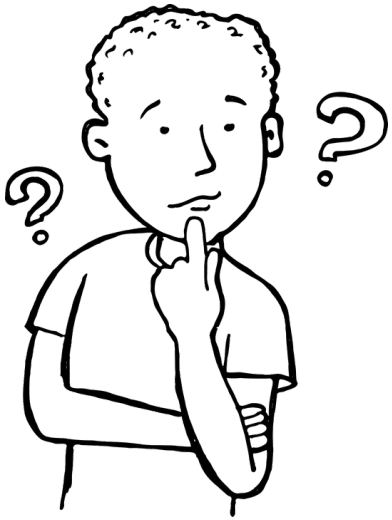
Gestor de Cursos y Rutas Académicas.

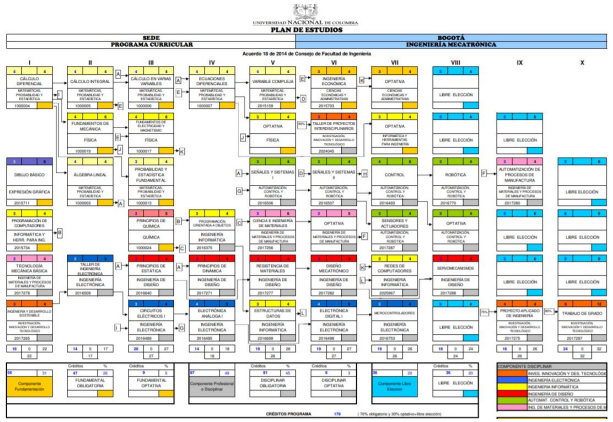
Dany Bañol Osorio.
Julián Bonilla Mórtigo
José Luis Rativa Medina

Equipo G

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ

Problema que se resolvió





Mantenimiento del servicio

En estos momentos se están realizando tareas de mantenimiento por lo que el servicio no se encuentra operativo.

Por favor inténtelo más tarde o, en caso de persistir el problema, diríjase al administrador

Rogamos disculpen las molestias que esta actuación pueda ocasionar.

Requerimientos funcionales

Menú de la Consola del prototipo inicial:

```
Bienvenido al gestor de rutas academicas
Opciones:
1. Organizar Asignaturas
2. Selecciona asignaturas para generar rutas
3. Buscar asignatura
4. Cargar un archivo txt para ver mis asignaturas
5. Consultar asignaturas disponibles
Que deseas hacer :
```

1) **Organizar Asignaturas y Ruta Académica:**

```
1
¿Cuantas asignaturas quieres planificar ?
2
Nombre de la asignatura:
POO
Nombre de prerrequisitos(separados por comas):
PB
Nombre de la asignatura:
PB
Nombre de prerrequisitos(separados por comas):

Introduzca la ruta del directorio donde quiere que se guarde su archivo:
C:\Users\josel\OneDrive\Documentos\Ing.Sis\2021-2\DataStructures\Proyecto\Prototipo2
Introduzca el nombre para su archivo:
Prueba
5
Semestre 1 --- PB
Semestre 2 --- POO
Presione 1 para volver al menu de opciones, 2 para salir del programa
```

Requerimientos funcionales

2) Seleccionar Asignaturas para generar rutas:

```
2
Introduzca el nombre de la asignatura que quiere cursar:
Fisica computacional
Si quiere cursar  "Física computacional"le recomendamos estudiar el programa: 2418
```

3) Buscar Asignaturas:

```
Que deseas hacer :
3
Introduzca el nombre de la asignatura que quiere buscar:
Fisica computacional
Codigo: 2027632
Nombre:  "Física computacional"
Creditos: 3
Componente:  "Formación Disciplinar o Profesional"
Presione 1 para volver al menu de opciones, 2 para salir del programa
```

4) Cargar archivo para ver asignaturas

```
4
Introduzca la ruta del directorio donde está su archivo:
C:\Users\josel\OneDrive\Documentos\dumps
Nombre de su archivo:
Prueba
4
Semestre 1 --- CALCULO DIFERENCIAL
Semestre 2 --- CALCULO INTEGRAL
Semestre 3 --- CALCULO VECTORIAL
Presione 1 para volver al menu de opciones, 2 para salir del programa
```

5) Consultar Asignaturas Disponibles:

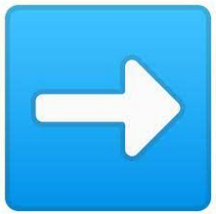
```
5
[
  {
    "codigo": 2015168,
    "nombre": "Fundamentos de matemáticas",
    "creditos": 4,
    "obligatoria": true,
    "prerrequisito": null,
    "correquisito": null,
    "componente": "Fundamentación",
    "programas": [2418]
  },
  {
    "codigo": 2015181,
    "nombre": "Sistemas numéricos",
    "creditos": 4,
    "obligatoria": true,
    "prerrequisito": "2015168",
    "correquisito": null,
    "componente": "Fundamentación",
    "programas": [2418]
  },
```

Implementación de nuevas estructuras de datos en la solución del problema planteado

Listas enlazadas

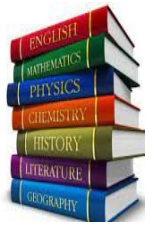
- Almacenamiento

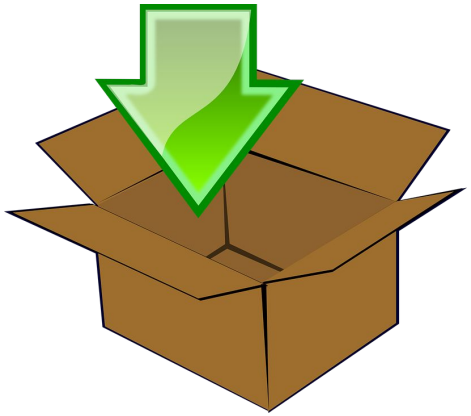
implementa a



Tablas Hash

- Almacenamiento





Prerrequisitos

Condición 1 Tipo M ¿Todas? [N] Número asignaturas [1]
1000005-B Cálculo Integral
2000916 Matematicas ii

Condición 2 Tipo M ¿Todas? [N] Número asignaturas [1]
1000003-B Álgebra Lineal
2000916 Matematicas ii

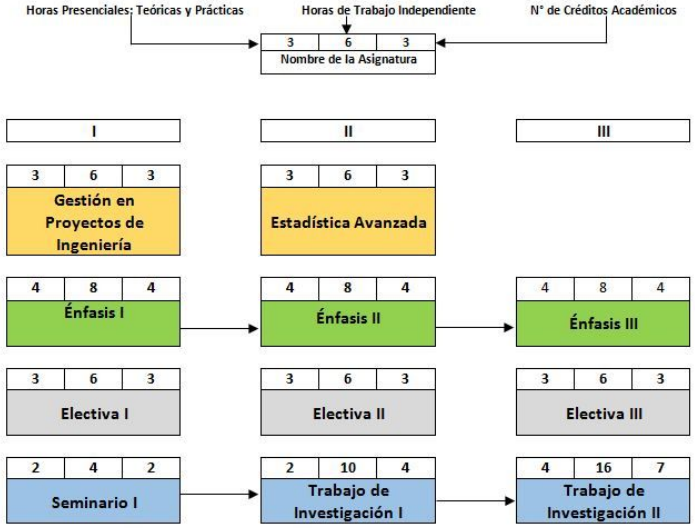
Implementación de nuevas estructuras de datos en la solución del problema planteado

Pila

- Lectura de archivo

Árbol AVL

- Asignación de semestre para asignaturas



Análisis comparativo del uso de las nuevas estructuras de datos implementadas (Gráficas)

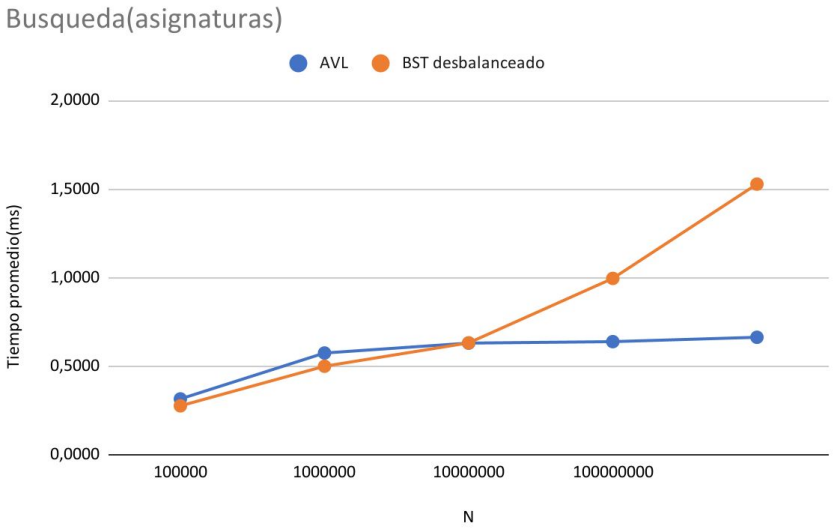


Gráfico 1. Tiempos de ejecución método "buscar" en árbol AVL y BST desbalanceado.

Gráfico 2. Tiempos de ejecución método "buscar" en una lista enlazada.

De la gráfica 1 y 2, podemos concluir que, con el método "buscar" un dato, el crecimiento en el árbol AVL tiende a comportarse de manera logarítmica, mientras que en el árbol BST tiende a ser lineal, al igual que en la lista enlazada .

Análisis comparativo del uso de las nuevas estructuras de datos implementadas (Gráficas)

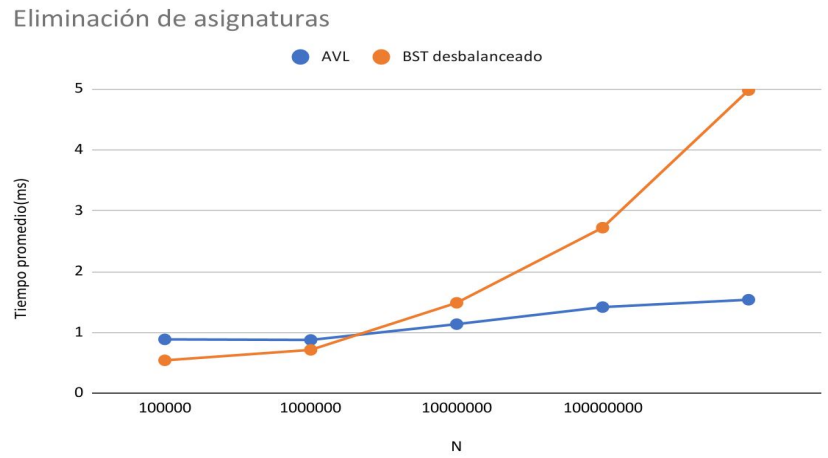


Gráfico 3. Tiempos de ejecución método "eliminar" en árbol AVL y BST desbalanceado.

Gráfico 4. Tiempos de ejecución método "eliminar" en Listas enlazadas.

De la gráfica 3 y 4, podemos concluir que, con el método "eliminar" un dato, el crecimiento en el árbol AVL tiende a comportarse de manera logarítmica, mientras que en el árbol BST y la lista enlazada tiende a ser lineal.

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE BOGOTÁ

Pruebas del uso de las nuevas estructuras de datos implementadas

Se compararon los árboles entre ellos y con la lista enlazada, la cual ya se tenía previamente para saber qué estructura de datos usar en el proyecto.

Nombre de la funcionalidad	Tipo(s) de estructura de datos	Cantidad de datos probados	Análisis realizado (Notación Big O)	Tiempos de ejecución (ms)
Búsqueda de datos (asignaturas)	Árbol AVL	10000	O(logn)	0,3167
		100000	O(logn)	0,5758
		1000000	O(logn)	0,6317
		10000000	O(logn)	0,6400
		100000000	O(logn)	0,6646
	Árbol BST	10000	O(n)	0,2777
		100000	O(n)	0,5012
		1000000	O(n)	0,6329
		10000000	O(n)	0,9968
		100000000	O(n)	1,5289
	Lista enlazada	10000	O(n)	1,189
		100000	O(n)	12,784
		1000000	O(n)	96,919
		10000000	O(n)	1056,623
		100000000	O(n)	8689,08

Tabla 1. Tiempos de ejecución del método “Buscar” en Árbol AVL, Árbol BST, Lista enlazada.

Nombre de la funcionalidad	Tipo(s) de estructura de datos	Cantidad de datos probados	Análisis realizado (Notación Big O)	Tiempos de ejecución (ms)
Eliminación de datos (asignaturas)	Árbol AVL	10000	O(logn)	0,8866
		100000	O(logn)	0,8771
		1000000	O(logn)	1,1366
		10000000	O(logn)	1,4167
		100000000	O(logn)	1,5377
	Árbol BST	10000	O(n)	0,5421
		100000	O(n)	0,7140
		1000000	O(n)	1,4872
		10000000	O(n)	2,7216
		100000000	O(n)	4,9806
	Lista enlazada	10000	O(n)	2,7263
		100000	O(n)	29,3203
		1000000	O(n)	222,2903
		10000000	O(n)	2.423,4480
		100000000	O(n)	24.339,1731

Tabla 2. Tiempos de ejecución del método “Eliminar” en Árbol AVL, Árbol BST, Lista enlazada.

Para búsqueda el mejor es el árbol AVL

Para eliminación el mejor es el árbol AVL

Pruebas y análisis comparativo del uso de las nuevas estructuras de datos implementadas- Árboles.

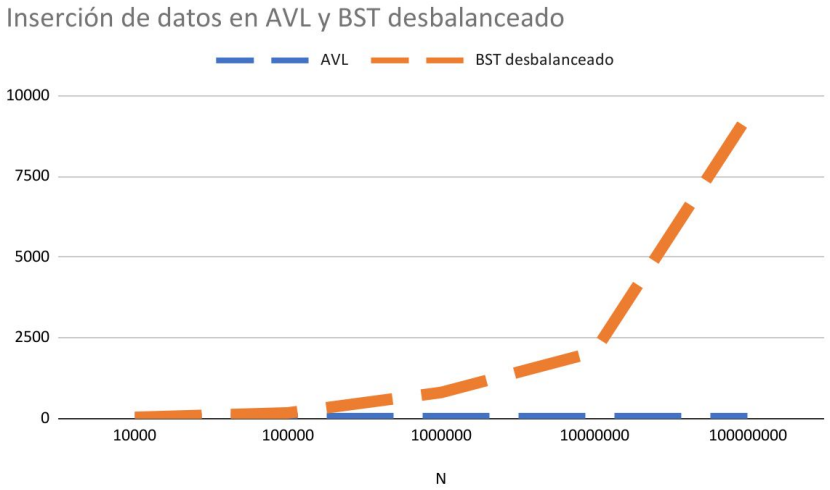


Gráfico 5. Tiempos de ejecución método "insertar" en árbol AVL y BST desbalanceado.

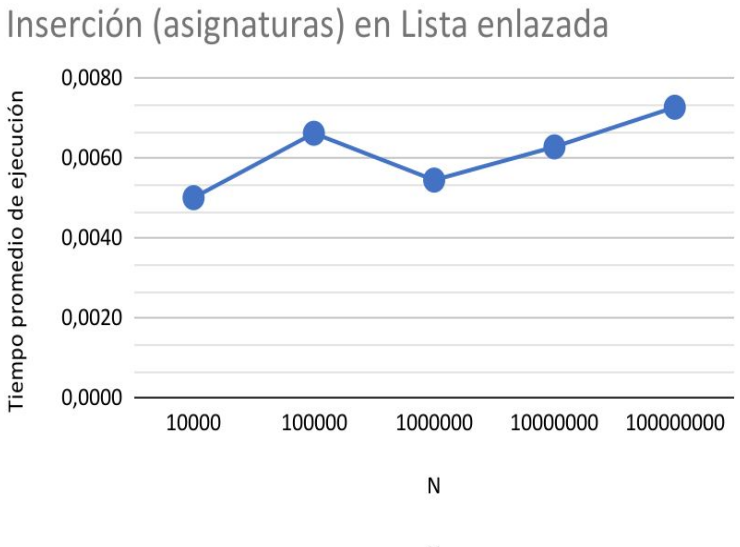


Gráfico 6. Tiempos de ejecución método "insertar" en Listas enlazadas.

De la gráfica 5 y 6, podemos concluir que, con el método "insertar" un dato, el crecimiento en el árbol AVL tiende a comportarse de manera logarítmica, mientras que en el árbol BST tiende a ser lineal y en la lista enlazada se comporta de manera constante.

Pruebas del uso de las nuevas estructuras de datos implementadas

Nombre de la funcionalidad	Tipo(s) de estructura de datos	Cantidad de datos probados	Análisis realizado (Notación Big O)	Tiempos de ejecución (ms)
Inserción de datos (Asignaturas)	Árbol AVL	10000	O(logn)	0,0242
		100000	O(logn)	0,0342
		1000000	O(logn)	0,0401
		10000000	O(logn)	0,0441
		100000000	O(logn)	0,0524
	Árbol BST	10000	O(n)	39,3727
		100000	O(n)	178,7520
		1000000	O(n)	811,5341
		10000000	O(n)	2061,2967
		100000000	O(n)	9358,2871
	Lista enlazada	10000	O(1)	0,0050
		100000	O(1)	0,0066
		1000000	O(1)	0,0055
		10000000	O(1)	0,0063
		100000000	O(1)	0,0073

Tabla 3. Tiempos de ejecución del método “Insertar” en Árbol AVL, Árbol BST, Lista enlazada.

Para inserción el mejor es la lista enlazada

Análisis comparativo de las Gráficas con respecto a la Complejidad Temporal

	Buscar Dato (Asignaturas)	Eliminar Dato (Asignaturas)	Insertar Datos (Asignaturas)
Árbol AVL	**O(log n)**	**O(log n)**	**O(log n)**
Árbol BST Desbalanceado	**O(n)**	**O(n)**	**O(n)**
Lista Linkeada	**O(n)**	**O(n)**	**O(1)**

Ya que el árbol AVL es mejor que el árbol desbalanceado y la lista enlazada en búsqueda y eliminación de datos, consideramos al árbol AVL cómo una mejor opción para almacenamiento de datos.

Pruebas y análisis comparativo del uso de las nuevas estructuras de datos implementadas- Árboles.

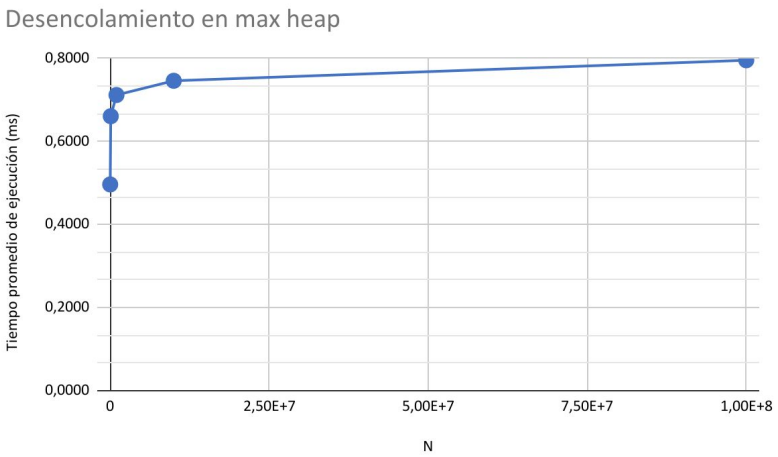


Gráfico 7 . Tiempos de ejecución método “desencolar” en max heap.

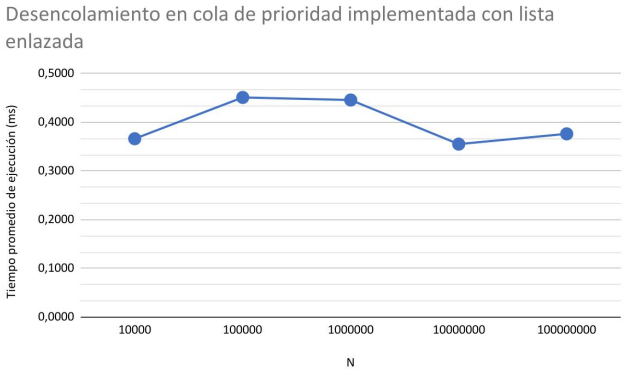


Gráfico 8. Tiempos de ejecución método “desencolar” en cola de prioridad implementadas con Listas enlazadas.

De la gráfica 7 y 8, podemos concluir que, con el método “Desencolar” un dato, el crecimiento en el Max Heap tiende a comportarse de manera logarítmica, mientras que en la cola de prioridad tiende a ser constante .

Pruebas y análisis comparativo del uso de las nuevas estructuras de datos implementadas- Heaps.

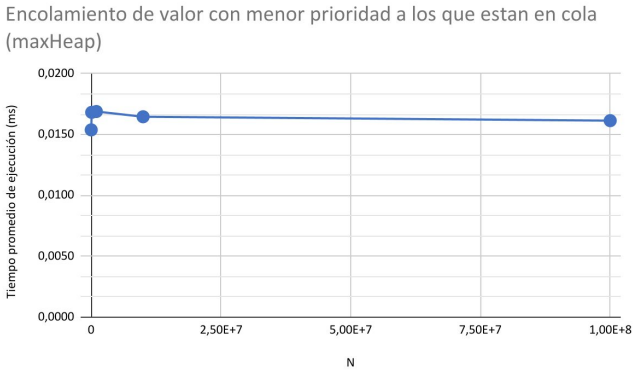


Gráfico 9 . Tiempos de ejecución método “encolar” con menor prioridad a los que están en cola (max heap).

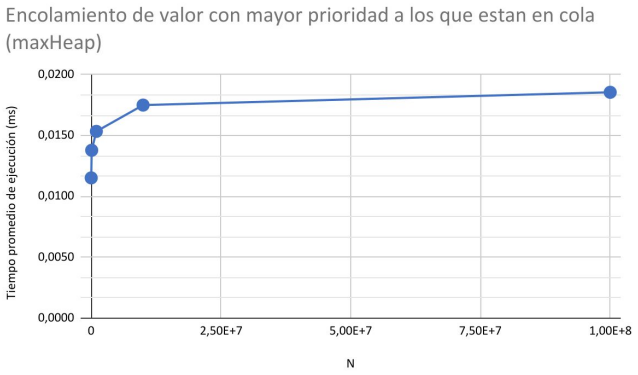


Gráfico 11 . Tiempos de ejecución método “encolar” con mayor prioridad a los que están en cola (max heap).

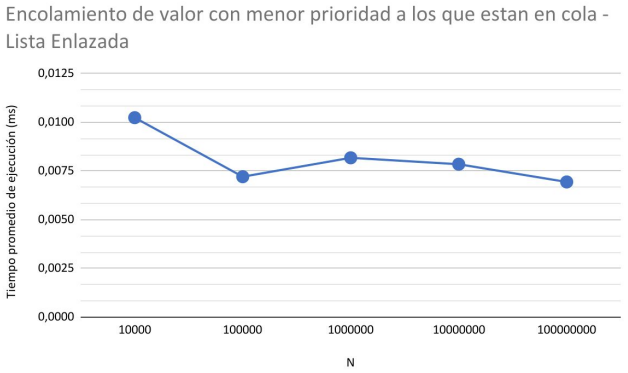


Gráfico 10. Tiempos de ejecución método “encolar” con menor prioridad a los que están en cola de prioridad implementadas con Listas enlazadas.

Gráfico 12. Tiempos de ejecución método “encolar” con mayor prioridad a los que están en cola de prioridad implementadas con Listas enlazadas.

Pruebas y análisis comparativo del uso de las nuevas estructuras de datos implementadas- Heaps.

Gráfico 13 . Tiempos de ejecución método "encolar" con valor de prioridad media con respecto a los que están en cola (max heap).

Gráfico 14. Tiempos de ejecución método "encolar" con valor de prioridad medio con respecto a los que están en cola de prioridad implementadas con Listas enlazadas.

De la gráfica 9 y 10, podemos concluir que, con el método "encolar valor con menor prioridad", el crecimiento en el Max Heap tiende a comportarse de manera constante al igual que en la cola de prioridad.

De las gráficas 11 y 12, podemos concluir, con el método "encolar valor con mayor prioridad", el crecimiento en el Max Heap tiende a comportarse de manera logarítmica, mientras que en la cola de prioridad tiende a ser lineal.

De las Gráficas 13 y 14, podemos decir que el crecimiento en el Max Heap tiende a ser logarítmico, mientras que en la cola de prioridad, tiende a ser lineal.

Pruebas del uso de las nuevas estructuras de datos implementadas

Nombre de la funcionalidad	Tipo(s) de estructura de datos	Cantidad de datos probados	Análisis realizado (Notación Big O)	Tiempos de ejecución (ms)
Desencolamiento	Max Heap	10000	O(logn)	0,4965
		100000	O(logn)	0,6608
		1000000	O(logn)	0,7119
		10000000	O(logn)	0,7461
		100000000	O(logn)	0,7955
	Cola de prioridad basada en Lista enlazada	10000	O(1)	0,3664
		100000	O(1)	0,4511
		1000000	O(1)	0,4459
		10000000	O(1)	0,3553
		100000000	O(1)	0,3764

Tabla 4. Tiempos de ejecución del método "Desencolar" en Max Heap y cola de prioridad basada en Lista enlazada.

.

Nombre de la funcionalidad	Nombre de la funcionalidad	Nombre de la funcionalidad	Análisis realizado (Notación Big O)	Tiempos de ejecución (ms)
Encolamiento de valor con mayor prioridad a los que estan en la cola (inicio de la cola)	Max Heap	10000	O(logn)	0,012
		100000	O(logn)	0,014
		1000000	O(logn)	0,015
		10000000	O(logn)	0,018
		100000000	O(logn)	0,02
	Cola de prioridad basada en Lista enlazada	10000	O(1)	0,0085
		100000	O(1)	0,0106
		1000000	O(1)	0,0111
		10000000	O(1)	0,0065
		100000000	O(1)	0,0086

Tabla 5. Tiempos de ejecución del método "Encolamiento de valor con mayor prioridad a los que están en la cola" en Max Heap y cola de prioridad basada en Lista enlazada.

Nombre de la funcionalidad	Nombre de la funcionalidad	Nombre de la funcionalidad	Análisis realizado (Notación Big O)	Tiempos de ejecución (ms)
Encolamiento de valor con menor prioridad a los que estan en la cola (final de la cola)	Max Heap	10000	O(1)	0,0154
		100000	O(1)	0,0168
		1000000	O(1)	0,0169
		10000000	O(1)	0,0165
		100000000	O(1)	0,0161
	Cola de prioridad basada en Lista enlazada	10000	O(1)	0,0102
		100000	O(1)	0,0072
		1000000	O(1)	0,0082
		10000000	O(1)	0,0079
		100000000	O(1)	0,0069

Tabla 6. Tiempos de ejecución del método "Encolamiento de valor con menor prioridad a los que están en la cola" en Max Heap y cola de prioridad basada en Lista enlazada.

Nombre de la funcionalidad	Nombre de la funcionalidad	Nombre de la funcionalidad	Análisis realizado (Notación Big O)	Tiempos de ejecución (ms)
Encolamiento promedio (posiciones internas de la cola)	Max Heap	10000	O(logn)	0,0066
		100000	O(logn)	0,0097
		1000000	O(logn)	0,0127
		10000000	O(logn)	0,0148
		100000000	O(logn)	0,0210
	Cola de prioridad basada en Lista enlazada	10000	O(n)	1,7673
		100000	O(n)	60,5152
		1000000	O(n)	169,1087
		10000000	O(n)	655,8323
		100000000	O(n)	3.314,6218

Tabla 7. Tiempos de ejecución del método "Encolamiento promedio" en Max Heap y cola de prioridad basada en Lista enlazada.

Análisis comparativo de las Gráficas con respecto a la Complejidad Temporal

	Desencolar	Encolamiento de valor con mayor prioridad a los que están en la cola (inicio de la cola)	Encolamiento de valor con menor prioridad a los que están en la cola (final de la cola)	Encolamiento promedio (posiciones internas de la cola)
Max-Heap	O(log n)	O(log n)	O(1)	O(log n)
Colas de prioridad basadas en linkedList	O(1)	O(1)	O(1)	O(n)

Puede parecer que la cola basada en lista enlazada es mejor que el montículo pero teniendo en cuenta que la mayoría de veces se encolan datos en las posiciones internas de una cola, la mejor opción para la cola de prioridad es el montículo ya que la operación que más se usa es O(logn) mientras que la misma operación es O(n) en la cola basada en lista enlazada.

Pruebas y análisis comparativo del uso de las nuevas estructuras de datos implementadas- Tablas Hash.

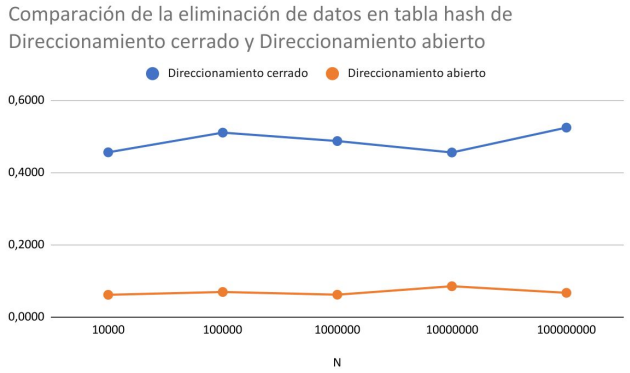


Gráfico 15 . Tiempos de ejecución método "eliminar" en Tablas Hash.

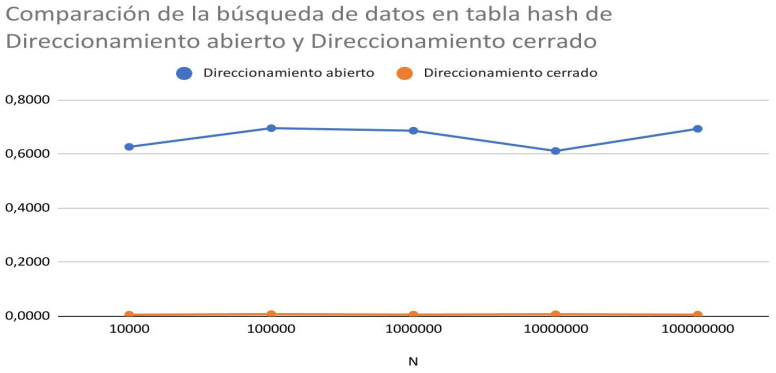


Gráfico 16. Tiempos de ejecución método "búsqueda" en tablas Hash.

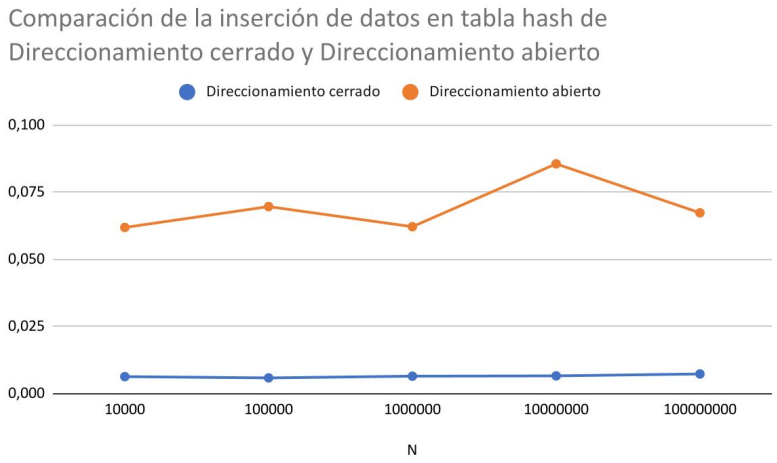


Gráfico 17. Tiempos de ejecución método "inserción" en tablas Hash.

De las Gráficas 15,16 y 17, se concluye que en las tablas hash de direccionamiento abierto y cerrado tienen un crecimiento constante al usar el método "eliminar", "buscar" e "insertar".

Pruebas del uso de las nuevas estructuras de datos implementadas

Al igual que se cambio la lista enlazada por el árbol AVL para almacenar la información de las asignaturas, al analizar la tabla hash se decidió cambiar el arbol AVL por la tabla hash para esta funcionalidad.

Nombre de la funcionalidad	Tipo(s) de estructura de datos	Cantidad de datos probados	Análisis realizado (Notación Big O)	Tiempos de ejecución (ms)
Eliminación de datos	Tabla Hash - Direccionamiento cerrado	10000	O(1)	0,4561
		100000	O(1)	0,5106
		1000000	O(1)	0,4873
		10000000	O(1)	0,4556
		100000000	O(1)	0,5245
	Tabla Hash - Direccionamiento abierto	10000	O(1)	0,0044
		100000	O(1)	0,0060
		1000000	O(1)	0,0063
		10000000	O(1)	0,0063
		100000000	O(1)	0,0047

Tabla 8. Tiempos de ejecución del método “Eliminar” en las tablas hash.

.

Nombre de la funcionalidad	Nombre de la funcionalidad	Nombre de la funcionalidad	Análisis realizado (Notación Big O)	Tiempos de ejecución (ms)
Búsqueda de datos	Tabla Hash - Direccionamiento cerrado	10000	O(1)	0,6268
		100000	O(1)	0,6959
		1000000	O(1)	0,6865
		10000000	O(1)	0,6116
		100000000	O(1)	0,6936
	Tabla Hash - Direccionamiento abierto	10000	O(1)	0,0060
		100000	O(1)	0,0081
		1000000	O(1)	0,0064
		10000000	O(1)	0,0078
		100000000	O(1)	0,0063

Tabla 9. Tiempos de ejecución del método “Buscar”en tablas hash.

Nombre de la funcionalidad	Nombre de la funcionalidad	Nombre de la funcionalidad	Análisis realizado (Notación Big O)	Tiempos de ejecución (ms)
Inserción de datos	Tabla Hash - Direccionamiento cerrado	10000	O(1)	0,0064
		100000	O(1)	0,0059
		1000000	O(1)	0,0066
		10000000	O(1)	0,0067
		100000000	O(1)	0,0074
	Tabla Hash - Direccionamiento abierto	10000	O(1)	0,0619
		100000	O(1)	0,0697
		1000000	O(1)	0,0622
		10000000	O(1)	0,0856
		100000000	O(1)	0,0674

Tabla 10. Tiempos de ejecución del método “insertar” en las tablas Hash.

Análisis comparativo de las Gráficas con respecto a la Complejidad Temporal

Tablas Hash	Eliminar Datos	Buscar Datos	Insertar datos
Close Hash	O(1)	O(1)	O(1)
Open Hash	O(1)	O(1)	O(1)

Ambas implementaciones de la tabla Hash son bastante buenas ya que cuentan con operaciones que ocurren en O(1) pero consideramos mejor al direccionamiento abierto ya que no necesita de una estructura de datos adicional además de que si no se tiene un control adecuado, las listas enlazadas del direccionamiento abierto pueden volverse muy grandes para grandes cantidades de datos ocasionando que las operaciones ya no ocurran en tiempo constante.

Conclusiones

Los tiempos de ejecución de un árbol balanceado(AVL) son notoriamente menores que los del árbol desbalanceado.

La tabla Hash con direccionamiento abierto es superior en las funciones de búsqueda y eliminación de datos con respecto al direccionamiento cerrado, así mismo, la función de inserción es superior en el direccionamiento cerrado, a pesar de que en las dos tablas el tiempo de ejecución es constante para las tres funciones.

La tabla Hash con direccionamiento cerrado, al depender de una estructura de datos adicional (lista enlazada), provoca que algunas funcionalidades sean más lentas con respecto al direccionamiento abierto.

Los tiempos de ejecución en los árboles AVL son menores que los tiempos de ejecución en las listas enlazadas para la mayoría de funciones.

Dificultades y lecciones aprendidas

Definir qué estructuras de datos usar para cada funcionalidad, y con esto aprendimos que es buena práctica hacer varias comparaciones entre diferentes estructuras para definir la “mejor” para cada problema.

Con respecto a las tablas Hash, hubo dificultades, puesto que en ciertas ocasiones calculaba el mismo hash para algunos (diferentes) valores y fue necesario realizar varios ajustes.

Una de las mayores dificultades, fue el tiempo, ya que este estuvo muy limitado, causando que no se pudiesen realizar algunas cosas como una interfaz gráfica para el prototipo.

Una de las lecciones más importantes, fue el aprender a manejar mejor el tiempo, y a distribuirlo de mejor manera para satisfacer todos los requerimientos.