Pregunta 2

0 / 10

Escribe una función en C++ que reciba un objeto de tipo `list<int>` por referencia (con `list` de STL), y devuelva el elemento central en la forma más eficientemente posible. Por ejemplo, si la lista contiene los elementos {4, 9, 3, 1, 6}, debe devolver el elemento 3; y si contiene {4, 9, 3, 1}, el elemento 9. Analiza la complejidad computacional de tu función. Justifica brevemente.

Pregunta 3

10 / 10

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.

Su respuesta
Puedo utiliar el algoritmo de Radix Sort, ya que al tener muchos elemento repetidos, es decir que la mayoria van a tener el mismo bit mas y menos significativo, facilitando enormemente su funcionamineto. La complejidad de este va a ser de O(na) ya que b es muy pequeno.

Pregunta 4

7 / 10

¿Cuáles son las diferencias fundamentales entre una lista enlazada y un arreglo en términos de la complejidad? Justifica brevemente.

Su respuesta
Una lista enlazada Es eficiente para la inserción y eliminación de elementos, pero no permite acceso por índice. Las listas enlazadas son O(n) para el acceso por índice, son O(1) para la inserción y eliminación. Los arreglos por otro lado, son muy utiles para acceder a elementos por indice

Comentarios para el estudiante

7/7/23 18:24

Pregunta 3:

Excelente.

Pregunta 4:

Y... las listas enlazadas requieren más memoria por los punteros de cada nodo; los arreglos requieren menos memoria.

Pregunta 5

0 / 10

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

Pregunta 6

5 / 10

¿Cuál es la complejidad computacional del recorrido de una tabla hash? Justifica brevemente.

Su respuesta
Para el recorrido de la tabla hash se necesita de una complejidad de O(n), ya que el acceso a los elementos es de O(1), pero por ejemplo al hacer rehashing, donde se tiene que recorrer la lista entera, se necesita ciclar por todos los elementos de la tabla, siendo este proceso de O(n).

7/7/23 18:24

Pregunta 6:

Es correcto que el recorrido de una tabla hash es $O(n)$, pero tu respuesta no define qué es n. Luego afirmas que "se necesita ciclar por todos los elementos de la tabla", dando a entender que n es la cantidad
de elementos almacenados en la tabla hash. Esto es incorrecto.

Pregunta 7

10 / 10

La función a continuación encuentra el índice del elemento máximo del arreglo `array`. Siendo que dispones de una máquina con múltiples cores, describe en palabras qué modificaciones realizarías sobre la función para mejorar su eficiencia.

```
int getMaxIndex(vector<int> array)
{
    int max = -1;
    int maxIndex = 0;

    for (int i = 0; i < array.size(); i++)
    {
        if (array[i] > max)
        {
            max = array[i];
            maxIndex = i;
        }
    }

    return maxIndex;
}
```

Su respuesta
Separaría al arreglo de tamaño n en múltiples pedazos, de esta forma hay menos comparaciones en el mismo momento. Luego por cada partición, lo que haría seria utilizar la estrategia de multithreading para resolver cada partición del arreglo y encontrar el elemento mas grande de cada uno, y luego comparo el elemento mas grande de cada uno de los resultados de cada thread, y obtengo el resultado que quiero. De esta forma al tener múltiples threads haciendo comparaciones, me permite hacer la comparación mas rápida y eficiente. Tendría mucho cuidado en la implementación con las operaciones atómicas, y por lo tanto usaria la herramienta de MUTEX para resolver esto.

Pregunta 8

0 / 10

Considera un grafo dirigido con ciclos. ¿Es posible utilizar el algoritmo de recorrido BFS para detectar la presencia de ciclos en el grafo? Explica tu respuesta y proporciona un ejemplo simple.

Pregunta 9

5 / 10

En un sistema de gestión de inventario de una tienda en línea, se desea ordenar los productos en función de su disponibilidad. ¿Qué algoritmo de ordenamiento sería más apropiado en este caso? Justifica brevemente.

Su respuesta
Utilizaría el pidheonholesort, ya que puedo agarrar cada item de la tienda por separado, e ir organizandolo dejando cada item de tal forma que me quedan separados aquellos que quedan en stock, con aquellos que no quedan en stock. Además, al separar los elementos de la tienda por clases (por ejemplo, articuos de cocina, bano, celulares) puedo ir ordenando cada elemento por indice, separando ademas los elementos por categoria y disponibilidad al mismo tiempo.

7/7/23 18:2

Pregunta 9:

Interpretaste "disponibilidad" como una decisión binaria. Entonces pidgeonhole sort es una buena elección. Pero no justificas por qué sería una buena elección.

Rta.: el algoritmo óptimo es pidgeonhole sort, ya que el espacio de búsquedas es extremadamente pequeño, de tamaño dos, por las posibilidades "si" y "no". Entonces el ordenamiento es $O(n)$. Otros algoritmos tienen complejidades mayores.

Pregunta 10

5 / 10

Pregunta 10

5 / 10

Supón que estás atrapado en un laberinto, con puertas y paredes dispuestas sobre una grilla. ¿Cómo puedes usar el algoritmo DFS para encontrar la salida? Discute las ventajas y desventajas de utilizar BFS en lugar de DFS. Justifica brevemente.

Su respuesta
Con el algoritmo de BFS voy a obtener un mejor resultado al pensar el camino como un grafo, usando el BFS voy recorriendo primero los caminos mas cercanos, y voy marcando el peso de cada camino hasta la meta y luego vuelvo para atrás reduciendo el peso y quedandome con el mejor camino. Pero con DFS si elijo un camino primero con un peso y termina siendo el total muy grande, me enteraria de esto al final del recorrido, y por lo tanto ternimo tardando de mas

No respondes a la pregunta de "Cómo puedes usar el algoritmo DFS para encontrar la salida". La respuesta es: Dado que el laberinto puede representarse con un grafo, un recorrido DFS asegura la visita de todos los nodos. Por tanto seguramente encontrará un nodo que se encuentra en el borde y que tenga una salida.

Es falso que con DFS "ternimo tardando de mas": esto depende de la estructura del árbol. Si el grafo se asemeja más a una estructura rizomática (https://es.wikipedia.org/wiki/Rizoma), será más eficiente BFS. Si el grafo posee muchas ramificaciones, será más eficiente DFS.

Pregunta 11

5 / 10

¿Cómo paralelizarías el algoritmo de ordenamiento mergesort a través del enfoque de computación distribuida? ¿Qué cuestiones particulares deberías atender? Justifica brevemente.

Su respuesta
Primero al separar el problema en multiples sub problemas voy a asignar cada sub problema a cada computadora por separado. El el medio utilizo un framework para implementar esta separacion de tareas entre las distintas computadoras. Y luego sincronizo para que se distribuyan las tareas, luego se espere a que todos terminen las comparaciones necesarias, y luego en orden se van juntando nuevamente los distintos subproblemas. En particular como mencione voy a tener mucho cuidado con no suceda un operacion de forma atomica, es decir que la operación de lectura y escritura no se realiza en un mismo paso.

En tu respuesta no explicas cómo separarías "el problema en multiples sub problemas" para el caso particular de merge-sort.

El problema de accesos atómicos se da en multithreading, no en computación distribuida.

Más problemas que deberías cuidar: el tráfico de red, los fallos de máquinas.

Todo lo demás, correcto.

Mostrar menos contenido