2023 Diciembre

En este caso no encontré respuestas chequeadas, así que no fueron revisadas por ningún profesor.

1ra fecha:

¿Por qué no es conveniente aplicar algoritmos de ordenamiento sobre listas enlazadas? Justifica brevemente.

Porque la función swap qué usamos para los algoritmos de ordenamiento solo están diseñadas para cambiar los valores de lugar en un arreglo contiguo. Para usar un algoritmo de ordenamiento debemos construir un algoritmo que permita además cambiar los punteros entre los elementos, lo qué complejiza la función swap demasiado y toma mucho tiempo de ejecución.

Para un conjunto de estudiantes con su respectiva edad, se necesita realizar búsquedas rápidas para obtener la edad de un estudiante dado su nombre. ¿Qué tipo de dato STL sería el más apropiado para almacenar esta información? Justifica brevemente.

Un unordered_map es el tipo de dato adecuado, ya que permite buscar la edad del estudiante mediante su nombre, el cual sería la clave del valor de la edad en el map. Usar unordered_map es más eficiente para la inserción y eliminación de estudiantes.

En un árbol binario de búsqueda, ¿cómo se puede encontrar el sucesor inmediato de un nodo dado? Describe el algoritmo en palabras y analiza su complejidad computacional.

Podemos usar un algoritmo DFS, que recorre el árbol de forma recursiva. Dado un nodo de entrada, este llama a una función qué lea el contenido del nodo antes o después de llamarse a sí misma para cada sucesor de un nodo. Tiene complejidad O(log(n)) por ser recursiva.

En el contexto de la búsqueda con hashing, ¿cuáles son las ventajas y desventajas de utilizar una función hash criptográfica en lugar de una función hash más simple? Justifica brevemente.

Una función hash criptográfica produce valores con un rango mucho más alto qué una función más simple, además de qué su salida es mucho menos predecible y por ende hace la tabla hash mucho más segura.

En el lenguaje SQL, ¿por qué nos podría interesar que una tabla refiera a otra tabla? Justifica brevemente.

Una tabla puede referir a otra tabla cuando nos interesa qué haya varias bases de datos modularizadas relacionadas entre sí. Por ejemplo, con la iMDB puede haber una tabla para una película, y una con todos los actores registrados. La película qué busquemos entonces refiere a la tabla de actores para obtener su información relevante sobre el actor.

En una red social representada en un grafo, ¿cómo podrías identificar grupos de amigos? Justifica brevemente.

Podemos identificar un grupo de amigos cómo un ciclo en el grafo.

En una tabla hash, ¿cómo afecta el factor de carga a la eficiencia de la búsqueda? Justifica brevemente.

Al haber un factor de carga demasiado alto, es más posible qué se produzcan colisiones.

Considera un escenario en el que tienes un arreglo de tamaño n con muchísimos elementos repetidos. ¿Puedes proponer un algoritmo de ordenamiento que tenga una complejidad temporal menor que $O(n \log n)$ en este caso? Justifica brevemente.

Se pueden dividir los elementos repetidos en “buckets”, subarreglos qué contengan los elementos repetidos en cuestión, y luego ordenarlos por cada valor. Para esto se podría usar una versión de Pigeonhole sort.

En un algoritmo divide-and-conquer, ¿cómo se garantiza que la recursión termine y no se produzca un bucle infinito? Justifica brevemente.

En un algoritmo divide-and-conquer siempre se debe revisar que el nuevo llamado de la función no haya llegado a su mínima versión del sub-problema que quiera resolver el algoritmo. Por ejemplo, en mergesort, se revisa siempre que el sub-arreglo que analiza cada llamado de la función tenga por lo menos un elemento.

En un grafo no dirigido con n vértices, ¿cuál es la complejidad computacional del algoritmo de recorrido BFS? ¿Y del algoritmo de recorrido DFS? Justifica brevemente.

Ambos algoritmos tienen complejidad O(n), ya que cada uno visita el nodo de un grafo solo una vez(siempre que se usen las “marcas” en el objeto de los nodos).

2da fecha

¿Cuál es la diferencia clave entre un algoritmo de búsqueda lineal y uno de búsqueda binaria? Justifica brevemente.

Un algoritmo de búsqueda lineal recorre un arreglo elemento por elemento con un ‘for’ hasta encontrar el índice dónde está.
Un algoritmo de busqueda binaria empieza con un índice, y mediante interpolación lineal o mediante la subdivisión del arreglo en arreglos más pequeños, encuentra el elemento de forma mucho más eficiente. Sin embargo, no puede usarse para un arreglo no ordenado.

Pregunta 3

10 puntos

Da un ejemplo concreto, de la vida real, de un problema algoritmico que deba resolverse con un grafo disperso. Justifica brevemente.

Use el editor para dar formato a la respuesta

La conexión entre dispositivos conectados en una red LAN puede ser representada con un grafo disperso dirigido. Recorriendo este grafo, podemos averiguar cuántos dispositivos intermediarios(servidor, router, etc.) hay entre dos dispositivos.

Pregunta 4

10 puntos

Describe el propósito de la técnica de programación dinámica.

Use el editor para dar formato a la respuesta

La técnica de programación dinámica sirve para resolver problemas con sub-problemas con respuesta predecibles. Puede ahorrar muchísimo tiempo de ejecución ya qué el resultado de cálculo o resolución algorítmica ya esta calculado, y solo debemos juntar las soluciones de los sub problemas en una solución integral.

¿Por qué es importante el concepto de notación "O-grande" al analizar algoritmos? Jusitifica brevemente.

Use el editor para dar formato a la respuesta

La notación O-grande describe la complejidad de un algoritmo, en base a factores cómo tiempo de ejecución, lecturas, escrituras o uso de la memoria. Es sumamente importante

porque provee una herramienta para evaluar algoritmos similares y saber para qué es bueno uno y para qué otro.

Pregunta 6

10 puntos

Define el concepto de "balanceo" en árboles binarios y explica su importancia. Justifica brevemente.

Use el editor para dar formato a la respuesta

Un árbol binario puede ser evaluado mediante la métrica del balance. Como un nodo puede tener hasta dos hijos, podemos evaluar el balance de este nodo con la diferencia entre la altura del subárbol(nodo descendiente + sus nodos descendientes) izquierdo contra el derecho. Un árbol no balanceado hace la búsqueda de elementos muy lenta y pesada, mientras qué uno si balanceado permite una búsqueda sencilla y rápida.

Pregunta 7

10 puntos

¿Cuándo es apropiado utilizar un algoritmo de fuerza bruta para resolver un problema? Justifica brevemente.

Use el editor para dar formato a la respuesta

La estrategia algorítmica de la fuerza bruta es apropiada para problemas simples, o problemas dónde la resolución de los sub-problemas afecten la respuesta de otros. Por ejemplo, resolución de ecuaciones o una simulación física.

Pregunta 8

10 puntos

¿En qué situaciones es preferible utilizar la búsqueda en amplitud (BFS) en lugar de la búsqueda en profundidad (DFS)? Justifica brevemente.

Use el editor para dar formato a la respuesta

Es preferible usar BFS cuando no se quiere usar el stack, o cuando se trabaja con un grafo cíclico. Usando BFS podemos encontrar el camino más corto entre dos nodos del grafo.

Pregunta 9

10 puntos

Dado un arreglo de números enteros ordenados en orden creciente, salvo por dos elementos que fueron intercambiados, describe en palabras (no en código) como ordenarías, con un algoritmo, el arreglo de menor a mayor en tiempo $O(n)$. Supón que no hay elementos duplicados en el arreglo.

Un ejemplo:

```
Datos iniciales: { 3, 8, 6, 7, 5, 9 }

Resultado: { 3, 5, 6, 7, 8, 9 }
```

Desde la izquierda hacia la derecha del arreglo, reviso por un elemento que rompa la secuencia de menor a mayor. Desde la derecha hacia la izquierda del arreglo, reviso por un elemento qué rompa con la secuencia de mayor a menor. Una vez que encuentro esos dos elementos, los intercambio de lugar.

Pregunta 10

10 puntos

Describe en palabras (no en código) cómo ordenarías un arreglo de valores binarios (cero o uno), de menor a mayor, con un algoritmo de complejidad computacional $O(n)$.

Un ejemplo:

```
Datos iniciales: { 1, 0, 1, 0, 1, 0, 0, 1 }

Resultado: { 0, 0, 0, 0, 1, 1, 1, 1 }
```

Creo dos cursores, uno qué va de izquierda a derecha y uno de derecha a izquierda. Cuando el de izquierda encuentra un 1 y el de derecha encuentra un 0, los cambia de lugar, y luego sigue con el resto del arreglo.