José María Martínez Carvajal

IA COMPARATIVA TADS

	Insert First	Insert At	Insert Last	Extract First	Extract Last	Extreact At	Concat
Vector	3.002 micros	3.0991 micros	0.348 micros	0.154 micros	0.125 micros	2.955 micros	9.230 micros
List	0.586 micros	0.705 micros	0.618 micros	0.360 micros	411.734 micros	0.643 micros	7.948 micros
DLList	0.623 micros	0.846 micros	0.635 micros	0.388 micros	0.642 micros	0.727 micros	8.017 micros

Primero de todo resaltar que las pruebas se han hecho para casos de 10000 elementos, 100 veces la operación en 100 ejecuciones diferentes, en el caso de las operaciones at se han realizado 5 insert previamente y se ha realizado el muestreo sobre 10005.

Como se puede aprecia en los datos, el vector es una estructura de datos idónea para una lectura de datos rápida. Pues la memoria del vector es contigua en memoria con lo cual ganamos velocidad gracias a la caché. Cabe notar que el extract at es mucho menos eficiente en el vector, pues la memoria debe reorganizarse para evitar saltos en la memoria. Esto también podría ser un problema al extraer al principio. Sin embargo, para evitar este problema el vector se ha implementado con un head dinámico, de esta forma, aunque se extraigan desde el principio no es necesario realojar la memoria.

Otra cosa a resaltar es la eficiencia en tiempo que gana la lista doblemente enlazada en referencia a la lista normal gracias a la adición del next. Pues el acceso al penúltimo nodo es directo y no es necesario realizar ningún bucle.

Pese a que la lista se ve muy beneficiada en las extracciones se ve más perjudicada en las inserciones, siempre y cuando no se realicen al final. Esto es debido a que las listas simplemente deben actualizar las direcciones de memoria, next y prev, necesarias para mantener la consistencia de datos. Mientras que el vector debe realojar la memoria. Sin embargo, esto no es así al insertar al final de este, pues en este caso puede añadirse sin afectar al resto de los nodos. Cabe notar que la implementación de un vector circular probablemente ayudaría a bajar los tiempos del insert First, pues el header podría actualizarse hacía el final del vector tras varias inserciones seguidas. También cabe destacar que con una distribución más heterogénea de acciones el insertFirst no se vería tan penalizado, pues por ejemplo si hubiese extracciones en el principio antes de las nuevas inserciones el header podría actualizarse sin necesidad de tener que realojar los datos. Sin embargo, es evidente que el vector por su propia naturaleza siempre se verá afectado al realizar inserciones y extracciones al final.

En definitiva, las conclusiones obtenidas con los resultados son.

- La ganancia al incluir el atributo previous a los Memory Nodes de la lista es muy notable, y va siendo más evidente cuando se manejan volúmenes de datos muy grandes, pues cada vez es más costoso llegar hasta el penúltimo elemento.
- Los vectores son estructuras de datos idóneas cuando queremos guardar datos que no van a sufrir grandes alteraciones en el transcurso del tiempo, puesto que las lecturas son su punto fuerte. Sin embargo, casos en las que los datos sean alterados por su extremo, sobre todo si es al final de este también salen bien paradas (como pordría ser el caso de un pull o una pila)
- Por último, las listas son estructuras de datos idóneas para cuando nuestro conjunto de datos va a sufrir constantes cambios durante el tiempo de ejecución. Como podría ser por ejemplo un conjunto de números que necesita ir ordenándose cada poco tiempo. O casos en los que los datos al ser usados deben ser extraídos de la estructura. Pues la reorganización de esta es sencilla y rápida mediante los punteros al no necesitar realojar la memoria.