Guía Precontrol 2

Profesores: Iván Sipiran
Nelson Baloian
Patricio Poblete
Auxiliares: Alonso Almendras, Albani Olivieri
Vicente Olivares, Ricardo Valdivia
Sebastián Acuña, Martín Paredes

P1. P2 2013-2 [Hash]

Se desean guardar enteros positivos (valor ≥ 0) en una tabla usando hashing. Los casilleros desocupados de la tabla se marcan con -1. Considere el algoritmo de inserción de un valor en la tabla de hashing utilizando linear probing, y suponga que la función `hash(valor)` ya está implementada.

Una variante del método de inserción que permite reducir la varianza en el tiempo de búsqueda es Robin Hood Hashing, en donde la estrategia de inserción es similar a linear probing, pero en cada casillero probado donde hay colisión el valor que se queda en dicho casillero es aquel que se encuentra más lejos de su posición original (la indicada por la función de hash). Implemente en Python una función de inserción usando Robin Hood Hashing. Puede suponer que al momento de invocar a la función de inserción la tabla no está llena (hay al menos un casillero desocupado).

P2. P3 2017-1 [Árboles]

El algoritmo usual de inserción en un árbol de búsqueda binaria (ABB) deja al nuevo elemento en el borde inferior del árbol. En este problema vamos a programar un algoritmo alternativo que deja al nuevo elemento como raíz del árbol resultante. Para esto, se compara el nuevo elemento con la raíz del árbol y, dependiendo de si es menor o mayor que ella, se inserta (recursivamente) en el subárbol izquierdo o derecho, respectivamente. Observando que después de esto el nuevo elemento debe estar como hijo directo de la raíz, se hace una rotación simple para dejarlo como raíz.

Escriba una función en pseudocódigo de encabezado `insertar_raiz(x, root)` que inserte un nuevo elemento x en un árbol cuya raíz es root, y que retorne la raíz del árbol resultante. Suponga que la llave x es distinta de todas las que están en el árbol.

P3. Ex 2020-1 [AVL]

El Prof. Frank N. Stein ha propuesto un nuevo algoritmo de ordenamiento utilizando un ABB (árbol de búsqueda binaria). Dado un arreglo de tamaño n, el algoritmo de ordenamiento consiste en los siguientes pasos:

a) Crear un ABB vacío.

b) Insertar los n elementos del arreglo en el ABB.

c) Realizar un recorrido inorden del árbol, mostrando en orden los nodos visitados.

¿Cuál es la complejidad en el mejor caso para este algoritmo? ¿Cuál es la complejidad en el peor caso para este algoritmo? Explique brevemente sus respuestas, indicando cuanto tiempo toma cada uno de los pasos del algoritmo.

Un alumno muy perspicaz le propone al Profesor usar un árbol AVL en vez del ABB en su algoritmo de ordenamiento. ¿Cuál es la complejidad en el peor caso para este nuevo algoritmo? Explique por qué. ¿Cuál es la complejidad de este nuevo algoritmo si los elementos del arreglo ya vienen ordenados.

Explique por qué (piense en cuánto cuesta insertar un nuevo elemento en el árbol AVL considerando que vienen ordenados, y luego calcule la complejidad total del algoritmo).

P4. Ex 2020-1 [Trie]

Dada una matriz binaria de tamaño $N \times M$, en esta pregunta abordaremos el problema de detectar filas duplicadas. Una forma de resolver este problema es comparar las filas en pares hasta encontrar las que sean idénticas.

La salida de un algoritmo que resuelva este problema debería ser el conjunto de pares de filas (i, j), tal que la fila i es idéntica a la fila j. En el peor caso, como la cantidad de pares que se pueden formar con N filas es $\frac{N(N-1)}{2}$ y la comparación de cada par de filas tomaría M comparaciones de números binarios, la complejidad de este algoritmo es $O(N^2 \times M)$.

Lo que se le pide es:

- Formule un algoritmo (pseudocódigo o descrito en palabras) que utilice un Trie para detectar las filas duplicadas de la matriz binaria. Su algoritmo debería tener complejidad $O(N \times M)$.
- Justifique por qué su algoritmo tiene complejidad $O(N \times M)$.

P5. Quickselect [Sort]

Al igual que Quicksort, fue desarrollado por Tony Hoare, y por eso también se conoce como El algoritmo de selección de Hoare. Quickselect se utiliza para entregar el índice de algún elemento en un arreglo, ocupando el mismo principio de reducción del problema que Quicksort. Esto es, se elige un elemento como pivote, y se particionan los datos en dos: si son menores o mayores que el pivote. La diferencia está en que, en lugar de continuar ordenando ambos lados, solo ordena el lado donde se encontraría el elemento buscado si estuviera.

¿Cuál es la complejidad de este algoritmo? ¿Cree usted que es un algoritmo eficiente? ¿Por qué?