INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

Act 3.4 - Actividad Integral de BST

Grupo: 601

Integrantes:

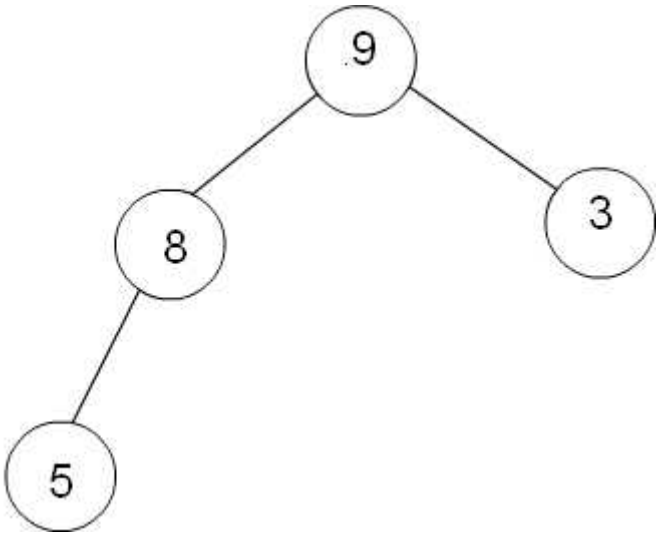
Daniel Emilio Fuentes Portaluppi - A01708302
José Diego Llaca Castro - A01704793
Iván Ricardo Paredes Avilez - A01705083

Profesor: Francisco Javier Navarro Barrón

Noviembre 28, 2022

Investigación:

Un montículo (heap en inglés) es una estructura de datos del tipo árbol con información perteneciente a un conjunto ordenado. Los montículos máximos tienen la característica de que cada nodo padre tiene un valor mayor que el de cualquiera de sus nodos hijos, mientras que en los montículos mínimos, el valor del nodo padre es siempre menor al de sus nodos hijos.
Puede considerarse como un árbol binario con ciertas restricciones, aunque con muchas ventajas. Un heap es un árbol binario casi completo. Se considera casi completo ya que está lleno en todos sus niveles excepto quizá el último que se va completando de izquierda a derecha. Otra diferencia es que un heap cada nodo tiene un valor menor igual (o mayor igual)que el valor asociado a cada uno de sus hijos, lo que se conoce como condición heap.



Un árbol cumple la condición de montículo si satisface dicha condición y además es un árbol binario completo. Un árbol binario es completo cuando todos los niveles están llenos, con la excepción del último, que se llena desde la izquierda hacia la derecha.

Una ventaja que poseen los montículos es que, por ser árboles completos, se pueden implementar usando arreglos, lo cual simplifica su codificación y libera al programador del uso de punteros.

La eficiencia de las operaciones en los montículos es crucial en diversos algoritmos de recorrido de grafos y de ordenamiento (heapsort).

Referencias:

HEAP. (s. f.). https://sites.google.com/site/arbolesheap/

Reflexiones:

Daniel Emilio Fuentes Portaluppi:

De las estructuras de datos vistas en clase los árboles son la que más me ha gustado. En esta actividad el haber utilizado un Min Heap porque nos ayudó bastante para poder llevar la cuenta del menor número de interacciones posibles. Los árboles binarios en general son muy útiles cuando no se utilizan un gran número de datos pero entre más aumentan estos últimos menos viables se vuelven. Por último hay que mencionar que los árboles heap también resultan muy útiles en los métodos de otros algoritmos como los grafos.

Esta actividad no fue sencilla pero considero que no nos presentó demasiada complejidad porque ya teníamos las bases de todo lo visto en clase. Me quedó con los aprendido y con la seguridad de que voy a ser capaz de poder implementarlos nuevamente más adelante tanto en mi vida escolar como profesional.

Iván Ricardo Paredes Avilez:

Esta actividad me permitió trabajar con estructuras de datos de tipo de árbol, como los heap, en un contexto similar al que nos enfrentaremos en la vida real si tuviéramos que emplear este tipo de estructura para manejar información de una empresa como lo es "International Seas, Ltd". Con ello, no solo he entendido cuál es la importancia de los árboles, sino que también tuve la oportunidad de practicar mis habilidades en programación e implementar el manejo de la estructura de datos en un lenguaje como C++. Ahora puedo darme cuenta de que esto es una actividad que deben desempeñar los ingenieros diariamente en diferentes empresas para que sus sistemas estén siempre funcionando al igual que tienen que saber que tipo de estructura de datos es más eficiente para cada aplicación.

José Diego Llaca Castro:

Como se puede observar en problemáticas tales como la mostrada en esta actividad, algunos métodos de ordenamiento pueden resultar muy útiles teniendo cantidades controladas de datos, sin embargo conforme aumenta la cantidad de datos se vuelven cada vez menos viables. Por lo anterior, es necesario que no solo nos centremos en un único método de ordenamiento que utilicemos como nuestra opción número uno.

Por ejemplo en este caso, en el cual se estaba utilizando un algoritmo Merge Sort para la organización de los datos, funcionaba hasta que empezaron a recibir un cantidad demasiado grande de pedidos lo cual nos obligó a reconsiderar nuestro método de ordenamiento.

Ahora una pregunta que surge de esto es ¿Porqué un BST resulta mejor que algún otro algoritmo de ordenamiento? Esto es sencillo de responder, pues como ya se estableció en este caso la cantidad de datos es preocupantemente alta, por lo que si se usa cualquier otro algoritmo los cuales son en gran parte de complejidad O(n) el tiempo necesario para ordenar

los datos sería innecesariamente largo, sin embargo al usarse un BST la máxima complejidad que alcanza tanto para añadir algún dato o buscar algún dato es de O(log n) por lo que resulta mucho más rápida cuando se trata de cantidades gigantescas de datos.

El caso anterior puede repetirse en múltiples proyectos, por lo que tener en nuestro arsenal una herramienta tan útil como los BST puede ayudarnos en múltiples ocasiones a manejar cantidades importantes de datos de manera rápida y efectiva como se demostró en esta actividad.