Memoria P4 Edat

Ejercicio 1c

Reflexiones sobre TAD BST

Al ejecutar el programa p4_e1.c se puede comprobar que los tiempos de creación y búsqueda varían mucho según si el programa se ejecuta en modo normal o en modo ordenado.

¿Por qué es así?

Esto se debe a la diferencia a la hora de insertar los datos, para insertarlo con el modo normal, la función pasa por una rama entera por lo que el coste es de O(N) (+ O(N) de tree_contains, pero como el coste sería el 2* O(N) sigue siendo proporcional) ya que el árbol no se encuentra balanceado y puede ser que todos los elementos se inserten en una sola rama (si todo el rato el siguiente es mayor que el actual), de esta forma para insertar el árbol entero el coste será $\sum_1^N O(N) = O(N^2)$.

Sin embargo, al hacerlo de la forma sorted, estamos ordenando los elementos partiendo un array por la mitad hasta encontrar el lugar del elemento por lo que su coste viene a ser $\log_2 N$ lo que se entiende como $O(log(N))$, por tanto, el coste para ordenar los elementos es de $\sum_1^N O(\log(N)) = O(N * \log(N))$. Y para insertarlos se recorre una rama del árbol $O(log(N))$ (ya que este está balanceado) hasta llegar al lugar, por tanto, al hacerlo con todos los elementos el coste de inserción es de nuevo $O(\mathrm{N} * log(N))$. Como al sumar los tiempos obtenemos 2 * $O(log(N))$ = $O(log(N))$ ya que son proporcionales.

Esta explicación es totalmente coherente con los tiempos obtenidos: 0.024416 segundos (normal) y 0.012027 segundos (sorted). Ya que el tiempo de normal crece más rápido que el de sorted.

¿Hay alguna propiedad del árbol que permita explicar este comportamient

La principal propiedad sería la profundidad del árbol, ya que como se ha mencionado previamente, para buscar o hacer la inserción de un elemento en el árbol (en la forma normal) se debe recorrer una rama entera, cuya profundidad puede ser N, de ahí que su coste sea O(N). En la forma sorted sin embargo la profundidad de cualquier rama es igual o un elemento menor, por lo que como cada nodo se va dividiendo en 2, la profundidad de cada rama será de $\log_2 N$ y por ello, el coste de inserción es $O(log(N))$.

Esta explicación es totalmente coherente con las profundidades obtenidas: 41 (normal) y 16 (sorted), la cual es notablemente menor a la primera.

Ejercicio 2

Reflexiones sobre el TAD SQ

¿Qué diferencias y similitudes hay entre el TAD SQ y el TAD Cola de la práctica anterior? □

El TAD SQ (SearchQueue) y el TAD Cola de la práctica anterior tienen en común que ambos se utilizan para almacenar elementos mediante punteros void y que permiten añadir y extraer elementos, pero se diferencian en su comportamiento y estructura interna. La Cola de la práctica anterior implementa un sistema de First In/First Out, es decir, mantiene el orden de llegada, mientras que el TAD SQ se basa en un Árbol Binario de Búsqueda (BST), lo que permite mantener los elementos ordenados según una función de comparación y extraer siempre el mínimo. Además, el SQ no admite duplicados, lo que no ocurre con la Cola.

¿Qué coste (aproximado) tiene la operación de extraer un elemento en el TAD SQ? ¿Sería posible hacer que esta operación fuera (1)?

La operación de extraer un elemento en el TAD SQ tiene un coste aproximado de $\mathcal{O}$(log n), ya que internamente utiliza un Árbol Binario de Búsqueda (BST) y para extraer el elemento mínimo es necesario buscar el nodo más a la izquierda y luego eliminarlo. No sería posible hacer que esta operación fuera $\mathcal{O}(1)$ porque ni la búsqueda ni la eliminación del mínimo pueden hacerse en tiempo constante sin sacrificar la propiedad

de orden. Para lograr una extracción en $\mathcal{O}(1)$, se requeriría cambiar la estructura interna por otra como un heap con puntero directo al mínimo, lo que implicaría renunciar a algunas propiedades del BST.