práctica para final

materia	prog II
creación	@June 15, 2023 11:09 AM

1. Dado un arreglo ordenado, considere una función que realice una búsqueda. Determinar:
    a. calcule orden de complejidad asintótica en ambos casos
    b. para un arreglo de 2048 elementos, cuántas interacciones se harán en el peor de los casos? justificar
2. Dada una lista generada al azar con valores cadena en el campo “DS”, borrar todas las ocurrencias de un elemento dado. El proceso será recursivo y deberá funcionar para cualquiera de las implementaciones vistas. NO se puede usar una lista aux para resolverlo.

    > Ejemplo: si la lista contiene (A, Z, B, H, K, R, A, H, Z) y se pide eliminar “H”, el resultado esperado será (A, Z, B, K, R, A, Z)

3. dada una lista que contiene registros con nombre de ciudad (DS) y cantidad de habitantes (DI), se pide realizar un programa que muestre el nombre de la ciudad o ciudades con mayor cantidad de habitantes. Las ciudades que difieran en más o en menos un 10% de habitantes, se consideran iguales. Indicar complejidad algorítmica

    > Por ejemplo dadas 3 ciudades: A=9100 h, B=10000 h, C=11000 h → salida sería B y C, son las ciudades con mayor cantidad de habitantes, pues la cantidad de habitantes de A difieren en un 20% con respecto a C

4. Qué tipos de recursividad existen?

---

1. Definir la estructura de datos necesaria para implementar una tabla hash con los siguientes requisitos:
   a. manejo de colisiones externo mediante listas dinámicas
   b. la tabla deberá almacenar como clave Tipo (A, B o C), Rubro un alfanumérico de 3 y Producto numérico de 4. Ejemplo Tipo=A, Rubro=ABF, Producto=1245
   c. encontrar función hash correspondiente para almacenar las claves
   d. el total de productos a almacenar no supera los 30000
2. Agregar al TAD una rutina que se llame `Arbol ArbolEspejo(Arbol)`, utilizar un recorrido iterativo (no recursivo). El algoritmo recibe “A” como árbol binario y la función retorna el nuevo árbol espejo.

---

1. nombrar y explicar las ventajas y desventajas de las diferentes formas de resolver las colisiones en las tablas hash
2. Definir la estructura de datos completa (sin implementación) necesaria para implementar una tabla hash con los siguientes requisitos:
   a. manejo de colisiones externo mediante zona de overflow (ZO)
   b. la tabla deberá almacenar como clave Tipo alfanumérico de 3 caracteres y Producto un numérico de 6. Ejemplo Tipo=PAX, Producto=321945
   c. encontrar función hash correspondiente para almacenar las claves
   d. el total de productos a almacenar no supera los 25000
3. Realizar función iterativa genérica que determine la diferencia sin repeticiones entre 2 conjuntos o multiconjuntos. Si el conjunto A = (1, 3, 5, 1, 2, 6) y B = (8, 3, 7, 5), la diferencia sin repeticiones de A-B = (1, 2, 6). La función se llamará `Conjunto RestaSinRepeticion(Conjunto A, Conjunto B)`

---

1. Dadas las siguientes claves generar un arbol ‘B’ de Orden 3, realizar todas las inserciones y eliminaciones identificando casos de unión o división de páginas.
   Insertar: 5, 8, 4, 12, 16, 50, 20, 5, 10, 90, 35, 24, 50, 18, 3
   Eliminar: 3, 18, 24
2. Realizar la implementacion de un Arbol Binario de Busqueda con Cursores.
3. Realizar dentro del TAD de Arboles una función recursiva genérica que reciba una posición del Arbol y devuelva la altura de la rama (la mas alta) de ese subArbol. La función debe estar implementada con lo realizado en el punto anterior. la funcion se invocara como `int AlturaDeNodo(ArbolBinario A, P:PosicionArbol)`
4. Definir la diferencia entre Tipo de Dato Primitivo, Tipo de Dato Definido por el usuario y TAD.

---

1. agregar al TAD una función que permita insertar un elemento ordenado de forma ascendente. Lista implementada mediante cursores. El campo por el cual se ordena es el numero entero (Campo DI del TipoElemento). La función se llamará como `void InsertarOrdenadoDI(Lista L, TipoElemento X)`
2. Definir la estructura de datos necesaria para implementar una tabla hash con los siguientes requisitos:
   a. manejo de colisiones externo mediante listas dinámicas
   b. la tabla deberá almacenar como clave un número de 5 dígitos más una fecha
   c. encontrar función hash correspondiente para almacenar un máximo de 150000 claves

- Retornar todos los nodos de un arbol N-Ario que pertenecen a un nivel determinado. La función recibirá como parámetros el árbol y el nivel, y retornará una lista.

---

1. generar una función recursiva que se llamará `buscarElementoPila(Pila pila, TipoElemento X, COMPARAPOR: CAMPOCOMPARAR)` que si el elemento buscado se encuentra en la pila devuelva su posición ordinal en la pila,

es decir: si la pila es (A,B,C,D) y se busca a 'C' entonces devuelve 3. (TOPE=A) NO SE DEBE DESTRUIR LA PILA ORIGINAL.

2. generar algoritmo iterativo genérico que indicará la cantidad de goleador/es de un torneo, devolviendo todos sus datos, los goles estarán en el campo DI, el DNI en el DR y el apellido en el DS. La función recibirá un argumento por referencia del tipo Lista L.
3.
    a. cómo será la complejidad del algoritmo del punto anterior sin tener en cuenta la implementación.
    b. cómo será la complejidad del algoritmo del punto anterior teniendo en cuenta la implementación.
4. Dada la siguiente definición sintáctica de lista:

    Estructura LISTA(L)

    ```
    Lista l_crear();
    void l_agregar(Lista lista, TipoElemento elemento)
    void l_borrar(Lista lista, int clave);
    ```

    CABEZA(lista) := ítem

    ```
    bool l_es_vacia(Lista lista);
    bool l_es_igual(Lista A, Lista B);
    ```

    definir la siguiente definición:

    `unsigned int cantidadParesLista(Lista);` (devuelve natural)

---

1. encontrar cantidad de claves mínimas para generar un árbol AVL donde se realicen las rotaciones en el siguiente orden: RID, RII, RDI, RDD - claves en orden y el árbol dibujado
2. agregar dentro del TAD de árboles binarios una función que permita determinar si un árbol es similar a otro. La función interna del TAD se llamará `bool arbol_similar (ArbolBinario A, Arbol Binario B)`. Debe ser lo más eficiente (tiempo y recursos)
3. agregar al TAD de listas con cursores una función que pueda insertar un dato (TE) según una posición ordinal (lógica). si puede realizar la inserción retornará la posición lista donde se insertó. La función se llama `int l_insertar_ordinal(Lista L, TipoElemento X, int pos)`

1. dada una lista, determinar la/las mayores claves de la misma. El proceso será genérico iterativo, se llamará `Lista RetornarMayores(Lista L)`. El resultado debe ser devuelto, no se puede mostrar dentro de la función. Complejidad
2. dada una pila determinar la menor clave de la misma. El dato de la clave es una cadena no repetible. La función será recursiva genérica. `Función TipoElemento MenorDePila(Pila P)`. No debe perderse pila original. En caso de que la pila esté vacía debe retornar el TipoElemento vacío. Complejidad
3. qué se debe tener en cuenta en el diseño de un proceso recursivo?
4. cuáles son las operaciones que no soportan apuntadores?

- Realizar una función genérica que permita determinar si un árbol binario cumple con las condiciones de un ABB. La función se llamará `bool EsArbolBinarioDeBusqueda(ArbolBinario)`

final 8/7 2023

1. Qué es factor de equilibrio y para qué se utiliza?
   es la diferencia de alturas entre el subárbol izquierdo y el subárbol derecho. los valores posibles para el FE son [-1;0;1]. fuera de esos valores significa que el árbol no está balanceado y se requiere reestructurarlo (desde el nodo cuyo FE se va de ese rango hacia abajo). se balancea el árbol para garantizar que la complejidad de la búsqueda se mantenga en O(log n), si no se parecería más a una lista, y los tiempos de búsqueda pueden degenerarse en O(n) (lineal)
2. Dentro de la TAD de listas cursores hacer la función `int insertarPosicionFisica(Lista L, TipoElemento X, int posicionFisica)`. Retornar la posicion ordinal o -1 si hay un error

> ❗ segun entendí, algunos hicieron eso de mover todo a la derecha, pero dijo que otra cosa que se podía hacer era que el elemento que estaba en esa posición moverlo al espacio libre, y hacer el enlace

```
int insertarPosicionFisica(Lista L, TipoElemento X, int PosicionFisica) {
  if (lista->cantidad == 0)
    return 0;
  else if (PosicionFisica >= 0 && PosicionFisica <= TAMANIO_MAXIMO -1) {
    // tomo el primer libre
    int p = lista->libre;
    lista->libre = lista->cursor[p].siguiente;
    // guardo el dato que hay actualmente en la pos fisica buscada
    lista->cursor[p].datos = lista->cursor[PosicionFisica].datos;
    // inserto el TE X en la pos fisica
    lista->cursor[PosicionFisica].datos = X;
    lista->cantidad++;

    // recorro lista para averiguar la pos ordinal de la pos fisica
    int pos_ordinal = 1;
    int actual = lista->inicio;
    while (actual != PosicionFisica && actual != NULO) {
      actual = lista->cursor[actual].siguiente;
      pos_ordinal++;
    }
    // conecto el elemento que estaba antes y mande a ocupar un libre
    lista->cursor[p].siguiente = lista->cursor[PosicionFisica].siguiente;
    // conecto el elemento de la pos fisica buscada con el anterior elemento
    lista->cursor[PosicionFisica].siguiente = p;

    return pos_ordinal;
  }
  return -1; // si la pos fisica no era valida, retorna error
}
```

3. Complejidad algoritmica del 2
4. árbol B orden 3
   insertar = 18, 14, 12, 22, 26, 20, 35, 10, 25, 24, 15, 18, 6
   eliminar = 14, 18, 20
5. Cuáles son las diferentes formas de implementar conjuntos?
   pueden implementarse tanto con listas como con árboles AVL

```
// arboles
struct ConjuntoRep {
    ArbolAVL arbolAvl;
};

struct ArbolAVLRep {
    NodoArbol raiz;
    int cantidad_elementos;
};
```

```
typedef struct ArbolAVLRep *ArbolAVL;

// listas
struct ConjuntoRep {
    Lista lista;
};

struct Nodo { // Se agrega el concepto de Nodo
    TipoElemento datos;
    struct Nodo *siguiente;
};

struct ListaRep { // Esta es la lista en esta implementación
    struct Nodo *inicio;
    int cantidad;
};

struct IteradorRep {
    struct Nodo *posicionActual;
};
```

```
int EliminarPosicionFisica(Lista lista, int PosicionFisica) {
  if (lista->cantidad == 0)
    return 0;
  if (PosicionFisica < 0 || PosicionFisica > TAMANIO_MAXIMO -1)
    return 0;
  int q;
  bool borrado = false;
  if (PosicionFisica == lista->inicio) {
    lista->inicio = lista->cursor[PosicionFisica].siguiente;
    lista->cursor[PosicionFisica].siguiente = lista->libre;
    lista->libre = PosicionFisica;
    borrado = true;
  } else {
    q = lista->inicio;
    int aux;
    while (q != NULO && lista->cursor[q].siguiente != NULO && !borrado) {
      aux = lista->cursor[q].siguiente;
      if (lista->cursor[q].siguiente == PosicionFisica){
        lista->cursor[q].siguiente = lista->cursor[PosicionFisica].siguiente;
        lista->cursor[PosicionFisica].siguiente = lista->libre;
        lista->libre = aux;
        borrado = true;
      } else {
        q = lista->cursor[q].siguiente;
      }
    }
  }
```

```
  if (borrado) {
    lista->cantidad--;
    return 1;
  } else {
    return 0;
    }
}
```

```
int EliminarPosicionFisica(Lista lista, int PosicionFisica) {
  if (lista->cantidad == 0)
    return 0;
  else if (PosicionFisica >= 0 && PosicionFisica <= TAMANIO_MAXIMO -1) {
    int actual = lista->inicio;
    if (lista->inicio == PosicionFisica) {
      lista->inicio = lista->cursor[actual].siguiente; // el nuevo inicio sera el siguient
e del inicio actual
      lista->cursor[actual].siguiente = lista->libre; // conecto el inicio eliminado con l
os libres
      lista->libre = actual; // el inicio eliminado es el primero de los libres
    }
    else {
      bool borrado = false;
      while (actual != NULO && !borrado) {
        if (lista->cursor[actual].siguiente == PosicionFisica) {
          lista->cursor[actual].siguiente = lista->cursor[PosicionFisica].siguiente; // co
necto el anterior de la posicion con el siguiente de la posicion a eliminar
          lista->cursor[PosicionFisica].siguiente = lista->libre // conecto el eliminado c
on los libres
          lista->libre = PosicionFisica; // ubico la posicion eliminada como el primer lib
re
          borrado = true;
        }
        else {
          actual = lista->cursor[actual].siguiente; // sigo recorriendo lista
        }
      }
    }
    lista->cantidad--;
    return 1;
  }
  else
    return 0;
}
```