Final febrero 2025: (el 58/120 es un 5,5):

1)
Describe en palabras cómo realizarías un recorrido preorden iterativo (no recursivo) de un árbol binario sin usar estructuras auxiliares más allá de una pila. Justifica brevemente.

Rta:
En caso de tratarse de un arbol binario de busqueda:
Primero creó la pila. Luego, inserto en esta el nodo raíz, para empezar a recorrerlo. Si el nodo tiene hijos, primero visito el nodo y a posterior coloco el hijo derecho y luego el izquierdo, esto se debe por la naturaleza del recorrido preorden. En el preorden, primero se analiza el nodo izquierdo, por ende como las pilas son de tipo LIFO, al arealizar el pop analizaran/visitaran primero el hijo izquierdo, respetando asi el preorden.

Correccion:
El algoritmo de recorrido in-orden de árboles binarios y árboles binarios de búsqueda es el mismo.

Tu justificación de cómo encolar el primer nodo está muy bien. Pero no explicas cómo proceder más allá del primer nodo.

Rta.: existen varias formas de encarar este problema. La más sencilla es la siguiente: inicializas el algoritmo con una pila vacía y apilas el nodo raíz. Mientras la pila no está vacía, sacas un nodo de la pila, lo procesas, y apilas su hijo derecho, si existe, y su hijo izquierdo, si existe. Continúas *iterativamente* el paso anterior hasta terminar.

2)

Dado un arreglo ordenado de números enteros con posibles valores repetidos, describe cómo ajustarías la búsqueda binaria para encontrar el último índice donde aparece un número dado. Justifica brevemente.

Rta: No se puse nada y por ende no hay corrección.

3)

Describe un algoritmo que permita encontrar el nodo con el valor mínimo en un árbol binario. Justifica brevemente.

Rta:

Si se trata de un arbol binario de busqueda, basta con partir del nodo raiz y de alli visitar al nodo hijo. Si el actual nodo hijo, tiene un hijo por la izquierda, muevo el puntero al hijo po rla izqueirda. Asi sucesivamente hasta llegar a una hoja del arbol. Ahora, si nos basamos en la definicion estricta de arbol binario, donde no estan rganizados por el valor de su clave/peso los hijos, se debería realizar lo siguiente:
Defino una variable con el valor de la raiz del arbol, en caso de no estar vacio, o en su defecto le doy un valor grande para asegurarme que no sea el menor valor el que elija.
Luego recorro todo el arbol, ya sea por DFS o BFS, para asi asegurarme de visitar todos sus nodos y procesarlos. En el procesamiento, lo que hago es comparar mi variable inicial, con el valor/peso del arbol, y en caso de ser menor, almacenar dicho valor en mi variable. Asi sucesivamente hasta que recorrer todo el arbol.

Correccion: No la hay, me puso un imbatible! (Anashei).

4)

Describe un algoritmo para rehashing cuando el factor de carga de una tabla hash excede un umbral dado, sin perder datos. Justifica brevemente.

Rta:

Primero verificaria el factor de carga, ya que si este supera el umbral, realizaria un rehashing. Para el rehashing, primero duplicaria el tama;o de la tabla, ya que de esta forma se puede reducir la densidad de elementos y por ende reducir las colisiones! Luego de crear la nueva tabla con el doble de tama;o, recorreria la vieja tabla para ir insertando los elementos utilizando mi nueva funcion de hash. Por ultimo, actualizaria la tabla, conviertiendo la nueva tabla en la official.

Correccion:
* No sólo cuando hay que agrandar la tabla, también cuando hay que achicarla!

5)

Describe cómo podrías usar std::map para implementar una función que cuenta la frecuencia de cada palabra en un texto. Analiza la complejidad en tiempo de tu algoritmo. Justifica brevemente.

Rta: No puse nada, por ende no hay corrección.

6) Describe un algoritmo de ordenamiento que funcione eficientemente en arreglos con muchos elementos duplicados, con una complejidad mejor que O(n log n). Justifica brevemente.

Rta:

candidatos para resolver este problema en forma eficiente son pidgeonhole sort y radix sort.

Correccion: Me falto explicar como implementarlos. (No lo hice por falta de tiempo).

7)

Describe cómo podrías usar std::unordered_set para eliminar duplicados de una lista de números enteros, manteniendo el orden relativo original. Justifica brevemente.

Rta: No puse nada, por ende no hay corrección.

8)

Justifica brevemente por qué una buena función hash debe distribuir los elementos uniformemente en una tabla hash. ¿Qué sucede con un mal diseño de función hash?

Rta:

El motivo de esto es para reducir las colisiones (2 claves diferentes generan el mismo indice). Ademas, debe distribuir los elementos uniformemente para mantener el O(1) para las operaciones de insercion, eliminacion y busqueda. En caso de estar mal dise;ada, agrupara grandes cantidades de elementos en pocas posiciones, incrementando la cantidad de colisiones. El efecto de esto es que para realizar las operaciones osbre los elementos, terminen con un costo computacional de O(n), ya que se deberan recorrer una lista de elementos colisionados.

Correccion: Y además: Si los elementos se agrupan en ciertas áreas de la tabla, se puede desencadenar un rehashing.

9)

Propon una estructura de datos que permita insertar, eliminar y buscar elementos en O(log n) en promedio. Justifica brevemente.

Rta:

Una estructura de dato que cumple con estos requisitos suelen los arboles bianrios de busqueda auto-balanceados. El motivo de esto, es que estos tipos de arboles tienen la propiedad de que mantienen la altura en un O(log (n)) y por ende lo mismo sucede con el costo de las operaciones de isnerci'on, eliminaci'on y busqueda. Por que las operaciones tienen el mismo costo computacional que la altura? Esto se debe a que cada operacion requieren ir desde la raiz hasta la hoja, luego como la altura es O(log (n)), lo mismo sucedera con las operaciones.

Correccion: Nada, me puso un extraordinario! (God, no?)

10)

Describe en palabras cómo podrías implementar una cola de prioridad con std::vector, incluyendo las operaciones push y pop, y analiza la complejidad de estas operaciones.

Rta: No puse nada, por ende no hay corrección.

11)

Describe en palabras un algoritmo para determinar si dos árboles binarios son idénticos en estructura y contenido. Justifica brevemente.

Rta:

Basicamente, se debe trabajar con un algoritmo recursivo, en donde se comienza desde la raiz y se va recorriendo nodo por nodo, en el mismo sentido, en ambos arboles. Si en la comparacion salta un peso distinto, se sabe que los arboles sin

distintos ya en contenido. Ademas, si en un nodo se tiene Null y en otro no, tambien se puede afirmar que son distintos en estructuras debido a que en uno se llego al final de una rama y en el otro no. Por ultimo, si al terminar la comparacion, se realizo el mismo recorrido y en las comparaciones siempre dieron lo mismo, se trata de arboles identidos.

Correccion:

* Tu respuesta tiene algunos aspectos confusos. Recuerda que los algoritmos deben estar precisamente definidos.
* En tu respuesta afirmas que se debe comprobar que los elementos de los sendos nodos coincidan. Pero antes de hacer esto, se debe verificar que ambos nodos existan.
* No describes explícitamente que después de comprobar la raíz, se debe hacer la misma comprobación para los subárboles izquierdos y derechos de manera recursiva.
* No explicitas qué devuelve la función recursiva, ni cómo se reúne la información de los nodos hijos.
* No justificas por qué este algoritmo funcionaría bien. Justificación: el método es eficiente porque se detiene en cuanto encuentra una discrepancia (no lo dices). El método también es exhaustivo porque en caso de ser los dos árboles idénticos, visita todos los nodos (no lo dices).

12)

Describe un algoritmo para calcular la altura de un árbol general sin usar estructuras de datos auxiliares. Justifica brevemente.

Rta:

Mi algoritmo se basaria en uno recursivo. En este algoritmo, lo primero que verifico es el caso base, el cual es si el arbol es vacio, es decir, si el nodo raiz es nulo. En este caso tomo su altura como 0. En caso de tener hijos, comienza la parte recursiva, que se basa en lo siguiente: Seria un algoritmo analogo a un recorrido postorden, basicamente se visitan primero cada uno de los hijos para luego volver al nodo actual. Luego, la altura de cada nodo se calcula como la altura maxima de los hijos +1. Por ultimo, se realiza esto hasta llegar a una hoja donde su altura seria 0, y de alli se propaga el valor maximo de la altura a la raiz!

Correccion:

* Redacción: tu algoritmo *es* recursivo.

* La explicación es comprensible, pero podría beneficiarse de una estructura más formal o de puntos claros para cada paso del algoritmo.
* La respuesta no incluye una justificación clara de por qué este método no usa estructuras de datos auxiliares. La justificación podría ser que no se necesita almacenar ninguna información adicional fuera de la recursividad para calcular la altura. Tampoco justifica por qué DFS es adecuado para este caso (porque explora cada rama hasta su máxima profundidad antes de retroceder).