Alexis Sánchez Orosco 181074 Estructuras de Datos Avanzadas

Implementación de Heap-Sort y Comparación de Desempeño con Merge y Trie Sort

Introducción

El Heap es una estructura binaria en donde todos sus niveles están llenos excepto posiblemente el ultimo. Todas las hojas están recargadas hacia la izquierda, es decir, no hay nodos con hijo derecho sin tener hijo izquierdo y no hay primos derechos a menos de que haya primos izquierdos. El min-heap tiene la estructura del heap y para cada nodo en el Heap, el elemento almacenado es menor o igual que el de sus hijos. Se puede realizar las operaciones de insertar, eliminar el mínimo o buscar el mínimo. La estructura heap también se puede implementar para un max-heap, en donde cada elemento es mayor o igual que el de sus hijos.

Descripción del Problema

La implementación de un Heap se logra mediante la manipulación de los datos de un arreglo del tamaño del número de datos. Debido a la estructura del min heap, al momento de eliminar un elemento se borra el elemento mas pequeño almacenado. De tal forma que si insertamos n datos a el min-heap, entonces al momento de eliminarlos de la estructura obtenemos los datos fuera del árbol de manera ordenada pues esa es la intención de eliminar del min-heap. Una vez programado el min-heap, se requiere ingresar conjunto de datos que pueden ir desde 50 hasta los 300,000 datos para evaluar el desempeño de este tipo de ordenamiento respecto a otros métodos tales como el Trie Sort y el Merge Sort. A pesar de que el Trie Sort esta diseñado para palabras y min-sort para valores, en este caso modificaremos el código para que ambas estructuras funcionen con palabras y de esta forma determinar cual tiene mejor desempeño. Para el caso del Merge sort, no tenemos problemas de diseño ya que funciona de la misma forma para palabras que para números.

Implementación Min-Heap

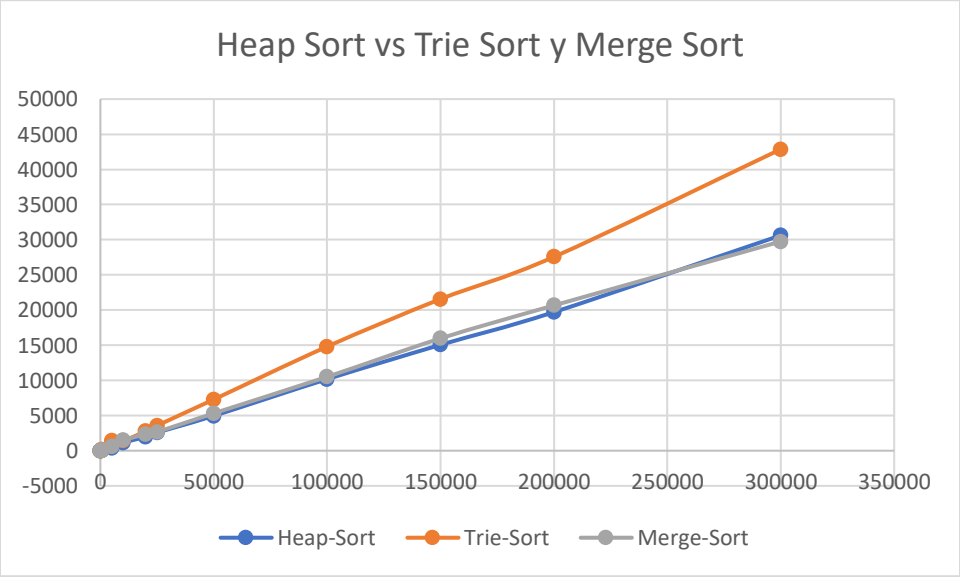
Para poder crear el min-heap, hacemos que el constructor pida un tamaño máximo el cual será el número de elementos máximos que se pueden almacenar en el arreglo que usaremos para modelar la estructura. Establecemos que el padre estará un cierta posición, sus hijos izquierdos al doble de esa posición y los hijos derechos al doble de la posición mas un unidad. Los elementos se insertan de manera normal y se va ordenando el arreglo en caso de que el elemento ingresado sea menor al padre. Una vez insertados los elementos requeridos, modificamos la estructura de tal forma que si no son una hoja los elementos deben ser menores o iguales que sus hijos.

Experimento realizado

En primer lugar, se hicieron algunas pruebas numéricas para comprobar el correcto funcionamiento del la estructura min-heap. Una vez revisado el buen funcionamiento, se pasó a probar con conjuntos de palabras a ordenar y poder así medir su rendimiento. Los mismo se realizó con el Trie Sort y Merge Sort y se recopilaron los siguiente datos que hablan de las diferencias en rendimiento. Un suceso que destacar es que debido a como

estaban programados el Heap Sort y Trie Sort, se ajustó el funcionamiento del Merge sort para que no solo ordenara los elementos sino también imprimiera los elementos. De esta forma, las tres formas de ordenar quedaban en igualdad de condiciones para medir cuidadosamente. Las pruebas se realizaron con los datos de manera desordenada.

	Tiempo en milisegundos		
Numero de datos	Heap-Sort	Trie-Sort	Merge-Sort
50	5	5	25
100	7	8	16
500	33	128	44
1000	67	131	69
5000	400	1463	594
10000	1123	1433	1478
20000	1990	2767	2350
25000	2580	3578	2665
50000	4962	7290	5321
100000	10178	14813	10511
150000	15062	21567	15971
200000	19714	27569	20666
300000	30619	42875	29754



Análisis de Resultados

En los datos que arrojo cada una de las pruebas, se puede ver en primer lugar como conforme avanza la cantidad de datos el tiempo aumenta. Hay un caso especial con el Merge Sort, en donde toma más tiempo ordenar 50 datos que 100 datos pero fuera de ese caso los datos cambian de forma creciente. Se puede notar tanto en los datos como en las graficas que el Trie Sort es el que toma mas tiempo para ordenar los datos. Una razón de esto puede ser la cantidad de pasos y procesos que se deben llevar a cabo para generar el árbol trie con las palabras

insertadas. La limitante de esta estructura es que resulta más complicado poder implementar esta solución para valores numéricos pues ocupa otro espacio importante en la memoria almacenar mas símbolos. El segundo mas lento por una ligera diferencia es el Merge Sort, el cual hace el proceso de ordenar muy rápido y en lo que se tarda es imprimir el arreglo que ha ordenado. Finalmente, se puede ver como el Heap Sort funcionó como el más rápido por milisegundos. A pesar de que el Heap Sort debe ingresar todos los elementos y luego eliminarlos para obtener el orden de los elementos, su desempeño es muy similar al del Merge Sort.

En conclusión, el Heap-Sort y el Merge-Sort son dos formas de ordenar datos de manera rápida respecto al Trie, el cual por sus limitaciones y estructura tarda un poco mas en su ordenamiento lexicográfico. Sumando a esto, hemos mencionado que el Trie solo funciona de manera eficiente para palabras pues cuando se quieren incluir símbolos de otra índole resulta mas complicado procesarlo en la memoria. El experimento deja la puerta abierta para evaluar como se comporta el heap-sort respecto a otros algoritmos de ordenamiento vistos previamente. Estas pruebas nos hicieron notar que cada uno de los estos procedimientos tienen usos particulares y no siempre se puede ordenar de la misma forma. Existen diversas estructuras de datos para ordenar diversos tipos de datos.