INTEGRANTE	ROL(ES)	ACTIVIDADES REALIZADAS (Listado)
Julian Bonilla Mórtigo	Líder	Consulta a miembros de equipo acerca de sus habilidades
	Experto	Coordinación de actividades
Danny Bañol Osorio	Observador	Pendiente de Actividades a realizar
	Animador	Charlas motivacionales
	Secretario	Organización de tareas
José Luis Rativa Medina	Coordinador	Programación de reuniones
	Investigador	Consultas acerca de blazor y aspnet.

XII. DIFICULTADES Y LECCIONES APRENDIDAS

Durante el desarrollo del proyecto se presentaron varias dificultades, al principio, el lenguaje de programación que se eligió para realizar el proyecto(C#), a pesar de que todos nos pusimos de acuerdo, un integrante del equipo no había trabajado nunca antes con este lenguaje, sin embargo, fue un "problema" fácil de solucionar con un poco de práctica.

Siguiendo en el orden de desarrollo, se complicó un poco el tema de las implementaciones de todas las estructuras de datos que eran requeridas, pues no teníamos la claridad de que estructura utilizar para las funciones definidas.

Dificultad en algunos conceptos, por ejemplo en "Linked List", donde fue necesario buscar bibliografía adicional para comprender mejor el tema e implementarlo de manera más óptima.

Una de las dificultades también que se presentaron, fue la organización y claridad de las cosas que se iban a realizar, así que aprendimos la lección de siempre tratar de planear con anterioridad las cosas que se van a realizar, dándole un orden específico y tiempos determinados para un mejor desarrollo de los trabajos en general.

Otra lección importante que aprendimos, fue a tener siempre varias alternativas al momento de resolver una tarea o problema y no esperar que las cosas salgan perfectamente a cómo se planearon, sino tener algún tipo de camino alternativo para poder cambiar o corregir errores de manera más veloz.

XIII. REFERENCIAS BIBLIOGRÁFICAS

[1] "Sólo el 28% de los estudiantes se gradúa a tiempo de la Universidad ",2010 Portafolio https://www.portafolio.co/economia/finanzas/28-estudiantes-gradua-universidad-144190

[2] Ming Chen, Xuan Huang, Hongyu Chen, Xuemei Su & Jasmine Yur-Austin (2021): Data driven course scheduling to ensure timely graduation, International Journal of Production Research, DOI: 10.1080/00207543.2021.1916118

[3] Gonzalo Gabriel Méndez, Luis Galárraga and Katherine Chiluiza. 2021. Showing Academic Performance Predictions during Term Planning: Effects on Students' Decisions, Behaviors, and Preferences. In CHI Conference on Human Factors in Computing Systems(CHI'21), May 8–13, 2021, Yokohama, Japan. ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3411764.3445718

[4] Guardex, "Build Progressive Web Applications with ASP.NET Core Blazor WebAssembly", *Microsoft,*[Online], Available: https://docs.microsoft.com/en-us/aspnet/core/blazor/progressive-web-app?view=aspnetcore-6.0&tabs=visual-studio

[5] Blazor," *Wikipedia*. Sep. 22, 2021. Accessed: Nov. 26, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Blazor&oldid=1045703149