ESAT INNOVATION SCHOOL | ESCUELA SUPERIOR DE ARTE Y TECNOLOGÍA

Algoritmos y Estructuras de datos

Comparativa

Nombre Asignatura:Algoritmos y estructuras de datos
Módulo Asociado: 2º VGP
Profesor: Aleix López
Curso: 2023/2024
Autor/es: Carlos Mazcuñán Blanes

1 .- Hipótesis comparativas

Vector:

- Insertar al principio (**InsertFirst()**) puede ser algo **lento**, ya que requiere en su mayoría de casos desplazar todos los elementos existentes para hacer hueco al nuevo elemento.
- Insertar al final (**InaserLast()**) debería de ser **rápido**, ya que solo inserta un elemento en el último índice del vector, sin tener que hacer ningún desplazamiento.
- La velocidad de insertar en una posición específica (**InsertAt()**) **variará** dependiendo de lo lejos del final que haya que insertar el nuevo elemento. Ya que cuánto más al principio, más elementos habrá que desplazar.
- Extraer al principio (**ExtractFirst()**), en su mayoría de los casos será **lento** ya que se elimina el primer elemento del vector desplazando todos los demás elementos y sobreescribiendo unos a otros,
- Extraer al final (**ExtractLast()**), suele ser **rápido** ya que solo hay que eliminar el último elemento del vector, sin desplazar nada.
- Extraer en una posición específica (**ExtractAt**) **variará** dependiendo de lo lejos del final que haya que extraer el nuevo elemento.
- Dado que se debe asignar memoria y copiar todos los elementos de ambos vectores, la velocidad de la operación **concat** dependerá en gran medida del tamaño de los vectores originales, es decir podría ser más lento para vectores grandes debido a la asignación de memoria y la copia de datos.

El **vector** permite un acceso aleatorio **rápido**, ya que los elementos están contiguos en memoria. No es tan eficiente para insertar y eliminar elementos en posiciones intermedias debido al desplazamiento requerido.

Lista:

- Insertar al principio (**InsertFirst**) y extraer al principio (**ExtractFirst**) deberían de ser operaciones **rápidas**, ya que solo se manipulan los punteros de los nodos del inicio.
- Insertar al final (**insertLast**) y extraer al final (**extractLast**) pueden ser **lentas**, ya que se debe recorrer la lista hasta el final.
- Insertar en una posición específica (**insertAt**) y extraer en una posición específica (**extractAt**) pueden ser lentas, ya que se necesita recorrer la lista hasta la posición deseada.
- La operación **concat** para una lista simple implica recorrer todos los nodos de la lista que se va a concatenar y crear nuevos nodos en la lista original para cada elemento, la velocidad de esta operación dependerá del tamaño de la lista que se va a concatenar y de la cantidad de elementos que contiene.

El adt de la lista tiene eficiencia para inserciones y eliminaciones en cualquier posición. Acceso secuencial eficiente, pero acceso aleatorio lento debido a la necesidad de recorrer la lista desde el principio.

Lista Doblemente Enlazada:

Similar a la lista simple, pero con la capacidad adicional de recorrer la lista hacia atrás, lo que puede mejorar la eficiencia y velocidad de algunas operaciones, como la extracción desde el final (**extractLast**), ya que tenemos un puntero al último nodo, y podemos recorrer la lista desde atrás.

La operación **concat** es similar a la lista enlazada, implica recorrer todos los nodos de la lista que se va a concatenar y crear nuevos nodos en la lista original para cada elemento.
Sin embargo, debido a la capacidad de la lista doblemente enlazada de recorrer la lista hacia atrás, podría haber una ligera mejora en el rendimiento en comparación con la lista enlazada estándar.

Sin embargo, los nodos adicionales y los punteros de retroceso pueden agregar una sobrecarga adicional en términos de memoria y procesamiento.

2 .- Velocidades y tiempos de los ADT

El tiempo **medio** de cada una de las operaciones se mide en **milisegundos(ms)**.

El número de operaciones por cada función es de **10.000**, excepto en la función **concat** que se reduce a unas **1000** operaciones, concatenando una lista o vector de 4 elementos sobre uno/a vacía.

El tipo de dato con el que se trabaja en los adt para hacer las comparativas es **int.**

	VECTOR	LIST	DLLIST
InsertFirst()	17.702011 ms	0.158210 ms	0.158470 ms
InsertLast()	0.008540 ms	0.154110 ms	0.318039 ms
InsertAt()	6.578240 ms	5.922080 ms	0.021880 ms
ExtractFirst()	16.842380 ms	0.006380 ms	6.973850 ms
ExtractLast()	0.006980 ms	9.423340 ms	0.006900 ms
ExtractAt()	7.636960 ms	5.313310 ms	5.433201 ms
Concat()	12.867200 ms	0.00500 ms	0.005500 ms