Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

Informe Tarea 1

IIC2133 - Estructuras de Datos y Algoritmos

Primer semestre, 2015

Alumno: Mithrandir

Parte 0: Funcionamiento del Programa

En el programa hay 3 *structs* principales: Order, List y CoolArray.

Order representa la estructura e información de un comando del programa, esto es: si un comando es "ADD AT repi 11", entonces su respectiva Order tendrá la siguiente información:

- `struct TipoOrden = AddAt`,
- `char[32] = 'repi'`, e
- `int y = 11`.

List representa una lista ligada. Tiene 3 valores guardados:

- `int count`
- `struct Node first`
- `struct Node last`

Donde Node contiene:

- `struct Node next`
- `char[32] value`

Esta List en particular es una lista ligada simple. Más tarde me di cuenta de que esta implementación implicaría una inversión de trabajo mucho mayor que para una lista ligada doble. Sin embargo seguí con ésta porque dentro de lo difícil que era, era bastante interesante.

Finalmente, CoolArray representa un array **bacán**. Esto, porque recuerda el tamaño de su arreglo de char* y la cantidad de elementos que guarda dentro.

El enfoque de mi diseño fue la flexibilidad de uso y de implementación del programa. Por lo tanto hay pocos o ningún *hack* como usar punteros a métodos o álgebra de punteros para obtener velocidad extra al ejecutar el programa.

Tampoco hay mucha programación *ad hoc* para cada tipo de tarea. Se trató de reutilizar la mayor cantidad de código posible. Por ejemplo cuando elimino una lista o un *Cool Array*, uso el método que elimina dado un índice. Los parámetros son la colección y el número 0. Luego repito esta acción hasta que la colección queda vacía. Una mejora en velocidad sería recorrer la estructura de inicio a fin, destruyendo cada elemento de ellas en el menor tiempo posible. Este detalle de implementación hace más fácil la eliminación de *bugs* y reduce su aparición, pues hay menos código. Sin embargo, se paga en velocidad.

Ojo que la falta de velocidad te descontará puntos.

Tampoco se usó "–" o "++" en líneas como array[i–], que si bien quitan líneas de código y muchas veces reducen las líneas de assembly ejecutadas, sólo crean errores invisibles cuando el programador se equivoca.

OK. No hay problema.

Parte 1: Tiempos de ejecución

Faltan gráficos para ilustrar mejor tus resultados

El programa fue probado usando el Test A subido al Siding. Los tiempos de ejecución fueron medidos con la librería **time.h**, usando el struct **clock_t** para almacenar tiempos y el método **clock** para obtenerlos.

Al usar una lista el programa tardó 2.661.767.976 ciclos de reloj en completar el test. El uso de un CoolArray tomó 1.812.318.028 ciclos. El uso de lista fue un 40 % más lento.

Para la próxima intenta medir el tiempo en segundos.

Al usar una lista la mayor demora fue al escribir el archivo de output pues leer cada elemento toma una cantidad de tiempo proporcional a la posición del elemento en la lista.

Al usar un arreglo la mayor demora fue al agregar los elementos pues el método que los agrega duplica el tamaño del arreglo cuando falta espacio.

Parte 2: Ventajas y Desventajas de cada técnica

Sería bueno ver una prueba empírica de las operaciones consideradas como lentas, para comprobar la teoría

Arreglo

El leer elementos en el arreglo es lo más fácil pues toma O(1). Sin embargo agregar elementos es O(n) en el peor caso (al duplicar el arreglo), y O(n) en el caso de agregar elementos dentro de la lista, pues eso implica mover muchos elementos 1 posición hacia afuera.

Eliminar elementos también es muy costoso, tanto al buscar un "value" como al buscar un índice (O(n)), pues de nuevo, hay que mover elementos 1 posición, hacia adentro. Buscar elementos es O(n) también, pues el arreglo no está ordenado. Invertir, finalmente, es O(n).

Lista

Muy bien! Excelente analogía.

Leer elementos en la lista es O(n), porque la lista se comporta como un *stream*. Agregar elementos sin embargo, es O(1) pues se agregan al final usando el puntero a `last`.

Agregarlos dentro de la lista es O(n), pero no tan terrible como el costo del arreglo pues el tiempo es utilizado en recorrer la lista y no en trasladar los elementos del arreglo.

Sería bueno separar toda esta información en 2 o 3 párrafos más cortos.

Eliminar elementos es costoso. Toma O(n) pues es necesario recorrer la lista. Sin embargo al igual que en la comparación anterior, la eliminación en el arreglo es mucho más costosa pues es necesario mover todos los elementos. Buscar elementos es O(n), aunque un poco más costoso que para el arreglo pues se usan punteros y no números para recorrer la lista. La inversión finalmente es O(n), pero mucho más lenta que la del arreglo. Esto porque siendo una lista ligada simple, es necesario copiar muchos punteros para cada elemento a fin de que la lista no pierda información.

Conclusión

El arreglo privilegia la lectura, la lista la escritura. En un entorno de poca entrada de datos pero mucha consulta el arreglo será superior porque el acceso a los datos es O(1). Un directorio o un buscador web serían buenos ambientes.

En un entorno de poca consulta pero mucha entrada de datos la lista será superior, pues su entrada es O(1). Un formulario web o un almacén de datos son ambientes adecuados.